INITIALIZATION USING UPDATE APPROXIMATION IS A *Silver Bullet* FOR EXTREMELY EFFICIENT LOW-RANK FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adapters have become standard for efficiently fine-tuning large language models, but they often fall short of achieving the performance of full fine-tuning. We propose a method, **LoRA S**ilver **B**ullet or **LoRA-SB**, that approximates full fine-tuning within low-rank subspaces using a carefully designed initialization strategy. We theoretically demonstrate that the architecture of LoRA-XS, which inserts a learnable $r \times r$ matrix between B and A while keeping other matrices fixed, provides the precise conditions needed for this approximation. We leverage its constrained update space to achieve optimal scaling for high-rank gradient updates while removing the need for scaling factor tuning. We prove that our initialization offers an optimal low-rank approximation of the initial gradient and preserves update directions throughout training. Concretely, **LoRA-SB** combines this initialization with a constrained low-rank adaptation mechanism, forming a co-designed system where both the update subspace and its optimization dynamics are jointly aligned with full fine-tuning. Extensive experiments across mathematical reasoning, commonsense reasoning, and language understanding tasks demonstrate that our approach exceeds the performance of LoRA (and baselines) while using **27-90** times fewer learnable parameters, and comprehensively outperforms LoRA-XS. Our findings establish that it is possible to simulate full fine-tuning in low-rank subspaces, and achieve significant parameter efficiency gains without sacrificing performance. Anonymous code is available at: https://anonymous.4open.science/r/lora-sb-anonymous-5BEE.

1 INTRODUCTION

Pre-trained language models have become central to natural language processing, achieving state-of-the-art performance across diverse tasks (35; 21; 1). While these models excel at general-purpose capabilities (4; 14), adapting them to specific downstream tasks often requires fine-tuning (FT). At the same time, full FT, while highly effective, is computationally expensive and impractical at scale.

Parameter-efficient fine-tuning (PEFT) has become vital for adapting large language models (LLMs) under computational constraints. Low-rank methods like LoRA (17) address this by reducing learnable parameters via low-rank updates, sparking advancements in optimization, initialization, structured matrices, and adaptive rank selection (52; 46; 45). However, these methods face trade-offs: either retain many parameters to match full FT or sacrifice performance for extreme efficiency (17; 10; 46). This raises a critical question: Can we design low-rank methods that achieve full FT-level performance while drastically reducing parameter counts?

Low-rank decomposition methods operate on a fundamental premise: FT requires learning only a low-rank update to the pre-trained weights. However, the **gradients** computed by these methods do not inherently possess this property. For instance, LoRA's gradients need explicit optimization at each step to better approximate the full FT gradient (46). Additionally, initialization has emerged as a critical factor in low-rank adaptation, as highlighted by recent works like PiSSA-LoRA (30) and LoRA-GA (45).

We analyze these limitations in the context of the architecture of LoRA-XS (2), which inserts a learnable $r \times r$ matrix between B and A while keeping other matrices fixed, and demonstrate that

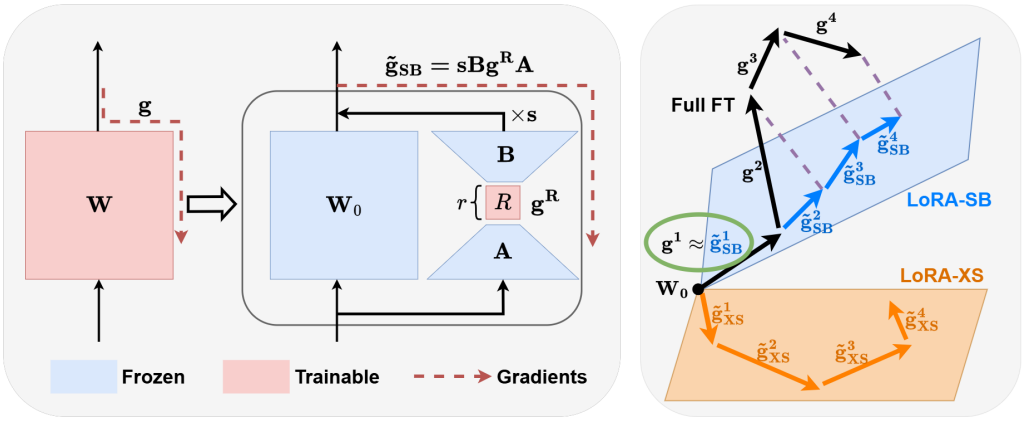


Figure 1: **LoRA-SB.** LoRA-XS (2) reduces parameters compared to LoRA (17) by inserting a learnable $r \times r$ matrix R between B and A, while keeping other matrices fixed, leading to $W = W_0 + sBRA$. Our method, LoRA-SB, uses the same architecture. We find that updating R using its gradients g^R is equivalent to updating the full FT matrix W with an equivalent gradient $\tilde{g}_{SB} = sBg^RA$. We initialize B, R, and A such that the equivalent gradient $\tilde{g}_{SB}$ provably best approximates the full FT gradient g in low rank subspaces **at each step**. In essence, we simulate the **entire full FT process** optimally within low-rank subspaces by **utilizing only the first full FT gradient** g_1.

these challenges are even more pronounced. While exploring solutions inspired by LoRA-based methods, we discover a remarkable property unique to LoRA-XS: through careful initialization of A and B, we can simulate the full FT optimization in low rank subspaces through **entire training**, as shown in Figure 1. Our initialization provides optimal scaling for approximating high-rank full FT gradients and eliminates need for tuning the hyperparameter α. While initialization plays a critical role, LoRA-SB is fundamentally a co-design of (1) the low-rank subspace itself, chosen via approximation of the first full-FT update, and (2) the optimization dynamics that use this constrained subspace with provably optimal gradient projection. As shown in Table 1, LoRA-SB is the only approach that jointly optimizes the low-rank subspace via FT-aligned initialization and the adaptation dynamics via FT-projected gradients, forming a principled co-design. **The peak memory usage of LoRA-SB never exceeds that of LoRA** or other baselines, and its training-time overhead relative to LoRA is negligible ($\approx 1.1\% - 1.3\%$). Our key contributions are:

- We formalize the limitations of LoRA-XS, showing how its constrained update space leads to suboptimal gradient approximation, initialization sensitivity, and scaling dependence.
- We propose an initialization strategy derived from using the first step of full FT, which provides an optimal approximation of the initial gradient and preserves update directions throughout.
- We prove our initialization makes gradient optimization scaling-independent and guarantees convergence by maintaining orthonormal bases, eliminating need for tuning the scaling factor α.
- Through extensive experiments on 4 models across 16 datasets covering mathematical reasoning, commonsense reasoning, and language understanding, we demonstrate that LoRA-SB surpasses LoRA while using **27-90x** less learnable parameters, and comprehensively outperforms LoRA-XS.

2 Methodology

2.1 Preliminaries

In standard FT, a pre-trained weight matrix $W \in \mathbb{R}^{m \times n}$ is updated using the update matrix ΔW as:

$$W = W_0 + \Delta W, \tag{1}$$

where W_0 is the pre-trained weight. This requires updating mn parameters per layer. LoRA posits that updates lie in a low-dimensional subspace, parameterizing ΔW as:

$$W = W_0 + sBA, \tag{2}$$

Method	Higher–Rank Updates Feasible?	Do Gradients Approximate Full Fine–Tuning?	Is Initialization Optimal (FT–Aligned)?
LoRA (17)	No - the number of learnable parameters scales as $r(m+n)$, making large r prohibitively expensive.	No - LoRA uses raw adapter gradients that do not match full fine–tuning and lack FT–aware correction.	No - initialization is standard random/zero/Kaiming and does not capture FT update directions.
LoRA-XS (2)	Yes - only r^2 parameters are learned, so r can be set much higher while remaining parameter–efficient.	No - gradients are restricted to the fixed subspace defined by frozen A and B, which may not reflect FT geometry.	No - initialization is derived from pretrained weight statistics rather than FT-aligned updates.
LoRA-Pro (46)	No - same $r(m+n)$ scaling as LoRA, so r cannot be increased cheaply.	Yes - applies a closed-form gradient transformation at every step to better approximate full FT.	No - initialization does not change and is not aligned with FT update directions.
LoRA-GA (45)	No - shares LoRA's $r(m+n)$ parameter scaling, so increasing r significantly is expensive.	No - after initialization, it relies on standard LoRA gradients without per-step FT-aware correction.	Yes - provides an optimized initialization designed to align the update subspace with FT at the start of training.
LoRA-SB (Ours)	Yes - same r^2 architecture as LoRA-XS, enabling much larger r at minimal parameter cost.	Yes - uses an equivalent gradient that matches the projection of full FT updates onto the low-rank subspace.	Yes - initialization uses the truncated SVD of the first FT update, giving the optimal rank-r FT-aligned subspace.

Table 1: Comparison of LoRA variants along three axes: (1) whether large ranks are feasible under their parameter scaling, (2) whether their gradients approximate full fine-tuning updates, and (3) whether their initialization is optimally aligned with full fine-tuning signals.

where $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times n}$ are trainable low-rank matrices with rank $r \ll \min(m,n)$, and s is a scaling factor (α/r) to stabilize training. This reduces the number of parameters from mn to $r(m+n)$. LoRA-XS efficiently parameterizes as:

$$W = W_0 + sBRA, \tag{3}$$

where B and A are fixed, and only $R \in \mathbb{R}^{r\times r}$ is trainable, reducing the number of parameters to r^2. We denote the full FT gradient: $g = \frac{\partial L}{\partial W}$; LoRA-XS gradient: $g^R_{\text{LoRA-XS}} = \frac{\partial L}{\partial R}$; L is the loss function.

2.2 Motivation

LoRA-XS (2) has significantly fewer learnable parameters than LoRA but performs suboptimally. LoRA-XS's architecture causes constraints on the type of updates it can learn. The subspace of learned updates is characterized in Lemma 1. This implies that while ΔW is constrained to be rank $\leq r$, it also needs to have column and row spaces defined by those of B and A, respectively. In contrast, LoRA can learn any update ΔW as long as rank$(\Delta W) \leq r$. Thus, the low expressivity of LoRA-XS as compared to LoRA can account for the performance drop.

Lemma 1. *Let ΔW be an update learned with LoRA-XS. Then, the set of all possible ΔW, say $\mathcal{W}_{LoRA-XS}$, is given as:*

$$\mathcal{W}_{LoRA-XS} = \{M \in \mathbb{R}^{m \times n} | \textit{Col}(M) \subseteq \textit{Col}(B) \wedge \textit{Row}(M) \subseteq \textit{Row}(A)\},$$

where $\textit{Col}(M)$ and $\textit{Row}(M)$ are column and row spaces of matrix M respectively.

Proof. See Appendix B.1. □

We identify three key limitations, which arise due to this and otherwise:

1) **Inadequate Gradient Approximation:** LoRA optimization is mathematically equivalent to full FT using a constrained low-rank gradient. The gradient of LoRA does not optimally approximate the full gradient, and needs to be tuned at each step. LoRA-Pro (46) finds that this results in suboptimal performances, and provides a closed form solution to optimize the gradients. In LoRA-XS, the gradient updates are restricted to an even more constrained low-rank space since A and B are fixed. We posit that the limitation becomes particularly severe when the ideal updates lie outside the space spanned by fixed A and B, and consequently has a larger impact on performance.

2) **Suboptimal Initialization:** While initialization impacts all low-rank methods, it becomes critical in LoRA-XS where A and B are frozen. Unlike LoRA where poor initialization can be compensated through training, LoRA-XS relies entirely on its initial subspace defined by A and B. Consider the zero initialization of the B matrix, for example. While LoRA may experience some performance degradation in this case (45; 30), the ideal low-rank update ΔW can still be reached through gradient descent. In fact, zero initialization for the B matrix is commonly used, including in the original LoRA paper (17). However, in LoRA-XS, this results in no learning, as the product BRA remains zero. LoRA-XS uses the most significant subspaces spanned by the columns of pre-trained weights for initialization, inspired by PiSSA (30). This initialization is not aligned well with FT because it fails to capture the specific subspaces relevant to the FT task.

3) **Scaling Factor Sensitivity:** The scaling factor s, present in almost every LoRA based method, requires tuning to maintain stability during training. This factor acts as a bridge between the low-rank and full-rank spaces, compensating for the dimensional mismatch in gradients. Poor tuning of s can lead to unstable training or slow convergence (rsLoRA (20)), adding complexity and potentially limiting practical deployment.

2.3 Approximation of the full FT gradient

As mentioned, LoRA optimization is equivalent to full FT using a constrained low-rank gradient. However, the update generated using the gradients of LoRA does not result in the same update which the low-rank gradient would have generated. The following holds true for LoRA-XS as well. To understand this, let us look at the change in weight W and its relationship with changing of low-rank matrix R, which can be simply given by $\mathrm{d}W = -sB(\mathrm{d}R)A$. This implies that updating R with gradient g^R is equivalent to updating W with low rank equivalent gradient $\tilde{g}$ in full FT (Definition 1).

Definition 1. *We define the equivalent gradient in LoRA-XS as: $\tilde{g} = sBg^RA$, where g^R is the gradient of L with respect to R.*

The equivalent gradient describes the virtual low-rank gradient of matrix W in LoRA-XS optimization process, despite W not being directly trainable. This gradient determines how updates to R affect W. To bridge the performance gap between LoRA-XS and full FT, we aim to minimize the discrepancy between the equivalent gradient $\tilde{g}$ and the full gradient g. First, we establish the relationship between gradients in LoRA-XS optimization in Lemma 2.

Lemma 2. *The gradient of the loss with respect to matrix R can be expressed in terms of the gradient with respect to the weight matrix W as: $g^R_{LoRA-XS} = sB^\top gA^\top$.*

Proof. See Appendix B.2. □

We now formulate our objective to minimize the distance between the equivalent gradient and the full gradient. We do not have access to the full FT gradient g during LoRA-XS based FT. Thus we need to find the ideal gradient with respect to R, given by g^R, and subsequently the optimal approximation $\tilde{g}$, in terms of the gradient which is available to us during training: $g^R_{LoRA-XS}$. Fortunately, this optimization problem admits a closed-form solution independent of g as described in Theorem 3.

Theorem 3. *For full-rank A and B matrices, the optimal solution for the objective $min_{g^R}||\tilde{g} - g||_F^2$, such that $\tilde{g} = sBg^RA$, is: $g^R = \frac{1}{s^2}(B^\top B)^{-1} g^R_{LoRA-XS}(AA^\top)^{-1}$.*

Proof. See Appendix B.3. □

The closed-form solution in Theorem 3 solves the optimization problem $\min_{g^R}||\tilde{g} - g||_F^2$, but by itself doesn't ensure the loss will decrease when updating R. Through Theorem 4, we prove that the change in loss is non-positive ($\Delta L \leq 0$). This property is fundamental to optimization as it guarantees consistent loss minimization throughout training.

Theorem 4. *Consider the update for matrix R using the solution derived in Theorem 3: $R \leftarrow R - \eta g^R$, where $\eta > 0$ is the (sufficiently small) learning rate. This update guarantees a reduction in the loss ΔL, given by: $\Delta L = -\eta \langle g^R_{LoRA-XS}, g^R \rangle_F + o(\eta) \leq 0$.*

Proof. See Appendix B.4. □

2.4 Initialization using update approximation

In FT, the primary goal is to update weights to better suit the target task. The initial gradient steps are particularly informative, as they indicate the direction of desired adaptation. We leverage this insight by using the first update step from full FT for initialization.

This approach offers two key advantages. First, it ensures the low-rank space captures the most relevant subspace for the target task rather than relying on pre-trained properties. Second, since A and B are fixed, initializing them to span the subspace of early adaptation increases the likelihood of capturing useful updates throughout training. This also ensures that the final update is learnt in the correct subspace, of which we have no apriori information besides the first full FT step. Our method is summarized as: set such initialization that best approximates the first step of full FT. Given a full FT update $\Delta W_{first-step}$, our initialization satisfies:

$$sB_{init}R_{init}A_{init} \approx \Delta W_{first-step} \tag{4}$$

The first step of full FT, for Adam-based optimizers such as AdamW, for sample x_i is:

$$\Delta W_{first-step} = -\eta \times \textbf{sign}(\nabla_W \mathcal{L}(W_0, x_i)) \tag{5}$$

However, the usage of a single sample may lead to noisy estimates. Instead, we compute a more stable initialization by averaging gradients over a subset of the training data:

$$\Delta W_{avg} = -\eta \textbf{sign}(\sum_{i=0}^{n \leq |\mathbb{X}|} \nabla_W \mathcal{L}(W_0, x_i)), \quad x_i \in \mathbb{X} \tag{6}$$

Since AdamW is used as the optimizer for both full FT and LoRA-SB training, we approximate its first update step using the sign of the summed gradients rather than their raw values (see Appendix C for details). This better captures the direction of adaptation required for the target task while being less sensitive to individual sample variations. We then use truncated SVD to obtain a low-rank approximation of ΔW_{avg}, and express it as $sBRA$. There exist infinite combinations of B and A which can obey this relationship. For instance, we can initialize B and A as US and $V^\top$ and keep R as I/s. This is equivalent to the B and A initialization in LoRA-XS but by approximating the update rather than the pre-trained matrix. The above process can be computed for any optimizer, by approximating the corresponding first step. We compute this specifically for AdamW since we use it.

2.5 Scaling Factor independence

The hyperparameter α is used in LoRA and other decomposition-based methods to tackle instability caused to improper scaling of the updates. The gradient scaling is accounted for, by adding a

hyperparameter to normalize the updates. The importance of scaling is shown in methods like rank stabilization (20). However, the full FT gradient g needs no such tuning. We claim that approximating the full FT gradient removes the need for introducing a scaling factor, as shown in Theorem 5.

Theorem 5. *The equivalent gradient $\tilde{g}$ is hyperparameter s independent for $\tilde{g} = sBg^RA$, but not for $\tilde{g} = sBg^R_{LoRA-XS}A$.*

Proof. See Appendix B.5. □

The scaling factor independence of the equivalent gradient eliminates the need for manual gradient scaling. Updates to W depend solely on this gradient (modulo learning rate), making any additional scaling redundant. This can be understood by examining the relationship with the full FT gradient g. Since g is naturally scaled for optimal weight updates, and our method approximates g in a constrained subspace, the equivalent gradient inherits appropriate scaling automatically. This property is unique to our gradient approximation approach and does not hold for standard LoRA-XS.

2.6 LoRA-SB: Update approximation initialization is a *silver bullet*

The solutions discussed independently address the gradient approximation and initialization problems, while also providing scaling factor independence. LoRA-SB, elegantly combines these solutions through a simple initialization strategy, derived from approximating the first full FT step:

$$U, S, V^\top \leftarrow \textbf{SVD}(\Delta W_{avg}) \tag{7}$$

$$B_{init} \leftarrow U[1:r], A_{init} \leftarrow V[1:r], R_{init} \leftarrow \frac{1}{s} S[1:r, 1:r] \tag{8}$$

By the Eckart-Young theorem (13; 32), this gives the optimal rank-r approximation of the full FT update. where U, S, V are obtained from truncated SVD of the averaged first update ΔW_{avg}. This initialization leads to several key advantages.

Simplified Gradient Optimization. Our initialization ensures B_{init} and A_{init} form orthonormal bases in $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively, leading to $B^\top B = AA^\top = I$. With fixed B and A matrices being orthonormal, the need for complex matrix inversions during training is eliminated, , as the optimal update step, derived in Equation 3, simplifies to:

$$g^R = \frac{1}{s^2}(B^\top B)^{-1} g^R_{LoRA-XS} (AA^\top)^{-1} = \frac{1}{s^2} g^R_{LoRA-XS}$$

Optimal Update Approximation. Our initialization guarantees that the first update optimally approximates the full FT weight updates: $sB_{\text{init}}R_{\text{init}}A_{\text{init}} \approx \Delta W_{avg}$. By the Eckart-Young theorem, this is the optimal rank-r approximation of the initial full FT update.

Scaling Factor Independence. As shown in Theorem 5, when gradient approximation is applied with orthonormal B and A, the hyperparameter s can be set to 1, resulting in guaranteed optimal gradient approximation at every step, without requiring any scaling factor:

$$\boxed{g^R = g^R_{\text{LoRA-XS}}} \tag{9}$$

Guaranteed Loss Reduction. Since B is a tall orthonormal and A a wide orthonormal matrix, they remain full rank throughout training. This ensures that dL remains negative (Theorem 4), guaranteeing stable optimization and convergence.

$$\Delta(sB_{init}R_{init}A_{init}) \approx \gamma \Delta W \tag{10}$$

Another heuristic which might lead to a good initialization is setting B and A, such that the first update also approximately matches the ΔW direction (Equation 10). Thankfully, we don't have to choose between the two. For SGD, we prove that setting B_{init} and A_{init} using Equations 7-8, results in the first update of LoRA-XS to best approximate the direction of the full FT update (Theorem 6).

Theorem 6. *If A_{init} and B_{init} are initialized using LoRA-SB for the first step of SGD optimizer, then the update given by LoRA-SB, $\Delta(B_{init}R_{init}A_{init})$, is the best low-rank approximation of full fine-tuning update, ΔW.*

Proof. See Appendix B.6. □

While Theorem 6 is stated for SGD, the result extends to other SGD-based optimizers such as AdamW. In practice, we use AdamW and approximate the first update by taking the sign of the averaged gradients, consistent with AdamW's first-step behavior. This produces an initialization whose SVD still yields the optimal rank-r approximation of the simulated full FT update.

Initialization Memory. To optimize GPU memory during **initialization**, we hook into the backward pass and compute the gradients layerwise, immediately discarding the computed gradients (29; 45). This ensures $O(1)$ memory usage, independent of the number of layers, keeping GPU memory well within limits. This guarantees that the memory required for LoRA-SB initialization never exceeds the memory needed for subsequent LoRA-SB fine-tuning, and that **the peak memory usage of the entire LoRA-SB algorithm never exceeds that of standard LoRA** and other baselines.

LoRA-SB Advantages over LoRA. Many properties described above are not achievable with standard LoRA methods. Even if B and A are initialized as orthonormal in LoRA, subsequent updates do not preserve this property because B and A are trainable. This results in several challenges in using LoRA (even with optimal gradient approximation) compared to LoRA-SB:

- Potential instability of $(B^\top B)^{-1}$ and $(AA^\top)^{-1}$, not guaranteed to remain non-singular throughout.
- Inability to ensure consistent loss reduction due to potential rank deficiency, B and A may not remain full-rank throughout training.
- Necessity to fine-tune the scaling factor hyperparameter α.
- Repeated re-computation of $B^\top B$ and $AA^\top$ is required at each optimizer step for accurate gradient approximation.

3 Experiments

We evaluate over 16 different datasets on 3 widely-used benchmarks, using models ranging from the 355 M RoBERTa-large model to the 9 B Gemma-2 model. Our setup spans both masked and autoregressive architectures, allowing us to comprehensively assess the effectiveness of LoRA-SB. Specifically, we fine-tune RoBERTa-large (27), Llama-3.2 3B (12), Mistral-7B (19), and Gemma-2 9B (43). **We compute the update approximation using only $1/1000$ (0.1%) of each dataset's total size**. This ensures that the training time overhead is minimal and has a negligible effect on efficiency. Detailed hyperparameter and dataset details are given in Appendix I and J, respectively.

Baselines. We compare LoRA-SB against full FT, LoRA (17), LoRA-XS (2), and several popular variants of LoRA - rsLoRA (20), PiSSA (30), DoRA (26), and LoRA-Pro (46).

Table 2: Comparison of FT methods on Mistral-7B and Gemma-2 9B across arithmetic benchmarks. # Params denotes the number of trainable parameters. Best results among PEFT methods are in **bold**.

Method	Rank	Mistral-7B			Gemma-2 9B		
		# Params	GSM8K (↑)	MATH (↑)	# Params	GSM8K (↑)	MATH (↑)
Full FT	-	7.24 B	63.87	17.65	9.24 B	79.23	38.02
LoRA	32	83.88 M	61.94	15.98	108.04 M	76.19	36.56
rsLoRA	32	83.88 M	62.15	16.24	108.04 M	76.84	36.88
PiSSA	32	83.88 M	62.43	16.52	108.04 M	77.12	37.04
DoRA	32	85.26 M	62.65	16.64	109.88 M	77.58	37.04
LoRA-GA	32	85.26 M	62.87	16.66	109.88 M	77.28	37.13
LoRA-Pro	32	83.88 M	63.07	17.32	108.04 M	78.26	37.53
LoRA-XS	32	0.23 M	54.28	13.36	0.30 M	74.07	34.62
LoRA-XS	64	0.92 M	57.08	15.62	1.20 M	75.02	36.46
LoRA-XS	96	2.06 M	58.53	16.42	2.71 M	75.21	36.98
LoRA-SB	32	0.23 M	58.91	15.28	0.30 M	75.44	36.66
LoRA-SB	64	0.92 M	60.73	16.28	1.20 M	76.65	37.14
LoRA-SB	96	2.06 M	**63.38**	**17.44**	2.71 M	**78.40**	**37.70**

Table 3: Comparison of FT methods on Llama-3.2 3B across eight commonsense reasoning datasets. # Params denotes the number of trainable parameters. Best results among PEFT methods are in **bold**.

Method	Rank	# Params	Accuracy (↑)								
			BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
Full FT	-	3.21 B	70.43	85.64	80.45	91.92	85.02	88.52	75.29	81.88	82.39
LoRA	32	48.63 M	70.03	85.20	79.12	90.71	82.24	86.91	74.32	**81.87**	81.30
rsLoRA	32	48.63 M	69.81	85.63	78.92	90.45	82.02	86.71	74.18	81.72	81.11
PiSSA	32	48.63 M	70.12	85.42	79.44	90.88	82.68	87.23	74.61	81.79	81.52
DoRA	32	49.40 M	70.43	85.63	79.68	90.76	82.90	87.61	74.87	82.04	81.74
LoRA-Pro	32	48.63 M	**71.28**	**85.81**	79.35	90.90	83.42	87.24	**75.32**	81.74	81.88
LoRA-XS	32	0.20 M	65.01	82.87	76.17	87.32	80.12	84.78	70.31	75.71	77.79
LoRA-XS	64	0.80 M	66.53	83.12	77.98	88.53	81.76	85.15	72.04	77.14	79.03
LoRA-XS	96	1.81 M	67.28	83.35	78.66	88.99	82.08	85.18	72.61	78.88	79.63
LoRA-SB	32	0.20 M	66.33	84.06	78.91	89.04	81.37	86.62	72.44	76.97	79.47
LoRA-SB	64	0.80 M	68.35	84.55	79.94	**91.68**	83.03	87.84	74.83	80.12	81.29
LoRA-SB	96	1.81 M	70.34	84.76	**80.19**	91.62	**84.61**	**87.92**	74.74	81.20	**81.92**

Table 4: Comparison of FT methods on RoBERTa-large across GLUE datasets. # Params denotes the number of trainable parameters. Best results among PEFT methods are in **bold**. We use Pearson correlation for STS-B, Matthew's correlation for CoLA, and accuracy for others.

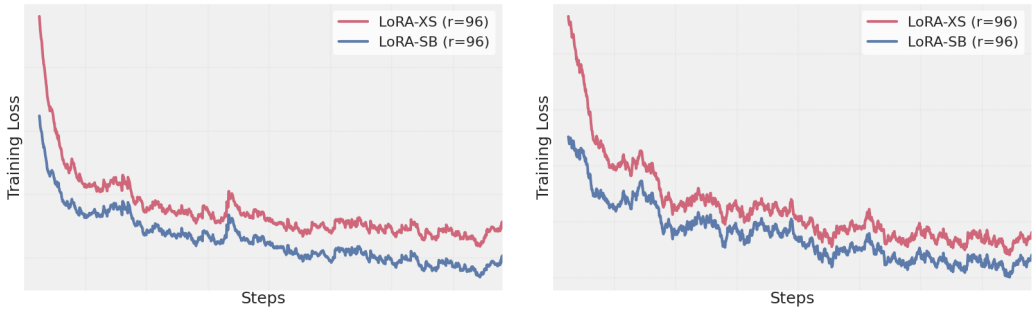
Method	Rank	# Params	CoLA Mcc ↑	RTE Acc ↑	MRPC Acc ↑	STS-B Corr ↑	QNLI Acc ↑	SST-2 Acc ↑	All Avg. ↑
Full FT	-	355.36 M	68.44	83.42	90.21	91.76	93.92	96.21	87.33
LoRA	8	2162.69 K	68.02	82.98	90.05	91.43	93.42	95.98	86.98
rsLoRA	8	2162.69 K	67.87	82.84	89.97	91.30	93.29	95.87	86.85
PiSSA	8	2162.69 K	68.22	83.14	90.10	91.59	93.55	96.03	87.10
DoRA	8	2260.99 K	68.05	83.04	89.93	91.34	93.11	95.82	86.88
LoRA-Pro	8	2162.69 K	67.98	**83.40**	**90.49**	91.38	93.37	95.98	87.10
LoRA-XS	8	6.14 K	61.07	75.23	86.21	89.29	92.44	94.72	83.16
LoRA-XS	16	24.57 K	63.32	79.06	86.28	90.36	93.69	95.76	84.70
LoRA-XS	24	55.20 K	66.27	80.14	88.48	90.77	93.21	95.89	85.79
LoRA-SB	8	6.14 K	63.57	78.43	88.72	90.59	92.95	95.07	84.88
LoRA-SB	16	24.57 K	64.36	82.31	89.71	91.24	**93.89**	95.87	86.23
LoRA-SB	24	55.20 K	**68.28**	83.03	90.12	**91.65**	93.75	**96.11**	**87.16**

3.1 Arithmetic Reasoning

We fine-tune Mistral-7B (19) and Gemma-2 9B (43) on 50K samples from MetaMathQA (50) and evaluate on GSM8K (8) and MATH (16). We apply LoRA modules to the key, value, query, attention output, and all fully connected weight matrices, training with ranks $r = \{32, 64, 96\}$. We present results in Table 2. LoRA-SB significantly outperforms LoRA-XS across all settings. LoRA-SB outperforms LoRA-based methods ($r = 32$) while using **40x** fewer trainable parameters for Mistral-7B and **90x** fewer for Gemma-2 9B at ranks $r = 96$ and $r = 64$, respectively. We present training loss curves comparing LoRA-SB and LoRA-XS in Figure 2. Thanks to superior initialization, LoRA-SB starts with a lower initial loss compared to LoRA-XS. Further, due to optimal gradient approximation, LoRA-SB maintains a consistently better loss throughout and converges to a superior final value.

3.2 Commonsense Reasoning

We fine-tune Llama-3.2 3B (12) on CommonSense170K, a dataset with eight commonsense reasoning tasks (18). LoRA modules are applied to the key, value, query, attention output, and all fully connected weight matrices, training with ranks $r = \{32, 64, 96\}$. We present the results in Table 3. LoRA-SB consistently outperforms LoRA-XS across all settings. In addition, LoRA-SB ($r = 96$) outperforms LoRA-based methods ($r = 32$) with **27x** fewer trainable parameters.



(a) Mistral-7B (b) Gemma-2 9B

Figure 2: Training loss curves for Mistral-7B and Gemma-2 9B, comparing LoRA-SB and LoRA-XS.

3.3 Natural Language Understanding

We fine-tune RoBERTa-large (27) on GLUE, a popular language understanding benchmark. LoRA modules are applied only to the self-attention layers, with ranks $r = \{8, 16, 24\}$. Results are shown in Table 4. LoRA-SB consistently outperforms LoRA-XS across all settings. Additionally, LoRA-SB ($r = 24$) outperforms LoRA-based methods ($r = 8$) with **39x** lesser trainable parameters.

4 Analysis

Optimal Initialization is Important!

To isolate the impact of initialization, we take truncated SVD on various matrices, including Kaiming initialization (15) and ΔW_{avg} with varying levels of Gaussian noise, as shown in Table 5. By applying truncated SVD, we ensure optimal gradient approximation, leading to initialization matrices B_{init} and A_{init} that form orthonormal bases in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively. This results in $B^TB = AA^T = I$, allowing us to isolate the effect of initialization. The results clearly demonstrate the significance of initialization, our approach consistently outperforms other variants.

Table 5: Comparison of initialization strategies using Mistral-7B on GSM8K and MATH. All methods ensure optimal gradient approximation, with differences arising solely from the initialization.

Initialization Method	**Accuracy (↑)**	
	GSM8K	**MATH**
trunc_SVD (Kaiming)	00.00	00.00
trunc_SVD ($\Delta W_{avg} + \mathcal{N}_{\mu=10^{-2}}$)	00.00	00.00
trunc_SVD ($\Delta W_{avg} + \mathcal{N}_{\mu=10^{-3}}$)	58.83	14.76
trunc_SVD ($\Delta W_{avg} + \mathcal{N}_{\mu=10^{-4}}$)	60.19	15.96
trunc_SVD ($\Delta W_{avg} + \mathcal{N}_{\mu=10^{-5}}$)	60.65	15.98
LoRA-SB; trunc_SVD (ΔW_{avg})	**63.38**	**17.44**

Why Do We Use 0.1% of the Dataset Size for Initialization?

We selected the 0.1% initialization dataset-size heuristic based on experiments that suggested it provides a good tradeoff between quality and efficiency. Specifically, we conducted ablations varying the number of samples used for initialization when fine-tuning Mistral-7B and Gemma-2 9B on 50k samples from MetaMathQA. The results (Table 6) show that once the sample count exceeds a modest threshold (25 samples or 0.05%), performance quickly plateaus, indicating that the learned subspace is already sufficiently representative. Using 0.1% of the training data (50 samples) consistently exceeds this threshold across tasks and models, while incurring negligible training time overhead.

Optimal Gradient Approximation is Important!

Table 6: Performance effect of number of samples used for initialization.

# Samples	Mistral-7B		Gemma-2 9B	
	GSM8K (↑)	MATH (↑)	GSM8K (↑)	MATH (↑)
1	62.13	15.55	76.03	35.77
5	62.78	16.86	77.49	37.24
25	63.28	17.30	78.18	37.70
50	63.38	17.44	78.40	37.70
100	63.34	17.25	78.22	37.45
200	63.45	17.36	78.43	37.87
500	63.40	17.52	78.54	37.63

We aim to examine the effect of optimal gradient approximation. Specifically, we want $B_{\text{init}}R_{\text{init}}A_{\text{init}} \approx \Delta W_{avg}$ without enforcing $B^TB = AA^T = I$. We achieve this through:

$$U, S, V^T \leftarrow \textbf{SVD}(\Delta W_{avg}) \tag{11}$$

$$B_{\text{init}} \leftarrow U[1:r]S[1:r, 1:r], A_{\text{init}} \leftarrow V[1:r], R_{\text{init}} \leftarrow I \tag{12}$$

This ensures that $B_{\text{init}}R_{\text{init}}A_{\text{init}} \approx \Delta W_{avg}$, but only $AA^T = I$, while $B^TB \neq I$. The setup is suboptimal for gradient approximation since we do not explicity use the closed-form solution derived in Theorem 3. We compare the resulting loss curves against LoRA-SB (which uses optimal gradient approximation) for Mistral-7B, as shown in Figure 3 in Appendix E. Although both start similarly due to effective initialization, LoRA-SB converges to significantly better values, demonstrating the advantage of optimal gradient approximation. Furthermore, LoRA-SB achieves higher accuracies on GSM8K and MATH, with scores of 63.38 and 17.44 compared to 55.87 and 12.74, respectively.

Training Time and Inference.

We provide detailed benchmarks of training time and inference performance in Appendix F and G, respectively. As shown, the initialization step in LoRA-SB introduces only a negligible training-time overhead compared to LoRA ($\approx 1.1\% - 1.3\%$).

5 Conclusion

In this work, we introduced LoRA-SB, which bridges the gap between low-rank PEFT and full FT. This is enabled by our initialization strategy, which approximates the first step of full FT and ensures that the most relevant subspaces for task-specific adaptation are captured. We achieve optimal gradient scaling and preserve update directions throughout training. Our approach ensures scaling factor independence by approximating the full FT gradient, thereby eliminating potential instability issues. Through extensive experiments, we demonstrate that our method outperforms LoRA (and baselines) using upto **90x** less parameters, and comprehensively outperforms LoRA-XS.

Reproducibility Statement

We have taken great care to guarantee the reproducibility of our work. Our open-source code is anonymously available at https://anonymous.4open.science/r/lora-sb-anonymous-5BEE and is also included in the supplementary material. Comprehensive details of the experimental setup are presented in Section 3 and Appendix I. All datasets used in this study are standard, publicly accessible benchmarks (see Appendix J for further information).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. (arXiv:2405.17604), October 2024. arXiv:2405.17604 [cs].

[3] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[4] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[5] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[6] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

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

[8] 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, 2021.

[9] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. (arXiv:2305.14314), May 2023. arXiv:2305.14314 [cs].

[10] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, March 2023.

[11] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*, 2005.

[12] 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.

[13] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[14] Yaru Hao, Haoyu Song, Li Dong, Shaohan Huang, Zewen Chi, Wenhui Wang, Shuming Ma, and Furu Wei. Language models are general-purpose interfaces. (arXiv:2206.06336), June 2022. arXiv:2206.06336 [cs].

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[16] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. (arXiv:2106.09685), October 2021. arXiv:2106.09685 [cs].

[18] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models, 2023.

[19] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[20] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora. (arXiv:2312.03732), November 2023. arXiv:2312.03732 [cs].

[21] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003, 2023.

[22] Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. Vera: Vector-based random matrix adaptation. (arXiv:2310.11454), January 2024. arXiv:2310.11454 [cs].

[23] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. (arXiv:2104.08691), September 2021. arXiv:2104.08691 [cs].

[24] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. (arXiv:2101.00190), January 2021. arXiv:2101.00190 [cs].

[25] Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates. (arXiv:2307.05695), December 2023. arXiv:2307.05695 [cs].

[26] 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:2402.09353), July 2024. arXiv:2402.09353 [cs].

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[29] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources, 2024.

[30] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. (arXiv:2404.02948), May 2024. arXiv:2404.02948 [cs].

[31] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[32] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59, 1960.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[34] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. (arXiv:2005.00247), January 2021. arXiv:2005.00247 [cs].

[35] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.

[36] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[37] Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-lora: Enhancing parameter efficiency of lora with weight tying. (arXiv:2311.09578), April 2024. arXiv:2311.09578 [cs].

[38] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[39] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[40] Raghav Singhal, Kaustubh Ponkshe, and Praneeth Vepakomma. Exact aggregation for federated and efficient fine-tuning of foundation models. *arXiv preprint arXiv:2410.09432*, 2024.

[41] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[42] Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. Improving lora in privacy-preserving federated learning, 2024.

[43] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[44] Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. Hydralora: An asymmetric lora architecture for efficient fine-tuning. (arXiv:2404.19245), May 2024. arXiv:2404.19245 [cs].

[45] Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. (arXiv:2407.05000), July 2024. arXiv:2407.05000 [cs].

[46] Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Lora-pro: Are low-rank adapters properly optimized? (arXiv:2407.18242), October 2024. arXiv:2407.18242 [cs].

[47] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[49] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models. (arXiv:2404.03592), May 2024. arXiv:2404.03592 [cs].

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

[51] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[52] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. (arXiv:2303.10512), December 2023. arXiv:2303.10512 [cs].

[53] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. (arXiv:2403.03507), June 2024. arXiv:2403.03507 [cs].

Appendix

Contents

A Related Work

Parameter-Efficient Fine-Tuning (PEFT). PEFT methods have become essential for adapting large pre-trained models under computational constraints. Early techniques like AdapterFusion (34) and Prefix-Tuning (24) enabled task-specific adaptation with minimal parameter updates. Advances like soft prompts (23) further reduced trainable parameter counts while maintaining strong performance. Recent approaches have explored operating directly on model representations (49).

Low-Rank Decomposition Methods. LoRA (17) demonstrated that weight updates during FT could be efficiently approximated using low-rank matrices, drastically reducing parameter counts. Building on this insight, variants such as QLoRA (9) and AdaLoRA (52) extended the paradigm through quantization and adaptive allocation strategies. The applicability of low-rank techniques has also been explored in pretraining with GaLore (53) and ReLoRA (25), highlighting the versatility of low-rank adaptation methods. LoRA-based methods have also been applied in other domains, such as efficient federated FT (42; 40).

Enhancing LoRA Performance. Recent efforts have focused on optimizing LoRA's performance. PiSSA (30) demonstrated improvements by initializing matrices with principal components of pre-trained weights. LoRA-Pro (46) and LoRA-GA (45) improved gradient approximation, aligning low-rank updates more closely with full FT. Methods like DoRA (26) and rsLoRA (20) introduced decomposition-based and scaling stabilization techniques to enhance learning stability and expand LoRA's utility.

Improving Efficiency in LoRA Variants. Efficiency-focused innovations have pushed LoRA toward more parameter savings. LoRA-XS (2) achieves this by inserting a small trainable weight matrix into frozen low-rank matrices. VeRA (22) shares low-rank matrices across layers, relying on scaling vectors for task-specific adaptation. Tied-LoRA (37) leverages weight tying to reduce parameter usage at higher ranks, while HydraLoRA (44) introduces an asymmetric architecture for improvement.

B PROOFS

In all the proofs below, we will use the notations defined in Section 2.

B.1 PROOF OF LEMMA 1

Lemma. *Let ΔW be an update learned with LoRA-XS. Then, the set of all possible ΔW, say $\mathcal{W}_{LoRA-XS}$, is given as:*

$$\mathcal{W}_{LoRA-XS} = \{M \in \mathbb{R}^{m\times n} | \textit{Col}(M) \subseteq \textit{Col}(B) \wedge \textit{Row}(M) \subseteq \textit{Row}(A)\},$$

where $\textit{Col}(M)$ and $\textit{Row}(M)$ are column and row spaces of matrix M respectively.

Proof. Since $\Delta W = BRA$, we have

$$\begin{aligned} \text{Col}(\Delta W) = \{y \in \mathbb{R}^m \mid y = BRAx,\, x \in \mathbb{R}^n\} \implies \\ \text{Col}(\Delta W) = \{y \in \mathbb{R}^m \mid y = Bz,\, z \in \text{Col}(RA)\} \subseteq \text{Col}(B). \end{aligned}$$

That is, we proved that

$$\text{Col}(\Delta W) \subseteq \text{Col}(B). \tag{13}$$

Following similar arguments, one can also show $\text{Row}(\Delta W) \subseteq \text{Row}(A)$. □

B.2 PROOF OF LEMMA 2

Lemma. *The gradient of the loss with respect to matrix R can be expressed in terms of the gradient with respect to the weight matrix W as:*

$$g^R_{LoRA-XS} = sB^\top g A^\top.$$

Proof. Let L be the loss function. We have already defined g and $g^R_{\text{LoRA-XS}}$ as:

$$g := \frac{\partial L}{\partial W} \quad \& \quad g^R_{\text{LoRA-XS}} := \frac{\partial L}{\partial R}. \tag{14}$$

The chain rule gives

$$\frac{\partial L}{\partial R} = \frac{\partial L}{\partial W}\frac{\partial W}{\partial R} \implies \frac{\partial L}{\partial R} = \frac{\partial L}{\partial W}\frac{\partial W}{\partial X}\frac{\partial X}{\partial R} \quad \text{for } X = RA \tag{15}$$

We know that for $W = sBX$:

$$\frac{\partial L}{\partial W}\frac{\partial W}{\partial X} = sB^\top g \implies \frac{\partial L}{\partial R} = sB^\top g \frac{\partial X}{\partial R} \tag{16}$$

Let $sB^\top g = y$. We know that when $X = RA$:

$$y\frac{\partial X}{\partial R} = yA^\top \implies \frac{\partial L}{\partial R} = yA^\top = sB^\top gA^\top \tag{17}$$

$$\text{Therefore,} \quad \boxed{g^R_{\text{LoRA-XS}} = sB^\top gA^\top} \tag{18}$$

□

B.3 Proof of Theorem 3

Theorem. *For full-rank A and B matrices, the optimal solution for the objective $min_{g^R}||\tilde{g} - g||_F^2$, such that $\tilde{g} = sBg^RA$, is: $g^R = \frac{1}{s^2}(B^\top B)^{-1}g^R_{LoRA-XS}(AA^\top)^{-1}$.*

Proof. Since we already defined the equivalent gradient $\tilde{g} := sBg^RA$, the minimization problem can be denoted as:

$$\arg\min_{g^R} F = \|sBg^RA - g\|_F^2 \tag{19}$$

For differentiable F,

$$\frac{\partial F}{\partial g^R} = 0 \implies 2(\tilde{g} - g)\cdot\frac{\partial \tilde{g}}{\partial g^R} = 0 \implies 2(sBg^RA - g)\cdot\frac{\partial(sBg^RA)}{\partial g^R} = 0 \tag{20}$$

Using the same trick from before and substituting $g^RA = X$, we get:

$$2sB^\top(sBg^RA - g)A^\top = 0 \implies B^\top(sBg^RA - g)A^\top = 0 \implies B^\top sBg^RAA^\top = B^\top gA^\top \tag{21}$$

From Lemma 2, we get:

$$B^\top gA^\top = g^R_{\text{LoRA-XS}}/s \implies B^\top sBg^RAA^\top = g^R_{\text{LoRA-XS}}/s \implies B^\top Bg^RAA^\top = g^R_{\text{LoRA-XS}}/s^2 \tag{22}$$

Now since B and A are full rank, multiplying both sides by $(B^\top B)^{-1}$ and $(AA^\top)^{-1}$ on the left and right side respectively gives:

$$(B^\top B)^{-1}(B^\top Bg^RAA^\top)(AA^\top)^{-1} = (B^\top B)^{-1}g^R_{\text{LoRA-XS}}(AA^\top)^{-1}/s^2 \tag{23}$$

$$\text{Therefore,} \quad \boxed{g^R = \frac{1}{s^2}(B^\top B)^{-1}g^R_{\text{LoRA-XS}}(AA^\top)^{-1}} \tag{24}$$

□

B.4 Proof of Theorem 4

Theorem. *Consider the update for matrix R using the solution derived in Theorem 3:*

$$R \leftarrow R - \eta g^R$$

where $\eta > 0$ is the (sufficiently small) learning rate. This update guarantees a reduction in the loss ΔL, given by:

$$\Delta L := L(W_0 + sB(R - \eta g^R)A) - L(W_0 + sBRA) = -\eta\langle g^R_{LoRA-XS}, g^R\rangle_F + o(\eta) \leq 0.$$

Proof. Assuming that L is differentiable, we use Taylor's theorem and get

$$\begin{aligned}\Delta L &\coloneqq L(W_0 + sB(R - \eta g^R)A) - L(W_0 + sBRA) \\ &= \left\langle \frac{\partial L}{\partial R}, -\eta g^R \right\rangle_F + o(\eta) \\ &= -\frac{\eta}{s^2}\langle g^R_{\text{LoRA-XS}}, (B^\top B)^{-1} g^R_{\text{LoRA-XS}} (AA^\top)^{-1}\rangle_F + o(\eta), \end{aligned} \tag{25}$$

where in the last step we also used the definition of $g^R_{\text{LoRA-XS}}$ and the result of Theorem 3. To prove $\Delta L \leq 0$ for small enough η, it is sufficient to show that

$$\langle g^R_{\text{LoRA-XS}}, (B^\top B)^{-1} g^R_{\text{LoRA-XS}} (AA^\top)^{-1}\rangle_F \geq 0. \tag{26}$$

Next, we note that matrices $B^\top B \in \mathbb{R}^{r\times r}$ and $AA^\top \in \mathbb{R}^{r\times r}$ are positive definite since they are positive semi-definite and matrices B and A are full-rank (i.e., with rank r) matrices, which means that $B^\top B$ and $AA^\top$ have non-zero eigenvalues. Therefore, $(B^\top B)^{-1}$ and $(AA^\top)^{-1}$ are also positive definite, implying that there exist matrices X and Y such that $(B^\top B)^{-1} = YY^\top$ and $(AA^\top)^{-1} = XX^\top$ (e.g., one can find such matrices using Cholesky decomposition). Then, we have

$$\begin{aligned}\langle g^R_{\text{LoRA-XS}}, (B^\top B)^{-1} g^R_{\text{LoRA-XS}} (AA^\top)^{-1}\rangle_F &= \langle g^R_{\text{LoRA-XS}}, YY^\top g^R_{\text{LoRA-XS}} XX^\top\rangle_F \\ &= \langle Y^\top g^R_{\text{LoRA-XS}} X, Y^\top g^R_{\text{LoRA-XS}} X\rangle_F \\ &= \|Y^\top g^R_{\text{LoRA-XS}} X\|_F^2 \geq 0.\end{aligned}$$

This concludes the proof. □

For our specific initialization where $(B^\top B) = I$, $(AA^\top) = I$, and $s = 1$, the result simplifies to:

$$\Delta L = -\eta\langle g^R_{\text{LoRA-XS}}, g^R_{\text{LoRA-XS}}\rangle_F + o(\eta) \leq 0. \tag{27}$$

B.5 Proof of Theorem 5

Theorem. *The equivalent gradient $\tilde{g}$ is hyperparameter s independent when*

$$\tilde{g} = sBg^R A \quad \textit{but not when} \quad \tilde{g} = sBg^R_{LoRA-XS} A.$$

Proof. Let g be the full fine-tuning gradient. We want to prove that $\tilde{g}$ does not depend on s, so we try to express it in terms of g which does not depend on the LoRA-XS training process or reparameterization.

1) For $\tilde{g} = sBg^R A$:

$$g^R = \frac{1}{s^2}(B^\top B)^{-1} g^R_{\text{LoRA-XS}} (AA^\top)^{-1} \implies \tilde{g} = \frac{s}{s^2} B(B^\top B^{-1}) g^R_{\text{LoRA-XS}} (AA^\top)^{-1} A \tag{28}$$

Now since $g^R_{\text{LoRA-XS}} = sB^\top g A^\top$:

$$\tilde{g} = \frac{1}{s} B(B^\top B^{-1}) sB^\top g A^\top (AA^\top)^{-1} A = B(B^\top B^{-1}) B^\top g A^\top (AA^\top)^{-1} A. \tag{29}$$

which is s-independent.

2) For $\tilde{g} = sBg^R_{\text{LoRA-XS}} A$

$$g^R_{\text{LoRA-XS}} = sB^\top g A^\top \implies \tilde{g} = sB(sB^\top g A^\top) A \implies \tilde{g} = s^2 BB^\top g A^\top A \tag{30}$$

which is not s-independent. □

B.6 PROOF OF THEOREM 6

Theorem. *If A_{init} and B_{init} are initialized using LoRA-SB for the first step of SGD optimizer, then the update given by LoRA-SB, $\Delta(B_{init}R_{init}A_{init})$, is the best low-rank approximation of full fine-tuning update, ΔW.*

Proof. Consider a gradient descent step with learning rate η and updates for R:

$$\Delta R = -\eta \nabla_R \mathcal{L}(R) \implies B\Delta R A = -\eta B \nabla_R \mathcal{L}(R) A. \tag{31}$$

To measure its approximation quality of update of the weights in full finetuning:

$$\Delta W = -\eta \nabla_W \mathcal{L}(W_0). \tag{32}$$

We use Frobenius norm of the difference between these two updates as a criterion:

$$\|B\Delta R A - \eta \nabla \mathcal{L}_W(W_0)\|_F = \eta \|B \nabla_R \mathcal{L}(R) A - \nabla \mathcal{L}_W(W_0)\|_F. \tag{33}$$

We have shown before that:

$$\nabla_R \mathcal{L} = B^\top \nabla_W \mathcal{L} A^\top. \tag{34}$$

The problem now becomes:

$$\min_{A_{\text{init}}, B_{\text{init}}} \|B^\top (B^\top \nabla_W \mathcal{L} A^\top) A - \nabla_W \mathcal{L}\|_F \quad \text{where } \nabla_W \mathcal{L} = USV^\top. \tag{35}$$

Using our initialization, we get:

$$\|BB^\top \nabla_W \mathcal{L} A^\top A - \nabla_W \mathcal{L}\|_F = \|U_{IR}U_{IR}^\top USV^\top V_{IR}V_{IR}^\top - USV^\top\|_F. \tag{36}$$

Moreover, we also have

$$U_{IR}U_{IR}^\top USV^\top V_{IR}V_{IR}^\top = \sum_{i=1}^{r} \sigma_i u_i v_i^\top. \tag{37}$$

The rank of W' such that

$$W' = U_{IR}U_{IR}^\top USV^\top V_{IR}V_{IR}^\top \tag{38}$$

is $\leq r$, since the corresponding ranks of B_{init} and A_{init} is r. Using the Eckart-Young Theorem, we find the optimal low-rank solution as:

$$W'^* = \underset{\text{rank}(W')=r}{\arg\min} \|W' - \nabla_W \mathcal{L}\|_F = \sum_{i=1}^{r} \sigma_i u_i v_i^\top. \tag{39}$$

Since we also get an identical expression, our solution is optimal. □

C SIMULATING THE FIRST STEP OF FULL FINE-TUNING UNDER ADAMW

Our initialization is designed to approximate the first update step that would occur during full fine-tuning using the AdamW optimizer, which is also used in LoRA-SB training. AdamW computes the parameter update using both first and second moment estimates of the gradient. At the first step, these moments are initialized to zero, so the update becomes:

$$\theta_1 = \theta_0 - \alpha \cdot \frac{g_1}{\sqrt{g_1^2 + \epsilon}} \approx -\alpha \cdot \text{sign}(g_1)$$

where g_1 is the gradient at the first step, ϵ is a small constant for numerical stability, and α is the learning rate. Due to zero-initialization and bias correction, the direction of the update is approximately the element-wise sign of the gradient.

To simulate this behavior in our low-rank initialization, we use:

$$\Delta W_{\text{avg}} = -\eta \cdot \text{sign}\left(\sum_{i=1}^{n} \nabla_W \mathcal{L}(W_0, x_i)\right)$$

This reflects the direction of the first AdamW step averaged over a mini-batch. By using the sign of the gradient sum, we ensure our initialization aligns with the dynamics of AdamW, leading to a consistent and faithful approximation of full fine-tuning updates within the low-rank subspace.

D ALGORITHM

We provide a pseudo-code implementation of our method in Algorithm 1.

Algorithm 1 LoRA-SB, PyTorch-like

```
def initSB(model, D)
    # Estimate gradient with n samples
    ΔW_avg ← est_grad(model, D, n)
    # Initialize B, R, A
    (B, R, A) ← trunc_SVD(ΔW_avg)
    # Convert to LoRA-SB model
    sb_model ← lora_SB(model, B, R, A)
    return sb_model

# Load pre-trained model
 model ← AutoModel(base_model)
# Initialize LoRA-SB with D
sb_model ← initSB(model, D)
# Train, only R trainable
trainer ← Trainer(sb_model,...)
trainer.train()
```

E OPTIMAL GRADIENT APPROXIMATION IS IMPORTANT!

As discussed in Section 4, optimal gradient approximation plays a key role in the effectiveness of LoRA-SB. In Figure 3, we compare the loss curves of models trained with and without this component on Mistral-7B. While both variants begin with similar performance due to effective initialization, LoRA-SB with optimal gradient approximation converges to substantially lower loss values, highlighting its contribution to improved optimization.

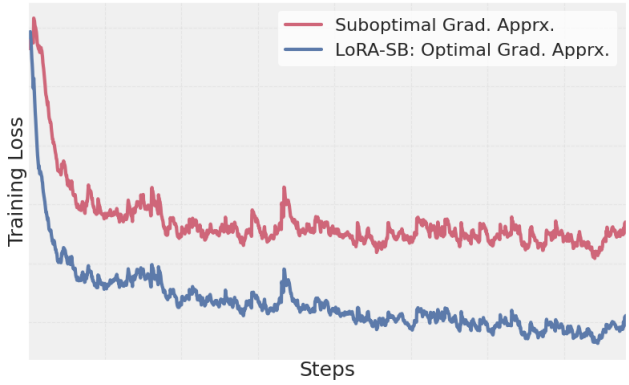


Figure 3: Training loss for Mistral-7B, highlighting the impact of optimal gradient approximation.

F TRAINING TIME OVERHEAD VS LORA-XS

As previously mentioned, we compute the update approximation using only $1/1000$ of the total training samples for each dataset. Table 7 presents the associated training time overhead for these computations, compared to LoRA-XS. The results show that the **additional overhead is negligible**, adding just 2–4 minutes compared to the total training time of 3–5 hours per epoch ($\approx 1.1\%$ to 1.3%). Additionally, the update computation is performed only once, at the beginning of the first epoch, prior to training. Notably, the initialization step is highly efficient, as we directly compute the **truncated**

SVD using optimized PyTorch libraries (`torch.svd_lowrank`). For reference, this **computation takes less than one second for each of the entire LLMs** used in our experiments.

Table 7: Training time overhead due to the initialization for various models on their respective tasks.

Model	Overhead	Training Time/Epoch
Mistral-7B	0:02:01	3:03:57
Gemma-2 9B	0:03:46	4:13:24
Llama-3.2 3B	0:03:54	4:54:31

G Inference Overhead vs LoRA

LoRA-SB introduces a minimal inference cost overhead due to the insertion of the $r \times r$ matrix R between B and A, and the need for higher ranks to achieve comparable performance to LoRA. We benchmark the inference-time FLOPs and MACs across various models and find that the overhead is negligible. This comparison is presented in Table 8, showing that the additional overhead of LoRA-SB is negligible.

Table 8: Inference cost comparison between LoRA-SB and LoRA across various models for a sequence length of 256. The minimum rank at which LoRA-SB matches or exceeds LoRA's performance is highlighted in **bold**.

Model	Method	Rank	MACs	FLOPs
RoBERTa-large	LoRA	8	77.86 G	155.79 G
	LoRA-SB	16	78.42 G	156.91 G
	LoRA-SB	24	78.97 G	158.01 G
Llama-3.2 3B	LoRA	32	0.84 T	1.67 T
	LoRA-SB	64	0.85 T	1.70 T
	LoRA-SB	96	0.86 T	1.72 T
Mistral 7B	LoRA	32	1.84 T	3.69 T
	LoRA-SB	64	1.86 T	3.73 T
	LoRA-SB	92	1.88 T	3.77 T
Gemma-2 9B	LoRA	32	3.89 T	7.77 T
	LoRA-SB	64	3.93 T	7.86 T
	LoRA-SB	96	3.97 T	7.94 T

H Additional Results

We present the standard deviation results across multiple runs in Tables 9 and 10.

I Experiment Details

We use PyTorch (33) and the HuggingFace Transformers library (48) for our implementations. We run all experiments on a **single NVIDIA A6000 GPU** and report results as the average of three random seeds. To save memory, we initialize base models in `torch.bfloat16` precision. We trained all models using the AdamW optimizer (28). **We compute the update approximation using only $1/1000$ of each dataset's total number of samples**. The samples are randomly selected from the training set in each run.

For arithmetic and commonsense reasoning tasks, we set up Mistral-7B, Gemma-2 9B, and Llama-3.2 3B with hyperparameters and configurations listed in Table 11. We adopted most settings from previous studies (18) but conducted our own learning rate sweep. Following LoRA-XS guidelines, we set $\alpha = r$ for their baseline configuration.

Table 9: Standard deviation results on Mistral-7B and Gemma-2 9B across arithmetic reasoning benchmarks.

Method	Rank	Mistral-7B			Gemma-2 9B		
		# Params	GSM8K	MATH	# Params	GSM8K	MATH
Full FT	-	7.24 B	0.32	0.22	9.24 B	0.27	0.19
LoRA	32	83.88 M	0.58	0.49	108.04 M	0.50	0.41
rsLoRA	32	83.88 M	0.64	0.44	108.04 M	0.46	0.39
PiSSA	32	83.88 M	0.56	0.47	108.04 M	0.53	0.36
DoRA	32	85.26 M	0.49	0.41	109.88 M	0.45	0.38
LoRA-GA	32	85.26 M	0.72	0.59	109.88 M	0.63	0.48
LoRA-Pro	32	83.88 M	0.44	0.33	108.04 M	0.38	0.31
LoRA-XS	32	0.23 M	0.92	0.77	0.30 M	0.78	0.63
LoRA-XS	64	0.92 M	0.85	0.70	1.20 M	0.72	0.57
LoRA-XS	96	2.06 M	0.78	0.66	2.71 M	0.68	0.52
LoRA-SB	32	0.23 M	0.70	0.52	0.30 M	0.58	0.45
LoRA-SB	64	0.92 M	0.60	0.47	1.20 M	0.50	0.39
LoRA-SB	96	2.06 M	0.52	0.39	2.71 M	0.43	0.33

Table 10: Standard deviation results for each metric on Llama-3.2 3B across commonsense reasoning benchmarks.

Method	Rank	# Params	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
Full FT	-	3.21 B	0.92	0.71	0.78	0.63	0.58	0.74	0.88	0.82	0.76
LoRA	32	48.63 M	1.24	1.05	0.96	0.88	0.83	1.02	1.16	1.11	1.03
rsLoRA	32	48.63 M	1.28	1.01	0.91	0.84	0.79	0.98	1.13	1.07	1.00
PiSSA	32	48.63 M	1.21	0.97	0.88	0.82	0.77	0.93	1.08	1.04	0.96
DoRA	32	49.40 M	1.15	0.93	0.85	0.80	0.76	0.92	1.05	1.02	0.94
LoRA-Pro	32	48.63 M	1.10	0.90	0.83	0.78	0.74	0.88	1.01	0.99	0.91
LoRA-XS	32	0.20 M	1.66	1.41	1.28	1.14	1.09	1.32	1.44	1.37	1.34
LoRA-XS	64	0.80 M	1.54	1.32	1.21	1.09	1.03	1.26	1.38	1.31	1.27
LoRA-XS	96	1.81 M	1.48	1.25	1.17	1.05	0.99	1.22	1.34	1.26	1.22
LoRA-SB	32	0.20 M	1.42	1.18	1.10	0.98	0.94	1.14	1.26	1.19	1.15
LoRA-SB	64	0.80 M	1.33	1.11	1.05	0.93	0.91	1.10	1.22	1.16	1.12
LoRA-SB	96	1.81 M	1.27	1.06	1.01	0.90	0.88	1.07	1.18	1.13	1.09

For the GLUE benchmark using RoBERTa-large, you can find the hyperparameter details in Table 12. We mostly adhered to the original configurations from the LoRA paper (17) but adjusted the learning rate through a sweep. In line with LoRA-XS settings, we fixed α at 16 for their baseline.

For all tasks, we followed the baseline configurations provided in the PiSSA (30), rsLoRA (20), DoRA (26), and LoRA-Pro (46) papers for our comparisons.

J Dataset Details

The **MetaMathQA** dataset (50) creates mathematical questions by rephrasing existing ones from different viewpoints, without adding new information. We assess this dataset using two benchmarks: **GSM8K** (8), which consists of grade-school math problems requiring multi-step reasoning, and **MATH** (16), which presents difficult, competition-level math problems. Evaluation focuses solely on the final numeric answer.

CommonSense170K is a comprehensive dataset that consolidates eight commonsense reasoning datasets (18). Each example is framed as a multiple-choice question where the model generates the correct answer without explanations. We use the prompt template from (18). The individual datasets used are described below:

Table 11: Hyperparameter settings for training Mistral-7B and Gemma-2 9B on MetaMathQA, and Llama-3.2 3B on COMMONSENSE170K.

	Mistral-7B / Gemma-2 9B	**Llama-3.2 3B**
Optimizer	AdamW	AdamW
Batch size	1	6
Max. Seq. Len	512	256
Grad Acc. Steps	32	24
Epochs	1	2
Dropout	0	0.05
Learning Rate	1×10^{-4}	2×10^{-3}
LR Scheduler	Cosine	Linear
Warmup Ratio	0.02	0.02

Table 12: Hyperparameter settings for RoBERTa-large on GLUE.

	CoLA	**RTE**	**MRPC**	**SST-2**	**QNLI**	**STS-B**
Optimizer	AdamW					
Batch size	128					
Max Seq. Len.	256					
Epochs	30	30	30	15	15	30
Dropout	0					
Learning Rate	1×10^{-3}					
LR Scheduler	Linear					
Warmup Ratio	0.06					

1. **HellaSwag** (51) challenges models to select the most plausible continuation of a given scenario from multiple possible endings.
2. **ARC Easy** (or **ARC-e**) (7) includes basic science questions at a grade-school level, offering simpler tasks to assess fundamental reasoning abilities.
3. **PIQA** (3) evaluates physical commonsense reasoning, where models must choose the best action to take in a hypothetical scenario.
4. **SIQA** (39) tests social commonsense reasoning by asking models to predict the social consequences of human actions.
5. **WinoGrande** (38) presents sentence completion tasks requiring commonsense reasoning to select the correct binary option.
6. **ARC Challenge** (or **ARC-c**) (7) consists of more complex science questions designed to challenge models with sophisticated reasoning, beyond simple co-occurrence patterns.
7. **OBQA** (31) features open-book, knowledge-intensive QA tasks that require multi-hop reasoning across multiple information sources.
8. **BoolQ** (6) involves answering yes/no questions based on real-world, naturally occurring queries.

The **GLUE Benchmark** is a comprehensive collection of tasks designed to evaluate natural language understanding (NLU) abilities. It included various datasets, including **STS-B** for measuring semantic textual similarity (5), **RTE** for recognizing textual entailment, **MRPC** for detecting paraphrases (11), **CoLA** for assessing linguistic acceptability (47), **SST-2** for sentiment analysis (41), and **QNLI** for question-answer inference (36). GLUE's broad scope makes it a standard benchmark for evaluating models like RoBERTa.

K USE OF LARGE LANGUAGE MODELS

LLMs are only used for small writing improvements, like polishing grammar and smoothing out phrasing.