LoRA Done RITE: Robust Invariant Transformation Equilibration for LoRA Optimization

Anonymous authors
Paper under double-blind review

Abstract

Low-rank adaption (LoRA) is a widely used parameter-efficient finetuning method for LLM that reduces memory requirements. However, current LoRA optimizers lack transformation invariance, meaning the actual updates to the weights depends on how the two LoRA factors are scaled or rotated. This deficiency leads to inefficient learning and sub-optimal solutions in practice. This paper introduces LoRA-RITE, a novel adaptive matrix preconditioning method for LoRA optimization, which can achieve transformation invariance and remain computationally efficient. We provide theoretical analysis to demonstrate the benefit of our method and conduct experiments on various LLM tasks with different models including Gemma 2B, 7B, and mT5-XXL. The results demonstrate consistent improvements against existing optimizers. For example, replacing Adam with LoRA-RITE during LoRA fine-tuning of Gemma-2B yielded 4.6% accuracy gain on Super-Natural Instructions and 3.5% accuracy gain across other four LLM benchmarks (HellaSwag, ArcChallenge, GSM8K, OpenBookQA).

1 Introduction

Low-Rank Adaptation (LoRA) (Hu et al., 2022) is a popular parameter-efficient fine-tuning method for Large Language Models (LLMs). By freezing the pretrained weights and injecting trainable low-rank decomposition matrices into each layer, LoRA significantly reduces memory requirements and mitigates overfitting in some limited data settings. More formally, let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ be a weight matrix in an LLM, LoRA freezes $\boldsymbol{W}$ and introduces a low-rank fine-tuned weight $\boldsymbol{Z}$ added to $\boldsymbol{W}$, where $\boldsymbol{Z}$ is represented by the multiplication of two thin matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, with a rank r,

$$\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{B}^T \in \mathbb{R}^{m \times n}, \boldsymbol{A} \in \mathbb{R}^{m \times r}, \boldsymbol{B} \in \mathbb{R}^{n \times r}. \tag{1}$$

These matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$, referred to as LoRA factors in this paper, have dimensions significantly smaller than the original weights. Recent research has explored numerous variations and improvements upon classic LoRA algorithm (Valipour et al., 2023; Zhang et al., 2023b; Liu et al., 2024; Yaras et al., 2024).

Despite being widely used in practice, we found that applying standard optimizers to LoRA leads to updates that are not "transformation invariant". By definition of LoRA in equation 1, the same update $\boldsymbol{Z}$ can be decomposed in multiple ways, i.e., $\boldsymbol{Z} = \boldsymbol{A}_1\boldsymbol{B}_1^T = \boldsymbol{A}_2\boldsymbol{B}_2^T$. Ideally, an optimizer should yield the same update to $\boldsymbol{Z}$ regardless of the specific factorization. However, commonly used optimizers like Adam (Kingma & Ba, 2014), Adagrad (Duchi et al., 2011), RMSProp (Tieleman & Hinton, 2012), and even second-order methods like Shampoo (Gupta et al., 2018), violate this principle when applied to LoRA. This violation not only presents a mathematical inconsistency but also leads to significant inefficiencies during training. In practice, we observe that one LoRA factor often dominates the optimization process, receiving substantial updates while the other remains nearly fixed. Although this can be partially mitigated by some recent proposed approaches such as employing different learning rates for two factors (Hayou et al., 2024), is there a more principled way to design an optimizer that inherently enforces transformation invariance for LoRA?

To address this challenge, we first prove that any form of diagonal preconditioner cannot achieve transformation invariance, which motivates the use of matrix preconditioners. However, existing matrix preconditioners like Shampoo lack transformation invariance and introduce significant computational and memory overhead. To overcome these limitations, we propose LoRA-RITE (Robust

Invariant Transformation Equilibration), a novel optimizer designed specifically for LoRA. LoRA-RITE employs a transformation-invariant preconditioner on the low-rank side, achieving transformation invariance without incurring substantial overhead. Furthermore, we demonstrate how to maintain this property when incorporating first and second moments, crucial for the practical effectiveness of adaptive optimization methods. Empirical evaluations across various datasets and models confirm the effectiveness of the proposed algorithm.

The contribution of this paper can be summarized below:

- We propose LoRA-RITE, the first adaptive matrix preconditioning optimizer for LoRA that is transformation-invariant, the property that is lacking for most existing optimizers when applying to LoRA. Theoretically, we provide a convergence analysis for our method.
- Despite utilizing matrix preconditioners, LoRA-RITE achieves little overhead in both memory and time compared to first-order optimizers, especially when the LoRA rank (r) is significantly smaller than the original matrix dimensions (m, n).
- The proposed optimizer leads to significantly improved performance across multiple datasets and architectures. For instance, when applied to the GSM8K (Cobbe et al., 2021) dataset with a Gemma 7B IT model (Gemma Team et al., 2024), LoRA-RITE achieves a 55.50% accuracy rate. This surpasses the widely-used Adam optimizer (Kingma & Ba, 2014) by a substantial margin (48.37%) and even outperforms the second-best optimizer on this dataset, Lamb (You et al., 2020) (50.64%), by approximately 5%.

2 TRANSFORMATION INVARIANCE FOR LORA OPTIMIZATION

This section introduces the concept of transformation invariance in LoRA training and demonstrates that most existing optimizers, when applied to LoRA, do not satisfy this property. This deficiency leads to inefficient learning in practice.

2.1 DEFINITION OF TRANSFORMATION INVARIANCE

As introduced in equation 1, LoRA adds a low-rank factor $\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{B}^T$ to the original weight matrix $\boldsymbol{W}$ and learns $\boldsymbol{A} \in \mathbb{R}^{m\times r}, \boldsymbol{B} \in \mathbb{R}^{n\times r}$ to minimize the fine-tuning loss. Observe that many different LoRA factors $(\boldsymbol{A}_1, \boldsymbol{B}_1), (\boldsymbol{A}_2, \boldsymbol{B}_2)$ can represent the same finetuned weight,

$$\boldsymbol{Z} = \boldsymbol{A}_1\boldsymbol{B}_1^T = \boldsymbol{A}_2\boldsymbol{B}_2^T. \tag{2}$$

When an optimizer is applied to train LoRA, it will produce different updates, $\delta\boldsymbol{A}_1, \delta\boldsymbol{B}_1$ or $\delta\boldsymbol{A}_2, \delta\boldsymbol{B}_2$, based on the specific parameterization used. Even though $(\boldsymbol{A}_1, \boldsymbol{B}_1)$ and $(\boldsymbol{A}_2, \boldsymbol{B}_2)$ represent the same finetuned weight $\boldsymbol{Z}$, those updates under different parameterizations can produce different updates to $\boldsymbol{Z}$. This suggests a serious mathematical inconsistency and implies that the update could be suboptimal under some parameterizations.

Based on this observation, we propose that LoRA optimization should ensure *transformation invariance*, defined as follows:

Definition 1 (Transformation Invariance)**.** *Let $(\boldsymbol{A}_1, \boldsymbol{B}_1)$ and $(\boldsymbol{A}_2, \boldsymbol{B}_2)$ be any two pairs of LoRA factors that satisfy equation 2. An optimizer exhibits transformation invariance if its updates, $(\delta\boldsymbol{A}_1, \delta\boldsymbol{B}_1)$ and $(\delta\boldsymbol{A}_2, \delta\boldsymbol{B}_2)$, satisfy*

$$(\boldsymbol{A}_1 + \delta\boldsymbol{A}_1)(\boldsymbol{B}_1 + \delta\boldsymbol{B}_1)^T = (\boldsymbol{A}_2 + \delta\boldsymbol{A}_2)(\boldsymbol{B}_2 + \delta\boldsymbol{B}_2)^T := \boldsymbol{Z} + \delta\boldsymbol{Z}. \tag{3}$$

This means the optimizer should produce the same update, $\delta\boldsymbol{Z}$, to the fine-tuned weights for any equivalent LoRA factorizations. To satisfy equation 3, the following equality should hold

$$\delta\boldsymbol{A}_1\boldsymbol{B}_1^T + \boldsymbol{A}_1\delta\boldsymbol{B}_1^T + \delta\boldsymbol{A}_1\delta\boldsymbol{B}_1^T = \delta\boldsymbol{A}_2\boldsymbol{B}_2^T + \boldsymbol{A}_2\delta\boldsymbol{B}_2^T + \delta\boldsymbol{A}_2\delta\boldsymbol{B}_2^T. \tag{4}$$

This leads to the following sufficient condition for transformation invariance, which will be used in our later derivations:

$$\delta\boldsymbol{A}_1\boldsymbol{B}_1^T = \delta\boldsymbol{A}_2\boldsymbol{B}_2^T,\ \boldsymbol{A}_1\delta\boldsymbol{B}_1^T = \boldsymbol{A}_2\delta\boldsymbol{B}_2^T,\ \delta\boldsymbol{A}_1\delta\boldsymbol{B}_1^T = \delta\boldsymbol{A}_2\delta\boldsymbol{B}_2^T. \tag{5}$$

As a special case, scalar scale invariance as introduced in Definition 2 is a weaker version of transformation invariance, which only requires that updates remain equivalent when the LoRA factors are scaled up or down by a constant factor. Formally, we define it as:

Definition 2 (Scalar Scale Invariance). *Let $(\boldsymbol{A}_1, \boldsymbol{B}_1)$ be a pair of LoRA factors and let $\boldsymbol{A}_2 = s\boldsymbol{A}_1, \boldsymbol{B}_2 = (1/s)\boldsymbol{B}_1$ for some nonzero constant s. An optimizer exhibits scalar scale invariance if its updates, $(\delta\boldsymbol{A}_1, \delta\boldsymbol{B}_1)$ and $(\delta\boldsymbol{A}_2, \delta\boldsymbol{B}_2)$, satisfy*

$$(\boldsymbol{A}_1 + \delta\boldsymbol{A}_1)(\boldsymbol{B}_1 + \delta\boldsymbol{B}_1)^T = (\boldsymbol{A}_2 + \delta\boldsymbol{A}_2)(\boldsymbol{B}_2 + \delta\boldsymbol{B}_2)^T.$$

Surprisingly, we will show that most commonly used optimizers, when applied to LoRA, do not even satisfy this weaker form of transformation invariance.

2.2 Existing Optimizers are not Scalar Scale Invariant

In this subsection we show both gradient descent and Adam are not scalar scale invariant, and in fact, almost all the existing optimizers are not scalar scale invariant when applying to LoRA.

For gradient descent, let $\boldsymbol{A}_2 = s\boldsymbol{A}_1, \boldsymbol{B}_2 = (1/s)\boldsymbol{B}_1$, by chain rule, since $\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{B}^T$, we have

$$\nabla\boldsymbol{A}_1 = \nabla\boldsymbol{Z}\boldsymbol{B}_1, \nabla\boldsymbol{A}_2 = \nabla\boldsymbol{Z}\boldsymbol{B}_2 = (1/s)\nabla\boldsymbol{A}_1.$$

Consequently, for gradient descent, we have

$$\nabla\boldsymbol{A}_2\boldsymbol{B}_2^T = (1/s)\nabla\boldsymbol{A}_1\boldsymbol{B}_2^T = (1/s^2)\nabla\boldsymbol{A}_1\boldsymbol{B}_1^T.$$

Therefore, the first term in equation 4 scales by $1/s^2$, and we can similarly derive that the second term scales by s^2 while the third term remain identical. Therefore, gradient descent is not scalar scale invariant, and the gradient can be arbitrary large when s goes to 0 or infinity.

Can this issue be mitigated by adaptive updates such as Adam? The answer is no. Let

$$\delta_{\text{Adam}}\boldsymbol{A}_1 := \nabla\boldsymbol{A}_1/\sqrt{\nabla\boldsymbol{A}_1 \circ \nabla\boldsymbol{A}_1}$$

be the Adam update, where all the operations are element-wise, and we omit the momentum part for simplicity. We have

$$\delta_{\text{Adam}}\boldsymbol{A}_2 = \frac{(1/s)\nabla\boldsymbol{A}_1}{(1/s)\sqrt{\nabla\boldsymbol{A}_1 \circ \nabla\boldsymbol{A}_1}} = \delta_{\text{Adam}}\boldsymbol{A}_1.$$

As a result,

$$\delta_{\text{Adam}}\boldsymbol{A}_2\boldsymbol{B}_2^T = \delta_{\text{Adam}}\boldsymbol{A}_1\boldsymbol{B}_2^T = (1/s)\delta_{\text{Adam}}\boldsymbol{A}_1\boldsymbol{B}_1^T,$$

which means Adam also does not satisfy scalar scale invariance. Actually, one can see most of the existing optimizers, such as Adagrad (Duchi et al., 2011), RMSProp (Tieleman & Hinton, 2012), and Shampoo (Gupta et al., 2018) are not scale or transformation invariance.

2.3 Benefits of Transformation Invariance

Why is transformation invariance important? Beyond the mathematical argument that different parameterizations of the same weight update should be equivalent, we demonstrate that transformation invariance leads to more efficient feature learning.

The concept of efficient feature learning, introduced in (Hayou et al., 2024), describes the asymptotic training behavior of LoRA as the network width grows. Specifically, for LoRA, the update to the matrix $\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{B}^T$ can be decomposed into three parts

$$\begin{aligned}\delta\boldsymbol{Z} &= (\boldsymbol{A} + \delta\boldsymbol{A})(\boldsymbol{B}^T + \delta\boldsymbol{B}^T) - \boldsymbol{A}\boldsymbol{B}^T \\ &= \delta\boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{A}\delta\boldsymbol{B}^T + \delta\boldsymbol{A}\delta\boldsymbol{B}^T,\end{aligned}$$

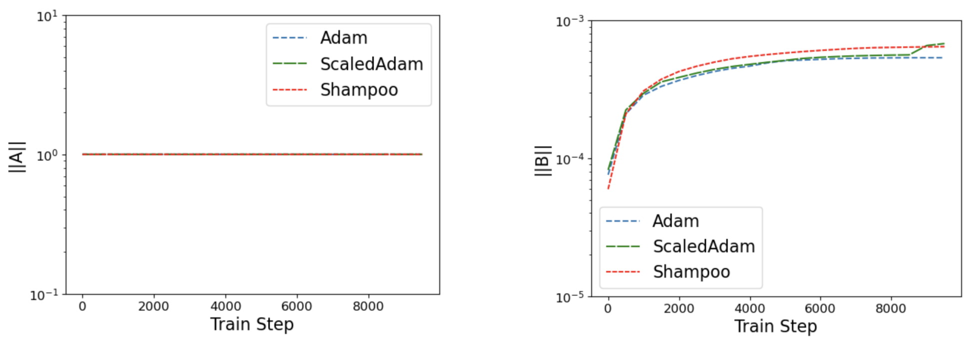
where the third term is typically negligible as it depends on the square of the learning rate. Efficient feature learning requires that both $\delta\boldsymbol{A}\boldsymbol{B}^T\boldsymbol{x}$ and $\boldsymbol{A}\delta\boldsymbol{B}^T\boldsymbol{x}$ is of $\boldsymbol{\theta}(n^0) = \boldsymbol{\theta}(1)$ with respect to the network width n, where $\boldsymbol{x}$ is the input embedding. In other words, let the scale be $\boldsymbol{\theta}(n^\alpha)$, it neither explodes ($\alpha > 0$) nor diminishes ($\alpha < 0$), when the network width n grows.

Hayou et al. (2024) show that conventional optimizer does not satisfy efficient feature learning. This can be seen from Figure 1, where the weight norm for factor $\boldsymbol{B}$ changes significantly while the weight norm for factor $\boldsymbol{A}$ barely changes.

Here we show that under mild assumptions, a transformation-invariant optimizer guarantees efficient feature learning. The proof is deferred to the appendix.

Theorem 1. *Any optimizer that is transformation-invariant and uses the same update rule for both $\boldsymbol{A}$ and $\boldsymbol{B}$ will achieve efficient feature learning.*

Beyond the efficient learning guarantee, in practice when training LoRA with existing optimizers, it's often the case that only one of the LoRA factors is updating properly, while the other remain almost unchanged, as shown in Figure 1. This is also a consequence of lacking scalar scale invariance, as the initial scale for the two LoRA factors can be very different (one from 0 while other from Gaussian random). we also show that the proposed algorithm as shown in the next section, which satisfies transformation invariant, achieves significantly improvements over previous LoRA optimizers in many empirical tasks.



(a) Weight norm of the $\boldsymbol{A}$ factor

(b) Weight norm of the $\boldsymbol{B}$ factor

Figure 1: The weight norm of $\boldsymbol{A}$ and $\boldsymbol{B}$ across different training steps.

3 Our Proposed Optimizer

3.1 Non-diagonal preconditioner is necessary for transformation invariance

We first show that using non-diagonal preconditioner matrix is necessary for achieving transformation invariance, which motivates the proposed algorithm.

Most existing optimizers utilize the following framework:

$$\mathrm{vec}(\delta \boldsymbol{A}) = \boldsymbol{P}\,\mathrm{vec}(\nabla \boldsymbol{A}), \tag{6}$$

where $\mathrm{vec}(\cdot)$ reshapes a matrix into a column vector and $\boldsymbol{P} \in \mathbb{R}^{mr \times mr}$ is a symmetric preconditioning matrix. Diagonal preconditioning methods like Adam and Adagrad assume $\boldsymbol{P}$ is a diagonal matrix, while matrix preconditioning methods such as Shampoo assume $\boldsymbol{P}$ is non-diagonal.

For transformation invariance property, we assume $\boldsymbol{A}_2 = \boldsymbol{A}_1\boldsymbol{R}, \boldsymbol{B}_2 = \boldsymbol{B}_1\boldsymbol{R}^{-T}$ where $\boldsymbol{R}$ is an invertible matrix, thus

$$\nabla \boldsymbol{A}_2 = \nabla \boldsymbol{Z}\boldsymbol{B}_2 = \nabla \boldsymbol{A}_1 \boldsymbol{R}^{-T}.$$

For simplicity, we consider the case where $n = m = 1$ to simplify the vectorization operation in equation 6, then we have

$$\delta \boldsymbol{A}\boldsymbol{B}^{T} = \nabla \boldsymbol{A}\boldsymbol{P}^{T}\boldsymbol{B}^{T}.$$

In this case, if we have two equivalent LoRA pairs $(\boldsymbol{A}_1, \boldsymbol{B}_1), (\boldsymbol{A}_2, \boldsymbol{B}_2)$ with their corresponding preconditioners $\boldsymbol{P}_1, \boldsymbol{P}_2$, transformation invariance implies

$$\delta \boldsymbol{A}_2\boldsymbol{B}_2^{T} = \nabla \boldsymbol{A}_2\boldsymbol{P}_2^{T}\boldsymbol{B}_2^{T} = \nabla \boldsymbol{A}_1(\boldsymbol{R}^{-T}\boldsymbol{P}_2\boldsymbol{R}^{-1})\boldsymbol{B}_1^{T} = \nabla \boldsymbol{A}_1\boldsymbol{P}_1^{T}\boldsymbol{B}_1^{T} \tag{7}$$

for arbitrary invertible $\boldsymbol{R}$. This implies $\boldsymbol{P}_1, \boldsymbol{P}_2$ cannot be diagonal, otherwise there exists an $\boldsymbol{R}$ to break the final equation of equation 7. We can easily extend this argument to the cases when $m, n > 1$ by looking at one row of $\boldsymbol{A}$ and $\boldsymbol{B}$.

Consequently, we have to adopt matrix preconditioning to achieve transformation invariance.

3.2 ACHIEVING TRANSFORMATION INVARIANCE

To achieve transformation invariance, we begin by recognizing that the LoRA weights, $\boldsymbol{A}$ and $\boldsymbol{B}$, can be decomposed into their respective orthogonal bases and magnitudes:

$$\boldsymbol{A} = \boldsymbol{U_A}\boldsymbol{R_A}, \quad \boldsymbol{B} = \boldsymbol{U_B}\boldsymbol{R_B},$$

where $\boldsymbol{U_A}$ and $\boldsymbol{U_B}$ can be obtained through QR decomposition. If $\boldsymbol{A}$ or $\boldsymbol{B}$ has a rank less than r, we can append orthogonal column vectors to $\boldsymbol{U_A}$ or $\boldsymbol{U_B}$ without affecting the results.

Observe that the gradients of $\boldsymbol{A}$ and $\boldsymbol{B}$,

$$\nabla \boldsymbol{A} = \nabla \boldsymbol{Z}\boldsymbol{B}, \quad \nabla \boldsymbol{B} = \nabla \boldsymbol{Z}^T\boldsymbol{A},$$

depend on both the basis and the magnitude. To achieve transformation invariance, we introduce the concept of "unmagnified gradients":

$$\bar{\nabla} \boldsymbol{A} \equiv \nabla \boldsymbol{Z}\boldsymbol{U_B} = \nabla \boldsymbol{A}\boldsymbol{R}_{\boldsymbol{B}}^{\dagger}, \quad \bar{\nabla} \boldsymbol{B} \equiv \nabla \boldsymbol{Z}^T\boldsymbol{U_A} = \nabla \boldsymbol{B}\boldsymbol{R}_{\boldsymbol{A}}^{\dagger}, \tag{8}$$

where $\boldsymbol{R}_{\boldsymbol{A}}^{\dagger}$ and $\boldsymbol{R}_{\boldsymbol{B}}^{\dagger}$ are the pseudo-inverse of $\boldsymbol{R_A}$ and $\boldsymbol{R_B}$. These unmagnified gradients, relying solely on the column spaces of $\boldsymbol{A}$ and $\boldsymbol{B}$, remain invariant to transformations of the LoRA weights. This invariance forms the cornerstone of our algorithm's ability to achieve transformation invariance.

Adaptive preconditioning methods like Adam have demonstrated superiority over non-adaptive methods like SGD. Furthermore, as established earlier, matrix preconditioning is crucial for achieving transformation invariance. Therefore, we propose utilizing these unmagnified gradients for adaptive matrix preconditioning. Additionally, we only precondition on the shorter side of size r, which ensures low time and memory complexity of the proposed method.

Since our update rule is symmetric for $\boldsymbol{A}$ and $\boldsymbol{B}$, for brevity, from now on we only describe the update rule for $\boldsymbol{A}$. For simplicity, let's first discuss the case without momentum. We propose the following update rule:

$$\delta \boldsymbol{A} = \eta \bar{\nabla} \boldsymbol{A}((\bar{\nabla} \boldsymbol{A})^T \bar{\nabla} \boldsymbol{A})^{-1/2}(\boldsymbol{R}_{\boldsymbol{B}}^{-T}), \tag{9}$$

where η is the learning rate. This update can be broken down into two parts. The first part

$$\bar{\nabla} \boldsymbol{A}((\bar{\nabla} \boldsymbol{A})^T \bar{\nabla} \boldsymbol{A})^{-1/2}$$

resembles the adaptive preconditioning mechanism in Adagrad, but employs matrix operations instead of element-wise operations. Crucially, the use of unmagnified gradients ensures this term remains consistent across all equivalent LoRA pairs, up to the choice of the basis.

The second part $\boldsymbol{R}_{\boldsymbol{B}}^{-T}$ adjusts the magnitude of the update for different LoRA pairs. Since $\boldsymbol{R}_{\boldsymbol{B}}^{-T}\boldsymbol{B}^T = \boldsymbol{U}_{\boldsymbol{B}}^T$, this effectively takes out the magnitude of $\boldsymbol{B}^T$ in $\delta \boldsymbol{A}\boldsymbol{B}^T$. We thus have

$$\begin{aligned} \delta \boldsymbol{A}_1\boldsymbol{B}_1^T &= \bar{\nabla} \boldsymbol{A}_1((\bar{\nabla} \boldsymbol{A}_1)^T \bar{\nabla} \boldsymbol{A}_1)^{-1/2}(\boldsymbol{R}_{\boldsymbol{B}_1}^{-T}\boldsymbol{B}_1^T) = \bar{\nabla} \boldsymbol{A}_1((\bar{\nabla} \boldsymbol{A}_1)^T \bar{\nabla} \boldsymbol{A}_1)^{-1/2}\boldsymbol{U}_{\boldsymbol{B}_1}^T \\ &= \bar{\nabla} \boldsymbol{A}_2(((\bar{\nabla} \boldsymbol{A}_2)^T \bar{\nabla} \boldsymbol{A}_2))^{-1/2}\boldsymbol{U}_{\boldsymbol{B}_2}^T = \delta \boldsymbol{A}_2\boldsymbol{B}_2^T. \end{aligned} \tag{10}$$

This demonstrates that our proposed method satisfies transformation invariance. Note that this simplified update rule does not yet incorporate accumulated first and second moments, which will be addressed in the following paragraphs.

Incorporating second moment. Adaptive optimizers typically employ accumulated second moments for preconditioning. A naive approach might involve replacing the $(\bar{\nabla} \boldsymbol{A})^T \bar{\nabla} \boldsymbol{A}$ term in equation 9 with its accumulated sum over history:

$$\sum_{t=1}^{T} (\bar{\nabla} \boldsymbol{A}_t)^T \bar{\nabla} \boldsymbol{A}_t.$$

However, since each $\bar{\nabla} \boldsymbol{A}_t$ is computed with respect to the basis at a specific step, directly summing them is mathematically unsound. Instead, we must account for the varying basis at each step. To achieve this, we accumulate the second moment as follows:

$$\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} = \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T + (\bar{\nabla} \boldsymbol{A}_t)^T \bar{\nabla} \boldsymbol{A}_t, \tag{11}$$

where $\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}$ is the accumulated second moment based on the previous basis at step $t-1$, and $\boldsymbol{P}_{\boldsymbol{A}_t} := (\boldsymbol{U}_{\boldsymbol{B}_t})^T\boldsymbol{U}_{\boldsymbol{B}_{t-1}}$ transforms it to the new basis at step t. During the adjustment,

$$\mathrm{Tr}(\boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T) \leq \mathrm{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}),$$

indicating a potential loss of information from the accumulated second moment. To quantify this loss, for symmetric positive definite matrices $\boldsymbol{X}_1, \boldsymbol{X}_2 \in \mathbb{R}^{r \times r}$, we define

$$d_\lambda(\boldsymbol{X}_1, \boldsymbol{X}_2) \equiv \max_i |\lambda_i(\boldsymbol{X}_1) - \lambda_i(\boldsymbol{X}_2)| \leq \min_{\boldsymbol{U}} \|\boldsymbol{X}_1 - \boldsymbol{U}\boldsymbol{X}_2\boldsymbol{U}^T\|,$$

where $\lambda_i(\boldsymbol{X})$ is the i-th eigenvalue of $\boldsymbol{X}$, and $\boldsymbol{U} \in \mathbb{R}^{r \times r}$ is an orthogonal matrix that reflects our freedom to choose the basis. We then define the "escaped mass" as

$$\rho_{\boldsymbol{A}_t} = \rho_{\boldsymbol{A}_{t-1}} + d_\lambda(\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}, \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T). \tag{12}$$

To compensate for this, we add $\rho_{\boldsymbol{A}_t}\boldsymbol{I}$ to our preconditioner, ensuring that

$$\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} + \rho_{\boldsymbol{A}_t}\boldsymbol{I}$$

monotonically increases under a suitable choice of basis, even though the choice of basis does not influence the actual update.

Finally, our unmagnified preconditioned step, when incorporating second moment, can be written as

$$\bar{\boldsymbol{S}}_{\boldsymbol{A}_t} = \bar{\nabla}\boldsymbol{A}_t(\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} + \rho_{\boldsymbol{A}_t}\boldsymbol{I})^{-1/2}. \tag{13}$$

Note that similar to Adam, we can turn equation 11 into the Exponential Moving Average (EMA) form, where we multiple the first term by $1-\beta_2$ and the second term by β_2, with the hyper-parameter $\beta_2 \in (0, 1)$ controls the decay rate. Additionally, for numerical stability we add a small $\epsilon\boldsymbol{I}$ to $\bar{\boldsymbol{V}}_{\boldsymbol{A}_t}$ before taking the inverse square root.

Incorporating first moment. Similar to the second moment, the first moment must also be adjusted for changes in the basis using a projection matrix. The update rule for maintaining the first moment can then be written as

$$\bar{\boldsymbol{M}}_{\boldsymbol{A}_t} = \beta_1\bar{\boldsymbol{M}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T + (1-\beta_1)\bar{\boldsymbol{S}}_{\boldsymbol{A}_t}.$$

Our final proposed update rule, incorporating both first and second moment, is

$$\delta\boldsymbol{A}_t = \bar{\boldsymbol{M}}_{\boldsymbol{A}_t}\boldsymbol{R}_{\boldsymbol{B}}^{-T}. \tag{14}$$

Algorithm 1 LoRA-RITE

1: Initialize: unmagnified first and second moment $\bar{\boldsymbol{M}}_{\boldsymbol{A}_0} = \boldsymbol{0}$, $\bar{\boldsymbol{V}}_{\boldsymbol{A}_0} = \boldsymbol{0}$
2: **for** $t = 1 \ldots T$ **do**
3: Compute the gradient $\nabla\boldsymbol{A}_t$;
4: QR decomposition over the LoRA factor $\boldsymbol{B}_t$: $\boldsymbol{B}_t = \boldsymbol{U}_{\boldsymbol{B}_t}\boldsymbol{R}_{\boldsymbol{B}_t}$;
5: Compute the unmagnified gradient $\bar{\nabla}\boldsymbol{A}_t = \nabla\boldsymbol{A}_t\boldsymbol{R}_{\boldsymbol{B}_t}^{-1}$ and $\boldsymbol{P}_{\boldsymbol{A}_t} = (\boldsymbol{U}_{\boldsymbol{B}_t})^T\boldsymbol{U}_{\boldsymbol{B}_{t-1}}$;
6: Update the unmagnified second moment $\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} = \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T + (\bar{\nabla}\boldsymbol{A}_t)^T\bar{\nabla}\boldsymbol{A}_t$;
7: Update the escaped mass $\rho_{\boldsymbol{A}_t} = \rho_{\boldsymbol{A}_{t-1}} + d_\lambda(\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}, \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T)$;
8: Compute the unmagnified precondition step $\bar{\boldsymbol{S}}_{\boldsymbol{A}_t} = \bar{\nabla}\boldsymbol{A}_t(\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} + \rho_{\boldsymbol{A}_t}\boldsymbol{I})^{-1/2}$;
9: Update the unmagnified first moment $\bar{\boldsymbol{M}}_{\boldsymbol{A}_t} = \beta_1\bar{\boldsymbol{M}}_{\boldsymbol{A}_{t-1}}\boldsymbol{P}_{\boldsymbol{A}_t}^T + (1-\beta_1)\bar{\boldsymbol{S}}_{\boldsymbol{A}_t}$;
10: Update model parameters $\boldsymbol{A}_{t+1} = \boldsymbol{A}_t - \eta_t\bar{\boldsymbol{M}}_{\boldsymbol{A}_t}\boldsymbol{R}_{\boldsymbol{B}}^{-T}$.
11: **end for**

The proposed algorithm, LoRA-RITE (**R**obust **I**nvariant **T**ransformation **E**quilibration for LoRA training), is summarized as Algorithm 1, where we show the updates for $\boldsymbol{A}$, and update for $\boldsymbol{B}$ can be derived in the same way. Note that we have shown that the main update rule of equation 13 satisfies transformation invariance, and this property can be extended even after adding the first and second moment into the algorithm, as shown in the following theorem (proof in Appendix).

Theorem 2. *In Algorithm 1, every unmagnified terms are consistent across all equivalent LoRA pairs. Consequently, Algorithm 1 is transformation invariant.*

Time and Space Complexity The time and space complexity of our algorithm is similar to first order methods like Adam when $r \ll m, n$. In each iteration of Algorithm 1, the dominant computational costs arise from (1) QR-decomposition for m-by-r and n-by-r matrices which takes $O(nr^2 + mr^2)$ time, (2) matrix inverses and roots for r-by-r matrices which takes $O(r^3)$ time, and (3) matmuls with time complexity $O(nr^2 + mr^2)$. Thus, the overall complexity per step is $O(mr^2 + nr^2 + r^3)$. It is only r times slower than Adam, and since r is very small, this overhead is negligible when comparing with the back-propagating time. The memory cost of our method is $O(mr + nr)$ which is the same as Adam. We summarize the time and space complexity of our method versus some commonly used optimizers in Table 7 in the Appendix.

3.3 Theoretical Analysis

Following previous work (Gupta et al., 2018; Feinberg et al., 2023), we provide a convergence analysis of the proposed algorithm within the online optimization framework (Hazan et al., 2016; Shalev-Shwartz et al., 2012).

In online convex optimization setting, a parameter $\boldsymbol{\theta}_t \in \mathcal{K}$ is chosen iteratively, where $\mathcal{K}$ is a convex decision set. After the decision of $\boldsymbol{\theta}_t$, a convex loss function f_t is revealed, potentially chosen adversarially. The regret accumulated by the algorithm up to step T is defined as

$$\text{Regret}_T = \sum\nolimits_{t=1}^{T} f_t(\boldsymbol{\theta}_t) - \min_{\boldsymbol{\theta}\in\mathcal{K}} \sum\nolimits_{t=1}^{T} f_t(\boldsymbol{\theta}).$$

In the online convex optimization analysis, we bound the first-order condition $\nabla\boldsymbol{\theta}_t^T(\boldsymbol{\theta}_t - \boldsymbol{\theta}^*)$ where $\boldsymbol{\theta}^*$ represents an arbitrary minimizer, and then use convexity to connect it to the loss function. However, due to the inherent structure of LoRA, loss functions f are not convex with respect to $\boldsymbol{\theta}$. Therefore, we directly bound the first-order condition instead.

We assume for the fine-tuned weight $\boldsymbol{Z}$ of each layer, the convex decision set imposes the following constrains:

$$\|\boldsymbol{A}\|_F \le D_{\boldsymbol{A}}, \|\boldsymbol{B}\|_F \le D_{\boldsymbol{B}},$$

where $\|\cdot\|$ denotes the Frobenius norm. Additionally, we assume the gradient satisfies $\|\nabla\boldsymbol{Z}\|_F \le G$. Following previous work, we analyze convergence in the simplified scenario where the first moment is omitted and the second moment is a summation, similar to Adagrad. For LoRA-RITE, our theoretical analysis yields the following result:

Theorem 3. *LoRA-RITE satisfies:*

$$\frac{1}{T}\sum\nolimits_{t=1}^{T} \frac{1}{\eta}\nabla\boldsymbol{\theta}_t^T(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t+1}) = O(GT^{-1/2}),$$

where η is a fixed constant learning rate.

This theorem shows that the method either converges to a particular stable solution or just move around in directions that does not change the function value, suggesting a form of convergence. To further strengthen the guarantee, we introduce an additional assumption:

Assumption 1. *Let*

$$\bar{\boldsymbol{X}}_{\boldsymbol{A}_t} = (\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} + \rho_{\boldsymbol{A}_t}\boldsymbol{I})^{-1/2}$$

be the unmagnified preconditioner $\boldsymbol{P}_{\boldsymbol{A}_t} = (\boldsymbol{U}_{\boldsymbol{B}_t})^T\boldsymbol{U}_{\boldsymbol{B}_{t-1}}$, and $\boldsymbol{Q}_{\boldsymbol{A}_t} = \boldsymbol{R}_{\boldsymbol{B}_t}^{-T}\boldsymbol{R}_{\boldsymbol{B}_{t-1}}^T$, then we have

$$\|\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1} - \boldsymbol{Q}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{Q}_{\boldsymbol{A}_t}^T\| \le \mu\|\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1} - \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{P}_{\boldsymbol{A}_t}^T\|.$$

This assumption essentially constrains the change in $\boldsymbol{R}_{\boldsymbol{B}_t}$ to be relatively smooth. Under this assumption, we can establish the following stronger convergence result:

Theorem 4. *Under Assumption 1, our proposed method satisfies:*

$$\frac{1}{T}\sum\nolimits_{t=1}^{T} \nabla\boldsymbol{\theta}_t^T(\boldsymbol{\theta}_t - \boldsymbol{\theta}^*) = O(GD_{\boldsymbol{A}}D_{\boldsymbol{B}}T^{-1/2}).$$

Our analysis closely resembles that of one-side matrix Adagrad. The key idea is to have a change of variable for both $\boldsymbol{A}$ and $\boldsymbol{B}$ such that all the quantities get replace by its unmagnified counterparts.

Compared to one-side matrix Adagrad, which has a regret bound of

$$O(G(D_{\boldsymbol{A}}^2 + D_{\boldsymbol{B}}^2)T^{-1/2}) \ge O(GD_{\boldsymbol{A}}D_{\boldsymbol{B}}T^{-1/2}),$$

our method has a better performance when the two LoRA factors exhibit imbalance magnitudes. This advantage is particularly relevant because previous work has shown that LoRA factors often exhibit such imbalances (Hayou et al., 2024), which can also be seen in Figure 1, providing an explanation for the strong empirical performance of our method.

4 Related Work

Related Optimizers Adaptive first-order optimizers like Adagrad (Duchi et al., 2011) utilize accumulated second moments, essentially diagonal preconditioners, to scale updates for each coordinate. This approach, adopted by optimizers like Adam (Kingma & Ba, 2014) and RMSProp (Tieleman & Hinton, 2012), has become the standard for training deep neural networks, including LoRA, and many other similar first-order methods have also been developed in the literature (Loshchilov & Hutter, 2017; Chen et al., 2024). However, as discussed in Section 3.1, these methods lack transformation invariance when applied to LoRA.

Several higher-order preconditioners have shown promise in various training scenarios (Shi et al., 2023). For example, Shampoo (Gupta et al., 2018) approximates the full second moment matrix using a Kronecker product, leading to the following preconditioned gradient:

$$L^{-1/4}GR^{-1/4}, \;\; L = L + GG^T, \; R = R + G^TG \tag{15}$$

where $L \in \mathbb{R}^{m\times m}, R \in \mathbb{R}^{n\times n}$ are the left and right preconditioner matrices, and $G \in \mathbb{R}^{m\times n}$ is the gradient. Many other higher-order methods follow this framework (Martens & Grosse, 2015; Morwani et al., 2024; Duvvuri et al., 2024). These methods incur $O(m^2 + n^2)$ additional memory overhead and require periodic computation of roots of L and R with $O(m^3 + n^3)$ computational cost. This complexity significantly exceeds that of our proposed method, as demonstrated in Table 7. Comparing equation 15 and equation 13 reveals that our method applies preconditioning only to the low-rank side of LoRA, resulting in negligible overhead. Furthermore, unlike our provably transformation-invariant approach, Shampoo-based methods lack this property.

Lars (You et al., 2017) and Lamb (You et al., 2020) are layer-wise adaptive optimization methods originally designed for large batch training. They dynamically adjust the update norm for each weight matrix based on its current norm, which ensure scalar scale invariance. Nonetheless, they still lacks transformation invariance.

Variants of LoRA As large language models (LLMs) grow in size, full fine-tuning on downstream tasks becomes increasingly resource-intensive. Parameter-efficient fine-tuning (PEFT) methods such as (Houlsby et al., 2019; He et al., 2022b;a; Lester et al., 2021; Li & Liang, 2021) have emerged to address this issue by reducing the number of trainable paramters. As a popular PEFT algorithm, LoRA (Hu et al., 2022) has been the subject of extensive research, with numerous variations and improvements proposed. One line of research focuses on dynamically adjusting the LoRA rank during training. This includes DyLoRA (Valipour et al., 2023), IncreLoRA (Zhang et al., 2023a), and AdaLoRA (Zhang et al., 2023b). Another approach involves enhancing LoRA performance through the addition of extra scaling matrices, which includes DoRA (Liu et al., 2024) and DeepLoRA (Yaras et al., 2024). These directions are orthogonal to our work.

Regarding LoRA optimization, Hayou et al. (2024) highlight the limitations of traditional optimizers as they fail to achieve efficient feature learning. To address this issue, they propose LoRA+, which uses two different learning rates $\eta_{\boldsymbol{A}}$ and $\eta_{\boldsymbol{B}}$ for LoRA weights. However, this leads to an extra hyperparameter to be tuned in practice. In contrast, Zhang & Pilanci (2024) propose the use of matrix preconditioning methods to achieve efficient feature learning. They propose the use of Riemannian gradient descent for LoRA optimization. As far as we know, Riemannian gradient descent is the only method in the literature that satisfies transformation invariance. However, similar to gradient descent, Riemannian gradient descent does not incorporate momentum and adaptivity, so it performs worse than Adam in their experiments. To improve the performance, they propose to combine Riemannian gradient descent with element-wise Adam, which becomes ScaledAdam. However, this combination makes ScaledAdam no longer transformation invariant.

5 Experiments

We evaluated the proposed LoRA optimizer against other optimizers across a range of datasets. This included the Super-Natural Instructions dataset, a comprehensive collection of diverse NLP tasks, as well as four standard LLM benchmarking datasets.

We compare the following optimizer:

- Adam (Kingma & Ba, 2014): The most widely used default optimizer for LoRA finetuning.

Table 1: Experimental results on the Super-Natural instruction dataset.

Model	Optimizer	Cause Effect Classification	Coreference Resolution	Title Generation	Data to Text	Global
Gemma-2B	Adam	58.93	77.06	51.30	55.52	50.51/74.54
	LoRA+	58.84	76.08	51.32	55.68	49.76/74.20
	ScaledAdam	58.71	77.55	51.16	55.69	49.40/74.01
	Shampoo	58.11	77.17	51.30	55.48	50.79/74.74
	Lamb	60.97	80.69	52.26	55.85	53.53/76.43
	LoRA-RITE	**61.26**	**82.02**	**52.26**	**55.98**	**55.11/77.12**
Gemma-7B	Adam	67.17	86.05	51.58	55.38	58.46/78.17
	LoRA+	65.50	86.67	51.51	55.34	58.19/78.29
	ScaledAdam	65.79	85.05	51.61	55.40	57.32/77.92
	Shampoo	66.29	85.62	51.86	55.43	57.99/78.27
	Lamb	69.62	86.57	51.87	55.5	57.79/78.18
	LoRA-RITE	**71.26**	**88.14**	**52.17**	**55.62**	**59.71/79.05**

Table 2: Experimental results on LLM benchmarking datasets.

Model	Optimizer	HellaSwag	ArcChallenge	GSM8K	OpenBookQA	Avg.
Gemma-2B	Adam	83.76	45.31	24.26	64.0	54.33
	LoRA+	83.75	45.31	23.65	64.4	54.28
	ScaledAdam	83.52	45.22	23.96	64.8	54.38
	Shampoo	83.26	44.88	23.35	63.6	53.77
	Lamb	86.60	47.35	26.76	68.0	57.18
	LoRA-RITE	**87.28**	**49.06**	**30.10**	**68.8**	**58.81**
Gemma-7B	Adam	94.07	54.78	48.37	77.60	68.71
	LoRA+	93.99	54.01	48.75	77.60	68.59
	ScaledAdam	93.31	52.90	48.07	75.80	67.52
	Shampoo	94.15	52.47	49.05	76.80	68.12
	Lamb	95.11	69.80	50.64	83.20	74.69
	LoRA-RITE	**95.59**	**71.76**	**55.50**	**84.80**	**76.91**

- LoRA+ (Hayou et al., 2024): Adam with different learning rate for $\boldsymbol{A}$ and $\boldsymbol{B}$. We set the learning of $\boldsymbol{B}$ to be 4 times large than $\boldsymbol{A}$, which is the value they used for decoder models.
- ScaledAdam (Zhang & Pilanci, 2024): A variant of Adam designed for LoRA optimization.
- Shampoo (Gupta et al., 2018): One of the most well-known adaptive matrix preconditioning method. To obtain a similar training time as the other methods, the block size is set to 512 and the preconditioners are updated every 100 steps.
- Lamb (You et al., 2020): A variant of Adam that normalizes the updates for each layer based on the norm of the parameters.
- LoRA-RITE: Our proposed optimizer that is transformation invariant.

For each optimizer applied on each dataset, we search for the best learning rate from $2 * 10^{-6}$ to $2 * 10^{-2}$. The other hyperparameters are listed in the appendix. For most of the experiments we chose rank $r = 16$ for LoRA, based on the ablation study over the rank at the end of experiments. We conduct experiments on Gemma (Gemma Team et al., 2024) 2B, 7B, and mT5-XXL (Xue et al., 2021) using TPUs.

Results on Super-Natural Instruction Dataset The Super-Natural instruction dataset (Wang et al., 2022) contains a collection of 1600+ NLP tasks, including both classification and generation tasks. We use a 10% split of the data for validation. Following (Wang et al., 2022), we use the exact match accuracy to evaluate classification and ROUGE-L score to evaluate generation tasks.

Table 1 presents the performance of individual fine-tuning on two classification and two generation tasks for 2,000 steps. It also includes the performance of fine-tuning on the global training set of over 1,600 tasks for 10,000 steps, reporting both exact match accuracy and ROUGE-L score evaluated on the global validation set. As shown in Table 1, our proposed method demonstrates superior performance across both classification and generation tasks. Compared to Adam, our method can achieve 2.3% to 4.9% accuracy improvements on the classification tasks and also shows significant improvements in the global training setting. Further, we found that Lamb performs well on some

Table 3: Ablation study on different ranks and different model architectures.

	Gemma-2B (rank=4)	Gemma-2B (rank=16)	mT5-XXL (rank=4)	mT5-XXL (rank=16)
Adam	63.00	64.0	72.00	72.20
ScaledAdam	63.00	64.8	70.80	74.60
Lamb	67.80	68.0	70.40	73.40
LoRA-RITE	**70.40**	**68.8**	**74.80**	**75.00**

Table 4: Number of training steps per second for different optimizers. LoRA-RITE has small overhead compared with first-order methods.

	Adam	LoRA+	ScaledAdam	Shampoo	Lamb	LoRA-RITE
Gemma-2B	0.948	0.930	0.917	0.837	0.929	0.878
Gemma-7B	0.120	0.120	0.114	0.112	0.116	0.114

of the datasets but there's still a significant gap between Lamb and LoRA-RITE. Since Lamb enforces scalar scale invariance but not transformation invariance, this result implicitly suggests that transformation invariance is crucial for achieving optimal performance.

Results on other LLM Benchmarking Datasets We also evaluate the performance on common LLM benchmarking datasets, including HellaSwag (Zellers et al., 2019), ArcChallenge (Clark et al., 2018), GSM8K (Cobbe et al., 2021), and OpenBookQA (Mihaylov et al., 2018). The summary information of these datasets is in the appendix. The results are presented in Table 2. We can observe that the trend is similar to the SuperNatural instruction results, where LoRA-RITE achieves the best performance on all the datasets, and Lamb is usually the second best optimizer.

Ablation Study We conduct an ablation study on the choice of different LoRA ranks and model architectures. Specifically, we considered rank 4 and 16 on both Gemma 2B (decoder only) and mT5-XXL (encoder-decoder) on the OpenBookQA dataset. As we can see from Table 3, our proposed method performs consistently well across different LoRA ranks. Furthermore, our method can be successfully applied to mT5-XXL which has an encoder-decoder architecture, showing the generalizability of the proposed optimizer.

Training Speed Comparison We compare the training speed of different optimizers. Table 4 shows the number of training steps per second for different optimizers with LoRA rank 16 on the OpenBookQA dataset using TPUv5e. As we can see, LoRA-RITE is only 8% slower than Adam on Gemma 2B, while the difference decreases to 5% when model size increased to 7B. Also, Shampoo is slower than LoRA-RITE in this case despite it recomputes the preconditioner with much lower frequency (once every 100 steps). This is due to our approach of preconditioning only the low-rank side of the LoRA factors.

6 Conclusion

Current LoRA optimization techniques lack transformation invariance, meaning equivalent LoRA parameterizations can yield significantly different updates. This hinders efficient feature learning and often leads to suboptimal solutions in practice. We introduce a novel, transformation-invariant optimization algorithm with comparable time and memory overhead to Adam. Our algorithm consistently achieves higher accuracy than existing LoRA optimizers across diverse datasets and models.

Limitations Although this work introduces a better optimizer for LoRA, it is important to acknowledge that LoRA itself has limitations. For instance, LoRA has smaller representational power and may result in a minor performance decrease compared to full fine-tuning. Also, how to select rank to strike a good trade-off between efficiency and accuracy may be non-trivial in practice.

The work focuses on addressing transformation-invariance when the optimization problem can be written in the form of $f(AB)$, and this assumption may not hold for other parameter-efficient structures beyond LoRA. Applying LoRA-RITE to ensure transformation invariance for the other more complicated LoRA variants will be an interesting future direction.

REFERENCES

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 2612–2626, 2022.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36, 2024.

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:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Sai Surya Duvvuri, Fnu Devvrit, Rohan Anil, Cho-Jui Hsieh, and Inderjit Dhillon. Caspr: Combining axes preconditioners through kronecker approximation for deep learning. In *Forty-first International Conference on Machine Learning*, 2024.

Vladimir Feinberg, Xinyi Chen, Y Jennifer Sun, Rohan Anil, and Elad Hazan. Sketchy: Memory-efficient adaptive regularization with frequent directions. *arXiv preprint arXiv:2302.03764*, 2023.

Thomas Mesnard Gemma Team, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, and et al. Gemma. 2024. doi: 10.34740/KAGGLE/M/3301. URL `https://www.kaggle.com/m/3301`.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. In *Forty-first International Conference on Machine Learning*, 2024.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022a.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2184–2190, 2022b.

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 *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017. URL `https://api.semanticscholar.org/CorpusID:53592270`.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

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 *EMNLP*, 2018.

Depen Morwani, Itai Shapira, Nikhil Vyas, Eran Malach, Sham Kakade, and Lucas Janson. A new perspective on shampoo's preconditioner. *arXiv preprint arXiv:2406.17748*, 2024.

Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

T. Tieleman and G. Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning. 2012.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287, 2023.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, 2022.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer, 2021. URL `https://arxiv.org/abs/2010.11934`.

Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

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*, 2019.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. In *Forty-first International Conference on Machine Learning*, 2024.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*, 2023a.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023b.

A Appendix

A.1 Hyperparameters

Table 5 shows our hyperparameters. We set weight decay and dropout probability to 0 as our early experiments suggest that setting a non-zero value does not improve the performance of the baselines.

Table 5: The setting for hyperparameters.

Hyperparameter	Value
Learning rate	$2 * 10^{-6}$ to $2 * 10^{-2}$
Weight decay	0
Dropout prob	0
LoRA target	q_{proj}, k_{proj} , v_{proj}, o_{proj}
LoRA rank	16
LoRA α	16
Batch size	16
Train step	2000
LR schedule	Linear decay
Warmup step	100
Evaluation period	100
Momentum β_1	0.9
Second moment β_2	0.999

A.2 Dataset

Table 6 shows the summary information of the LLM benchmarking datasets. We use the test set to evaluate ArcChallenge, as it is much larger than the development set.

Table 6: Summary information of the LLM benchmarking datasets.

Dataset	#Train	#Dev	#Test	Split for Eval
HellaSwag	39905	10042	10003	Dev
ArcChallenge	1119	299	1172	Test
GSM8K	7473	NA	1319	Test
OpenBookQA	4957	500	500	Dev

A.3 Proof of Theorem 1

Let $\|\boldsymbol{A}_1\| = \boldsymbol{\theta}(n^a)$, $\|\boldsymbol{B}_1\| = \boldsymbol{\theta}(n^b)$, $\|\nabla \boldsymbol{Z}\| = \boldsymbol{\theta}(n^c)$, $\eta = \boldsymbol{\theta}(n^d)$, where η is the learning rate and n is the network width. Since $\boldsymbol{Z} = \boldsymbol{A}_1 \boldsymbol{B}_1^T$, from chain rule we know $\nabla \boldsymbol{A} = \nabla \boldsymbol{Z} \boldsymbol{B}$ and $\nabla \boldsymbol{B} = \nabla \boldsymbol{Z}^T \boldsymbol{A}$. Since the update rule is symmetric, we can express the updates as

$$\|\delta \boldsymbol{A}_1\| = \boldsymbol{\theta}(n^{xa+yb+zc+d}), \|\delta \boldsymbol{B}_1\| = \boldsymbol{\theta}(n^{xb+ya+zc+d}).$$

If the update rule is scalar scale invariant, then for any $\boldsymbol{A}_2 = n^\delta \boldsymbol{A}_1, \boldsymbol{B}_2 = n^{-\delta} \boldsymbol{B}_1$ we have

$$\|\delta \boldsymbol{A}_1\| \|\boldsymbol{B}_1\| = \|\delta \boldsymbol{A}_2\| \|\boldsymbol{B}_2\|,$$

which means

$$xa + (y+1)b + zc + d = x(a+\delta) + (y+1)(b-\delta) + zc + d,$$

thus $x\delta - (y+1)\delta = 0$ for all δ, which means $y = x - 1$. Consequently, we have

$$\|\delta \boldsymbol{A}_1\| \|\boldsymbol{B}_1\| = \boldsymbol{\theta}(n^{xa+(y+1)b+sc+d}) = \boldsymbol{\theta}(n^{xa+xb+sc+d}).$$

Table 7: Time and space complexity comparison for LoRA optimization.

Algorithm	Time Complexity	Space Complexity
Forward/Backward	$\Omega(nm)$	$\Omega(nm)$
Full Matrix Adagrad (Duchi et al., 2011)	$O(m^3r^3 + n^3r^3)$	$O(m^2r^2 + n^2r^2)$
Adam (Kingma & Ba, 2014)	$O(mr + nr)$	$O(mr + nr)$
Lamb (You et al., 2020)	$O(mr + nr)$	$O(mr + nr)$
Shampoo (Gupta et al., 2018)	$O(m^3 + n^3 + r^3)$	$O(m^2 + n^2 + r^2)$
KFAC (Martens & Grosse, 2015)	$O(m^3 + n^3 + r^3)$	$O(m^2 + n^2 + r^2)$
ScaledAdam (Zhang & Pilanci, 2024)	$O(mr^2 + nr^2)$	$O(mr + nr)$
LoRA-RITE (our proposed)	$O(mr^2 + nr^2)$	$O(mr + nr + r^2)$

Similarly, we have

$$\|\boldsymbol{A}_1\|\|\delta \boldsymbol{B}_1\| = \boldsymbol{\theta}(n^{xb+(y+1)a+sc+d}) = \boldsymbol{\theta}(n^{xb+xa+sc+d}).$$

Since these two are equal, we can achieve efficient feature learning

$$\|\boldsymbol{A}\|\|\delta \boldsymbol{B}\|\|\boldsymbol{x}\| = \|\delta \boldsymbol{A}\|\|\boldsymbol{B}\|\|\boldsymbol{x}\| = \boldsymbol{\theta}(1),$$

where $\boldsymbol{x}$ is the input vector, by selecting a proper learning rate $\eta = \boldsymbol{\theta}(n^d)$.

A.4 Proof of Theorem 2

For matrix $\boldsymbol{X}_{\boldsymbol{A}} \in \mathbb{R}^{m\times r}$, $\boldsymbol{H}_{\boldsymbol{A}} \in \mathbb{R}^{r\times r}$, we call them consistent if

$$\boldsymbol{X}_{\boldsymbol{A}} \boldsymbol{U}_{\boldsymbol{B}}^T \in \mathbb{R}^{m\times n}$$

and

$$\boldsymbol{U}_{\boldsymbol{B}} \boldsymbol{H}_{\boldsymbol{A}} \boldsymbol{U}_{\boldsymbol{B}}^T \in \mathbb{R}^{n\times n}$$

are respectively the same across all equivalent LoRA pairs.

First, one should note the fact that

$$\boldsymbol{U}_{\boldsymbol{B}} \boldsymbol{U}_{\boldsymbol{B}}^T$$

is the same across all equivalent pairs. Thus,

$$\boldsymbol{U}_{\boldsymbol{B}} (\bar{\nabla} \boldsymbol{A})^T \bar{\nabla} \boldsymbol{A} \boldsymbol{U}_{\boldsymbol{B}}^T = \boldsymbol{U}_{\boldsymbol{B}} \boldsymbol{U}_{\boldsymbol{B}}^T \nabla \boldsymbol{Z}^T \nabla \boldsymbol{Z} \boldsymbol{U}_{\boldsymbol{B}} \boldsymbol{U}_{\boldsymbol{B}}^T$$

implies $(\bar{\nabla} \boldsymbol{A})^T \bar{\nabla} \boldsymbol{A}$ is consistent.

This combined with the fact that $\boldsymbol{P}_{\boldsymbol{A}_t} \bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}} \boldsymbol{P}_{\boldsymbol{A}_t}^T$ is consistent if $\bar{\boldsymbol{V}}_{\boldsymbol{A}_{t-1}}$ is consistent and that $\bar{\boldsymbol{V}}_{\boldsymbol{A}_0} = \boldsymbol{0}$ implies $\bar{\boldsymbol{V}}_{\boldsymbol{A}_t}$ is consistent.

Lastly, since

$$\boldsymbol{U}_{\boldsymbol{B}} (\bar{\boldsymbol{V}}_{\boldsymbol{A}_t} + \rho_{\boldsymbol{A}_t} \boldsymbol{I})^{-1/2} \boldsymbol{U}_{\boldsymbol{B}}^T = (\boldsymbol{U}_{\boldsymbol{B}} \bar{\boldsymbol{V}}_{\boldsymbol{A}_t} \boldsymbol{U}_{\boldsymbol{B}}^T + \rho_{\boldsymbol{A}_t} \boldsymbol{U}_{\boldsymbol{B}} \boldsymbol{U}_{\boldsymbol{B}}^T)^{-1/2},$$

both $\bar{\boldsymbol{S}}_{\boldsymbol{A}_t}$ and $\bar{\boldsymbol{M}}_{\boldsymbol{A}_t}$ are consistent, which completes our proof.

A.5 Proof of Theorem 3

For convenience, for matrix $\boldsymbol{X} \in \mathbb{R}^{m\times r}$, $\boldsymbol{H} \in \mathbb{R}^{r\times r}$, we define

$$\|\boldsymbol{X}\|_{\boldsymbol{H}} = \mathrm{Tr}(\boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^T)^{1/2}.$$

We also utilize the following lemma for online optimization.

Lemma 1 (Lemma 5.13 Hazan et al. (2016))**.** *For online optimization, if $\boldsymbol{\theta}_t$ is updated as $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{X}_t \boldsymbol{g}_t$, then we have*

$$\sum_{t=1}^{T} \nabla \boldsymbol{\theta}_t^T (\boldsymbol{\theta}_t - \boldsymbol{\theta}^*) \leq \frac{1}{2\eta} \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_*\|^2_{\boldsymbol{X}_1^{-1}} + \frac{\eta}{2} \sum_{t=1}^{T} (\boldsymbol{g}_t)^T \boldsymbol{X}_t \boldsymbol{g}_t$$
$$+ \frac{1}{2\eta} \sum_{t=2}^{T} (\boldsymbol{\theta}_t - \boldsymbol{\theta}_*)^T (\boldsymbol{X}_t^{-1} - \boldsymbol{X}_{t-1}^{-1}) (\boldsymbol{\theta}_t - \boldsymbol{\theta}_*).$$

Lemma 2 (Lemma 5.13, 5.14 Hazan et al. (2016)). *For arbitrary matrix $\boldsymbol{G}_t \in \mathbb{R}^{m\times r}$, $\boldsymbol{H}_t = \sum_{i=1}^t \boldsymbol{G}_i^T \boldsymbol{G}_i$, we have*

$$\sum_{t=1}^T \|\boldsymbol{G}_t\|_{\boldsymbol{H}_t^{-1/2}} \le 2\operatorname{Tr}(\boldsymbol{H}_T^{1/2})$$

Proof of Theorem 3

Since we are preconditioning each layer independently, all three terms in Lemma 1 can be written as summation over the L layers. For simplicity, from now on we omit the summation and the subscript for layers.

For our method, the preconditioner $\boldsymbol{X}_{\boldsymbol{A}_t}$ is as follows,

$$\boldsymbol{X}_{\boldsymbol{A}_t} = \boldsymbol{R}_{\boldsymbol{B}_t}^{-1}\bar{\boldsymbol{V}}_{\boldsymbol{A}_t}^{-1/2}\boldsymbol{R}_{\boldsymbol{B}_t}^{-T}.$$

We define the unmagnified preconditioner

$$\bar{\boldsymbol{X}}_{\boldsymbol{A}_t} = \bar{\boldsymbol{V}}_{\boldsymbol{A}_t}^{-1/2}.$$

Then by Lemma 2, for the $\boldsymbol{A}$ factor, we have

$$\begin{aligned}\sum_{t=1}^T \operatorname{vec}(\nabla\boldsymbol{A}_t)^T\operatorname{vec}(\delta\boldsymbol{A}_t) &= \sum_{t=1}^T \operatorname{Tr}(\nabla\boldsymbol{A}_t^T\delta\boldsymbol{A}_t)\\ =&\eta\sum_{t=1}^T\|\nabla\boldsymbol{A}_t\|^2_{\boldsymbol{X}_{\boldsymbol{A}_t}} = \eta\sum_{t=1}^T\|\bar{\nabla}\boldsymbol{A}_t\|^2_{\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}} \le 2\eta\operatorname{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_T}^{1/2}).\end{aligned} \tag{16}$$

Since

$$\operatorname{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_T}^{1/2}) = O(GT^{1/2}),$$

this completes our proof.

Proof of Theorem 4

We continue from the proof of Theorem 3 and utilize Lemma 1. We already bound the second term in Theorem 3, so we only need to bound the third term.

For the third term, we have

$$\begin{aligned}\|\boldsymbol{A}_t-\boldsymbol{A}_*\|^2_{\boldsymbol{X}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{X}_{\boldsymbol{A}_{t-1}}^{-1}} &= \|(\boldsymbol{A}_t-\boldsymbol{A}_*)\boldsymbol{R}_{\boldsymbol{B}_t}^T\|^2_{\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{Q}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{Q}_{\boldsymbol{A}_t}^T}\\ \le& D_{\boldsymbol{A}}^2 D_{\boldsymbol{B}}^2\|\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{Q}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{Q}_{\boldsymbol{A}_t}^T\| \le \mu D_{\boldsymbol{A}}^2 D_{\boldsymbol{B}}^2\|\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{P}_{\boldsymbol{A}_t}^T\|,\end{aligned}$$

where the last inequality comes from our assumption.

Consequently, since

$$\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1} \succeq \boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{P}_{\boldsymbol{A}_t}^T,$$

we have

$$\sum_{t=1}^T\|\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{P}_{\boldsymbol{A}_t}^T\| \le \sum_{t=1}^T\operatorname{Tr}(\bar{\boldsymbol{X}}_{\boldsymbol{A}_t}^{-1}-\boldsymbol{P}_{\boldsymbol{A}_t}\bar{\boldsymbol{X}}_{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{P}_{\boldsymbol{A}_t}^T) \le \operatorname{Tr}(\bar{\boldsymbol{X}}_{\boldsymbol{A}_T}^{-1}) = \operatorname{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_T}^{1/2}).$$

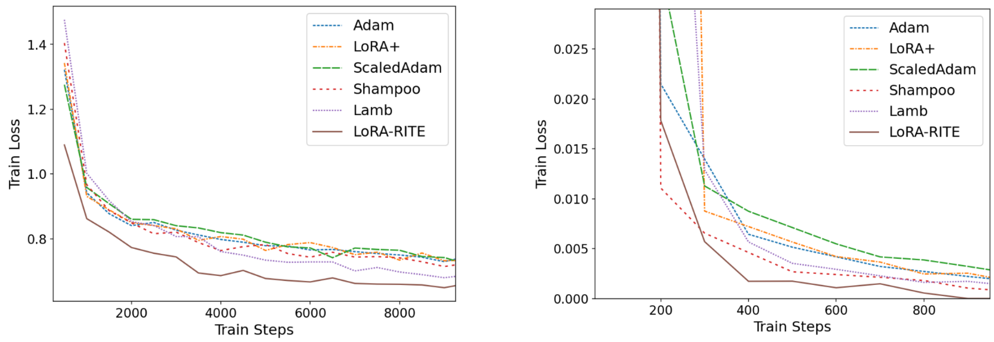
Summing up the second and third term, we get

$$(2\eta + \frac{1}{\eta}\mu D_{\boldsymbol{A}}^2 D_{\boldsymbol{B}}^2)\operatorname{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_T}^{1/2}).$$

Choosing $\eta = (1/\sqrt{2})\mu^{1/2}D_{\boldsymbol{A}}D_{\boldsymbol{B}}$, we have

$$2\sqrt{2}\mu^{1/2}D_{\boldsymbol{A}}D_{\boldsymbol{B}})\operatorname{Tr}(\bar{\boldsymbol{V}}_{\boldsymbol{A}_T}^{1/2}) = O(D_{\boldsymbol{A}}D_{\boldsymbol{B}}GT^{-1/2}),$$

which completes the proof.



(a) Super-Natural instruction

(b) OpenBookQA

Figure 2: The training loss curve for the Super-Natural instruction dataset and the OpenBookQA dataset.

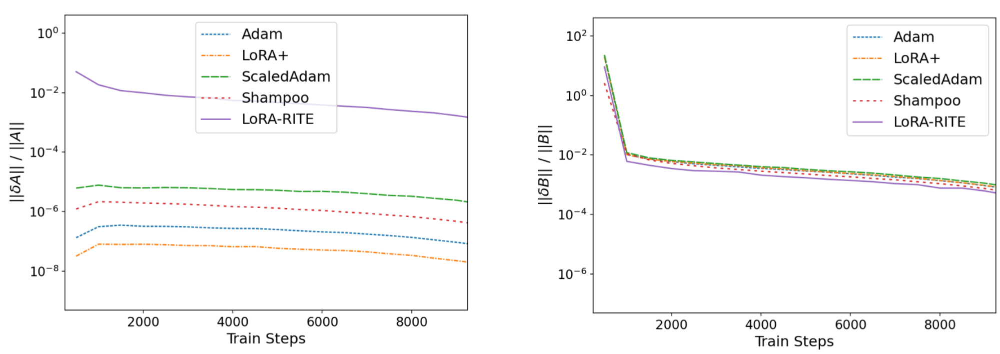
A.6 Training Loss Curve Visualization

To cross-validate the effectiveness of LoRA-RITE, we plot the training loss curve of each method for the Super-Natural instruction dataset and the OpenBookQA dataset. Figure 2 shows that LoRA-RITE has the lowest training loss, which demonstrates the effectiveness of our method.

A.7 Update Magnitude Visualization

To visualize the update magnitude of the two LoRA factors, we plot the update norm divided by the weight norm, $\|\delta \boldsymbol{A}\|/\|\boldsymbol{A}\|$ and $\|\delta \boldsymbol{B}\|/\|\boldsymbol{B}\|$.

Figure 3 and Figure 4 show that for conventional optimizers, factor $\boldsymbol{A}$ barely changes, while LoRA-RITE is able to learn the factor $\boldsymbol{A}$ effectively. This demonstrates the importance of transformation invariance.



(a) Update magnitude of the $\boldsymbol{A}$ factor

(b) Update magnitude of the $\boldsymbol{B}$ factor

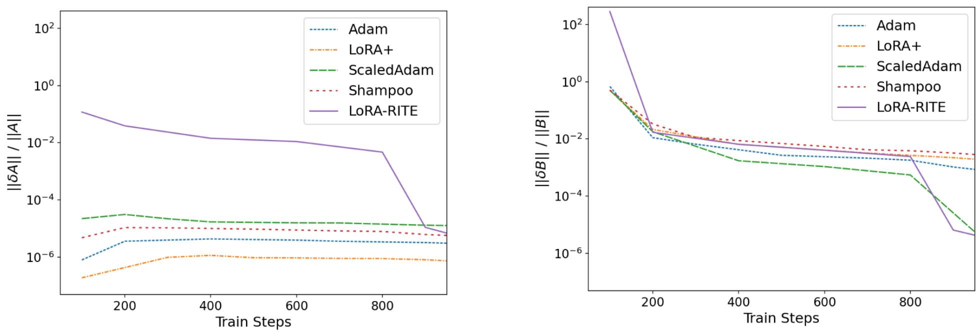
Figure 3: The update magnitude of $\boldsymbol{A}$ and $\boldsymbol{B}$ for the Super-Natural instruction dataset.

A.8 Ablation Study on Different Ranks

To study the effect of different LoRA ranks, we conduct additional ablation study on different datasets.

As we can see from Table 8, higher rank generally improves LoRA performance, approaching full fine-tuning. This explains why the performance gap between LoRA-RITE and other methods narrows at higher ranks, as they all converge towards the results of full fine-tuning.

Additionally, one can observe that LoRA has inherent regularization properties. As noted in previous research (Chen et al., 2022), this means that sometimes a lower rank can actually lead to



(a) Update magnitude of the $\boldsymbol{A}$ factor (b) Update magnitude of the $\boldsymbol{B}$ factor

Figure 4: The update magnitude of $\boldsymbol{A}$ and $\boldsymbol{B}$ for the OpenBookQA dataset.

Table 8: Ablation study on different ranks on Gemma-2B on the LLM benchmarking datasets.

Optimizer	Rank	HellaSwag	ArcChallenge	GSM8K	OpenBookQA	Avg.
Adam	$r = 4$	81.83	42.32	20.92	63.0	52.02
	$r = 16$	83.76	45.31	24.26	64.0	54.33
	$r = 64$	84.56	46.67	26.08	67.0	56.08
ScaledAdam	$r = 4$	81.95	44.80	21.15	63.0	52.73
	$r = 16$	83.52	45.22	23.96	64.8	54.38
	$r = 64$	84.42	48.21	26.61	67.0	56.56
Lamb	$r = 4$	86.01	46.67	25.25	67.8	56.43
	$r = 16$	86.60	47.35	26.76	68.0	57.18
	$r = 64$	87.83	47.53	29.04	62.8	56.80
LoRA-RITE	$r = 4$	87.08	49.57	29.49	70.4	59.14
	$r = 16$	87.28	49.06	30.10	68.8	58.81
	$r = 64$	87.89	49.91	31.46	68.8	59.52

better performance. This effect depends on factors like model generalization and training data size. This explains why LoRA-RITE achieves better performance at rank 4 instead of 16 and why Lamb achieves better performance at rank 16 than rank 64.

A.9 Best Learning Rate for Different Optimizers

In Table 9, we list the best learning rate for each optimizer on the LLM benchmarking datasets. We observe that LoRA-RITE and Lamb usually prefer a larger learning rate than the other baselines.

Table 9: Best Learning Rate for Different Optimizers on LLM benchmarking datasets.

Model	Optimizer	HellaSwag	ArcChallenge	GSM8K	OpenBookQA
Gemma-2B	Adam	1e-5	5e-5	1e-5	5e-5
	LoRA+	1e-5	5e-5	1e-5	5e-5
	ScaledAdam	5e-5	5e-5	1e-5	2e-4
	Shampoo	1e-5	5e-5	5e-5	5e-5
	Lamb	5e-3	5e-3	5e-3	5e-3
	LoRA-RITE	2e-4	1e-3	2e-4	2e-4