STABLE-LORA: STABILIZING FEATURE LEARNING OF LOW-RANK ADAPTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-Rank Adaption (LoRA) is a widely adopted parameter-efficient method for fine-tuning Large Langauge Models. It updates the weight matrix as $W = W_0 + sBA$, where W_0 is the original frozen weight, s is a scaling factor and A,B are trainable low-rank matrices. Despite its robust empirical effectiveness, the theoretical foundations of LoRA remain insufficiently understood, particularly with respect to feature learning stability. In this paper, we first establish that, LoRA can, in principle, naturally achieve and sustain stable feature learning (i.e., be self-stabilized) under appropriate hyper-parameters and initializations of A and B. However, we also uncover a fundamental limitation that the necessary non-zero initialization of A compromises self-stability, leading to suboptimal performances, while neither hyper-parameter tuning nor alternative initializations can resolve this instability. To address this challenge, we propose Stable-LoRA, a weight-shrinkage optimization strategy that dynamically enhances stability of LoRA feature learning. By progressively shrinking A during the earliest training steps, Stable-LoRA is theoretically proved and empirically validated to effectively eliminate instability of LoRA feature learning while preserving the benefits of the non-zero start. Experiments show that Stable-LoRA consistently outperforms other baselines across diverse models and tasks, with no additional memory usage and only negligible computation overheads.

1 INTRODUCTION

Low-Rank Adaption (LoRA) (Hu et al., 2022) is an effective and widely adopted parameter-efficient method for fine-tuning large language models (LLMs). Unlike full fine-tuning, which updates all model parameters, LoRA freezes the original weight matrix W_0 and introduces two low-rank trainable matrices, A and B, with the weight matrix updated by the multiplication of A and B. Formally,

$$W = W_0 + sBA, W_0 \in \mathbb{R}^{m\times n}, A \in \mathbb{R}^{r\times n}, B \in \mathbb{R}^{m\times r}$$

where s is a scaling factor. By choosing $r << \min(m, n)$, the number of trainable parameters is reduced from mn to $(m + n)r$, substantially lowering computation and memory overhead while retaining strong learning capacity.

The effectiveness of LoRA has been demonstrated through massive experiments across various models and tasks, and recent studies (Hayou et al., 2024b; Zhang & Pilanci, 2024; Kalajdzievski, 2023) have also begun to theoretically explore the fine-tuning dynamics of LoRA. However, no prior work has established a theoretical explanation for such robust effectiveness. In this paper, we first provide a theoretical analysis showing that, with appropriate hyper-parameters and initializations of A and B, LoRA can naturally achieves stable feature learning with respect to model width n (informally, the learned features scale as $\Theta(n^0)$). Furthermore, once this stability is achieved, it will be sustained throughout the entire training process. Such self-stabilizing property provides a theoretical foundation for the observed effectiveness and robustness of LoRA.

According to the analysis, the ideal initialization for ensuring self-stabilization is to set both A and B to zero. However, this leads to practical issues of saddle-point halting (Zhang & Pilanci, 2024), information loss and gradient vanishing/explosion (He et al., 2015). The mostly-adopted and theoretically proven-effective (Hayou et al., 2024a) solution is to initialize only B to zero and A non-zero. Nevertheless, we both theoretically and empirically demonstrate that such a non-zero

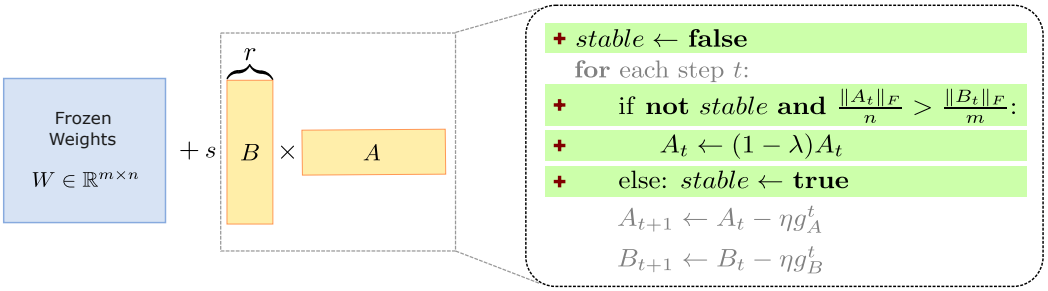


Figure 1: Illustration of Stable-LoRA. The weight-shrinkage operation is emphasized as a patch to the gradient-descent procedure.

initialization A_0 compromises stable feature learning and hence causes suboptimal performances, while tuning hyper-parameters or altering initial values cannot resolve it. This observation motivates the design of novel LoRA optimization strategies that must go beyond tuning or initialization.

To address this problem, we propose Stable-LoRA, a weight-shrinkage strategy for LoRA optimization that dynamically enhances the stability of feature learning. We observe from theoretical perspectives that the initialization-induced instability is a long-term problem whereas others are short-termed. Therefore, Stable-LoRA adopts the non-zero A_0 for its benefits and progressively shrinks A as training proceeds. Specifically, a shrinkage ratio λ ($0 < \lambda < 1$) is applied to A in the earliest steps of training, updating A according to

$$A_{t+1} = (1-\lambda)A_t - \eta g_A^t$$

(as shown in Figure 1).This exponential decay diminishes the instability introduced by A_0 while still preserving its advantages for early training. Shrinkage stops once the stability condition is satisfied—specifically, when the average norm of A becomes no larger than that of B (see Section 4). We also theoretically proved that sufficient shrinkage of A guarantees the prevention of potential instability, thereby ensuring stable feature learning throughout training.

We evaluated Stable-LoRA across different model architectures and tasks, where it uniformly outperforms AdamW and other baselines. Importantly, Stable-LoRA incurs no additional memory usage and introduces only negligible computational overhead—properties that are particularly important in the resource-constrained scenarios where LoRA is most commonly applied.

2 Preliminary

2.1 Feature learning of LoRA

Consider training a weight matrix W with input Z, such that the output is $Y = WZ$. In LoRA, the original weight is frozen as W_0 and two low-rank trainable matrices A and B are introduced, so that the updated weight becomes $W = W_0 + sBA$, where s is a scaling factor. Given a learning rate η, the parameter updates at training step t are:

$$A_{t+1} = A_t - \eta g_A^t, B_{t+1} = B_t - \eta g_B^t$$

, where g_A and g_B are the optimizer-processed gradients. The change in output after these updates is given by:

$$\begin{aligned} \Delta Y_t &= s(A_t - \eta g_A^t)(B_t - \eta g_B^t)Z - sA_tB_tZ \\ &= -s\eta g_B^t A_t Z - s\eta B_t g_A^t Z + s\eta^2 g_B^t g_A^t Z \end{aligned} \tag{1}$$

. We are particularly interested in the properties of ΔY_t, as it serves as the "learned feature" at step t. Specifically, ΔY_t directly influences the inputs to downstream layers and ultimately the model's output, representing the contribution of LoRA updates to the model.

2.2 Stable feature learning

As neural networks continue to scale, understanding their training dynamics with respect to parameter growth becomes increasingly important (Hayou et al., 2024b; Zhang & Pilanci, 2024; Hayou

et al., 2019). Much of this analysis has focused on the regime of model width, since in most architectures the parameter count is dominated by width, while depth (i.e., the number of layers) plays a comparatively smaller role. In this regime, a desirable property is that learned features remain "stable" as width increases—they neither explode nor vanish numerically. Such stability is crucial for ensuring meaningful representations can be learned, thereby allowing the model to achieve its full performance potential upon trained tasks.

In the context of LoRA, stable feature learning requires that the output update ΔY_t does not scale positively or negatively with model width n, otherwise it would explode or vanish as n increases. Formally, this requirement can be expressed as $\Delta Y_t = \Theta(n^0) = \Theta(1)$.

Definition 1. *(LoRA stable feature learning) LoRA feature learning is stable, if $\Delta Y_t = \Theta(1)$ for all training steps t.*

Note that Definition 1 is slightly different from similar concepts in prior works (Hayou et al., 2024b; Zhang & Pilanci, 2024). It does not require intermediate representations (e.g. $A_t Z$) to individually scale as $\Theta(1)$, but only constrains the final output update. This relaxation is motivated by practical considerations: for an actual (finite) n, the scale of intermediate representations can compensate for each other through multiplicative interactions and yield an overall stable output. For example, it is acceptable for components of ΔY_t to scale as $U = \Theta(n)$ and $V = \Theta(n^{-1})$, as long as $\Delta Y_t = UV = \Theta(1)$ is ensured.

2.3 γ-FUNCTION

For convenience of notation, we introduce γ-function to characterize the scaling behavior of variables with respect to the model width n. It is defined as follows: for a real-valued scalar variable v, we have $v = \Theta(n^{\gamma[v]})$. For a k-dimensional tensor variable $\vec{v} = (v_0, \cdots, v_{k-1})$, we define $\gamma[\vec{v}] := \max(v_i, 0 < i < k)$, which captures the dominant scaling behavior among its components.

By definition, γ-function obeys the following properties under element-wise operations:

Multiplication: For two real-valued variables v and v', $\gamma[v \times v'] = \gamma[v] + \gamma[v']$

Addition: For two real-valued variables v and v', $\gamma[v + v'] = \max(\gamma[v], \gamma[v'])$

With this notation, the condition for stable feature learning can be succinctly expressed as $\gamma[\Delta Y] = 0$.

2.4 OPTIMIZED GRADIENT

Modern optimizers such as Adam and AdamW (Kingma & Ba, 2014) are generally preferred over Stochastic Gradient Descent (SGD) in fine-tuning scenarios. These optimizers typically normalize gradients through momentum mechanisms (e.g., exponential moving averages in Adam), which effectively prevents the entries of gradients from becoming excessively small or large. In the following analysis, we assume that the normalized gradients have all entries to be $\Theta(1)$, a condition theoretically justified by the internal dynamics of such optimizers and commonly observed in practice (Hayou et al., 2024b). In the context of LoRA, this assumption applies individually to optimized gradients of each low-rank matrices, i.e., $g_A, g_B = \Theta(1)$.

3 LoRA IS SELF-STABILIZED

In this section, we present a theoretical analysis of LoRA fine-tuning dynamics, showing that LoRA is self-stabilized with potential appropriate choices of hyper-parameters and initializations A_0 and B_0.

Recall from Definition 1 and Equation (1) that stable feature learning requires all 3 components of ΔY to be $\Theta(1)$. Formally, using the multiplication property of γ-function and the results that $\gamma[g_A] = \gamma[g_B] = 0$ (as established in Section 2.4), we have the following contraints:

$$\begin{cases} \gamma[s] + \gamma[\eta] + \gamma[A_t Z] = 0 \quad (\delta_1 = \Theta(1)) \\ \gamma[s] + \gamma[\eta] + \gamma[B_t] + \gamma[g_A^t Z] = 0 \quad (\delta_2 = \Theta(1)) \\ \gamma[s] + 2\gamma[\eta] + \gamma[g_A^t Z] = 0 \quad (\delta_3 = \Theta(1)) \end{cases} \tag{2}$$

Readers may notice that it is sufficient for $\Delta Y_t = \Theta(1)$ if just one component is $\Theta(1)$ and the others are $o(1)$. As $\gamma[\delta_1] \geq \gamma[\delta_3]$ and $\gamma[\delta_2] \geq \gamma[\delta_3]$ always hold (later explained in Section 3.1), it suffices to only justify why δ_1 and δ_2 are restricted to be $\Theta(1)$:

Assume that $\delta_1 = o(1)$. To maintain $\Delta Y_t = \Theta(1)$, we must have $\delta_2 = \Theta(1)$, implying that the output update is dominated by δ_2. This situation corresponds to fixing the matrix B and only training A ($g_B = 0$ in Equation (1)), which is clearly suboptimal compared to training with both matrices. The same argument applies if $\delta_2 = o(1)$. Therefore, for effective and balanced learning, both δ_1 and δ_2 must scale as $\Theta(1)$.

Now we focus on the value of $\gamma[A_tZ]$, $\gamma[B_t]$ and $\gamma[g_A^tZ]$, which ultimately correlate to the choices of s and η.

3.1 Value of $\gamma[g_A^tZ]$, $\gamma[A_tZ]$ and $\gamma[B_t]$.

We begin by stating an assumption on the value of $\gamma[g_A^tZ]$ and some clarification of its soundness.

Assumption 1. *With optimized gradient $g_A^t \in \mathbb{R}^{r\times n}$ and input $Z \in \mathbb{R}^{n\times *}$, we have $\gamma[g_A^tZ] = 1$.*

For Stochastic Gradient Descent (SGD), this claim holds as $g_A^tZ = \frac{\partial L}{\partial A_t}Z = (d(A_tZ) \times Z)Z = \Theta(n)$. We leave the detailed explanation for other optimizers in Appendix B and use the conclusion directly afterwards. Equation (2) is then refined as:

$$\begin{cases} \gamma[s] + \gamma[\eta] + \gamma[A_tZ] = 0 \\ \gamma[s] + \gamma[\eta] + \gamma[B_t] + 1 = 0 \\ \gamma[s] + 2\gamma[\eta] + 1 = 0 \end{cases} \tag{3}$$

Next, we analysis the value of $\gamma[A_tZ]$ and $\gamma[B_t]$ by induction. Recall from Section 2.1 and the addition property of γ-function, we have

$$\begin{cases} \gamma[A_tZ] = \max(\gamma[A_{t-1}Z], \gamma[\eta] + 1), \Longrightarrow \gamma[A_tZ] \geq \gamma[\eta] + 1 \\ \gamma[B_t] = \max(\gamma[B_{t-1}], \gamma[\eta]), \Longrightarrow \gamma[B_t] \geq \gamma[\eta] \end{cases} \tag{4}$$

, which immediately implies that $\gamma[\delta_1] \geq \gamma[\delta_3]$ and $\gamma[\delta_2] \geq \gamma[\delta_3]$ (used as a conclusion above). It is quite intuitive that δ_3 is less significant than δ_1 and δ_2, as δ_3 is quadratic in the typically small learning rate η; indeed, this term is often even neglected in some prior analysis (e.g., Yen et al. (2025)).

3.2 Impact of A_0 and B_0.

Based on the induction relations of Equation (4), we have the following two equivalent condition pairs:

$$\begin{cases} \gamma[A_tZ] = \gamma[\eta] + 1 \Longleftrightarrow \gamma[A_0Z] \leq \gamma[\eta] + 1 \\ \gamma[B_t] = \gamma[\eta] \Longleftrightarrow \gamma[B_0] \leq \gamma[\eta] \end{cases} \tag{5}$$

, which indicates that the γ-values of the components are closely related to the initializations A_0 and B_0.

Importantly, to satisfy Equation (3) , Equation (5) must both hold or neither: if only one of them is an equation, then we will definitely have $\gamma[A_tZ] \neq \gamma[B_t] + 1$, which leads to an undesirable situation that $\gamma[\delta_1] \neq \gamma[\delta_2]$. Hence, there are only two acceptable cases for A_0 and B_0:

- **Case 1.** $\gamma[A_0Z] \leq \gamma[\eta] + 1$ and $\gamma[B_0] \leq \gamma[\eta]$
- **Case 2.** $\gamma[A_0Z] > \gamma[\eta] + 1$ and $\gamma[B_0] > \gamma[\eta]$

Among them, Case 2 is undesirable because the initial values dominate the training results, overriding contributions from learned updates. In contrast, Case 1 ensures that gradient-based updates govern the learning process. More importantly, if Case 1 is satisfied, the left-hand side conditions of Equation (5) are also satisfied, which in turn ensures that all constraints in Equation (3) are met simultaneously. This leads to a unified expression of δs:

$$\gamma[\delta_1] = \gamma[\delta_2] = \gamma[\delta_3] = \gamma[s] + 2\gamma[\eta] + 1 \tag{6}$$

Therefore, with appropriate initializations satisfying Case 1, tuning s and η such that $\gamma[s] + 2\gamma[\eta] + 1 = 0$ results in $\gamma[\Delta Y_t] = 0$, and stable feature learning will be naturally (without any extra operations) achieved and sustained throughout training, validating its empirical robust effectiveness.

Theorem 3.1. *(Self-stability of LoRA) LoRA can naturally achieve and sustain stable feature learning, if the hyper-parameters s and η are tuned such that $\gamma[s] + 2\gamma[\eta] + 1 = 0$, and the initializations A_0 and B_0 satisfy $\gamma[A_0 Z] \leq \gamma[\eta] + 1$ and $\gamma[B_0] \leq \gamma[\eta]$.*

4 STABLE-LORA

Case 1 suggests that an ideal initialization strategy is to set both A_0 and B_0 to zero, ensuring that $\gamma[A_0] = \gamma[B_0] = -\infty$, which guarantees the satisfaction of Case 1 with arbitrary η. However, while stable feature learning is a necessary condition for effective training, it is not sufficient on its own. With setting $B_0 = 0$ feasible, initializing $A_0 = 0$ introduces two empirical issues: (1) the combination $A = 0$ and $B = 0$ is a saddle point with zero gradient, leading to halting of training; (2) the initial input to B is $A_0 Z = 0$, resulting in complete information loss for learning B and possible gradient vanishing/explosion. The common solution is to set $B_0 = 0$ and sample the entries of A_0 from a distribution with $\sigma^2 = n^{-1}$, which addresses both issues and has been theoretically (Hayou et al., 2024a) and empirically (He et al., 2015) shown to be beneficial.

From the perspective of feature learning, this initialization yields $\gamma[B_0] = -\infty < \gamma[\eta]$ for arbitrary η. According to Theorem 3.1, we must also have $\gamma[A_0 Z] \leq \gamma[\eta] + 1$ to ensure stability. However, due to the non-zero entries in A_0, this condition imposes a lower bound on η: it cannot be arbitrarily small, but must be sufficiently large to absorb the magnitude of $A_0 Z$. In practical scenario where learning rate is usually small, this condition on η is typically violated (as supported by empirical results in Section 5.2). Moreover, this issue cannot be resolved solely by altering initialization or tuning hyper-parameters: adjusting A_0 cannot reduce $A_0 Z$ uniformly across the batch, since Z varies while A_0 is fixed; tuning s is also ineffective since s is not involved in the condition. Consequently, these limitations motivate the development of novel optimization strategies.

We discover that the problem of instable feature learning is fundamentally different from other issues: it is a long-term problem whereas others are short-termed. While the saddle-point and gradient-vanishing/explosion issues arise only at the beginning of training, they naturally resolve as training proceeds with parameters moving away from the saddle point and meaningful signals being propagated to B. In contrast, feature-learning instability occurs at the start if $\gamma[A_0 Z] > \gamma[\eta] + 1$ and persists throughout training due to the induction in Equation (4). This leads to a key idea: instead of modifying A_0 from the beginning (which would exacerbate the other two problems), we can gradually reduce its negative impact as the training proceeds. The solution would be optimal if A_0 can serve its early-stage positive role while its adverse effects diminish to a desirable level over time.

Based on this, we propose Stable-LoRA, a weight-shrinkage optimization strategy applied to matrix A in earliest steps of training, to mitigate instability of LoRA feature learning. An overview of Stable-LoRA is shown in Figure 1, and the detailed procedure is provided in Algorithm 1. Specifically, at an early step t, before parameter updates, A first shrinks as

$$A_{t+1} = (1 - \lambda) A_t - \eta g_A^t$$

where λ ($0 < \lambda < 1$) is the shrinkage ratio, after which A continues to be updated with g_A^t. Shrinkage is applied at every step until a stable condition is met: A achieves a comparable average norm scale to B, i.e. $\|A\|_F / n \leq \|B\|_F / m$ (with r in denominators canceled). The design of stable condition is motivated by the terms $\delta_1 = s\eta g_B^t A_t Z$ and $\delta_2 = s\eta B_t g_A^t Z$, which reach similar scales when A_t and B_t do. Since $\gamma[\delta_2] = 0$ is always ensured, satisfying this condition also guarantees $\gamma[\delta_1] = 0$. The practical effectiveness of this stable condition has been demonstrated in Section 5.3.

Stable-LoRA can robustly prevent instable feature learning for any learning rate η: after N steps of shrinking, we have

$$
\begin{aligned}
A_N &= (1-\lambda)A_{N-1} - \eta g_A^{N-1} \\
&= (1-\lambda)^2 A_{N-2} - (1-\lambda)\eta g_A^{N-2} - \eta g_A^{N-1} \\
&= \cdots \\
&= (1-\lambda)^N A_0 - \eta g_A^{N-1} - \eta((1-\lambda)g_A^{N-2} + \cdots + (1-\lambda)^{N-1} g_A^0) \\
&= (1-\lambda)^N A_0 - \eta g_A^{N-1} - \eta\Delta
\end{aligned}
$$

With $\gamma[(1-\lambda)^k] \leq \gamma[1] = 0$ for all $k \in \mathbb{Z}^+$, we have $\gamma[\Delta Z] \leq \gamma[g_A^* Z] = 1$. Therefore,

$$
\begin{aligned}
\gamma[A_N Z] &= \max(N\gamma[1-\lambda] + \gamma[A_0 Z], \gamma[\eta] + \gamma[g_A^{N-1} Z], \gamma[\eta] + \gamma[\Delta Z]) \\
&= \max(N\gamma[1-\lambda] + \gamma[A_0 Z], \gamma[\eta] + 1)
\end{aligned}
$$

With N and/or λ sufficiently large, $N\gamma[1-\lambda] + \gamma[A_0 Z]$ will finally drop below $\gamma[\eta] + 1$, and hence stable feature learning is achieved from step $N+1$ onward and persists throughout the rest of training.

Stable-LoRA is orthogonal to existing optimization strategies such as gradient optimization (like AdamW) and weight decay, as formally described in Algorithm 1. More importantly, Stable-LoRA incurs negligible overheads during training: it requires no additional memory usage, since the shrinkage operation can be done in-place (and should be, for acceleration) with the pre-shrinkage value no longer used after that. This property is particularly crucial since LoRA is commonly used in memory-constrained scenarios. Computation overhead arising from computing the Frobenius norms $\|\cdot\|_F$ and performing shrinkage is negligible (as shown in Table 4), as it is (1) relatively light-weight compared to other operations and (2) limited to the earliest shrinkage steps.

5 Experiments

We evaluated Stable-LoRA and other baselines under these experimental settings:

Datasets. The tasks involve two fine-tuning scenarios: multi-choice question answering (QA) and chain-of-thought (CoT) reasoning. The QA datasets include HellaSwag (Zellers et al., 2019), SocialIQa (Sap et al., 2019), OpenbookQA, ARC-Easy and ARC-Challenge. For CoT reasoning, we focus on mathematical tasks where models are trained on MetaMathQA (Yu et al., 2023) and evaluated on GSM8K (Cobbe et al., 2021). The exact match accuracy is used as evaluation metric for all tasks.

Models. The experiments are conducted on the 0.5B and 1.5B models from Qwen-2 (Yang et al., 2024) and 1B and 3B models from LLaMA-3.2 (Dubey et al., 2024), for demonstrating the broad effectiveness of our method.

Baselines. Besides AdamW, we compared our proposed method against several other baselines, including stable feature learning methods of LoRA+ (Hayou et al., 2024b) and Riemann Preconditioned Optimization (Zhang & Pilanci, 2024), and a newly-introduced optimizer LoRA-RITE (Yen et al., 2025). LoRA+ claims to achieve stable feature learning by setting learning rate of B larger than A. Riemann Preconditioned Optimization adopts matrix preconditioning on g_A and g_B. LoRA-RITE achieves invariant transformation equilibration of LoRA using unmagnified gradients.

Configurations. Unless otherwise stated, we use the following training configurations. We train q_{proj} and v_{proj} of the attention block with rank $r = 8$. We conducted careful tuning of hyper-parameters by searching η from 5e-5 to 8e-4 and s from 2.0 to 64.0. Each value of A_0 is sampled from $[-1/n, 1/n]$ following (He et al., 2015) and $B_0 = 0$. For LoRA+, we set $\eta_B = 4\eta_A$ following its recommendation for decoder-only models. We search the shrinkage ratio over $\lambda \in [0.001, 0.002, 0.005]$ and report the best result (results for each value of λ are in Table 8). AdamW is adopted as the gradient optimizer. Each reported accuracy is the mean of 3 random runs. More detailed configurations are specified in corresponding subsections or Table 6.

Model	Method	HellaS.	SIQA	ObQA	ARC-E	ARC-C	Avg.
0.5B	AdamW	64.11	67.04	63.00	66.92	47.10	61.63
	LoRA+	63.40	67.91	64.40	67.72	46.33	61.95
	Riemann	59.93	66.27	63.60	66.84	47.27	60.66
	LoRA-RITE	62.22	66.53	65.20	66.84	45.22	61.20
	Stable-LoRA	**66.62**	**67.98**	**66.27**	**68.62**	**48.41**	**63.58**
1B	AdamW	82.33	70.88	72.27	75.42	50.85	70.35
	LoRA+	82.26	71.08	71.93	75.88	51.11	70.45
	Riemann	77.41	70.37	69.00	73.99	48.98	67.95
	LoRA-RITE	81.43	70.68	68.80	76.05	52.30	69.85
	Stable-LoRA	**83.80**	**71.92**	**72.80**	**77.20**	**54.44**	**72.03**
1.5B	AdamW	87.90	76.66	82.20	85.77	69.97	80.50
	LoRA+	87.92	77.18	82.20	85.77	69.80	80.57
	Riemann	86.46	76.51	82.00	85.56	68.94	79.89
	LoRA-RITE	87.70	76.25	82.20	85.90	68.94	80.20
	Stable-LoRA	**88.41**	**77.35**	**83.53**	**86.68**	**71.47**	**81.49**
3B	AdamW	93.26	79.73	81.80	87.67	72.24	82.94
	LoRA+	93.19	79.94	82.80	87.79	71.93	83.13
	Riemann	92.25	79.38	80.80	87.25	71.25	82.19
	LoRA-RITE	92.87	79.69	83.40	88.01	71.08	83.01
	Stable-LoRA	**93.51**	**80.07**	**83.60**	**88.29**	**73.04**	**83.70**

Table 1: Task accuracies of models on question-answering tasks.

5.1 Main results

5.1.1 Results of multi-choice question answering

Table 1 presents the results on the QA datasets. As demonstrated, Stable-LoRA consistently outperforms other methods across models and datasets, achieving up to a 3.59% increase in accuracy. While other baselines may boost performances on specific tasks or models, the improvements are inconsistent and the results occasionally fall below AdamW, indicating that the instability issue remains not fully resolved. In contrast, Stable-LoRA offers not only improved accuracies but also greater robustness across tasks and models.

5.1.2 Results of chain-of-thought reasoning

Chain-of-Thought (CoT) is a widely used approach in tasks that require multi-step reasoning. We trained models to learn to reason in CoT format spontaneously without explicit prompting (i.e., giving the question directly without instructing the model to "think step by step"). We used math-reasoning datasets as representative reasoning tasks. The results in Table 2 show that Stable-LoRA again outperforms all baselines, maintaining its performance advantages in CoT tasks.

5.1.3 Ablations

We conducted ablation studies about target modules of training. The results of training $q_{proj}, k_{proj}, v_{proj}, o_{proj}$ of the 0.5B model is in Table 3, and that of 1B in Table 9. All the results show that Stable-LoRA can robustly increase task accuracies under different LoRA configuration settings.

5.2 Dynamic analysis

Figure 2 provides a dynamic analysis of the training process by visualizing the change of Frobenius norms $||\cdot||_F$ of matrices A and B. During training with AdamW ($\lambda = 0$), $||B||_F$ grows rapidly from small values (g_B is large and B is small), while $\|A\|_F$ increases steadily but remains larger (g_A small and A large), causing output changes induced by δ_1 to dominate over δ_2. This phenomenon indicates that **the problem of $\gamma[A_tZ] > \gamma[\eta] + 1$ does occur in practice**. Stable-LoRA declines $||A||_F$

Methods	1B,1000 steps	1B,2000 steps	3B,1000 steps	3B,2000 steps
AdamW	22.57	26.56	51.25	53.26
LoRA+	22.14	25.55	50.49	53.07
Riemann	19.56	21.68	49.73	51.33
LoRA-RITE	21.68	25.02	50.27	53.25
Stable-LoRA	**23.43**	**27.22**	**51.93**	**54.06**

Table 2: Task accuracies of models on (mathematical) chain-of-thought reasoning tasks.

Targets	Method	HellaS.	SIQA	ObQA	ARC-E	ARC-C	Avg.
qkvo	AdamW	67.12	67.96	64.67	67.47	46.33	62.17
	LoRA+	66.86	68.01	65.60	68.06	48.42	63.39
	Riemann	63.31	67.5	64.80	68.35	47.95	62.38
	LoRA-RITE	63.81	67.40	64.80	67.59	47.01	62.12
	Stable-LoRA	**68.42**	**68.49**	**66.33**	**69.28**	**49.17**	**64.34**

Table 3: Task accuracies of training $q_{proj}, k_{proj}, v_{proj}, o_{proj}$ of the 0.5B model on QA datasets.

while keeping $||B||_F$ unaffected, confirming our claim that the benefit of non-zero A_0 is preserved. Furthermore, $||A||_F$ never drops below its initial value, indicating that the negative influence of initialization persists throughout training.Stable-LoRA effectively suppresses this effect and hence promotes more stable feature learning.

5.3 Memory and computational costs.

Stable-LoRA introduces no additional memory usage compared to LoRA, as the shrinkage operation is conducted in-place. Table 4 compares the training time of Stable-LoRA with baseline methods. The results show that Stable-LoRA incurs only a marginal (0.6%) increase in training time, indicating that the scalar-matrix multiplication involved in shrinkage is far less costly than gradient computation and parameter updates. Moreover, since this extra computation occurs only during the earliest training steps (with a sufficiently large λ), the overall overhead is even more negligible.

5.4 Justification for the stable condition.

To justify the stable condition, we conducted experiments where the stable condition is removed and A shrinks whenever $\|A\|_F/n > \|B\|_F/m$. Table 5 compares task accuracies with and without stopping at the stable condition, and the results show that further shrinkage beyond the stable condition does not lead to noticeable improvements and can even degrade performances, which aligns with our previous analysis.

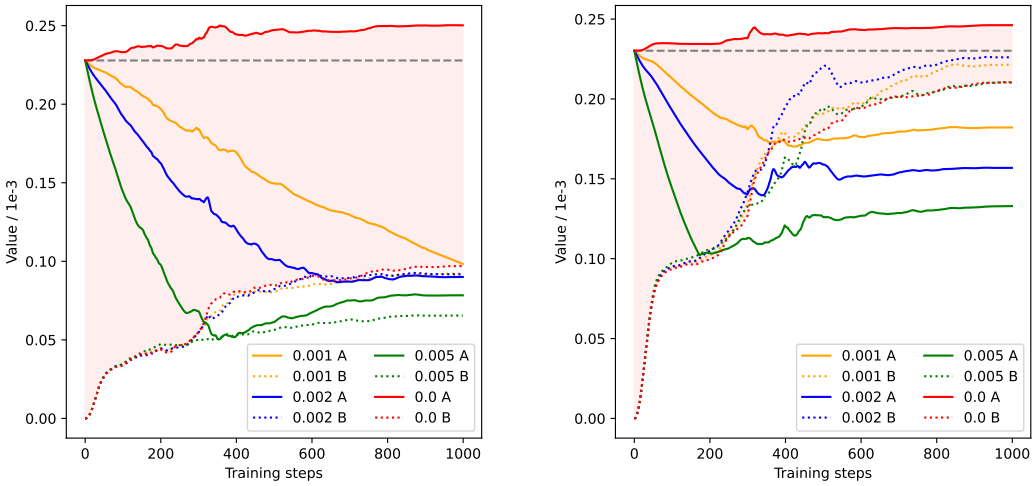
6 Related works

6.1 Stable feature learning.

There are existing works established upon initialization schemes for stable feature learning, from the perspective of width and depth. In scenario of width, (Glorot & Bengio, 2010) proposed Xavier initialization to stabilize the variance of activations, and (He et al., 2015) improved it for non-linear activation functions (like leaky ReLu). (Yang & Hu, 2021) introduced μP parameterization for ensuring feature learning in the infinite-width scenario. Related literature about the depth limit in-

Method	AdamW	LoRA+	Riemann	LoRA-RITE	Stable-LoRA
Time(s)	217.4	217.4	235.5	317.3	218.8
+%	-	+0.0%	+8.3%	+46.0%	+0.6%

Table 4: Comparison of training time of different methods on 0.5B and HellaSwag.



(a) q_{proj} (b) v_{proj}

Figure 2: Averaged norm of A and B on the 0.5B model and HellaSwag. While the scale of B is smaller, it grows more rapidly than A ($|g_B| > |g_A|$), indicating a practical violation of feature learning stability.

Method	HellaS.	SIQA	ObQA	ARC-E	ARC-C	Avg.
Stable-LoRA	**66.62**	**67.98**	**66.27**	68.62	48.41	**63.58**
Stable-LoRA*	66.60	67.84	65.67	**68.77**	**48.81**	63.54

Table 5: Task accuracies of Stable-LoRA with/without the stable condition. * represents "without the stable condition".

cludes (Hayou, 2022; Schoenholz et al., 2016; Yang, 2019) etc.. Stable-LoRA is specifically targeted at width scenario, so it is theoretically discussed upon the width-related initialization method of (He et al., 2015) and shows empirically strong results.

6.2 Stable feature learning for LoRA.

The concept of LoRA stable feature learning originates from LoRA+ (Hayou et al., 2024b), which suggests choosing a larger learning rate for B than A ($\eta_B > \eta_A$). (Zhang & Pilanci, 2024) proposed a matrix-preconditioned optimizer to achieve stabilization. Beyond width-related stability, (Kalajdzievski, 2023) studies the stability with respect to rank r and recommends using scaling factor $s = \alpha/\sqrt{r}$ rather than α/r. Our definition of stable feature learning is slightly different from the above-mentioned work, where we do not demand intermediate states to be $\Theta(n^0)$ due to practical considerations.

7 Conclusion

This paper addresses the challenge of stabilizing feature learning in Low-Rank Adaptation (LoRA). We first establish that, under appropriate hyper-parameters and initializations of A and B, LoRA can in principle be self-stabilized during the training process regardless of model width, which provides a theoretical foundation for the robustness and effectiveness of LoRA. However, we further reveal that the non-zero A_0 compromises this self-stability which leads to performance degradation. Stable-LoRA is hence proposed as a weight-shrinkage strategy that mitigates instability caused by A_0 while preserving its benefits. Stable-LoRA shows superiority over various tasks and models, with no additional memory usage and only marginal computation overhead.

REFERENCES

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

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

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Soufiane Hayou. On the infinite-depth limit of finite-width neural networks. *arXiv preprint arXiv:2210.00688*, 2022.

Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *International conference on machine learning*, pp. 2672–2680. PMLR, 2019.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics. *Advances in Neural Information Processing Systems*, 37:117015–117040, 2024a.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, pp. 17783–17806. PMLR, 2024b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5254–5276, 2023.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*, 2023.

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

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

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, Jianxin Yang, 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, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Liu Yang, Steve Hanneke, and Jaime Carbonell. A theory of transfer learning with applications to active learning. *Machine learning*, 90(2):161–189, 2013.

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 *The Thirteenth International Conference on Learning Representations*, 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 *The Twelfth International Conference on Learning Representations*, 2023.

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*, pp. 4791–4800, 2019.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 59641–59669, 2024.

A Details of Algorithm

Algorithm 1 shows the detailed procedure of Stable-LoRA, where the orthogonality of our method to gradient optimizers and weight decay is demonstrated.

Algorithm 1 Stable-LoRA

Input: Learning rate η, shrink rate λ, weight decay rate w, initializations A_0, B_0
Shrink A if the stable condition is not satisfied
$stable \leftarrow$ **false**
for training step t **do**
  if not $stable$ **and** $\frac{\|A\|_F}{n} > \frac{\|B\|_F}{m}$ **then**
    $A_t = (1-\lambda)A_t$
  else
    $stable \leftarrow$ **true**
  end if
  # Update parameters with optimized gradients and weight decay
  $A_{t+1} = A_t - \eta g_A^t - \eta w A_t$, $B_{t+1} = B_t - \eta g_B^t - \eta w B_t$
end for

Note that Stable-LoRA is conceptually different from weight decay from several perspectives:

- **Theoretical motivation and formulation.** Weight decay is based on a Bayesian prior assuming that model weights follow a Gaussian distribution centered at zero. It introduces an additional regularization term in the loss function, so the rate w is multiplied by η and then applied to the parameters.

  $$A = A - \eta w A = (1 - \eta w)A$$

  In contrast, Stable-LoRA directly shrinks the weights using λ which is independent of the learning rate:

  $$A = A - \lambda A = (1 - \lambda)A$$

  With $\eta << 1$ in almost all cases, Stable-LoRA results in significantly faster decay compared to weight decay.
- **Scope of application.** Weight decay is applied uniformly to all trainable parameters, and Stable-LoRA targets only at the matrix A, with the explicit purpose of reducing the influence of A_0.
- **Application schedule.** Weight decay is applied throughout the entire training process, while Stable-LoRA is only applied before the stable condition achieved.

B Proof of Assumption 1

Assumption 1 has been well-studied and adopted in existing works (Hayou et al., 2019; Zhang & Pilanci, 2024; Hayou et al., 2024b), yet we show the explanation of its rationality below.

Assumption. *With optimized gradient $g_A^t \in \mathbb{R}^{r\times n}$ and input $Z \in \mathbb{R}^{n\times *}$, we have $\gamma[g_A^t Z] = 1$.*

Consider an extremely simplified optimizer, which normalizes each entry of the gradient to its sign, i.e.,

$$g_A^t = \text{sign}(\frac{\partial L_t}{\partial A}),$$

where $\frac{\partial L_t}{\partial A}$ denotes the raw (non-optimized) gradient of A at step t. By the chain rule, we have

$$\frac{\partial L_t}{\partial A} = sB_t^T dY_t \times Z,$$

where dY_t is the gradient of the output Y_t. Define

$$S^t = sB_t^T dY_t,$$

so that the gradient becomes

$$\frac{\partial L_t}{\partial A} = S^t \times Z = (S_i^t Z_j)_{ij}$$

Therefore we have

$$g_A^t = \text{sign}(\frac{\partial L_t}{\partial A}) = \text{sign}(S^t \times Z) = \text{sign}(S^t) \times \text{sign}(Z)$$

Hence,

$$g_A^t Z = (\text{sign}(S^t) \times \text{sign}(Z))Z = (\text{sign}(Z)^T Z)\text{sign}(S^t)$$

Since $\text{sign}(Z)^T Z = \Theta(n)$ always holds and $S^t = \Theta(1)$ if it is a stable gradient, we conclude that $g_A^t Z = \Theta(n)$, the same as $\gamma[g_A^t Z] = 1$.

As more sophisticated optimizers generally preserve the sign of gradient (Yang et al., 2013), this assumption is well justified and serves as a foundation for our subsequent analysis.

C Details of Experiments

C.1 Hyper-parameters

The hyper-parameters of experiments (except for specified ones) are listed in Table 6. To find the optimal combination, η and s are thoroughly searched across wide ranges. The weight decay ratio w is set to 0.01 to prevent overfitting, and the dropout ratio is 0.0 as setting a non-zero value of it cannot boost performance in our experiments.

Hyper-parameter	Value
Learning rate η	[5e-5,1e-4,2e-4,3e-4,4e-4,6e-4,8e-4]
Scale factor s	2.0 to 64.0 (interval 2.0)
Target modules	q_{proj}, v_{proj}
Rank r	8
Weight decay ratio w	0.01
Dropout ratio	0.0
Batch size	16
Training steps	1000
Learning rate scheduler	linear
Warm-up steps	100
AdamW β_1, β_2	0.9, 0.999

Table 6: Hyper-parameters.

C.2 Datasets

Table 7 contains the detailed information of datasets. All datasets include test sets but no validation sets; therefore we apply weight decay to mitigate overfitting. For the QA datasets, we use the templated versions provided by LLM-Adapter (Hu et al., 2023).

D Additional results

D.1 Results of different λs

We show results of different λs on QA datasets in Table 8. The results show that Stable-LoRA can robustly out-perform AdamW with a variety of λ, demonstrating the principle effectiveness of the method. λ should be considered as a newly-introduced hyper-parameter, which could and should be tuned for optimal performances. The results of $\lambda = 0.005$ is often suboptimal, suggesting that an excessively larger λ could result in loss of information from previous training steps and hence downgraded performances.

Dataset	#Train	#Test
HellaSwag	39905	10042
SIQA	33410	1954
ObQA	4957	500
ARC-E	2251	2376
ARC-C	1119	1172
MetaMathQA	40000	1319

Table 7: Detailed information about datasets.

Model	Method	HellaS.	SIQA	ObQA	ARC-E	ARC-C	Avg.
0.5B	AdamW	64.11	67.04	63.00	66.92	47.10	61.63
	Stable-LoRA$_{\lambda=0.001}$	66.14	67.36	<u>65.73</u>	**68.62**	**48.41**	<u>63.25</u>
	Stable-LoRA$_{\lambda=0.002}$	<u>66.37</u>	<u>67.88</u>	**66.27**	<u>68.42</u>	<u>48.24</u>	**63.44**
	Stable-LoRA$_{\lambda=0.005}$	**66.62**	**67.98**	65.07	68.17	48.04	63.18
1B	AdamW	82.33	70.88	71.27	75.42	50.85	70.15
	Stable-LoRA$_{\lambda=0.001}$	**83.80**	**71.92**	<u>72.13</u>	**77.20**	**54.44**	**71.90**
	Stable-LoRA$_{\lambda=0.002}$	<u>83.66</u>	<u>71.87</u>	**72.33**	<u>77.06</u>	<u>54.35</u>	<u>71.85</u>
	Stable-LoRA$_{\lambda=0.005}$	83.70	71.79	72.00	76.84	53.84	71.63
1.5B	AdamW	87.90	76.66	82.20	85.77	69.97	80.50
	Stable-LoRA$_{\lambda=0.001}$	**88.41**	<u>77.23</u>	**83.53**	**86.68**	**71.47**	**81.46**
	Stable-LoRA$_{\lambda=0.002}$	<u>88.38</u>	77.21	<u>83.40</u>	<u>86.62</u>	<u>71.25</u>	<u>81.37</u>
	Stable-LoRA$_{\lambda=0.005}$	88.22	**77.35**	83.33	86.45	70.11	81.09
3B	AdamW	93.26	79.73	81.80	87.67	72.24	82.94
	Stable-LoRA$_{\lambda=0.001}$	<u>93.48</u>	<u>79.92</u>	83.27	**88.29**	**73.04**	**83.60**
	Stable-LoRA$_{\lambda=0.002}$	**93.51**	79.84	**83.60**	88.02	**73.04**	**83.60**
	Stable-LoRA$_{\lambda=0.005}$	93.47	**80.07**	<u>83.40</u>	<u>88.15</u>	72.47	83.51

Table 8: Task accuracies of different λs on QA datasets. The secondary best results are <u>underlined</u>.

D.2 More ablation studies

Table 9 shows the results of ablation studies on the $q_{proj}, k_{proj}, v_{proj}, o_{proj}$ of the 1B model on QA datasets, where Stable-LoRA shows uniform superiority of performances across the involved tasks.

Targets	Method	HellaS.	SIQA	ObQA	ARC-E	ARC-C	Avg.
qkvo	AdamW	84.21	72.77	74.40	75.97	51.54	71.78
	LoRA+	84.43	73.08	73.80	75.80	53.30	72.08
	Riemann	82.15	71.44	73.40	75.51	51.28	70.76
	LoRA-RITE	83.03	72.36	69.80	76.56	54.95	71.34
	Stable-LoRA	**85.71**	**73.61**	74.40	**77.92**	**55.75**	**73.46**

Table 9: Task accuracies of training $q_{proj}, k_{proj}, v_{proj}, o_{proj}$ of the 1B model on QA datasets.

E Limitations

Stable-LoRA has shown empirical superiority through extensive experiments on LoRA, while its effectiveness on other variants of parameter-efficient fine-tuning methods remains uncertain. The experiments adopted the commonly used initializations of $A_0 \sim [-1/n, 1/n]$ and $B_0 = 0$ rather than other strategies. AdamW was employed as gradient optimizer while performances with other optimizers have not been tested. These choices of experimental settings are all representatives of the most common and relevant scenarios, reflecting our focus on practicability.

F DECLARATION OF LLM USAGE

LLMs have been used to polish the writing of this paper, while the core idea is fully originated from human beings.