Learning Rate Matters: Vanilla LoRA May Suffice for LLM Fine-tuning

Yu-Ang Lee [1] **Ching-Yun Ko** [2] **Pin-Yu Chen** [2] **Mi-Yen Yeh** [1 3]

Abstract

Low-Rank Adaptation (LoRA) is the prevailing approach for efficient large language model (LLM) fine-tuning. Building on this paradigm, recent studies have proposed alternative initialization strategies and architectural modifications, reporting substantial improvements over vanilla LoRA. However, these gains are often demonstrated under fixed or narrowly tuned hyperparameter settings, despite the known sensitivity of neural networks to training configurations. In this work, we systematically re-evaluate nine representative LoRA variants alongside vanilla LoRA through extensive hyperparameter searches. Across tasks spanning mathematical reasoning, commonsense reasoning, code generation, and instruction following at diverse model scales, we find that different LoRA methods favor distinct learning rate ranges. Crucially, once learning rates are properly tuned, all methods achieve similar peak performance (within 1–2%), with only subtle rank-dependent behaviors. These results suggest that vanilla LoRA remains a competitive baseline and that improvements reported under a single training configuration may not reflect consistent methodological advantages. Finally, a second-order analysis attributes the differing optimal learning rate ranges to variations in the largest Hessian eigenvalue, aligning with classical learning theories. Our code is publicly available at `https://github.com/yuang-lee/lr-matters-lora`.

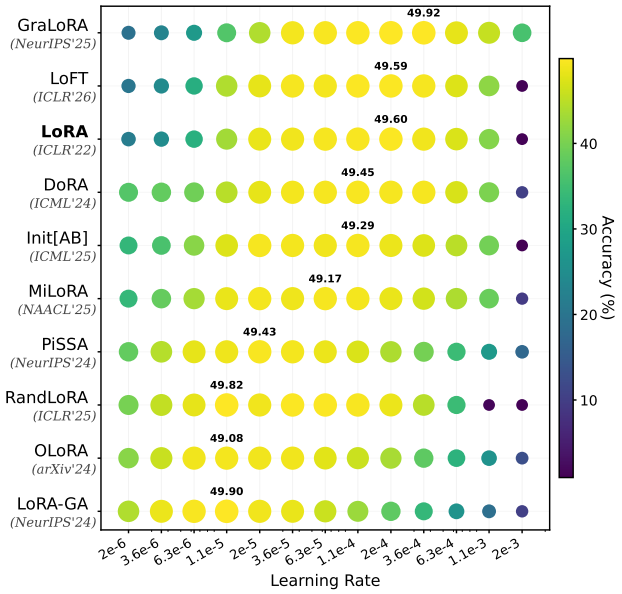


Figure 1. Performance of Qwen3-0.6B fine-tuned on mathematical reasoning tasks across learning rates. Different methods reach a similar performance level once the learning rate is properly tuned. Each point is averaged over three training runs. We annotate the peak accuracy of each method and sort methods by their optimal learning rate ranges. Results for other model–task combinations and training setups are reported in Sec. 4.3 and Appendix Sec. A.

1. Introduction

Despite the rapidly growing capabilities of pretrained large language models (LLMs), fine-tuning remains a fundamental step for adapting these models to specialized applications in diverse domains such as medicine (Anisuzzaman et al., 2025) and finance (Djagba & Saley, 2025). However, modern LLMs typically contain billions of parameters, making full-parameter fine-tuning (Full FT) prohibitively expensive in terms of memory and computation. These constraints have motivated sustained research interest in developing parameter-efficient fine-tuning (PEFT) methods, which allow task-specific learning while updating only a small fraction of parameters.

Even though PEFT methods span diverse design paradigms, ranging from prompt-based approaches (Li & Liang, 2021; Lester et al., 2021) to adapter-based methods (Houlsby et al., 2019; He et al., 2021), low-rank adaptation (LoRA), introduced by Hu et al. (2022), has emerged as the de facto standard. Inspired by the low intrinsic dimensionality observed in pretrained models (Li et al., 2018), Hu et al. (2022) hypothesize that task-specific parameter updates can be well approximated by low-dimensional structures. Consequently, they inject pairs of trainable decomposition matrices into selected layers while keeping the pretrained weights frozen. After training, these learned low-rank adapters can be merged into the original backbone, thereby incurring no

[1]National Taiwan University [2]IBM Research [3]Academia Sinica. Correspondence to: <r12946015@ntu.edu.tw, cyko@ibm.com, pin-yu.chen@ibm.com, miyen@iis.sinica.edu.tw>.

additional inference latency.

Even with its popularity, LoRA has been shown to underperform Full FT on challenging tasks in programming and mathematics (Biderman et al., 2024). This gap has in turn spurred recent efforts toward advanced LoRA variants (Zhu & Nguyen, 2025), with promising performance improvements reported. On Llama (Touvron et al., 2023), for example, Meng et al. (2024a) presented around a 10% accuracy improvement on GSM8K (Cobbe et al., 2021) by modifying LoRA initialization strategies, while Liu et al. (2024a) reported substantial gains of 37.2% on commonsense reasoning tasks by separately learning magnitude and directional updates of pretrained weight matrices.

Yet, the results in a majority of work along this line were obtained with hyperparameters directly inherited from prior studies, or only fine-tuned in a narrow range. To be specific, in Figure 2, we collect 54 LoRA publications from major AI conferences and journals over the past three years, and include 10 high-impact or recently released preprints, to investigate whether their training involved tuning key hyperparameters—namely, learning rate, batch size, and rank. The statistics over a total of 64 prior studies clearly reveal that hyperparameter search is not a standard practice in the field, with only one paper simultaneously considering three hyperparameters and fewer than 30% tuning the learning rate, raising questions about the extent to which the reported gains can be attributed to genuine methodological improvements, particularly given the well-known sensitivity of neural networks to training configurations (LeCun et al., 2002; Bengio, 2012). This is especially critical when it comes to LoRA on LLMs, where careful learning rate tuning is demonstrated to be essential for eliciting strong performance, and optimal settings are contingent on both the base model and the target problem (Biderman et al., 2024; Schulman & Lab, 2025; Yan et al., 2025).

To address the above concern, we select nine representative advanced LoRA variants and conduct a large-scale hyperparameter search, benchmarking them against vanilla LoRA in a head-to-head manner. Under a unified evaluation protocol, we surprisingly find that once the learning rate is properly tuned, all methods peak at similar performance levels, exhibiting no systematic advantages over the vanilla LoRA. For example, in Figure 1, we fine-tune Qwen3 (Yang et al., 2025) using all ten LoRA methods, with the learning rate varied over three orders of magnitude; under a fixed rank of 128 and batch size of 64, all methods achieve accuracies within a narrow 0.83% range. Moreover, different methods operate under disparate learning rate ranges (e.g., a 10× difference between PiSSA (Meng et al., 2024a) and LoRA in Figure 1), suggesting that success under a single training configuration cannot be taken as evidence of robust and reliable improvements. This phenomenon is not

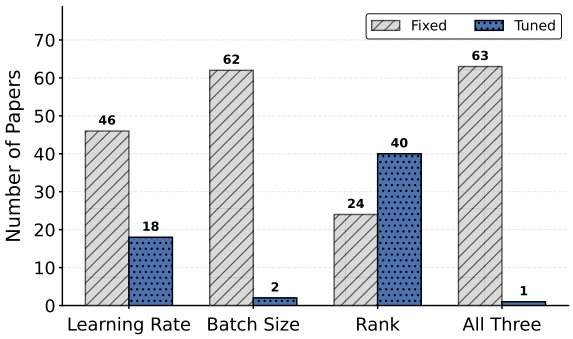


Figure 2. Frequency of advanced LoRA-based PEFT studies, categorized by whether learning rate or batch size tuning was applied and whether comparisons with vanilla LoRA across different ranks were conducted. Crucially, fewer than 30% of the surveyed works tuned the learning rate, and only one covered all three dimensions. Refer to Appendix Sec. B for detailed data counts.

isolated; we consistently observe such performance parity across four task types over diverse LoRA ranks, training durations, and model scales ranging from 0.6B to 13B (e.g., Appendix Figure 6 presents similar results on commonsense reasoning tasks with Gemma-3 (Team et al., 2025)). Notably, within these marginal performance variations, rank-dependent behaviors emerge: some advanced variants may slightly outperform LoRA at higher ranks while lagging behind at lower ones (or vice versa), highlighting the importance of verifying improvements across the entire rank spectrum.

By delving into the fundamentals in learning theories (LeCun et al., 1992; Lewkowycz et al., 2020), we provide an explanation for the importance of fine-tuning learning rates during LoRA finetuning and uncover the reasons behind different desirable learning rate ranges among various LoRA methods. Specifically, we demonstrate that PiSSA (Meng et al., 2024a), OLoRA (Büyükakyüz, 2024), and LoRA-GA (Wang et al., 2024c) exhibit significantly larger maximum Hessian eigenvalues compared to vanilla LoRA, which theoretically justifies their requirement for a lower learning rate. Based on the extensive tuning experiments, we also derive five *practical heuristics* for hyperparameter tuning in LoRA-based methods, particularly regarding how desirable learning-rate ranges interact with batch size (***I–II***) and how such ranges can be inferred from the eigenvalues of the loss Hessian (***III***). In addition, we show how performance improvements can be expected across LoRA ranks (***IV***) and training durations (***V***).

In summary, our work reveals the insufficiency of hyperparameter tuning in many prior LoRA studies and provides a systematic empirical re-evaluation of their best achievable performance. By explaining the observed performance trends and differences in optimal learning rate ranges via Hessian analysis, we hope to encourage future LoRA research to adopt more comprehensive hyperparameter tuning

protocols. Meanwhile, with the provided practical guidelines on LoRA hyperparameters, we aim to help practitioners with limited computational resources avoid unnecessarily exhaustive hyperparameter searches. Concretely, the main contributions of this paper are organized around the following questions:

- ***What is the problem?*** We conduct a comprehensive audit of advanced LoRA PEFT studies and identify a recurring issue: the majority of works lack thorough hyperparameter tuning, despite this being a standard requirement. In fact, according to Figure 2, only 1 out of 64 papers simultaneously considers three hyperparameters, while 46 present results under a fixed learning rate.
- ***Why does it matter?*** Without proper hyperparameter tuning, conclusions may be ungrounded. Concretely, through extensive experimentation, we demonstrate that while different methods require distinct optimal learning rate ranges, they yield comparable peak performance when configured to their optimal settings. For example, on Qwen3-0.6B, averaged over three runs, the top-performing method (GraLoRA) leads the runner-up (LoRA-GA) by only 0.02%, and the least effective method (OLoRA) by 0.83% (cf. Figure 1).
- ***Which learning rate to use?*** By analyzing the eigenvalues of the loss Hessian across various initialization-based LoRA variants, we find that the optimal learning rate is generally negatively correlated with the maximum eigenvalue, aligning with classical learning theories.
- ***How can we tune LoRA methods efficiently?*** Since fair comparison requires method-specific tuning but exhaustive searches are computationally expensive, we identify practical heuristics across learning rate, batch size, LoRA rank, and training duration. Together with the Hessian eigenvalue analysis, these findings help narrow the hyperparameter search space.

2. Related Work

2.1. Systematic Empirical Re-evaluation of Prior Claims

Incomplete performance evaluation remains a persistent concern (Sculley et al., 2018; Lipton & Steinhardt, 2019). For instance, Melis et al. (2017) revealed that two published improvements to the vanilla LSTM (Graves, 2012), originally attributed to complex network designs, were in fact due to more careful hyperparameter tuning. Under a fair tuning protocol, the standard LSTM emerged as the best-performing architecture. In the same vein, Lin et al. (2023) pointed out that simple baselines such as linear SVM (Boser et al., 1992) are competitive with BERT (Devlin et al., 2019)-based methods for text classification, sometimes even outperforming them with a clear gap.

Such systematic empirical re-evaluation has also been done across diverse machine learning subfields. Examples include not only traditional topics such as image classification (Chatfield et al., 2014), graph neural networks (Shchur et al., 2018), generative adversarial networks (Lucic et al., 2018), recommender systems (Ferrari Dacrema et al., 2019), metric learning (Musgrave et al., 2020), and neural network pruning (Blalock et al., 2020), but also more recent areas like optimizers (Schmidt et al., 2021), reinforcement learning (Eimer et al., 2023), preference optimization (Ahrabian et al., 2025), and model merging (Hitit et al., 2025).

While empirical studies benchmarking LoRA with other PEFT methods such as prefix tuning (Li & Liang, 2021) and BitFit (Zaken et al., 2022) exist (He et al., 2021; Hu et al., 2023; Zheng et al., 2024b; Männistö et al., 2025; Belanec et al., 2026), few studies specifically focus on comparing LoRA and its advanced variants. More concerning, training hyperparameters in many of these works were kept fixed without method-specific optimization. Therefore, practitioners are left without clear and reliable guidance when choosing LoRA-based methods.

2.2. LoRA Hyperparameter Tuning

Theories regarding LoRA's lack of Lipschitz smoothness (Sun et al., 2024; Malinovsky et al., 2024) and its spurious loss landscape (Liu et al., 2025) point toward its intrinsic sensitivity to hyperparameter variations. Consequently, many research efforts have been invested in finding optimal training setups, such as learning rate (Hayou et al., 2024b), rank (Zhang et al., 2025a), initializations (Hayou et al., 2024a), scaling factor (Kalajdzievski, 2023), dropout (Lin et al., 2024), and adapter placements (Fomenko et al., 2024; Hayou et al., 2025). Despite these insights into individual hyperparameters, establishing unified configuration guidelines remains an ongoing pursuit. For example, recent works have sought to derive practical "rules of thumb" through extensive, joint evaluations across multiple hyperparameter dimensions (Biderman et al., 2024; Schulman & Lab, 2025). Addressing the computational bottleneck of such extensive searches, Yan et al. (2025) took a system-level approach, optimizing hardware resources to maximize training throughput specifically for LoRA hyperparameter tuning.

Despite these efforts to optimize LoRA hyperparameters, few studies have examined whether LoRA and its variants require distinct hyperparameter settings. Most relevant to our work is Zhang et al. (2025h), which provided a helpful theoretical framework revealing the interplay between LoRA initialization strategies and hyperparameters such as learning rate and scaling factor. Specifically, Zhang et al. (2025h) noted an intriguing phenomenon where LoRA and two of its initialization variants, PiSSA (Meng et al., 2024a) and MiLoRA (Wang et al., 2024b), exhibit performance shifts across two learning rates ($2e^{-4}$ and $2e^{-5}$). In contrast, our

work expands this investigation to a broader set of recent LoRA variants and conducts comprehensive multivariate hyperparameter tuning to identify the best-performing configuration for each method. Moreover, we also provide intuitions of the underlying factors driving the observed performance differences and trends.

3. Background: LoRA PEFT Methods

3.1. Low-Rank Adaptation

Given a pretrained neural network layer parameterized by $W_{\text{pre}} \in \mathbb{R}^{m\times n}$, LoRA introduced two trainable matrices: the down-projecting $A \in \mathbb{R}^{r\times n}$ and the up-projecting $B \in \mathbb{R}^{m\times r}$ ($r \ll \min(m,n)$). For layer input $x \in \mathbb{R}^n$, the output $h \in \mathbb{R}^m$ is computed as:

$$h = W_{\text{pre}}x + \gamma_r BAx, \tag{1}$$

where $\gamma_r = \frac{\alpha}{r}$ serves as a rank-dependent scaling factor with α being a tunable hyperparameter. At initialization, the two trainable matrices B and A are set to $B_0 = 0$ and $A_0 \sim \mathcal{N}(0, \sigma^2)$ (i.e., Kaiming initialization (He et al., 2015)), ensuring that fine-tuning starts exactly from the pretrained checkpoint.

3.2. Representative LoRA Variants

In this paper, we consider nine representative LoRA variants spanning diverse optimization mechanisms, which we organize into three categories: (1) *Initialization Variants* (OLoRA (Büyükakyüz, 2024), PiSSA (Meng et al., 2024a), MiLoRA (Wang et al., 2024b), Init[AB] (Li et al., 2025), LoRA-GA (Wang et al., 2024c)), (2) *Architecture Modifications* (DoRA (Liu et al., 2024a), GraLoRA (Jung et al., 2026), RandLoRA (Albert et al., 2025)), and (3) *Optimization Adjustments* (LoFT (Tastan et al., 2025)). We describe their key design principles below and defer the detailed design rationales and formulas to Appendix Sec. C.

Initialization Variants. This category comprises methods that explore improved initialization strategies for LoRA (Meng et al., 2024a; Büyükakyüz, 2024; Wang et al., 2024c;b; Li et al., 2025; Yang et al., 2024a; Paischer et al., 2024; Zhang et al., 2025g). Methods along this line can be further distinguished by whether their initialization requires task data, yielding *data-free* and *data-informed* subcategories, both of which are considered in this work. Specifically, within the data-free subcategory, OLoRA (Büyükakyüz, 2024) applies QR decomposition to W_{pre} to initialize A and B using first-r columns of Q and first-r rows of R, respectively. PiSSA (Meng et al., 2024a) and MiLoRA (Wang et al., 2024b), on the other hand, leverage the singular value decomposition (SVD) of W_{pre} to inform the initialization of LoRA adapters, with PiSSA selecting the top-r principal components and MiLoRA adopting the minor ones. Several works have also theoretically analyzed the initialization strategies of LoRA (Hayou et al., 2024a; Xu et al., 2025; Li et al., 2025). In particular, Init[AB] (Li et al., 2025) showed that randomly initializing both LoRA matrices using Kaiming initialization can provide better advantage by balancing stability, training efficiency, and hyperparameter robustness. Turning to the *data-informed* subcategory, LoRA-GA (Wang et al., 2024c) uses one-step full-gradient information to initialize LoRA adapters. Let $G = -\nabla_{W_{\text{pre}}}\mathcal{L} \in \mathbb{R}^{m\times n}$ denote the sampled full gradient with respect to W_{pre}. LoRA-GA computes the SVD of G and initializes LoRA adapters in a disjoint manner, using the top-r right singular vectors for A and the $(r+1)$-th through $2r$-th left singular vectors for B.

Note that since $B_0A_0 \neq 0$ for all the initialization variants discussed above, the base weight is replaced by a *residual matrix* so that fine-tuning starts from the pretrained weights. Specifically, the residual matrix is defined as $W_{\text{res}} = W_{\text{pre}} - B_0A_0$, and the modified forward pass becomes:

$$h = W_{\text{res}}x + \gamma_r BAx.$$

Architectural Modifications. Besides investigating initialization strategies, a large body of literature has also focused on architecture-level improvements, e.g., (Jung et al., 2026; Albert et al., 2025; Kopiczko et al., 2023b; Liu et al., 2023b; Zhong et al., 2026) available in the PEFT library (Mangrulkar et al., 2022). By modifying vanilla LoRA's forward design (i.e., Eq. 1), these methods improve fine-tuning effectiveness either by sustaining performance with greater parameter efficiency (Kopiczko et al., 2023b; Bałazy et al., 2024; Li et al., 2024; Gao et al., 2024; Yang et al., 2024b) or by achieving higher accuracy under a similar trainable-parameter budget (Liu et al., 2024a; Jung et al., 2026; Albert et al., 2025; Huang et al., 2025; Jiang et al., 2024). We focus on methods in the latter category, as large differences in trainable-parameter counts relative to LoRA make the direct head-to-head comparisons non-trivial. For example, VeRA substantially reduces the number of trainable parameters from $(m+n)r$ to $m+r$ per layer. In particular, we select DoRA, RandLoRA, and GraLoRA, as they require no more than one additional architectural hyperparameter. This contrasts with other modification strategies such as BoFT (Liu et al., 2023b) and PEANuT (Zhong et al., 2026), which involve multiple architectural choices, namely (m, b) and depth/activation function, respectively, and could therefore rapidly expand the hyperparameter search space. Due to space constraints, we defer the details of their forward designs to Appendix Sec. C.2.

Optimization Adjustments. More recent studies have started to improve LoRA by directly adjusting its optimization dynamics. For example, LoRA+ (Hayou et al., 2024b) assigns different learning rates to A and B, while *scaled*

AdamW (Zhang & Pilanci, 2024) introduces a small preconditioner into each gradient step. LoRA-Pro (Wang et al., 2024e) further adjusts the gradients of LoRA so that the induced update better approximates the full fine-tuning gradient. More recently, LoFT (Tastan et al., 2025) aligns the optimizer's internal dynamics with full fine-tuning by projecting Adam (Kingma, 2014)'s first- and second-moment estimates into the same low-rank subspace, narrowing the performance gap between LoRA and full fine-tuning.

4. Learning Rate Matters, Really

4.1. Motivation

For the trainable LoRA parameters across layers, collectively denoted as θ, the update rule of Stochastic Gradient Descent (SGD) at step t is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}(\boldsymbol{\theta}_t),$$

where η is the learning rate and $\mathbf{g}(\boldsymbol{\theta}_t) \triangleq \nabla \mathcal{L}(\boldsymbol{\theta}_t)$ is the gradient of loss function $\mathcal{L}$. While setting η too large causes the optimization step to overshoot, leading to instability or divergence, a value that is too small is insufficient to escape suboptimal local minima or affect convergence rate. To analyze this formally, consider the local geometry characterized by the Hessian $\mathbf{H}(\boldsymbol{\theta}_t) \triangleq \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t)$. According to classical learning theories (LeCun et al., 1992), the optimal learning rate η^* for efficient learning is intrinsically tied to the curvature of the loss landscape at $\boldsymbol{\theta}$, typically scaling inversely with the Hessian's maximum eigenvalue:

$$\eta^* \propto \frac{1}{\lambda_{\max}(\mathbf{H}(\boldsymbol{\theta}))}. \tag{2}$$

Notably, LoRA initialization variants establish specific training starting points $\boldsymbol{\theta}_0$, resulting in distinct $\mathbf{g}(\boldsymbol{\theta}_0)$, $\mathbf{H}(\boldsymbol{\theta}_0)$, and subsequent training trajectories compared to vanilla LoRA. Similarly, while LoRA variants based on architectural modifications or gradient adjustments could share the same $\mathbf{g}(\boldsymbol{\theta}_0)$ and $\mathbf{H}(\boldsymbol{\theta}_0)$ as vanilla LoRA, their subsequent gradient and Hessian evolution throughout training may naturally deviate from LoRA due to the unique forward designs or update rules. Therefore, different methods theoretically require their respective calibrations of η to ensure efficient convergence, motivating our decision to perform learning rate tuning for a fair and reliable head-to-head comparison across methods.

4.2. Experimental Setup

Since model choices, training configurations, and dataset partitioning vary across papers, we establish a unified experimental framework that accommodates all methods fairly. We describe the key components below, with additional implementation details deferred to Appendix D.

Pretrained Models. We consider four decoder-only models spanning diverse scales: Qwen3-0.6B (Yang et al., 2025), Gemma-3-1B (Team et al., 2025), and Llama-2-7B and 13B (Touvron et al., 2023). This selection includes both recently released ones (Qwen3, Gemma-3) and an older but widely used model family (Llama-2) in prior art on this subject, enabling us to validate results across LLMs with diverse pretrained capabilities.

Fine-tuning Tasks. We train models on four canonical tasks: commonsense reasoning, mathematical reasoning, code generation, and instruction following. The dataset setup follows prior LoRA studies. Specifically, for commonsense reasoning, we leverage the 15k training examples compiled by Hu et al. (2023), which comprise eight general questio n-answering subtasks. For mathematical reasoning, we use 100k subsampled training examples from MetaMathQA (Yu et al., 2023) and evaluate models on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). For code generation, we use 104k subsampled training examples from CodeFeedback (Zheng et al., 2024a) and evaluate models on HumanEval (Chen, 2021) and MBPP (Austin et al., 2021). For instruction following, we train models on 52k Alpaca (Taori et al., 2023) examples and evaluate them using the IFEval framework (Zhou et al., 2023). Unless otherwise specified, we report mean accuracy over the testing datasets.

Hyperparameter Settings. We consider batch sizes (B) in {16, 32, 64, 128, 256, 512} and ranks (r) in {4, 8, 16, 32, 64, 128, 256}. The learning rates (η) are swept uniformly on a logarithmic scale from 10^{-3} to 10^{-6}, with four values per order of magnitude: $1.1247\times10^*$, $2.0000\times10^*$, $3.5566\times10^*$, $6.3246\times10^*$, yielding up to 16 grid points for the learning rate alone. To maintain the computational feasibility of this study, batch size and rank are tuned only for selected model-task combinations; conversely, learning rates are tuned across all combinations, with ranges defined to ensure inclusion of optimal performance. See Appendix Table 4 for a summary of models, tasks, and their corresponding hyperparameter search ranges. Other configurations, such as epoch, adapter placement, and learning rate scheduler, remain fixed across all experiments and are listed in Appendix Table 5. Specifically for the scaling factor γ_r, we follow Meng et al. (2024a) by setting α equal to r in all our experiments. This results in $\gamma_r = 1$ for all r, effectively factoring out the need to tune this hyperparameter (refer to Appendix Sec. E for further discussion).

4.3. Results and Observations

We begin by discussing results under fixed rank $r = 128$ in Sec. 4.3.1, and then in Sec. 4.3.2, we analyze how the methods perform across different ranks.

Methods	Batch Size	Learning Rate											
		1.1e-5	2e-5	3.6e-5	6.3e-5	1.1e-4	2e-4	3.6e-4	6.3e-4	1.1e-3	2e-3	3.6e-3	6.3e-3
LoRA	16	$9.78_{\pm0.36}$	$11.16_{\pm0.28}$	$13.58_{\pm0.18}$	$15.48_{\pm0.15}$	$18.43_{\pm0.14}$	$\mathbf{20.00}_{\pm0.26}$	$19.93_{\pm0.65}$	$17.99_{\pm0.55}$	$11.71_{\pm0.49}$	$1.52_{\pm0.19}$	$1.27_{\pm0.59}$	$1.07_{\pm0.27}$
	64	$6.88_{\pm0.04}$	$9.12_{\pm0.39}$	$10.79_{\pm0.37}$	$13.23_{\pm0.25}$	$15.65_{\pm0.57}$	$17.54_{\pm0.29}$	$19.73_{\pm0.16}$	$\mathbf{20.46}_{\pm0.79}$	$19.83_{\pm0.91}$	$13.33_{\pm0.81}$	$1.48_{\pm0.48}$	$0.00_{\pm0.00}$
	128	$5.70_{\pm0.34}$	$6.95_{\pm0.23}$	$9.41_{\pm0.44}$	$11.43_{\pm0.40}$	$13.68_{\pm0.77}$	$15.92_{\pm0.45}$	$18.58_{\pm0.44}$	$19.60_{\pm0.09}$	$\mathbf{20.32}_{\pm0.28}$	$16.95_{\pm2.70}$	$0.09_{\pm0.16}$	$0.00_{\pm0.00}$
DoRA	16	$9.89_{\pm0.24}$	$11.16_{\pm0.51}$	$13.84_{\pm0.41}$	$15.61_{\pm0.11}$	$18.21_{\pm0.45}$	$20.11_{\pm0.26}$	$\mathbf{20.96}_{\pm0.57}$	$18.34_{\pm0.20}$	$11.90_{\pm0.29}$	$4.89_{\pm0.99}$	$0.93_{\pm0.12}$	$1.16_{\pm0.15}$
	64	$6.72_{\pm0.09}$	$9.19_{\pm0.19}$	$10.53_{\pm0.20}$	$13.45_{\pm0.31}$	$15.72_{\pm0.32}$	$17.66_{\pm0.20}$	$19.96_{\pm0.05}$	$\mathbf{20.82}_{\pm0.32}$	$19.87_{\pm0.91}$	$13.53_{\pm1.64}$	$1.52_{\pm0.45}$	$0.34_{\pm0.23}$
	128	$5.55_{\pm0.11}$	$7.21_{\pm0.18}$	$9.72_{\pm0.17}$	$11.58_{\pm0.25}$	$13.98_{\pm0.33}$	$16.19_{\pm0.46}$	$18.25_{\pm0.23}$	$19.67_{\pm0.71}$	$\mathbf{20.33}_{\pm0.64}$	$12.86_{\pm10.03}$	$0.13_{\pm0.23}$	$0.02_{\pm0.03}$
Init[AB]	16	$9.73_{\pm0.35}$	$12.10_{\pm0.14}$	$14.41_{\pm0.49}$	$16.73_{\pm0.37}$	$18.38_{\pm0.53}$	$20.39_{\pm0.38}$	$\mathbf{20.55}_{\pm0.40}$	$18.34_{\pm0.48}$	$11.94_{\pm0.31}$	$1.48_{\pm0.24}$	$1.16_{\pm0.31}$	$1.45_{\pm0.17}$
	64	$6.51_{\pm0.22}$	$9.15_{\pm0.12}$	$11.28_{\pm0.20}$	$13.20_{\pm0.24}$	$15.88_{\pm0.39}$	$17.89_{\pm0.30}$	$20.08_{\pm0.26}$	$\mathbf{20.98}_{\pm0.33}$	$19.31_{\pm0.75}$	$13.97_{\pm0.03}$	$2.74_{\pm3.83}$	$0.07_{\pm0.12}$
	128	$6.06_{\pm0.35}$	$7.05_{\pm0.33}$	$9.53_{\pm0.22}$	$11.81_{\pm0.08}$	$13.98_{\pm0.79}$	$16.46_{\pm0.39}$	$18.36_{\pm0.21}$	$20.37_{\pm0.39}$	$\mathbf{20.66}_{\pm0.39}$	$17.85_{\pm0.84}$	$4.40_{\pm7.46}$	$0.00_{\pm0.00}$
MiLoRA	16	$12.44_{\pm0.07}$	$13.77_{\pm0.25}$	$16.28_{\pm0.24}$	$18.45_{\pm0.47}$	$20.04_{\pm0.19}$	$\mathbf{20.63}_{\pm0.67}$	$19.40_{\pm0.80}$	$15.72_{\pm0.49}$	$10.22_{\pm0.42}$	$2.03_{\pm0.95}$	$1.35_{\pm0.43}$	$1.56_{\pm0.65}$
	64	$8.82_{\pm0.40}$	$11.25_{\pm0.20}$	$13.16_{\pm0.11}$	$15.54_{\pm0.29}$	$17.43_{\pm0.24}$	$19.56_{\pm0.33}$	$\mathbf{20.03}_{\pm0.59}$	$19.60_{\pm0.78}$	$17.93_{\pm0.90}$	$13.65_{\pm0.07}$	$4.97_{\pm0.40}$	$0.00_{\pm0.00}$
	128	$7.32_{\pm0.33}$	$9.57_{\pm0.24}$	$11.76_{\pm0.33}$	$13.54_{\pm0.12}$	$16.02_{\pm0.16}$	$18.39_{\pm0.26}$	$19.70_{\pm0.34}$	$\mathbf{19.99}_{\pm0.66}$	$19.53_{\pm0.47}$	$16.83_{\pm0.73}$	$7.45_{\pm1.00}$	$0.57_{\pm0.81}$
PiSSA	16	$14.30_{\pm0.18}$	$16.10_{\pm0.27}$	$18.31_{\pm0.12}$	$19.90_{\pm0.21}$	$\mathbf{20.61}_{\pm0.28}$	$19.09_{\pm0.20}$	$16.10_{\pm0.64}$	$13.25_{\pm0.55}$	$8.41_{\pm0.13}$	$4.67_{\pm0.29}$	$2.50_{\pm1.27}$	$0.96_{\pm0.15}$
	64	$11.11_{\pm0.05}$	$13.67_{\pm0.17}$	$15.56_{\pm0.33}$	$18.11_{\pm0.23}$	$19.52_{\pm0.48}$	$\mathbf{20.68}_{\pm0.77}$	$20.59_{\pm0.32}$	$19.11_{\pm0.86}$	$15.53_{\pm0.37}$	$9.57_{\pm0.72}$	$5.78_{\pm0.37}$	$0.33_{\pm0.46}$
	128	$9.42_{\pm0.38}$	$11.80_{\pm0.28}$	$14.40_{\pm0.11}$	$16.23_{\pm0.38}$	$18.60_{\pm0.21}$	$19.61_{\pm0.44}$	$\mathbf{20.65}_{\pm0.44}$	$19.21_{\pm1.15}$	$16.91_{\pm0.19}$	$13.87_{\pm0.97}$	$6.28_{\pm0.49}$	$1.19_{\pm0.36}$

Table 1. Performance of Gemma-3-1B on mathematical reasoning task across varying batch sizes and learning rates ($r = 128$). Results are reported as mean $\pm$ standard deviations over three independent runs. Best results are highlighted in **bold**, and configurations achieving $\geq 18.5\%$ accuracy (i.e., $\approx 90\%$ of the maximum) are shaded in green ( ). While all methods achieve comparable peak accuracies, the optimal learning rates vary depending on both the fine-tuning method and batch size.

4.3.1. Similar Performance Levels

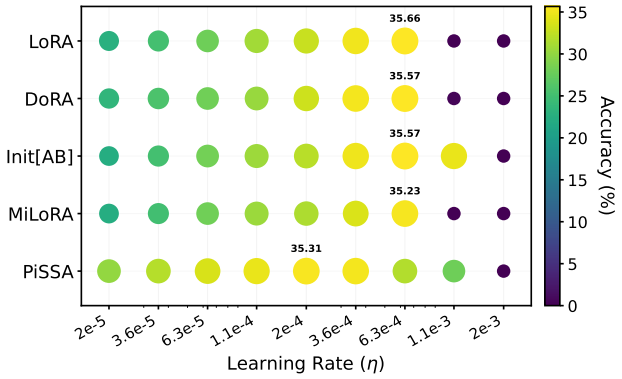
To further examine whether the performance parity among LoRA methods observed on Qwen3-0.6B (Figure 1 and Appendix Figure 6) generalizes to diverse model–task combinations, Table 1, Figure 3, and Appendix Figure 7 present the results for Gemma-3-1B, Llama-2-7B, and Llama-2-13B, respectively. To maintain the computational feasibility of this study, we select four popular and recently published LoRA variants from the nine methods considered previously, namely PiSSA, MiLoRA, Init[AB], and DoRA. Through comprehensive hyperparameter searches, we consistently observe across different model scales and tasks that these methods peak at performance levels similar to vanilla LoRA. Specifically, the performance gaps across all methods remain small: 0.52% for Gemma-3-1B on math, 0.43% and 1.75% for Llama-2-7B on math and code, respectively, and 1.81% for Llama-2-13B on math. It is also important to note that while peak performance is similar, the optimal learning rates vary. In particular, PiSSA requires a lower learning rate compared to vanilla LoRA across all model-task combinations, while other methods fall within a similar range to LoRA, typically within the same order of magnitude. Beyond the optimal learning rates, a closer inspection of the full learning rate spectrum also reveals intriguing method-specific behaviors. For instance, we observe that PiSSA tends to remain effective at larger learning rates where other methods diverge (cf. Figure 3).

Note that in Table 1, the joint optimization of learning rate and batch size indicates that tuning the learning rate is significantly more critical than tuning the batch size for obtaining best performance in both LoRA and its variants, consistent with early findings for neural networks (Bengio, 2012). For example, with PiSSA, fixing the learning rate at 2×10^{-5} and tuning only the batch size yields a suboptimal maximum accuracy of 16.1%. In contrast, by fixing the batch size to any value in $\{16, 64, 128\}$ and tuning the learning rate, the model achieves substantially higher performance around 20.6%. Moreover, we observe that the optimal learning rate scales proportionally with batch size, aligning with the "scaling rule" established in SGD literature (Goyal et al., 2017; Hoffer et al., 2017). This offers several *practical heuristics* for LoRA hyperparameter tuning: ***I.*** Under limited computational resources, one may consider prioritizing learning rate tuning while fixing the batch size to a small or medium value.[1] ***II.*** If additional resources are available and practitioners wish to explore different batch sizes, even though further performance gains are likely to be marginal once the learning rate has been properly tuned, we remind practitioners to keep in mind the scaling relationship between batch size and learning rate, which can help guide initial learning rate selection when configuring different batch sizes. More numerical results, example model responses, and practical learning heuristics (***III–V***) are provided in Appendix F, Sec. G, and Sec. H, respectively.

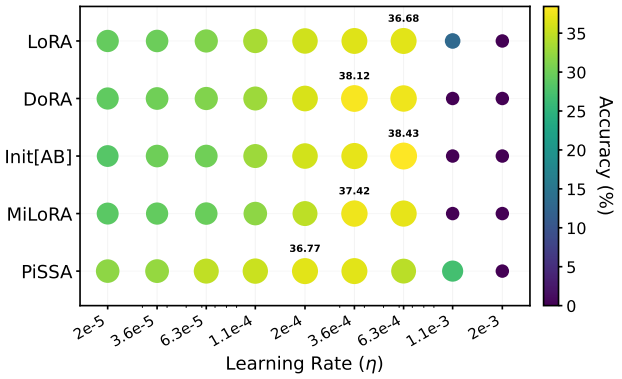
4.3.2. Performance Comparison Across Ranks

Next, we extend our analysis by varying adapter ranks for Gemma-3-1B, as shown in Figure 4. The results indicate

[1]When the batch size is set too large, the best achievable performance of LoRA methods under comprehensive learning rate tuning starts to decay, as we show in Appendix A.4.

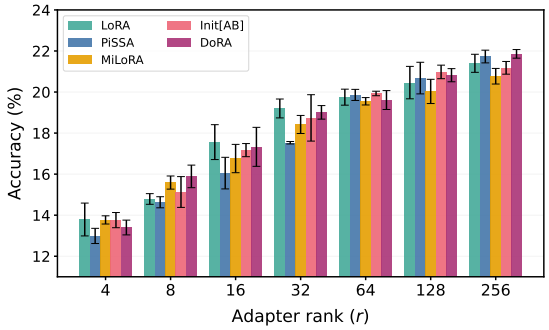


(a) Mathematical Reasoning

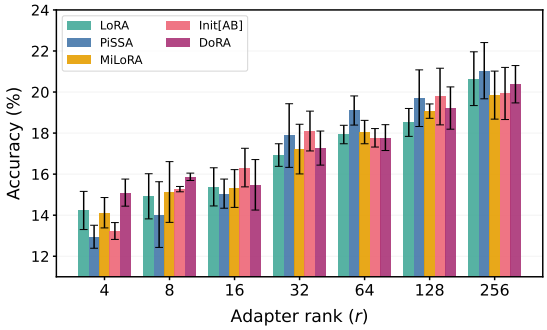


(b) Code Generation

Figure 3. Performance of Llama-2-7B on mathematical reasoning and code generation tasks across varying learning rates ($r = 128$, $B = 128$). Notably, PiSSA peaks at lower learning rates but remains effective at larger learning rates on both tasks (e.g., 1.1×10^{-3}), where other methods diverge. Results scaling up to Llama-2-13B are provided in Appendix Figure 7.



(a) Mathematical Reasoning



(b) Code Generation

Figure 4. Best achievable performance of LoRA and its advanced variants across adapter ranks on Gemma-3-1B ($B = 64$). With properly tuned learning rates, all methods exhibit similar performance improvement trends as the rank increases, though subtle rank-dependent behaviors emerge. Results are reported with means and standard deviations over three independent runs.

that the previously observed performance parity persists across a wide range of rank settings, with the maximum performance differences among methods being only 1.67% (Math, $r = 32$) and 2.15% (Code, $r = 4$). Interestingly, however, we observe the relative performance of variants compared to LoRA fluctuates across different ranks within these margins.

In particular, PiSSA initially underperforms vanilla LoRA before gradually overtaking it as the rank increases. Taking the math task as an example (Figure 4a), PiSSA exhibits performance deficits of up to 1.67% at low ranks ($r \leq 32$), but narrows the gap to within 0.11% at $r = 64$ and shifts to a slight gain of 0.22% and 0.33% at $r = 128$ and 256, respectively. In contrast, MiLoRA shows an opposite trend, where it tends to outperform vanilla LoRA at lower ranks but fails to sustain this advantage as the rank increases. Figure 4b indicates that this rank-dependent dynamics extend to the coding task. For Init[AB], we observe that it tends to outperform LoRA at medium ranks, e.g., achieving maximal gains of 0.52% on math and 1.26% on code at $r = 128$. Yet, the success does not translate to either lower or higher rank scenarios, where Init[AB] typically performs similarly to vanilla LoRA. As for DoRA, we observe performance gains against LoRA specifically in low-rank regimes, peaking at 1.1% on math and 0.95% on code at $r = 8$.

Similar performance comparisons across ranks were also conducted on Llama-2-7B, with results deferred to Appendix Sec. A.2. Beyond adapter ranks, we also validate our findings under different numbers of training samples and training epochs, with results deferred to Appendix Sec. A.3.

5. Understanding the Optimal Learning Rate via Hessian Analysis

5.1. Sharpness-Learning Rate Relationship

The Hessian of the loss function has been the subject of numerous studies. Geometrically, its top eigenvalue (denoted as $\lambda_{\max}$ for brevity) at a given point represents the maximal curvature of the loss landscape along any direction, com-

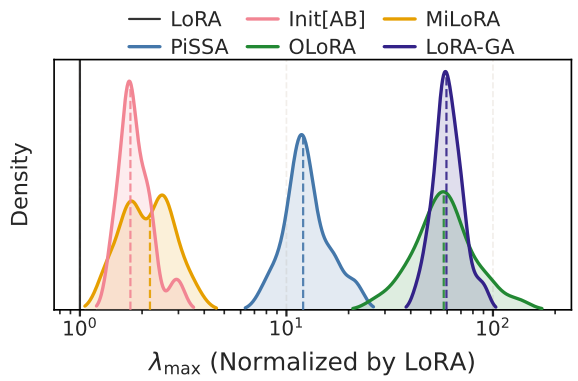


Figure 5. Distributions of the ratios of the top MetaMath loss Hessian eigenvalues relative to LoRA for query projection matrices across Transformer layers on Qwen3 ($r = 128$). Dashed lines indicate the medians. With an adapter rank of $r = 128$, PiSSA, OLoRA, and LoRA-GA exhibit significantly higher maximum Hessian eigenvalues. In contrast, MiLoRA and Init[AB] exhibit eigenvalues slightly larger than those of LoRA. Refer to Appendix Sec. I for more details on the Hessian analysis.

monly referred to as sharpness (Dinh et al., 2017; Lyu et al., 2022; Luo et al., 2024). This metric is closely linked to the optimal learning rate, a connection that originates from the Gauss-Newton method for convex optimization and further elucidated by LeCun et al. (1992) in the context of neural networks. Specifically, it was shown that an efficient learning rate theoretically falls within $1/\lambda_{\max} \leq \eta^* < 2/\lambda_{\max}$ under quadratic approximation, whereas rates exceeding $2/\lambda_{\max}$ lead to divergence. More recently, Lewkowycz et al. (2020) identified a "catapult" learning regime characterized by $2/\lambda_{\max} \leq \eta^* \leq 12/\lambda_{\max}$, in which modern architectures achieve optimal performance. More research has further explored the intricacies of the interplay between $\lambda_{\max}$ and η^* with a consensus that these two quantities exhibit an inversely proportional relationship (Pan et al., 2021; Cohen et al., 2021; Kalra & Barkeshli, 2024).

5.2. Sharpness Analysis in LoRA

For our LoRA fine-tuning problem, we leverage the downstream MetaMathQA dataset to compute the Hessian matrix of the loss function and focus exclusively on the trainable LoRA parameters (Yang et al., 2023; Zhao et al., 2024b; Yu et al., 2025b). Instead of concatenating LoRA parameters across all layers, we follow standard LLM practices to estimate $\lambda_{\max}$ in a block-wise manner (Zhang et al., 2024b; Wang et al., 2025a; Ilin, 2025) at the initialization point. Formally, we calculate the layer-wise metric as $\lambda^l_{\max} = \lambda_{\max}(\mathbf{H}^l)$, where $\mathbf{H}^l$ represents the Hessian corresponding to parameters $\theta^l = \{B^l_0, A^l_0\}$, with l indexing matrix types and Transformer layers. The Lanczos algorithm (Lanczos, 1950) and Hessian-vector products are used to estimate the top eigenvalue without explicitly forming $\mathbf{H}$. Implementation details are provided in Appendix I.1. While LoRA architectural modifications and gradient adjustments may share LoRA's initialization, implying identical initial Hessians, their unique forward designs and update rules may lead to distinct Hessian evolution throughout training, we thus defer their investigation to future work.

Specifically, let the Hessian for the Query projection matrix in the i-th layer be $\mathbf{H}^{Q,i}_t$ with t denotes varying LoRA methods. We further denote their corresponding maximum eigenvalues by $\lambda^{Q,i}_{\max,t}$. Then, we normalize the maximum eigenvalues from LoRA initialization variants by that from LoRA, and plot the distribution across layers in Figure 5, i.e. $\lambda^{Q,i}_{\max,t}/\lambda^{Q,i}_{\max,\text{LoRA}}$ for t =Init[AB], MiLoRA, PiSSA, OLoRA, LoRA-GA and $i = 1, \ldots, L$. The results reveal that all methods initialize trainable parameters in a higher curvature state than vanilla LoRA. Most notably, OLoRA and LoRA-GA exhibit up to $100\times$ higher curvature, explaining the reason behind their requirement for a much lower learning rate ($18.2\times$ lower) in Figure 1. Similar patterns apply to other methods. In particular, PiSSA exhibits $\approx 10\times$ higher curvature, which is consistent with its requirement for a $10\times$ lower learning rate. For Init[AB] and MiLoRA, however, the eigenvalue magnitudes are more similar to those of vanilla LoRA ($\approx 2\times$ higher), supporting their lower optimal learning rates by factors of $1.8\times$ and $3.2\times$ in Figure 1, respectively. Detailed $\lambda_{\max}$ values and Hessian analyses on other models and matrix types are provided in Appendix Sec. I.2 and Sec. I.3.

6. Conclusion

Motivated by the increasing number of LoRA variants and the insufficient hyperparameter tuning in many studies, in this work, we conducted a systematic re-evaluation of nine LoRA PEFT methods under a unified evaluation protocol. Based on the comprehensive hyperparameter experiments, we conclude that **improper learning rate gives a false sense of LoRA advancements**. In our studies, we also pointed out the scenarios where one might be in favor of a specific variant that, albeit likely to produce generally comparable performance, marginally outperforms other variants. It is worth noting that these improvements often lack universality, with vanilla LoRA frequently matching or even outperforming them. By elucidating the disparate optimal learning rate ranges through Hessian analysis, we hope our study encourages future PEFT research to adopt a more comprehensive hyperparameter search protocol, ensuring reliable advancements in the field. We also hope that the five practical heuristics derived from our experimentation will help practitioners reduce the computational burden associated with LoRA tuning. We acknowledge that this paper is subject to several limitations, primarily due to computational constraints. Due to page limits, we defer the detailed discussion of these limitations to Appendix Sec. J.

Impact Statement

The objective of this paper is to advance Machine Learning research. Although there may be potential societal consequences of our work, we do not identify any specific ones that must be specifically highlighted here.

References

Ahrabian, K., Lin, X., Patra, B., Chaudhary, V., Benhaim, A., Pujara, J., and Song, X. A practical analysis of human alignment with* po. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 8013–8021, 2025.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Albert, P., Zhang, F. Z., Saratchandran, H., Rodriguez-Opazo, C., Hengel, A. v. d., and Abbasnejad, E. Randlora: Full-rank parameter-efficient fine-tuning of large models. *arXiv preprint arXiv:2502.00987*, 2025.

Anisuzzaman, D., Malins, J. G., Friedman, P. A., and Attia, Z. I. Fine-tuning large language models for specialized use cases. *Mayo Clinic Proceedings: Digital Health*, 3 (1):100184, 2025.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Azizi, S., Kundu, S., and Pedram, M. Lamda: Large model fine-tuning via spectrally decomposed low-dimensional adaptation. *arXiv preprint arXiv:2406.12832*, 2024.

Bałazy, K., Banaei, M., Aberer, K., and Tabor, J. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Belanec, R., Pecher, B., Srba, I., and Bielikova, M. Peft-bench: A parameter-efficient fine-tuning methods benchmark. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3035–3054, 2026.

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pp. 437–478. Springer, 2012.

Biderman, D., Portes, J., Ortiz, J. J. G., Paul, M., Greengard, P., Jennings, C., King, D., Havens, S., Chiley, V., Frankle, J., et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Bini, M., Girrbach, L., and Akata, Z. Decoupling angles and strength in low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

Büyükakyüz, K. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.

Cahill, E., Irving, A., Johnston, C., Sexton, J., Collaboration, U., et al. Numerical stability of lanczos methods. *Nuclear Physics B-Proceedings Supplements*, 83:825–827, 2000.

Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

Chavan, A., Liu, Z., Gupta, D., Xing, E., and Shen, Z. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*, 2023.

Chen, A., Cheng, J., Liu, Z., Gao, Z., Tsung, F., Li, Y., and Li, J. Parameter-efficient fine-tuning via circular convolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2004–2019, 2025.

Chen, M. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Cohen, J. M., Kaur, S., Li, Y., Kolter, J. Z., and Talwalkar, A. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065*, 2021.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Ding, C., Li, J., Dong, S., Gao, X., He, Y., and Gong, Y. Sulora: Subspace low-rank adaptation for parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5334–5349, 2025.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pp. 1019–1028. PMLR, 2017.

Djagba, P. and Saley, A. Y. Exploring large language models for financial applications: Techniques, performance, and challenges with finma. *arXiv preprint arXiv:2510.05151*, 2025.

Dong, H., Zhu, W., Song, G., and Wang, L. Aurora: Breaking low-rank bottleneck of lora with nonlinear mapping. *arXiv preprint arXiv:2505.18738*, 2025.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Edalati, A., Tahaei, M., Kobyzev, I., Nia, V. P., Clark, J. J., and Rezagholizadeh, M. Krona: Parameter-efficient tuning with kronecker adapter. In *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pp. 49–65. Springer, 2025.

Eimer, T., Lindauer, M., and Raileanu, R. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pp. 9104–9149. PMLR, 2023.

Fan, C., Lu, Z., Liu, S., Gu, C., Qu, X., Wei, W., and Cheng, Y. Make lora great again: Boosting lora with adaptive singular values and mixture-of-experts optimization alignment. *arXiv preprint arXiv:2502.16894*, 2025.

Ferrari Dacrema, M., Cremonesi, P., and Jannach, D. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 101–109, 2019.

Fomenko, V., Yu, H., Lee, J., Hsieh, S., and Chen, W. A note on lora. *arXiv preprint arXiv:2404.05086*, 2024.

Gangwar, N., Deshmukh, R., Shavlovsky, M., Li, H., Mittal, V., Ying, L., and Kani, N. Giva: Gradient-informed bases for vector-based adaptation. *arXiv preprint arXiv:2604.21901*, 2026.

Gao, Z., Wang, Q., Chen, A., Liu, Z., Wu, B., Chen, L., and Li, J. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

Golub, G. H., Underwood, R. R., and Wilkinson, J. H. The lanczos algorithm for the symmetric ax= λ bx problem., 1972.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Graves, A. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Hao, Y., Cao, Y., and Mou, L. Flora: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*, 2024.

haonan he, Ye, P., Ren, Y., yuan yuan, LuyangZhou, ShucunJu, and lei chen. GoRA: Gradient-driven adaptive low rank adaptation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=d1dL1ymD6N.

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

Hayou, S., Ghosh, N., and Yu, B. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024b.

Hayou, S., Ghosh, N., and Yu, B. Plop: Precise lora placement for efficient finetuning of large models. *arXiv preprint arXiv:2506.20629*, 2025.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. 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.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Hitit, O. K., Girrbach, L., and Akata, Z. A systematic study of model merging techniques in large language models. *arXiv preprint arXiv:2511.21437*, 2025.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. 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.

Huang, Q., Ko, T., Zhuang, Z., Tang, L., and Zhang, Y. Hira: Parameter-efficient hadamard high-rank adaptation for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ilin, I. Hessian of perplexity for large language models by pytorch autograd (open source). *arXiv preprint arXiv:2504.04520*, 2025.

Jiang, T., Huang, S., Luo, S., Zhang, Z., Huang, H., Wei, F., Deng, W., Sun, F., Zhang, Q., Wang, D., et al. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024.

Jung, Y., Ahn, D., Kim, H., Kim, T., and Park, E. GraloRA: Granular low-rank adaptation for parameter-efficient fine-tuning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=8wvOMQ2Olw.

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

Kalra, D. S. and Barkeshli, M. Why warmup the learning rate? underlying mechanisms and improvements. *Advances in Neural Information Processing Systems*, 37: 111760–111801, 2024.

Kang, J. and Yin, Q. Miss: Revisiting the trade-off in lora with an efficient shard-sharing structure, 2025. URL https://arxiv.org/abs/2409.15371.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023a.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023b.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of research of the National Bureau of Standards*, 45(4):255–282, 1950.

LeCun, Y., Simard, P., and Pearlmutter, B. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. *Advances in neural information processing systems*, 5, 1992.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–50. Springer, 2002.

Lee, D., Park, J., Kim, M., and Kwon, J. Abm-lora: Activation boundary matching for fast convergence in low-rank adaptation. *arXiv preprint arXiv:2511.19145*, 2025.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Lewkowycz, A., Bahri, Y., Dyer, E., Sohl-Dickstein, J., and Gur-Ari, G. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

Li, S., Luo, X., Tang, X., Wang, H., Chen, H., Luo, W., Li, Y., He, X., and Li, R. Beyond zero initialization: Investigating the impact of non-zero initialization on lora fine-tuning dynamics. *arXiv preprint arXiv:2505.23194*, 2025.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Li, Y., Han, S., and Ji, S. Vb-lora: Extreme parameter efficient fine-tuning with vector banks. *Advances in Neural Information Processing Systems*, 37:16724–16751, 2024.

Lialin, V., Shivagunde, N., Muckatira, S., and Rumshisky, A. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Liao, B. and Monz, C. 3-in-1: 2d rotary adaptation for efficient finetuning, efficient batching and composability. *Advances in Neural Information Processing Systems*, 37: 35018–35048, 2024.

Lin, Y., Ma, X., Chu, X., Jin, Y., Yang, Z., Wang, Y., and Mei, H. Lora dropout as a sparsity regularizer for overfitting control. *arXiv preprint arXiv:2404.09610*, 2024.

Lin, Y.-C., Chen, S.-A., Liu, J.-J., and Lin, C.-J. Linear classifier: An often-forgotten baseline for text classification. *arXiv preprint arXiv:2306.07111*, 2023.

Lipton, Z. C. and Steinhardt, J. Troubling trends in machine learning scholarship: Some ml papers suffer from flaws that could mislead the public and stymie future research. *Queue*, 17(1):45–77, 2019.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=1qvx610Cu7.

Liu, M., Si, C., and Jia, Y. Flexlora: Entropy-guided flexible low-rank adaptation. *arXiv preprint arXiv:2601.22905*, 2026.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024a.

Liu, W., Qiu, Z., Feng, Y., Xiu, Y., Xue, Y., Yu, L., Feng, H., Liu, Z., Heo, J., Peng, S., et al. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*, 2023b.

Liu, X.-H., Du, Y., Wang, J., and Yu, Y. On the optimization landscape of low rank adaptation methods for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Liu, Z., Lyn, J., Zhu, W., Tian, X., and Graham, Y. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*, 2024b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.

Luo, H., Truong, T., Pham, T., Harandi, M., Phung, D., and Le, T. Explicit eigenvalue regularization improves sharpness-aware minimization. *Advances in Neural Information Processing Systems*, 37:4424–4453, 2024.

Lyu, K., Li, Z., and Arora, S. Understanding the generalization benefit of normalization layers: Sharpness reduction. *Advances in Neural Information Processing Systems*, 35: 34689–34708, 2022.

Malinovsky, G., Michieli, U., Hammoud, H. A. A. K., Ceritli, T., Elesedy, H., Ozay, M., and Richtárik, P. Randomized asymmetric chain of lora: The first meaningful theoretical framework for low-rank adaptation. *arXiv preprint arXiv:2410.08305*, 2024.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., Bossan, B., and Tietz, M. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

Männistö, J., Attieh, J., and Tiedemann, J. A comparative study of peft methods for python code generation. In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pp. 390–396, 2025.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024a.

Meng, X., Dai, D., Luo, W., Yang, Z., Wu, S., Wang, X., Wang, P., Dong, Q., Chen, L., and Sui, Z. Periodiclora: Breaking the low-rank bottleneck in lora optimization. *arXiv preprint arXiv:2402.16141*, 2024b.

Musgrave, K., Belongie, S., and Lim, S.-N. A metric learning reality check. In *European Conference on Computer Vision*, pp. 681–699. Springer, 2020.

Nikdan, M., Tabesh, S., Crnčević, E., and Alistarh, D. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*, 2024.

Paige, C. C. Practical use of the symmetric lanczos process with re-orthogonalization. *BIT Numerical Mathematics*, 10(2):183–195, 1970.

Paischer, F., Hauzenberger, L., Schmied, T., Alkin, B., Deisenroth, M. P., and Hochreiter, S. Parameter efficient fine-tuning via explained variance adaptation. *arXiv preprint arXiv:2410.07170*, 2024.

Pan, R., Ye, H., and Zhang, T. Eigencurve: Optimal learning rate schedule for sgd on quadratic objectives with skewed hessian spectrums. *arXiv preprint arXiv:2110.14109*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Qin, P., Zhang, R., and Xie, P. BidoRA: Bi-level optimization-based weight-decomposed low-rank adaptation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=v2xCm3VYl4.

Quercia, A., Bangun, A., Assent, I., and Scharr, H. Least but not last: Fine-tuning intermediate principal components for better performance-forgetting trade-offs. *arXiv preprint arXiv:2602.03493*, 2026.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

Renduchintala, A., Konuk, T., and Kuchaiev, O. Tied-lora: Enhancing parameter efficiency of lora with weight tying. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8694–8705, 2024.

Schmidt, R. M., Schneider, F., and Hennig, P. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, pp. 9367–9376. PMLR, 2021.

Schulman, J. and Lab, T. M. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

Sculley, D., Snoek, J., Wiltschko, A., and Rahimi, A. Winner's curse? on pace, progress, and empirical rigor. 2018.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Shi, S., Huang, S., Song, M., Li, Z., Zhang, Z., Huang, H., Wei, F., Deng, W., Sun, F., and Zhang, Q. Reslora: Identity residual mapping in low-rank adaption. *arXiv preprint arXiv:2402.18039*, 2024.

Si, C., Shi, Z., Zhang, S., Yang, X., Pfister, H., and Shen, W. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations*, 2024.

Sun, Y., Li, Z., Li, Y., and Ding, B. Improving lora in privacy-preserving federated learning. *arXiv preprint arXiv:2403.12313*, 2024.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Tastan, N., Laskaridis, S., Takác, M., Nandakumar, K., and Horváth, S. Loft: Low-rank adaptation that behaves like full fine-tuning. *arXiv preprint arXiv:2505.21289*, 2025.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

Wang, F., Jiang, J., Park, C., Kim, S., and Tang, J. Kasa: Knowledge-aware singular-value adaptation of large language models. *arXiv preprint arXiv:2412.06071*, 2024a.

Wang, H., Li, Y., Wang, S., Chen, G., and Chen, Y. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. *arXiv preprint arXiv:2406.09044*, 2024b.

Wang, J., Wang, M., Zhou, Z., Yan, J., Wu, L., et al. The sharpness disparity principle in transformers for accelerating language model pre-training. *arXiv preprint arXiv:2502.19002*, 2025a.

Wang, Q. and Shen, S. Activation-guided low-rank parameter adaptation for efficient model fine-tuning. *IEEE Access*, 2025.

Wang, S. and Kanwar, P. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 4(1), 2019.

Wang, S., Yu, L., and Li, J. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024c.

Wang, S., Yu, L., and Li, J. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024d.

Wang, X., Chen, T., Ge, Q., Xia, H., Bao, R., Zheng, R., Zhang, Q., Gui, T., and Huang, X.-J. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, 2023.

Wang, X., Qi, Y., and Xu, B. Losia: Efficient high-rank fine-tuning via subnet localization and optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6707–6726, 2025b.

Wang, Y., Meng, F., Zhang, X., Jiang, F., Tang, P., and Zhang, M. Hd-pissa: High-rank distributed orthogonal adaptation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6526–6539, 2025c.

Wang, Z., Liang, J., He, R., Wang, Z., and Tan, T. Lora-pro: Are low-rank adapters properly optimized? *arXiv preprint arXiv:2407.18242*, 2024e.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Wu, Y., GAO, K., Li, L., and Wu, Y. Stable-loRA: Stabilizing feature learning of low-rank adaptation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=xSa19DAieH.

Wu, Z., Arora, A., Wang, Z., Geiger, A., Jurafsky, D., Manning, C. D., and Potts, C. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024.

Xiong, Y. and Xie, X. Oplora: Orthogonal projection lora prevents catastrophic forgetting during parameter-efficient fine-tuning. *arXiv preprint arXiv:2510.13003*, 2025.

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., and Tian, Q. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

Xu, Z., Min, H., MacDonald, L. E., Luo, J., Tarmoun, S., Mallada, E., and Vidal, R. Understanding the learning dynamics of lora: A gradient flow perspective on low-rank adaptation in matrix factorization. *arXiv preprint arXiv:2503.06982*, 2025.

Yan, M., Wang, Z., Jia, Z., Venkataraman, S., and Wang, Y. Plora: Efficient lora hyperparameter tuning for large models. *arXiv preprint arXiv:2508.02932*, 2025.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yang, A. X., Robeyns, M., Wang, X., and Aitchison, L. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023.

Yang, Y., Li, X., Zhou, Z., Song, S., Wu, J., Nie, L., and Ghanem, B. Corda: Context-oriented decomposition adaptation of large language models for task-aware parameter-efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:71768–71791, 2024a.

Yang, Y., Zhou, J., Wong, N., and Zhang, Z. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. *arXiv preprint arXiv:2402.11417*, 2024b.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pp. 581–590. IEEE, 2020.

Ye, J., haonan he, Li, M., Han, F., Chen, T., and Ye, P. Gradient intrinsic dimensionality alignment: Narrowing the gap between low-rank adaptation and full fine-tuning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=kObvnQ6pUx.

Yin, B., Yang, X., and Wang, X. Don't forget the nonlinearity: Unlocking activation functions in efficient fine-tuning. *arXiv preprint arXiv:2509.13240*, 2025.

Yin, F., Ye, X., and Durrett, G. Lofit: Localized fine-tuning on llm representations. *Advances in Neural Information Processing Systems*, 37:9474–9506, 2024.

Yu, J., Zhang, Y., Wang, B., Lin, P., Liu, Y., and Feng, S. Ssmlora: Enhancing low-rank adaptation with state space model. *arXiv preprint arXiv:2502.04958*, 2025a.

Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J. T., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Yu, X., Xie, C., Zhao, Z., Fan, T., Xue, L., and Zhang, Z. Prunedlora: Robust gradient-based structured pruning for low-rank adaptation in fine-tuning, 2025b. URL https://arxiv.org/abs/2510.00192.

Yuan, S., Liu, H., and Xu, H. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. *Advances in Neural Information Processing Systems*, 37:113484–113518, 2024.

Zaken, E. B., Goldberg, Y., and Ravfogel, S. Bitfit: https://www.overleaf.com/project/69f3ee837e6dfa2926effa24simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, 2022.

Zhang, C., Wang, K., and Gu, Y. Beyond low-rank tuning: Model prior-guided rank allocation for effective transfer in low-data and large-gap regimes. *arXiv preprint arXiv:2507.00327*, 2025a.

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

Zhang, H. Droplora: Sparse low-rank adaptation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2508.17337*, 2025.

Zhang, H., Huang, B., Li, Z., Xiao, X., Leong, H. Y., Zhang, Z., Long, X., Wang, T., and Xu, H. Sensitivity-lora: Low-load sensitivity-based fine-tuning for large language models. *arXiv preprint arXiv:2509.09119*, 2025b.

Zhang, J., You, J., Panda, A., and Goldstein, T. LoRA without forgetting: Freezing and sparse masking for low-rank adaptation. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025c. URL https://openreview.net/forum?id=aGOQYJfz6H.

Zhang, J.-C., Xiong, Y.-J., Xia, C.-M., Zhu, D.-H., and Zhan, H.-J. Lora2: Multi-scale low-rank approximations for fine-tuning large language models. *Neurocomputing*, 650:130859, 2025d.

Zhang, L., Zhang, L., Shi, S., Chu, X., and Li, B. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023a.

Zhang, Q., Chen, M., Bukharin, A., Karampatziakis, N., He, P., Cheng, Y., Chen, W., and Zhao, T. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023b.

Zhang, Q., Chu, C., Peng, T., Li, Q., Luo, X., Jiang, Z., and Huang, S.-L. Lora-da: Data-aware initialization for low-rank adaptation via asymptotic analysis. *arXiv preprint arXiv:2510.24561*, 2025e.

Zhang, R., Qiang, R., Somayajula, S. A., and Xie, P. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113*, 2024a.

Zhang, X., Chen, G.-Z., Li, S., Liu, Z., Chen, C. P., and Zhang, T. An orthogonal high-rank adaptation for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18826–18844, 2025f.

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., and Luo, Z. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37: 131786–131823, 2024b.

Zhang, Y., Liu, F., and Chen, Y. Lora-one: One-step full gradient could suffice for fine-tuning large language models, provably and efficiently. *arXiv preprint arXiv:2502.01235*, 2025g.

Zhang, Z., Li, H., Zhang, Y., Gong, G., Wang, J., Liu, P., Jiang, Q., and Hu, J. The primacy of magnitude in low-rank adaptation. *arXiv preprint arXiv:2507.06558*, 2025h.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024a.

Zhao, Y., Dang, S., Ye, H., Dai, G., Qian, Y., and Tsang, I. W. Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer. *arXiv preprint arXiv:2402.15173*, 2024b.

Zheng, T., Zhang, G., Shen, T., Liu, X., Lin, B. Y., Fu, J., Chen, W., and Yue, X. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024a.

Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z., Feng, Z., and Ma, Y. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024b.

Zhong, Y., Zhao, J., and Zhou, Y. Low-rank interconnected adaptation across layers. In *Findings of the Association*

for Computational Linguistics: ACL 2025, pp. 17005–17029, 2025.

Zhong, Y., Jiang, H., Li, L., Nakada, R., Liu, T., Zhang, L., Yao, H., and Wang, H. Peanut: Parameter-efficient adaptation with weight-aware neural tweakers. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 2054–2065, 2026.

Zhou, H., Lu, X., Xu, W., Zhu, C., Zhao, T., and Yang, M. Lora-drop: Efficient lora parameter pruning based on output evaluation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 5530–5543, 2025.

Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D., and Hou, L. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Zhu, M. and Nguyen, P. H. A survey of lora algorithm variations for language models. In *International Conference on Applications of Natural Language to Information Systems*, pp. 275–290. Springer, 2025.

Zi, B., Qi, X., Wang, L., Wang, J., Wong, K.-F., and Zhang, L. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*, 2023.

Zou, H., Zang, Y., Xu, W., Zhu, Y., and Ji, X. Flylora: Boosting task decoupling and parameter efficiency via implicit rank-wise mixture-of-experts. *arXiv preprint arXiv:2510.08396*, 2025.

Contents

A Additional Experiments **18**

A.1 Learning Rate Tuning on More Model–task Combinations . . . 18

A.2 Varying Adapter Ranks on Llama . . . 19

A.3 Varying Training Duration . . . 20

A.4 Scaling Batch Size to 512 for LoRA and PiSSA . . . 21

A.5 Fine-tuning on Instruction Following Tasks for LoRA and DoRA . . . 21

B Comprehensive Study of Hyperparameters in Prior Work **21**

B.1 Survey Criteria . . . 21

B.2 Comprehensive List of Papers . . . 22

C Detailed Formulas of Selected LoRA Variants **27**

C.1 Initialization Variants . . . 27

C.2 Architectural Modifications . . . 28

C.3 Optimization Adjustments . . . 29

D Fine-tuning Implementation Details **30**

D.1 Models . . . 30

D.2 Hyperparameter Search Ranges . . . 30

D.3 Fixed Training Hyperparameters . . . 30

D.4 Data, Code, Libraries, and Hardware . . . 30

E On LoRA Scaling Factor **31**

F Details of Hyperparameter Search Results **32**

F.1 Qwen3-0.6B . . . 32

F.2 Gemma-3-1B . . . 33

F.3 Llama-2-7B . . . 33

G Example Model Responses **36**

H Practical Heuristics for LoRA Hyperparameter Tuning **39**

I Hessian Computation Details **41**

I.1 Lanczos Algorithm Implementation Details . . . 41

I.2 Hessian Results on Gemma and Llama . . . 43

I.3 Detailed $\lambda_{\max}$ Values . . . 43

J Limitations and Future Work **44**

A. Additional Experiments

In this section, we present fine-tuning results for various LoRA methods under additional model–task combinations (Sec. A.1), diverse training durations (Sec. A.3), and various LoRA ranks for Qwen3 and Llama-2 (Sec. A.2). Moreover, we scale up the batch size in Sec. A.4 to examine performance under lower-stochasticity regimes and include instruction-following tasks in Sec. A.5 to cover diverse task types.

A.1. Learning Rate Tuning on More Model–task Combinations

Analogously to Figure 1, where we present all ten LoRA methods for Qwen3-0.6B on mathematical reasoning tasks, Figure 6 presents the corresponding comparison under commonsense reasoning tasks. With proper learning rate tuning, all methods consistently peak at a similar performance level ($\approx 37\%$). We note that, although the detailed ordering of the optimal learning rate ranges across methods may differ slightly from that shown in Figure 1, the overall trend remains broadly similar. For example, {DoRA, LoFT, GraLoRA} share slightly higher or comparable learning rate ranges relative to LoRA, whereas {PiSSA, Init[AB], RandLoRA, OLoRA, MiLoRA} tend to exhibit lower ranges. Notably, LoRA-GA consistently requires substantially lower learning rate ranges than LoRA, by more than $10\times$ in both Figure 1 and Figure 6. This relatively stable relationship between each LoRA variant and vanilla LoRA across different model–task combinations suggests that practitioners who wish to use different LoRA variants may use their known relative learning rate ranges with respect to LoRA as a practical prior to guide learning rate tuning, without necessarily conducting an exhaustive learning rate sweep as in our study or Hessian estimation as in Sec. 5.

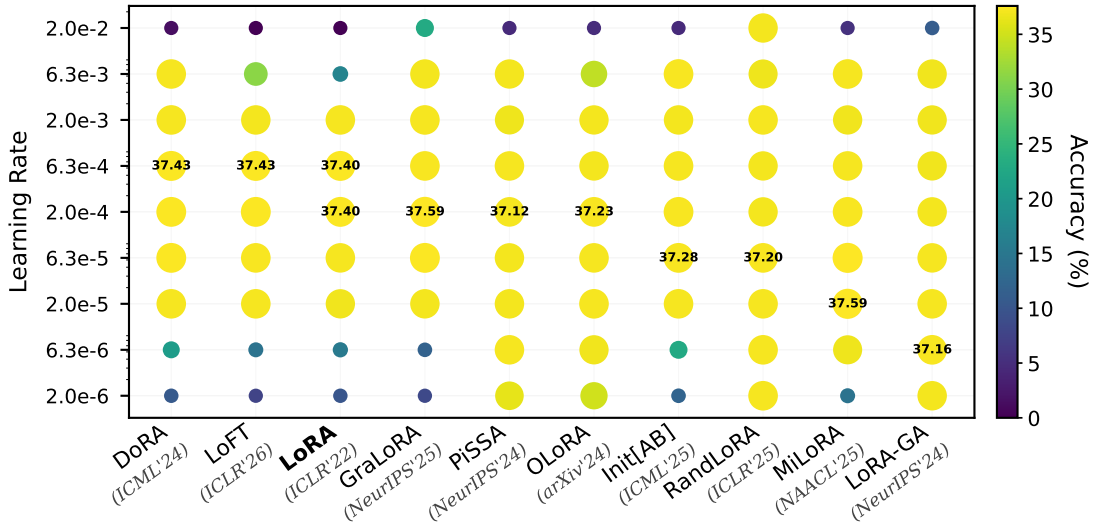
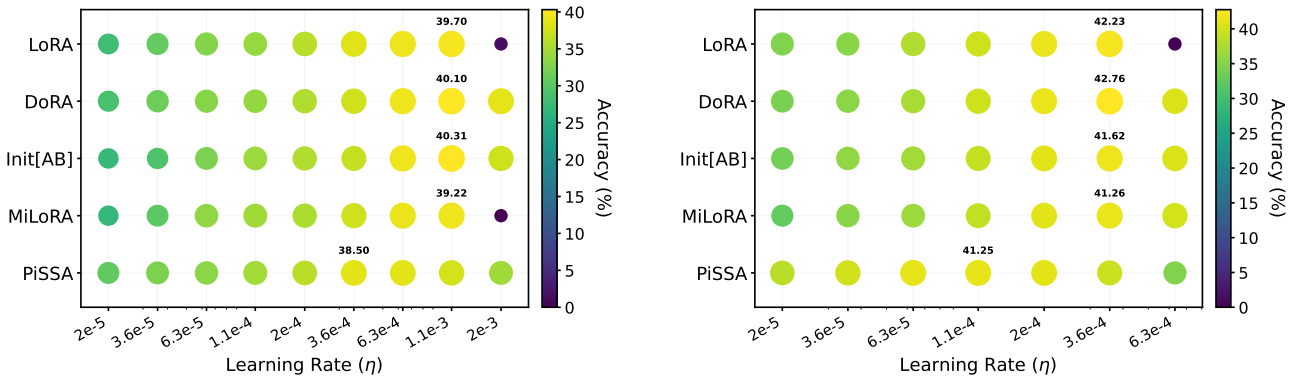


Figure 6. Performance of Gemma-3-1B fine-tuned on commonsense reasoning tasks across learning rates ($r = 4, B = 64$). Different methods reach a similar performance level once the learning rate is properly tuned. Each point is averaged over three independent training runs, and methods are sorted by their optimal learning rate ranges.

Similarly to Figure 3, where we present the performance of Llama-2-7B for LoRA, DoRA, Init[AB], MiLoRA, and PiSSA, Figure 7 scales the analysis up to Llama-2-13B. The results validate the generalizability of our conclusions to larger model scales, where the largest gains over vanilla LoRA are 0.61% for Init[AB] when $r = 8$ and 0.53% for DoRA when $r = 128$.

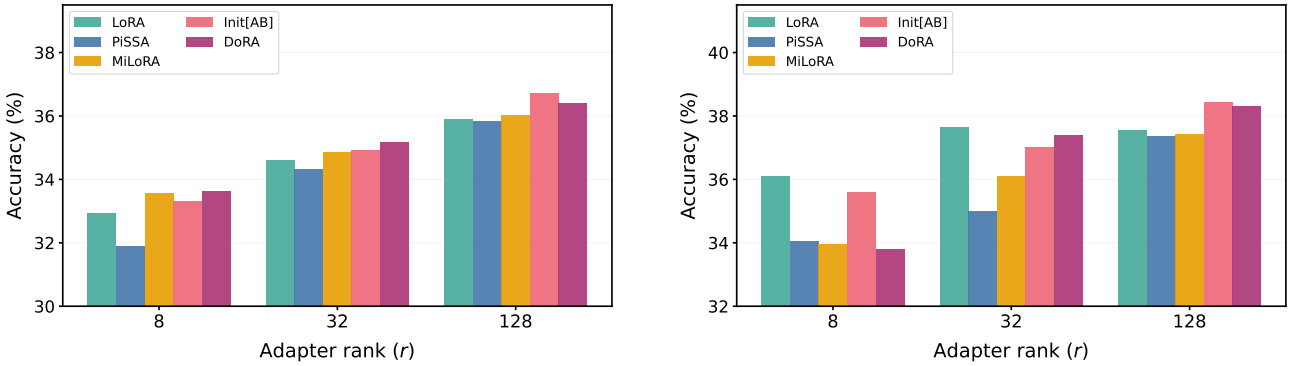


(a) $r = 8$ *(b)* $r = 128$

Figure 7. Performance of Llama-2-13B on mathematical reasoning tasks across varying learning rates and ranks $r \in \{8, 128\}$ ($B = 64$).

A.2. Varying Adapter Ranks on Llama

In Sec. 4.3.2, we analyzed the behavior of LoRA PEFT methods as the adapter rank varies on Gemma across math and code tasks. Here, we present a corresponding analysis on Llama in Figure 8, reporting the best performance achieved under joint optimization of learning rate and batch size ($B \in \{16, 128\}$). Similar trends can be observed, where all methods exhibit comparable performance improvement trends as the rank increases. Although under specific tasks or rank settings, one might favor a particular variant that marginally outperforms others, it is worth noting again that these improvements often lack universality, with vanilla LoRA frequently matching or even outperforming them.



(a) Mathematical Reasoning *(b)* Code Generation

Figure 8. Best achievable performance of LoRA and its advanced variants across adapter ranks on Llama-2-7B ($B \in \{16, 128\}$).

A.3. Varying Training Duration

We examine whether different methods consistently peak at comparable performance levels under varying training durations. Specifically, we vary the training duration by (1) scaling the number of MetaMathQA training samples from 5k up to the full 395k (Figure 9), and (2) varying the number of training epochs in {1, 2, 3} with a fixed 100k MetaMathQA training samples (Figure 10). With accuracies averaged over three independent runs, the results show that vanilla LoRA and its variants exhibit similar improvement trends as the number of training samples or epochs increases, with their performance generally falling within one another's standard-deviation range.

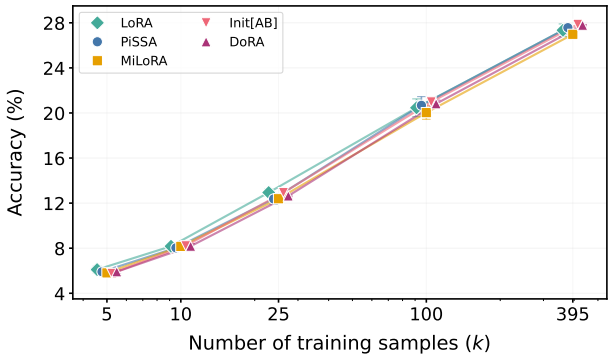


Figure 9. Best achievable performance of LoRA and its variants across different training sample sizes on mathematical reasoning with Gemma-3-1B ($r = 128, B = 64$). Once the learning rate is properly tuned, all methods exhibit nearly identical improvement trends as the number of training samples increases. Results are reported with mean and standard deviation over three runs.

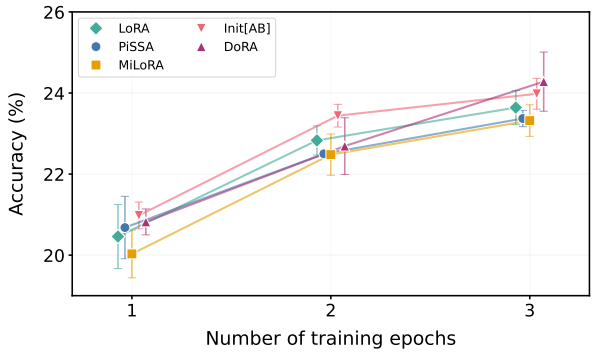


Figure 10. Best achievable performance of LoRA and its variants across different numbers of training epochs on mathematical reasoning with Gemma-3-1B ($r = 128, B = 64$). Results are reported with means and standard deviations over three independent runs. Once the learning rate is properly tuned, all methods exhibit nearly identical improvement trends as the number of training epochs increases, with their performance largely falling within each other's standard deviation ranges.

A.4. Scaling Batch Size to 512 for LoRA and PiSSA

In Table 1, we jointly optimize the learning rate and batch size for Gemma-3-1B on mathematical reasoning tasks. However, the batch sizes considered there remain within standard stochastic training regimes ($B \in \{16, 64, 128\}$). To further examine the behavior of LoRA methods under larger-batch, lower-stochasticity regimes, we scale B up to 512 in Figure 11. The results reveal an intriguing phenomenon: even after learning-rate tuning, the best achievable performance of both LoRA and PiSSA begins to decay when the batch size reaches $B \geq 256$. This suggests that, while learning-rate tuning should be prioritized, the batch size should still be kept within a relatively small-to-medium range, as stated in *practical heuristic* ***I***.

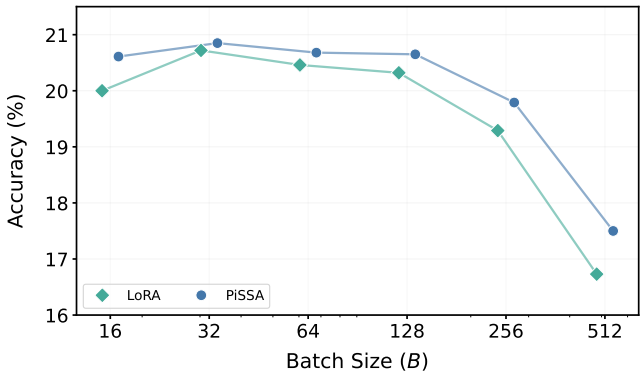


Figure 11. Best achievable performance of LoRA and PiSSA across different batch sizes on mathematical reasoning with Gemma-3-1B ($r = 128$). While at $B \leq 128$, both LoRA and PiSSA can reach $\approx 20\%$ accuracy with proper learning rate tuning, the performance upper bound gradually decays when B increases to 512.

A.5. Fine-tuning on Instruction Following Tasks for LoRA and DoRA

Beyond the commonsense reasoning, mathematical reasoning, and code generation tasks considered previously, we extend our evaluation to instruction-following tasks. Specifically, we train Qwen3-0.6B on Alpaca (Taori et al., 2023), while evaluating models' prompt-level strict accuracy under the IFEval (Zhou et al., 2023) framework.

	2e-06	**6.3e-06**	**2e-05**	**6.3e-05**	**2e-04**	**6.3e-04**	**2e-03**
LoRA	26.84	26.63	26.93	**28.20**	27.32	26.97	15.30
DoRA	26.99	26.41	26.41	26.44	**27.42**	26.54	16.86

Table 2. The prompt-level strict accuracy on IFEval with Qwen3-0.6B ($r = 128$, $B = 64$). Both DoRA and LoRA achieve a similar performance level under proper learning rate tuning.

B. Comprehensive Study of Hyperparameters in Prior Work

B.1. Survey Criteria

To generate the statistics presented in Figure 2, we curated a dataset comprising 64 papers, consisting of 54 studies published at major AI conferences or journals, 6 high-impact arXiv preprints (exceeding 40 citations), and 4 recent preprints released within the last six months. For each paper, we examined whether the authors reported performance metrics under learning rate or batch size tuning, and whether comparisons across different ranks were provided. The selection criteria for inclusion were as follows:

1. The primary objective of the proposed method is to enhance fine-tuning effectiveness (i.e., aiming for higher accuracy with equivalent trainable parameter counts or sustained performance with greater parameter efficiency).
2. Vanilla LoRA is explicitly employed as a baseline for performance comparison.

In assessing hyperparameter tuning, **our analysis focuses exclusively on decoder-only LLMs**, excluding encoder-only (Devlin et al., 2019), encoder-decoder (Raffel et al., 2020) architectures, Vision Transformers (Dosovitskiy, 2020), and

Vision-Language Models (Alayrac et al., 2022), as these lie outside the scope of this work. Consequently, papers lacking experiments on decoder-only LLMs are excluded from our statistics (e.g., Zhang et al. (2023b); Liu et al. (2023b); Xu et al. (2025); Zhang et al. (2025d); Edalati et al. (2025); Yuan et al. (2024)). Moreover, we exclude papers focusing on objectives other than PEFT efficiency, such as parameter-efficient pretraining (Lialin et al., 2023), continual learning (Wang et al., 2023; Zhao et al., 2024a), and quantization (Dettmers et al., 2023; Xu et al., 2023).

Given that some studies may tune hyperparameters only for their proposed methods while leaving baselines untuned (e.g., by adopting settings from prior work without modification), **we rigorously verified whether the vanilla LoRA baseline underwent tuning**. Specifically, we consider the learning rate and batch size to be "tuned" only if they are evaluated across at least three distinct values. Consequently, studies such as MiLoRA (which compared two sets of hyperparameter setups) or LoRA-GA (Wang et al., 2024d) (which tested learning rate in {1e-5, 5e-5}) do not qualify as tuning under our criteria. For rank, we require comparisons across at least two distinct values. Crucially, if a study varies the rank for its proposed method but benchmarks against a fixed-rank vanilla LoRA, we do not classify the baseline rank as tuned (e.g., Bałazy et al. (2024); Yuan et al. (2024); Li et al. (2024)).

During the data curation process, we observed that verifying specific hyperparameter tuning details can be non-trivial in some cases. The difficulty arises from discrepancies between paper versions (e.g., ablation studies on hyperparameters added to the Appendix post-publication), incomplete descriptions of experimental setups, or underspecified hyperparameter settings. Additionally, we frequently observed papers performing hyperparameter tuning only on smaller encoder-only LLMs (e.g., RoBERTa (Liu et al., 2019)). In strict adherence to our inclusion criteria, we do not categorize these instances as tuned. Moreover, we observed that many prior LoRA studies that did involve hyperparameter tuning reported only the final optimal performance, leaving unclear whether the adopted search ranges covered the optimal configurations for each method. While these ambiguities complicate binary categorization, we have made every effort to ensure accuracy. We emphasize that the statistics is curated solely to present the current state of hyperparameter tuning practices in LoRA PEFT research, and we welcome future contributions or corrections to further refine this collection.

B.2. Comprehensive List of Papers

A comprehensive list of the reviewed papers is detailed in Table 3. Specifically, "arXiv Date" marks the release date of the first version on arXiv (denoted as "–" if unavailable). "Pub. Date" refers to the formal publication date of the venue, where "–" indicates the paper has not yet been formally published. Note that the table is sorted primarily by Pub. Date, followed by arXiv Date.

Table 3. Publication dates, venues, and experimental configurations in prior LoRA PEFT studies. The table summarizes decoder-only LLMs and tasks, noting whether the vanilla LoRA baseline involved learning rate (**LR**) or batch size (**BS**) tuning and offered comparisons across different **Ranks**. A positive entry (✓) indicates the configuration was provided for at least one model-task combination; ✗ denotes otherwise. The symbol ∗ denotes a workshop paper.

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
DyLoRA (2023)	2022-10	2023-05	EACL	GPT-2 Medium	NLG	✗	✗	✓
GLoRA (2023)	2023-06	–	arXiv	Llama-1-7B Llama-2-7B	NLG	✗	✗	✗
LoRA-FA (2023a)	2023-08	–	arXiv	Llama-1-7B Llama-2-7B	Commonsense	✗	✗	✗
Laplace-LoRA (2023)	2023-08	2024-05	ICLR	Llama-1-7B Llama-2-7B	Commonsense	✗	✗	✗
VeRA (2023a)	2023-10	2024-05	ICLR	GPT-2 Medium/Large Llama-1-7B/13B Llama-2-7B/13B	NLG Instruction Following	✓	✗	✓
BoFT (2023b)	2023-11	2024-05	ICLR	Llama-2-7B	Instruction Following Math	✗	✗	✓
MoRA (2024)	2024-05	–	arXiv	Llama-2-7B/13B	UUID Math Instruction Following	✓	✗	✓
Delta-LoRA (2023)	2023-09	–	arXiv	GPT-2 Medium	NLG	✗	✗	✗

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
Tied-LoRA (2024)	2023-11	2024-06	NAACL	GPT-2B-001 Llama-2-7B	NLG Commonsense Math	✗	✗	✓
LoRETTA (2024b)	2024-02	2024-06	NAACL	Llama-2-7B/13B/70B	NLG GLUE	✗	✗	✗
AutoLoRA (2024a)	2024-03	2024-06	NAACL	GPT-2 Medium	NLG	✗	✗	✗
ALoRA (2024b)	2024-05	2024-06	NAACL	GPT2-Large Llama-2-7B	NLG GLUE Instruction Following	✗	✗	✓
RoSA (2024)	2024-01	2024-07	ICML	Llama-2-7B	NLG Math Code Instruction Following	✓	✗	✗
LoRA+ (2024b)	2024-02	2024-07	ICML	GPT-2 Llama-1-7B	GLUE Instruction Following	✓	✗	✗
scaled AdamW (2024)	2024-02	2024-07	ICML	GPT-2 Medium Mistral-7B-V0.1	NLG GLUE	✓	✗	✓
DoRA (2024a)	2024-02	2024-07	ICML	Llama-1-7B/13B Llama-2-7B Llama-3-8B	Commonsense	✗	✗	✓
FLORA (2024)	2024-02	2024-07	ICML	GPT-2 -base/XL	Summarization Translation	✓	✗	✓
FourierFT (2024)	2024-05	2024-07	ICML	GPT-2 Medium/Large Llama-1-7B/13B Llama2-7B/13B	NLG Instruction Following	✓	✓	✓
ResLoRA (2024)	2024-02	2024-08	ACL	Llama-2-7B	Math Commonsense	✗	✗	✓
PLoRA (2024b)	2024-02	–	arXiv	Llama-1-7B	Instruction Following Math	✓	✗	✓
OLoRA (2024)	2024-06	–	arXiv	Mistral-7B LLaMA-2-7B Tiny Llama-1.1B Gemma-2B OPT-1.3B	Commonsense Instruction Following	✗	✗	✓
LamDA (2024)	2024-06	2024-11	EMNLP	Llama-2-7B	NLG Math Commonsense	✗	✗	✓
PiSSA (2024a)	2024-04	2024-12	NeurIPS	Llama-2-7B/13B Llama-3-8B/70B Mistral-7B-v0.1 Gemma-7B Qwen1.5-7B Yi-1.5-34B DeepSeek-MoE-16B Mixtral-8x7B	Math Code Instruction Following	✗	✗	✓
VB-LoRA (2024)	2024-05	2024-12	NeurIPS	GPT-2 Medium/Large Llama-2-7B/13B Mistral-7B-v0.1 Gemma-7B	NLG Math Instruction Following	✗	✗	✗
HRA (2024)	2024-05	2024-12	NeurIPS	Llama-2-7B	Math	✗	✗	✗
CorDA (2024a)	2024-06	2024-12	NeurIPS	Llama-2-7B/13B Llama-3-8B Gemma-2-9B	Math Code Instruction Following World Knowledge	✗	✗	✓

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
LoRA-GA (2024d)	2024-07	2024-12	NeurIPS	Llama-2-7B	Math Code Instruction Following	✗	✗	✗
RoAd (2024)	2024-09	2024-12	NeurIPS	Llama-1-7B/13B Llama-2-7B Llama-3-8B	Math Commonsense	✗	✗	✓
LoRA-drop (2025)	2024-02	2025-01	COLING	Llama-2-7B	NLG Summarization GLUE Math	✗	✗	✗
AG-LoRA (2025)	–	2025-01	IEEE Access	Llama-1-7B	Commonsense	✗	✗	✓
LoRA-Pro (2024e)	2024-07	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Llama-3.1-8B	Math Code Code Instruction Following	✓	✗	✓
LoRA-Dash (2024)	2024-09	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Qwen2.5-7B	GLUE Commonsense	✗	✗	✓
KaSA (2024a)	2024-09	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Qwen2.5-7B	GLUE Commonsense Instruction Following	✗	✗	✓
RandLoRA (2025)	2025-02	2025-04	ICLR	GPT-2 Medium Qwen2-0.5B Phi3-3B Llama3-8B	NLG Commonsense	✗	✗	✓
DeLoRA (2025)	2025-03	2025-04	ICLR	Llama-2-7B Llama3-8B	Commonsense	✓	✗	✓
HiRA (2025)	–	2025-04	ICLR	Llama-2-7B Llama-3-8B	Math Commonsense Dialogue Generation	✗	✗	✓
MiLoRA (2024b)	2024-06	2025-04	NAACL	Llama-2-7B Llama-3-8B Qwen2.5-7B	Math Commonsense Instruction Following	✗	✗	✗
SSMLoRA (2025a)	2025-02	2025-04	NAACL	GPT-2 Llama-2-7B/13B	GLUE	✓	✗	✓
MiSS (2025)	2024-09	2025-07	ICML∗	Llama2-7B/13B Mistral-7B Qwen3-4B Llama-3.2-3B	Math Code Instruction Following	✗	✗	✓
LoRA-One (2025g)	2025-02	2025-07	ICML	Llama-2-7B	Math Code Instruction Following	✓	✓	✗
Init[AB] (2025)	2025-05	2025-07	ICML	Llama-3-8B	Arithmetic Commonsense	✓	✗	✗
Lily (2025)	2024-07	2025-07	ACL	Llama-3-8B	Commonsense	✗	✗	✗
C3A (2025)	2024-07	2025-07	ACL	Llama-2-7B Llama-3-8B/70B Mistral-7B Mistral-8x7B	Math Code Commonsense	✗	✗	✗
SuLoRA (2025)	–	2025-07	ACL	Llama-2-7B	Instruction Following	✗	✗	✓
BiDoRA (2025)	2024-10	2025-08	TMLR	GPT-2 Medium	NLG	✗	✗	✗

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
HD-PiSSA (2025c)	2025-05	2025-11	EMNLP	Llama-2-7B Llama-3-8B Mistral-7b-v0.1	Math Code	✗	✗	✓
LoSiA (2025b)	2025-07	2025-11	EMNLP	Gemma 2B Llama-2-7B/13B	Math Code Commonsense Instruction Following	✓	✗	✓
Sensitivity-LoRA (2025b)	2025-09	2025-11	EMNLP	GPT-2 Large Qwen2.5-7B/32B Llama-3.1-8B	NLG Instruction Following	✗	✗	✗
OHoRA (2025f)	–	2025-11	EMNLP	Llama-2-7B Llama-3-8B Gemma-7B Llama-3.1-8B-Inst	Math Code Commonsense Instruction Following	✗	✗	✓
EVA (2024)	2024-10	2025-12	NeurIPS	Llama-2-7B Gemma-2-70B Llama-3.1-8B/70B	Math Code Commonsense	✓	✗	✓
GoRA (2026)	2024-10	2025-12	NeurIPS	Llama-3.1-8B Llama-2-7B	Math Code Instruction Following	✗	✗	✗
AuroRA (2025)	2025-05	2025-12	NeurIPS	Llama-3-8B	Commonsense	✗	✗	✓
GraLoRA (2026)	2025-05	2025-12	NeurIPS	Llama-3.1-8B/70B Llama-3.2-3B Qwen-2.5-1B/7B	Math Code Commonsense	✗	✗	✓
FlyLoRA (2025)	2025-10	2025-12	NeurIPS	Llama-3.1-8B Qwen-2.5-7B/14B	MMLU Science Math Code	✗	✗	✓
DropLoRA (2025)	2025-08	–	arXiv	Llama-2-7B Llama-3-8B	Math Code Commonsense Instruction Following	✗	✗	✓
PrunedLoRA (2025b)	2025-09	–	arXiv	Llama-3-8B		✓	✗	✓
LoRA-DA (2025e)	2025-10	–	arXiv	Llama-2-7B	Math Commonsense	✗	✗	✓
ABM-LoRA (2025)	2025-11	–	arXiv	Llama-2-7B	Instruction Following	✗	✗	✓
LoFT (2025)	2025-05	2026-04	ICLR	GPT-2-base/Large Llama-1-7B Llama-2-7B Llama-3-8B Llama-3.1-70B	NLG Math Code Commonsense	✗	✗	✓
FlexLoRA (2026)	2026-01	2026-04	ICLR	Llama-3-8B	Commonsense	✗	✗	✗
Stable-LoRA (2026)	2026-05	2026-04	ICLR	Qwen-2-0.5B/1B Llama-1-7B Llama-3.1-8B Llama-3.2-1B/3B	Math Commonsense	✓	✗	✗
RaLoRA (2026)	–	2026-04	ICLR	LLaMA-3.1-8B	Math Code Instruction Following	✗	✗	✓
GiVA (2026)	2026-04	2026-05	AISTATS	Qwen-2-0.5B OLMo-2-7B Phi-3-3.8B Mistral-7B	Math Code Commonsense Instruction Following	✓	✗	✗

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
PEANuT (2026)	2024-10	2026-08	KDD	Llama-2-7B Llama-3-8B Qwen-3-8B	Math Commonsense	✗	✗	✗

C. Detailed Formulas of Selected LoRA Variants

C.1. Initialization Variants

OLoRA. Büyükakyüz (2024) proposed to improve the convergence behavior of LoRA by initializing the adaptation subspace with an orthonormal basis obtained from QR decomposition of pretrained weight matrices. OLoRA first performs QR decomposition:

$$W_{\text{pre}} = QR,$$

where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper triangular. Let $Q_r \in \mathbb{R}^{m\times r}$ denote the first r columns of Q, and $R_r \in \mathbb{R}^{r\times n}$ denote the first r rows of R. OLoRA initializes LoRA adapters as

$$B_0 = Q_r, \quad A_0 = R_r, \tag{3}$$

such that the initialized adapter component satisfies

$$B_0 A_0 = Q_r R_r. \tag{4}$$

To start fine-tuning from the pretrained weights, the *residual matrix* is defined as

$$W_{\text{res}} = W_{\text{pre}} - B_0 A_0 = W_{\text{pre}} - Q_r R_r, \tag{5}$$

and the forward pass becomes:

$$h = W_{\text{res}} x + BAx. \tag{6}$$

PiSSA. Aiming to address the potential slow convergence of LoRA, Meng et al. (2024a) proposed to initialize BA with the top-r principal components of the pretrained weight matrix and showed that this approach achieves faster convergence with loss and gradient norm curves similar to those of full fine-tuning. Suppose W_{pre} admits SVD into $\sum_i \sigma_i u_i v_i^T$ where σ_i are singular values in descending order, the LoRA adapter is initialized as:

$$B_0 A_0 = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

with

$$B_0 = \sum_{i=1}^{r} \sqrt{\sigma_i} u_i e_i^T, \quad A_0 = \sum_{i=1}^{r} \sqrt{\sigma_i} e_i v_i^T, \tag{7}$$

where $e_i \in \mathbb{R}^r$ denotes the i-th standard basis vector. To start fine-tuning from the pretrained weights, the *residual matrix* is defined as

$$W_{\text{res}} = W_{\text{pre}} - B_0 A_0 = \sum_{i=r+1}^{\min(m,n)} \sigma_i u_i v_i^T.$$

MiLoRA. Concurrent to PiSSA, Wang et al. (2024b) proposed an analogous approach targeting adaptation to new tasks while maximally retaining the pretrained model's knowledge. Instead of using principal components, MiLoRA initializes the low-rank adapters using bottom-r minor components:

$$B_0 A_0 = \sum_{i=\min(m,n)-r+1}^{\min(m,n)} \sigma_i u_i v_i^T,$$

with B_0 and A_0 defined analogously to Eq. 7. The residual matrix retains principal components:

$$W_{\text{res}} = W_{\text{pre}} - B_0 A_0 = \sum_{i=1}^{\min(m,n)-r} \sigma_i u_i v_i^T.$$

Wang et al. (2024b) showed experimentally that MiLoRA achieves superior downstream performance with less catastrophic forgetting.

Init[AB]. Several works have theoretically analyzed the initialization strategies of LoRA (Hayou et al., 2024a; Xu et al., 2025). In particular, Hayou et al. (2024a) confirmed that Init[A] (i.e., randomly initializing A only, the default LoRA setting) generally leads to better performance than Init[B] (randomly initializing B only). Li et al. (2025) further showed that initializing both matrices (i.e., Init[AB]) can provide even better advantage by balancing stability, training efficiency, and hyperparameter robustness. Specifically, Init[AB] initializes both matrices as $B_0 \sim \mathcal{N}(0, \sigma^2)$ and $A_0 \sim \mathcal{N}(0, \sigma^2)$. Since $B_0 A_0 \neq 0$ in this case, the residual matrix W_{res} is similarly introduced and utilized. Note that Li et al. (2025) also proposed a variant, Init[AB+], which does not require W_{res} and shows no discernible performance difference.

LoRA-GA. Wang et al. (2024c) proposed to accelerate LoRA convergence by aligning the first-step update of the low-rank product with the full fine-tuning gradient. Let $G = \nabla_W \mathcal{L}(W_{\text{pre}})$ denote the gradient of the loss with respect to the full weight matrix, computed on a sampled downstream batch. Instead of decomposing the pretrained weight matrix as in PiSSA and MiLoRA, LoRA-GA performs SVD on the gradient matrix:

$$G = U\Sigma V^T.$$

The goal is to initialize A and B such that the update of the low-rank product approximates the full fine-tuning update:

$$\Delta(BA) \approx \zeta \Delta W,$$

for some positive scaling constant ζ. Following the solution derived in LoRA-GA, the adapter matrices are initialized using selected singular directions of G:

$$B_0 = \frac{\sqrt{\zeta}}{\eta} U_{\mathcal{I}_B}, \quad A_0 = \frac{\sqrt{\zeta}}{\eta} V_{\mathcal{I}_A}^T,$$

where η denotes the scaling factor in LoRA-GA, and $\mathcal{I}_A$ and $\mathcal{I}_B$ are selected index sets of singular directions with $|\mathcal{I}_A| = |\mathcal{I}_B| = r$. Since $B_0 A_0 \neq 0$, the residual matrix is introduced as

$$W_{\text{res}} = W_{\text{pre}} - \eta B_0 A_0,$$

and the forward pass becomes:

$$h = W_{\text{res}} x + \eta BAx.$$

A notable follow-up is LoRA-One (Zhang et al., 2025g), which aims to address several limitations of LoRA-GA identified through theoretical analysis by using the top-r singular directions of G together with additional preconditioning and scaling choices.

C.2. Architectural Modifications

DoRA. Liu et al. (2024a) proposed to learn *magnitude* and *directional* updates of W_{pre} separately. Formally, the modified forward pass becomes:

$$h = \gamma_r \left(\frac{m}{\|W_{\text{pre}} + BA\|_c} \odot (W_{\text{pre}} + BA) \right) x, \tag{8}$$

where B and A are responsible for directional updates and initialized as vanilla LoRA, while $m \in \mathbb{R}^{1\times n}$ is an additional trainable magnitude vector initialized with $m_0 = \|W_{\text{pre}}\|_c$. Note that $\|.\|_c$ denotes taking the column-wise norm of a matrix, while $\odot$ denotes element-wise multiplication with broadcasting across columns. With only a marginal increase in trainable parameters introduced by the vector m, DoRA has been shown to consistently outperform LoRA, especially in regimes where the rank is small.

GraLoRA. Jung et al. (2026) proposed GraLoRA to address the structural bottleneck of vanilla LoRA, where increasing the rank does not always lead to better performance and may even distort gradient propagation under skewed input-channel statistics. Instead of applying a single global low-rank adapter to the entire weight matrix, GraLoRA partitions the weight matrix into a $k \times k$ grid of sub-blocks, each equipped with its own local low-rank adapter. Let k denote the number of splits along both the output and input dimensions. For $W_{\text{pre}} \in \mathbb{R}^{m\times n}$, the update is written as:

$$\Delta W_{\text{GraLoRA}} = \begin{bmatrix} B_{1,1}A_{1,1} & B_{1,2}A_{1,2} & \cdots & B_{1,k}A_{1,k} \\ B_{2,1}A_{2,1} & B_{2,2}A_{2,2} & \cdots & B_{2,k}A_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{k,1}A_{k,1} & B_{k,2}A_{k,2} & \cdots & B_{k,k}A_{k,k} \end{bmatrix},$$

where

$$B_{i,j} \in \mathbb{R}^{\frac{m}{k}\times\frac{r}{k}}, \quad A_{i,j} \in \mathbb{R}^{\frac{r}{k}\times\frac{n}{k}}.$$

The forward pass becomes:

$$h = W_{\text{pre}} x + \gamma_r \Delta W_{\text{GraLoRA}} x.$$

When $k = 1$, GraLoRA reduces to vanilla LoRA. When $k > 1$, the total number of trainable parameters remains identical to vanilla LoRA under the same overall rank:

$$k^2 \left(\frac{m}{k}\frac{r}{k} + \frac{r}{k}\frac{n}{k} \right) = (m+n)r.$$

By introducing localized adapters, GraLoRA provides more fine-grained control over different subregions of the weight matrix and reduces global gradient distortion caused by input outliers. Moreover, the effective rank of the resulting update can increase up to kr, thereby enhancing expressive capacity without increasing the trainable parameter count. Jung et al. (2026) showed that this design consistently improves over vanilla LoRA across multiple tasks, especially under larger-rank settings.

RandLoRA. Albert et al. (2025) proposed RandLoRA to examine whether the performance gap between LoRA and full fine-tuning stems from the reduced number of trainable parameters or from the low-rank constraint itself. The key idea is to maintain parameter efficiency while enabling full-rank updates through fixed random low-rank bases and trainable diagonal scaling matrices. Following their notation, let $W_{\text{pre}} \in \mathbb{R}^{m\times n}$ with $n \leq m$. RandLoRA represents the update as a sum of low-rank random basis components:

$$\Delta W_{\text{RandLoRA}} = \sum_{j=1}^{q} B_j \Lambda_j A \Gamma_j,$$

where $B_j \in \mathbb{R}^{m\times r_b}$ and $A \in \mathbb{R}^{r_b\times n}$ are non-trainable random matrices, while $\Lambda_j \in \mathbb{R}^{r_b\times r_b}$ and $\Gamma_j \in \mathbb{R}^{n\times n}$ are trainable diagonal matrices. The forward pass becomes:

$$h = W_{\text{pre}}x + \gamma_r \Delta W_{\text{RandLoRA}} x.$$

Since only the diagonal scaling matrices are optimized, the number of trainable parameters is substantially restricted. When $q = n/r_b$, RandLoRA can form a full-rank update with

$$q(r_b + n) = n + \frac{n^2}{r_b}$$

trainable parameters. This differs from vanilla LoRA, which directly learns a single rank-r product BA and is therefore constrained to rank at most r. RandLoRA's design rationale is to trade the flexibility of learning an arbitrary low-rank basis for broader spectral coverage through multiple fixed random bases. Albert et al. (2025) showed that RandLoRA can reduce or even eliminate the gap between LoRA and full fine-tuning in settings where full-rank updates are beneficial, while maintaining parameter and memory efficiency during training.

C.3. Optimization Adjustments

LoFT. Tastan et al. (2025) proposed LoFT to reduce the optimization gap between LoRA and full fine-tuning by aligning the optimizer's internal dynamics with those of updating all model weights. Following their notation, write the adapted weight as

$$W = W_0 + UV^T,$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ are trainable low-rank factors. For a scalar loss $f(W)$, a full fine-tuning gradient descent step is:

$$W^+ = W - \eta \nabla_W f(W).$$

In contrast, simultaneous updates of U and V induce the full-space update:

$$W^+ = W - \eta \left(\nabla_W f(W) VV^T + UU^T \nabla_W f(W)\right) + \eta^2 \nabla_W f(W) VU^T \nabla_W f(W),$$

where the last term is absent in full fine-tuning. LoFT therefore uses alternating updates, updating only one low-rank factor at a time. When updating U, this gives:

$$W^+ = W - \eta \nabla_W f(W) VV^T.$$

To address the scale ambiguity $UV^T = (cU)(V/c)^T$, LoFT applies scaled gradients:

$$\widetilde{\nabla}_U f(W) = \nabla_U f(W)(V^TV)^{-1}, \quad \widetilde{\nabla}_V f(W) = \nabla_V f(W)(U^TU)^{-1},$$

which yield the projected full-space update

$$W^+ = W - \eta \nabla_W f(W) P_V, \quad P_V = V(V^TV)^{-1}V^T.$$

Beyond gradient alignment, LoFT calibrates Adam-style optimizer states in the same low-rank subspace. For the first moment, when updating U, it uses:

$$m_k^U = \beta_1 m_{k-1}^U C_k^V + (1-\beta_1)\widetilde{\nabla}_U f(W_k), \quad C_k^V = (V_{k-1}^T V_k)(V_k^T V_k)^{-1}.$$

An analogous update is applied when updating V. For the second moment, LoFT similarly accumulates cross-terms for variance recalibration after projection:

$$p_k^U = \beta_2 p_{k-1}^U (C_k^V \otimes C_k^V) + (1-\beta_2)\left(\widetilde{\nabla}_U f(W_k) \bullet \widetilde{\nabla}_U f(W_k)\right),$$

where $\otimes$ denotes the Kronecker product and $\bullet$ denotes the transposed Khatri–Rao product. Together, alternating updates, scaled gradients, and optimizer-state calibration allow LoFT to better approximate full fine-tuning dynamics. Tastan et al. (2025) showed that LoFT narrows the performance gap between LoRA and full fine-tuning without increasing inference cost.

D. Fine-tuning Implementation Details

D.1. Models

Following standard practice in PEFT research to ensure results purely reflect the training data, we use base versions (not instruction-tuned) for all models (Hu et al., 2022; Meng et al., 2024a; Wang et al., 2024b; Li et al., 2025; Liu et al., 2024a). Specifically, we utilize the official checkpoints hosted on Hugging Face (Wolf et al., 2019): `Qwen3-0.6B-Base`[2], `gemma-3-1b-pt`[3], and `Llama-2-7b-hf`[4].

D.2. Hyperparameter Search Ranges

See Table 4 for hyperparameter search ranges for each model-task combination. For all experiments on Qwen and Gemma, we conduct three independent trainings and report the mean and standard deviation.

Model	Task	Rank (r)	Batch (B)	Learning Rate (η)
Qwen3-0.6B	Math	8	{16, 64, 128}	1.1247e-5 – 6.3246e-3
		128	64	2.0000e-6 – 2.0000e-3
	Inst	128	64	2.0000e-6 – 2.0000e-3
Gemma-3-1B	CS	4	64	2.0000e-6 – 2.0000e-2
	Math	{4, 8, 16, 32, 64, 128, 256}	64	1.1247e-5 – 6.3246e-3
		128	{16, 64, 128}	1.1247e-5 – 6.3246e-3
	Code	{4, 8, 16, 32, 64, 128, 256}	64	1.1247e-5 – 6.3246e-3
Llama-2-7B	Math	{8, 32, 128}	{16, 128}	2.0000e-5 – 3.5566e-3
	Code	{8, 32, 128}	{16, 128}	2.0000e-5 – 3.5566e-3
Llama-2-13B	Math	{8, 128}	64	2.0000e-5 – 2.0000e-3

Table 4. Summary of models, tasks, ranks, and hyperparameter search ranges. Learning rates are tuned evenly in logarithmic scale: {1.1247e*, 2.0000e*, 3.5566e*, 6.3246e*} per order of magnitude.

D.3. Fixed Training Hyperparameters

Except for tunable hyperparameters (i.e., learning rate and batch size), all other training configurations remain fixed and the same for all experiments; the values are summarized in Table 5. Note that these configurations primarily follow PiSSA, thus may differ from those of other considered PEFT methods. For example, MiLoRA, DoRA, and Init[AB] employ linear decay instead of cosine annealing for learning rate scheduling. Additionally, MiLoRA and DoRA use fixed warmup steps (100) rather than 3% of total training steps, and apply a dropout rate of 0.05 instead of no dropout. Furthermore, while we place low-rank adapters on all linear layers, these methods exclude output projections (`out_proj`) or gate matrices (`gate_proj`) in several of their experiments.

D.4. Data, Code, Libraries, and Hardware

We use the PiSSA codebase (Meng et al., 2024a) as the core framework and extend it into a unified implementation covering all ten LoRA-based methods considered in this study. Specifically, LoRA, OLoRA, PiSSA, LoRA-GA, DoRA, GraLoRA, and RandLoRA are implemented using the built-in interfaces of the official PEFT library (Mangrulkar et al., 2022). For MiLoRA, Init[AB], and LoFT, we refer to their official codebases[6] for essential functions and implementation details, which are then integrated into the same unified experimental codebase.

Note that while the original LoRA paper used Kaiming Normal initialization, we follow its official implementation and the widely-used PEFT library to use Kaiming Uniform instead in our experiments. The results are expected to be similar (cf. Meng et al. (2024a, Table 2)). Additionally, while Li et al. (2025) also proposed a variant, Init[AB+], which does not require W_{res} and shows no discernible performance difference, we chose to implement the default Init[AB].

For commonsense reasoning tasks, we leveraged the dataset released by Hu et al. (2023)[7]. For mathematical reasoning and code generation tasks,

[2] https://huggingface.co/Qwen/Qwen3-0.6B-Base
[3] https://huggingface.co/google/gemma-3-1b-pt
[4] https://huggingface.co/meta-llama/Llama-2-7b-hf
[5] Following PiSSA's codebase, normalization layers and `gate_proj` matrices of the pretrained model are converted back to Float32 after the BFloat16 sampling and before training.
[6] https://github.com/sufenlp/MiLoRA
https://github.com/Leopold1423/non_zero_lora-icml25
https://github.com/tnurbek/loft
[7] https://github.com/AGI-Edgerunners/LLM-Adapters/blob/main/ft-training_set/commonsense_15k.json

Configuration	Value
Epoch	1
α	r
Optimizer	AdamW (Loshchilov & Hutter, 2017)
LR scheduler	Cosine annealing with warmup
Warmup ratio	3%
Dropout	None
Weight Decay	None
Adapter placement	All linear layers
Base model precision	BFloat16 (Wang & Kanwar, 2019)[5]
Adapter precision	Float32
Max sequence length	512

Table 5. Fixed training configurations across all experiments. α equals the LoRA rank r, resulting in a scaling factor $\gamma_r = 1$. Adapters are applied to all linear projection layers (`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`).

we used the preprocessed dataset released by Meng et al. (2024a)[8]. For instruction-following tasks, we trained models on Alpaca-cleaned, which is publicly available on HuggingFace[9].

PyTorch (Paszke et al., 2019) version 2.7.1 is used for implementation. All experiments are conducted on four GPUs (either 4× Nvidia RTX 3090 or 4× Nvidia A6000). We employ DeepSpeed (Rasley et al., 2020) for parallel training and vLLM (Kwon et al., 2023) for parallel inference. During inference, we apply greedy decoding (i.e., temperature set to 0), and utilize the EvalPlus (Liu et al., 2023a) framework to evaluate pass@1 for code generation tasks.

All fine-tuning experiments (except those for Llama) are conducted with three independent runs and reported with means and standard deviations. The sources of randomness are controlled by explicitly fixing random seeds across Python, NumPy, and PyTorch using the code snippet shown below.

```
def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True
```

E. On LoRA Scaling Factor

The configuration of the LoRA alpha parameter (α) generally follows two paradigms: (1) setting it to a fixed constant across LoRA ranks (typically 32 or 64), or (2) scaling it with the LoRA rank, often following $\alpha = r$ or $\alpha = 2r$, which results in a scaling factor γ_r of 1 or 2, respectively. The configurations adopted for decoder-LLMs in the considered LoRA variants are summarized as follows:

- **PiSSA:** $\gamma_r = 1$ ($\alpha = r$ for all r).
- **MiLoRA:** $\gamma_r = 2$ for vanilla LoRA; $\gamma_r = 1$ for both MiLoRA and PiSSA.
- **Init[AB]:** $\gamma_r = 1$ ($r = 16$ and $\alpha = 16$).
- **DoRA:** $\gamma_r = 2$ ($\alpha = 2r$ for all r).

Evidently, the methods considered in this paper adhere to the second paradigm. Consequently, we adopt the setting $\alpha = r$ ($\gamma_r = 1$) for all our experiments. We refer readers to several prior studies that have explored the optimal LoRA scaling factor: Kalajdzievski (2023) argued that γ_r should scale with the square root of r ($\gamma_r = \alpha/\sqrt{r}$), rather than linearly ($\gamma_r = \alpha/r$), though the optimal α setup remains unclear. Empirically, Biderman et al. (2024) demonstrated through joint sweeps of α and learning rates that $\alpha = 2r$ ($\gamma_r = 2$) is the optimal choice, with $\alpha = r$ ($\gamma_r = 1$) performing only marginally worse (cf. Biderman et al. (2024, Appendix B.2, Figure S3)). Notably, Zhang et al. (2025h) recently unified the learning rate, scaling factor, and initialization under a single theoretical framework, suggesting that tuning the learning rate is theoretically equivalent to tuning the scaling factor (Fan et al., 2025). This further validates our decision to fix the scaling factor in our experiments.

[8] https://huggingface.co/datasets/fxmeng/pissa-dataset
[9] https://huggingface.co/datasets/yahma/alpaca-cleaned

F. Details of Hyperparameter Search Results

Sections F.1, F.2, and F.3 present detailed hyperparameter search results on Qwen, Gemma, and Llama, respectively. Note that Tables 7, 11, and 14 correspond to the detailed numerical results for Figures 1, 3a, and 3b in the main text, respectively. Additionally, Tables 9, 10, and 11 provide the hyperparameter search details across adapter ranks for Appendix Figure 8a, while Tables 12, 13, and 14 correspond to Appendix Figure 8b.

F.1. Qwen3-0.6B

F.1.1. Mathematical Reasoning

Methods	Batch Size	Learning Rate											
		1.12e-05	2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03	3.56e-03	6.32e-03
LoRA	16	$38.94_{\pm1.02}$	$46.49_{\pm1.17}$	$47.78_{\pm0.47}$	$47.85_{\pm0.42}$	$48.13_{\pm0.40}$	$48.90_{\pm0.41}$	$48.41_{\pm0.59}$	$\mathbf{49.05}_{\pm0.49}$	$47.64_{\pm0.86}$	$44.00_{\pm0.45}$	$26.59_{\pm21.82}$	$5.03_{\pm5.33}$
	64	$29.53_{\pm0.18}$	$33.02_{\pm0.28}$	$39.15_{\pm0.33}$	$46.49_{\pm0.25}$	$48.16_{\pm0.24}$	$48.39_{\pm0.29}$	$48.95_{\pm0.23}$	$\mathbf{48.99}_{\pm0.40}$	$48.73_{\pm0.10}$	$48.14_{\pm0.42}$	$43.92_{\pm0.48}$	$1.28_{\pm0.05}$
	128	$22.30_{\pm0.30}$	$30.22_{\pm1.63}$	$33.64_{\pm0.13}$	$40.64_{\pm1.48}$	$47.88_{\pm0.61}$	$48.38_{\pm0.19}$	$48.38_{\pm0.01}$	$48.69_{\pm0.57}$	$\mathbf{48.72}_{\pm0.04}$	$48.33_{\pm0.54}$	$31.32_{\pm26.20}$	$1.13_{\pm0.58}$
DoRA	16	$42.03_{\pm1.63}$	$47.36_{\pm0.59}$	$48.10_{\pm0.12}$	$48.10_{\pm0.48}$	$48.29_{\pm0.08}$	$\mathbf{48.80}_{\pm0.64}$	$48.60_{\pm0.17}$	$48.70_{\pm0.05}$	$46.30_{\pm0.29}$	$42.67_{\pm0.19}$	$35.93_{\pm0.44}$	$1.31_{\pm0.19}$
	64	$38.69_{\pm1.24}$	$37.60_{\pm1.22}$	$40.65_{\pm1.58}$	$47.06_{\pm0.50}$	$48.41_{\pm0.31}$	$48.03_{\pm0.18}$	$\mathbf{49.07}_{\pm0.03}$	$48.87_{\pm0.82}$	$48.55_{\pm0.43}$	$47.31_{\pm0.48}$	$44.61_{\pm0.00}$	$1.10_{\pm0.31}$
	128	$33.37_{\pm1.56}$	$36.85_{\pm0.98}$	$36.56_{\pm0.41}$	$43.03_{\pm1.11}$	$48.46_{\pm0.54}$	$47.94_{\pm0.30}$	$48.30_{\pm0.24}$	$48.41_{\pm0.72}$	$\mathbf{48.59}_{\pm0.10}$	$47.75_{\pm0.47}$	$46.08_{\pm0.10}$	$14.91_{\pm23.56}$
Init[AB]	16	$36.53_{\pm2.20}$	$41.67_{\pm1.99}$	$45.47_{\pm1.30}$	$48.07_{\pm0.70}$	$48.28_{\pm0.72}$	$\mathbf{48.66}_{\pm0.31}$	$48.53_{\pm0.48}$	$48.18_{\pm0.49}$	$46.79_{\pm0.19}$	$42.32_{\pm0.39}$	$38.65_{\pm0.80}$	$20.67_{\pm27.96}$
	64	$35.78_{\pm0.54}$	$35.15_{\pm1.09}$	$37.85_{\pm0.08}$	$40.04_{\pm1.50}$	$45.03_{\pm1.13}$	$48.34_{\pm0.29}$	$48.53_{\pm0.07}$	$48.45_{\pm0.50}$	$\mathbf{48.68}_{\pm0.15}$	$47.11_{\pm0.43}$	$43.13_{\pm0.80}$	$1.36_{\pm0.06}$
	128	$31.34_{\pm1.30}$	$32.44_{\pm0.70}$	$36.21_{\pm0.32}$	$35.29_{\pm1.90}$	$41.73_{\pm1.33}$	$47.06_{\pm1.29}$	$48.38_{\pm0.60}$	$\mathbf{48.57}_{\pm0.39}$	$48.12_{\pm0.01}$	$48.34_{\pm0.66}$	$46.79_{\pm0.22}$	$0.99_{\pm0.52}$
MiLoRA	16	$39.42_{\pm0.87}$	$45.09_{\pm0.19}$	$44.76_{\pm0.78}$	$45.92_{\pm0.81}$	$49.16_{\pm0.37}$	$\mathbf{49.36}_{\pm0.09}$	$48.93_{\pm0.37}$	$48.08_{\pm0.06}$	$46.09_{\pm0.55}$	$43.39_{\pm0.52}$	$25.53_{\pm20.93}$	$1.47_{\pm0.07}$
	64	$32.25_{\pm1.20}$	$38.33_{\pm1.24}$	$45.72_{\pm0.88}$	$44.08_{\pm1.50}$	$47.32_{\pm0.05}$	$48.69_{\pm0.32}$	$\mathbf{49.40}_{\pm0.01}$	$49.06_{\pm0.18}$	$48.90_{\pm0.37}$	$47.02_{\pm0.38}$	$43.98_{\pm1.06}$	$1.07_{\pm0.50}$
	128	$30.94_{\pm0.32}$	$33.71_{\pm0.34}$	$35.03_{\pm0.69}$	$40.49_{\pm0.39}$	$44.27_{\pm0.12}$	$48.08_{\pm0.42}$	$48.65_{\pm0.18}$	$\mathbf{49.37}_{\pm0.19}$	$49.33_{\pm0.52}$	$48.12_{\pm0.74}$	$46.91_{\pm0.06}$	$1.24_{\pm0.33}$
PiSSA	16	$47.10_{\pm0.28}$	$44.80_{\pm0.71}$	$46.45_{\pm0.61}$	$\mathbf{48.37}_{\pm0.36}$	$48.30_{\pm0.26}$	$47.42_{\pm0.22}$	$45.47_{\pm0.36}$	$42.43_{\pm0.03}$	$38.83_{\pm0.40}$	$33.14_{\pm0.41}$	$24.63_{\pm0.98}$	$14.82_{\pm19.10}$
	64	$44.20_{\pm0.12}$	$44.12_{\pm0.69}$	$47.54_{\pm0.50}$	$48.25_{\pm0.14}$	$\mathbf{48.46}_{\pm0.59}$	$48.43_{\pm0.12}$	$47.94_{\pm0.39}$	$47.23_{\pm0.19}$	$43.94_{\pm0.04}$	$40.15_{\pm0.20}$	$35.37_{\pm0.55}$	$18.90_{\pm15.52}$
	128	$39.49_{\pm2.26}$	$43.64_{\pm0.51}$	$43.61_{\pm0.50}$	$46.42_{\pm0.10}$	$\mathbf{48.51}_{\pm0.31}$	$48.24_{\pm0.18}$	$48.06_{\pm0.72}$	$47.70_{\pm0.36}$	$45.56_{\pm0.15}$	$43.54_{\pm0.41}$	$38.85_{\pm0.18}$	$33.36_{\pm0.44}$

Table 6. Performance of **Qwen3-0.6B** fine-tuned on **mathematical reasoning** tasks with **rank = 8**.

Methods	Batch Size	Learning Rate											
		2.00e-06	3.56e-06	6.32e-06	1.12e-05	2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03
LoRA	64	$21.48_{\pm0.53}$	$24.52_{\pm0.44}$	$31.94_{\pm0.58}$	$43.07_{\pm1.16}$	$48.37_{\pm0.19}$	$48.81_{\pm0.14}$	$49.27_{\pm0.34}$	$49.46_{\pm0.56}$	$\mathbf{49.60}_{\pm0.18}$	$48.95_{\pm0.20}$	$47.08_{\pm0.22}$	$40.76_{\pm0.96}$	$1.27_{\pm0.15}$
DoRA	64	$36.84_{\pm0.86}$	$38.19_{\pm1.74}$	$39.29_{\pm0.36}$	$44.72_{\pm0.96}$	$48.09_{\pm0.74}$	$49.01_{\pm0.44}$	$49.25_{\pm0.33}$	$\mathbf{49.45}_{\pm0.25}$	$49.33_{\pm0.45}$	$49.32_{\pm0.71}$	$46.92_{\pm0.28}$	$40.24_{\pm0.62}$	$10.14_{\pm15.64}$
Init[AB]	64	$33.40_{\pm2.16}$	$35.81_{\pm1.72}$	$41.20_{\pm1.83}$	$47.27_{\pm0.89}$	$49.27_{\pm0.13}$	$49.02_{\pm0.11}$	$48.81_{\pm0.17}$	$\mathbf{49.29}_{\pm0.23}$	$48.51_{\pm0.44}$	$47.37_{\pm0.39}$	$44.81_{\pm0.34}$	$39.41_{\pm0.49}$	$0.93_{\pm0.41}$
MiLoRA	64	$33.62_{\pm0.39}$	$38.35_{\pm0.46}$	$43.37_{\pm0.41}$	$48.30_{\pm0.27}$	$49.08_{\pm0.23}$	$48.74_{\pm0.32}$	$\mathbf{49.17}_{\pm0.38}$	$49.08_{\pm0.14}$	$48.22_{\pm0.20}$	$46.57_{\pm0.73}$	$43.91_{\pm0.58}$	$38.70_{\pm0.34}$	$9.48_{\pm14.40}$
PiSSA	64	$38.13_{\pm0.84}$	$44.51_{\pm0.26}$	$48.11_{\pm0.14}$	$48.77_{\pm0.12}$	$\mathbf{49.43}_{\pm0.19}$	$49.09_{\pm0.16}$	$48.44_{\pm0.12}$	$47.10_{\pm0.32}$	$43.84_{\pm0.28}$	$39.66_{\pm0.36}$	$34.37_{\pm0.39}$	$27.18_{\pm1.29}$	$17.35_{\pm0.57}$

Table 7. Performance of **Qwen3-0.6B** fine-tuned on **mathematical reasoning** tasks with **rank = 128**.

F.2. Gemma-3-1B

F.2.1. Mathematical Reasoning

Methods	Batch Size	Learning Rate											
		1.12e-5	2.00e-5	3.56e-5	6.32e-5	1.12e-4	2.00e-4	3.56e-4	6.32e-4	1.12e-3	2.00e-3	3.56e-3	6.32e-3
LoRA	16	$9.78_{\pm0.36}$	$11.16_{\pm0.28}$	$13.58_{\pm0.18}$	$15.48_{\pm0.15}$	$18.43_{\pm0.14}$	$\mathbf{20.00}_{\pm0.26}$	$19.93_{\pm0.65}$	$17.99_{\pm0.55}$	$11.71_{\pm0.49}$	$1.52_{\pm0.19}$	$1.27_{\pm0.59}$	$1.07_{\pm0.27}$
	64	$6.88_{\pm0.04}$	$9.12_{\pm0.39}$	$10.79_{\pm0.37}$	$13.23_{\pm0.25}$	$15.65_{\pm0.57}$	$17.54_{\pm0.29}$	$19.73_{\pm0.16}$	$\mathbf{20.46}_{\pm0.79}$	$19.83_{\pm0.91}$	$13.33_{\pm0.81}$	$1.48_{\pm0.48}$	$0.00_{\pm0.00}$
	128	$5.70_{\pm0.34}$	$6.95_{\pm0.23}$	$9.41_{\pm0.44}$	$11.43_{\pm0.40}$	$13.68_{\pm0.77}$	$15.92_{\pm0.45}$	$18.58_{\pm0.44}$	$19.60_{\pm0.09}$	$\mathbf{20.32}_{\pm0.28}$	$16.95_{\pm2.70}$	$0.09_{\pm0.16}$	$0.00_{\pm0.00}$
DoRA	16	$9.89_{\pm0.24}$	$11.16_{\pm0.51}$	$13.84_{\pm0.41}$	$15.61_{\pm0.11}$	$18.21_{\pm0.45}$	$20.11_{\pm0.26}$	$\mathbf{20.96}_{\pm0.57}$	$18.34_{\pm0.20}$	$11.90_{\pm0.29}$	$4.89_{\pm0.99}$	$0.93_{\pm0.12}$	$1.16_{\pm0.15}$
	64	$6.72_{\pm0.09}$	$9.19_{\pm0.19}$	$10.53_{\pm0.20}$	$13.45_{\pm0.31}$	$15.72_{\pm0.32}$	$17.66_{\pm0.20}$	$19.96_{\pm0.05}$	$\mathbf{20.82}_{\pm0.32}$	$19.87_{\pm0.91}$	$13.53_{\pm1.64}$	$1.52_{\pm0.45}$	$0.34_{\pm0.23}$
	128	$5.55_{\pm0.11}$	$7.21_{\pm0.18}$	$9.72_{\pm0.17}$	$11.58_{\pm0.25}$	$13.98_{\pm0.33}$	$16.19_{\pm0.46}$	$18.25_{\pm0.23}$	$19.67_{\pm0.71}$	$\mathbf{20.33}_{\pm0.64}$	$12.86_{\pm10.03}$	$0.13_{\pm0.23}$	$0.02_{\pm0.03}$
Init[AB]	16	$9.73_{\pm0.35}$	$12.10_{\pm0.14}$	$14.41_{\pm0.49}$	$16.73_{\pm0.37}$	$18.38_{\pm0.53}$	$20.39_{\pm0.38}$	$\mathbf{20.55}_{\pm0.40}$	$18.34_{\pm0.48}$	$11.94_{\pm0.31}$	$1.48_{\pm0.24}$	$1.16_{\pm0.31}$	$1.45_{\pm0.17}$
	64	$6.51_{\pm0.22}$	$9.15_{\pm0.12}$	$11.28_{\pm0.20}$	$13.20_{\pm0.24}$	$15.88_{\pm0.39}$	$17.89_{\pm0.30}$	$20.08_{\pm0.26}$	$\mathbf{20.98}_{\pm0.33}$	$19.31_{\pm0.75}$	$13.97_{\pm0.03}$	$2.74_{\pm3.83}$	$0.07_{\pm0.12}$
	128	$6.06_{\pm0.35}$	$7.05_{\pm0.33}$	$9.53_{\pm0.22}$	$11.81_{\pm0.08}$	$13.98_{\pm0.79}$	$16.46_{\pm0.39}$	$18.36_{\pm0.21}$	$20.37_{\pm0.39}$	$\mathbf{20.66}_{\pm0.39}$	$17.85_{\pm0.84}$	$4.40_{\pm7.46}$	$0.00_{\pm0.00}$
MiLoRA	16	$12.44_{\pm0.07}$	$13.77_{\pm0.25}$	$16.28_{\pm0.24}$	$18.45_{\pm0.47}$	$20.04_{\pm0.19}$	$\mathbf{20.63}_{\pm0.67}$	$19.40_{\pm0.80}$	$15.72_{\pm0.49}$	$10.22_{\pm0.42}$	$2.03_{\pm0.95}$	$1.35_{\pm0.43}$	$1.56_{\pm0.65}$
	64	$8.82_{\pm0.40}$	$11.25_{\pm0.20}$	$13.16_{\pm0.11}$	$15.54_{\pm0.29}$	$17.43_{\pm0.24}$	$19.56_{\pm0.33}$	$\mathbf{20.03}_{\pm0.59}$	$19.60_{\pm0.78}$	$17.93_{\pm0.90}$	$13.65_{\pm0.07}$	$4.97_{\pm0.40}$	$0.00_{\pm0.00}$
	128	$7.32_{\pm0.33}$	$9.57_{\pm0.24}$	$11.76_{\pm0.33}$	$13.54_{\pm0.12}$	$16.02_{\pm0.16}$	$18.39_{\pm0.26}$	$19.70_{\pm0.34}$	$\mathbf{19.99}_{\pm0.66}$	$19.53_{\pm0.47}$	$16.83_{\pm0.73}$	$7.45_{\pm1.00}$	$0.57_{\pm0.81}$
PiSSA	16	$14.30_{\pm0.18}$	$16.10_{\pm0.27}$	$18.31_{\pm0.12}$	$19.90_{\pm0.21}$	$\mathbf{20.61}_{\pm0.28}$	$19.09_{\pm0.20}$	$16.10_{\pm0.64}$	$13.25_{\pm0.55}$	$8.41_{\pm0.13}$	$4.67_{\pm0.29}$	$2.50_{\pm1.27}$	$0.96_{\pm0.15}$
	64	$11.11_{\pm0.05}$	$13.67_{\pm0.17}$	$15.56_{\pm0.33}$	$18.11_{\pm0.23}$	$19.52_{\pm0.48}$	$\mathbf{20.68}_{\pm0.77}$	$20.59_{\pm0.32}$	$19.11_{\pm0.86}$	$15.53_{\pm0.37}$	$9.57_{\pm0.72}$	$5.78_{\pm0.37}$	$0.33_{\pm0.46}$
	128	$9.42_{\pm0.38}$	$11.80_{\pm0.28}$	$14.40_{\pm0.11}$	$16.23_{\pm0.38}$	$18.60_{\pm0.21}$	$19.61_{\pm0.44}$	$\mathbf{20.65}_{\pm0.44}$	$19.21_{\pm1.15}$	$16.91_{\pm0.19}$	$13.87_{\pm0.97}$	$6.28_{\pm0.49}$	$1.19_{\pm0.36}$

Table 8. Performance of **Gemma-3-1B** fine-tuned on **mathematical reasoning** tasks with **rank=128**.

F.3. Llama-2-7B

F.3.1. Mathematical Reasoning

Methods	Batch Size	Learning Rate									
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03	3.56e-03
LoRA	16	23.16	24.55	26.05	28.73	30.70	32.18	**32.94**	32.02	27.71	0.00
	128	16.12	18.60	21.46	23.49	25.91	28.21	30.31	32.28	**32.78**	1.97
DoRA	16	22.74	24.34	26.20	28.44	30.62	**33.20**	32.71	32.43	1.59	1.98
	128	16.07	18.98	21.57	23.70	26.16	28.54	30.20	32.80	**33.62**	0.00
Init[AB]	16	20.89	23.36	27.08	29.25	31.24	**33.30**	32.78	31.34	27.38	0.04
	128	15.79	17.64	20.17	22.96	25.42	28.03	30.45	32.32	**32.96**	31.08
MiLoRA	16	21.12	23.45	25.61	28.38	30.59	32.49	**33.22**	32.46	27.56	0.00
	128	15.72	18.51	20.70	22.58	25.32	26.76	29.87	31.48	**33.55**	0.26
PiSSA	16	22.66	26.30	28.20	30.12	**31.91**	31.62	30.57	28.76	0.84	0.92
	128	18.94	21.60	22.80	26.23	28.14	30.61	31.64	**31.86**	31.53	0.48

Table 9. Performance of **Llama-2-7B** fine-tuned on **mathematical reasoning** with **rank=8**.

Methods	Batch Size	Learning Rate								
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03
LoRA	**16**	25.03	27.77	29.67	32.11	33.73	33.84	**34.18**	27.01	1.31
	128	19.93	22.25	24.08	26.29	29.13	32.19	33.24	**34.62**	0.00
DoRA	**16**	25.41	27.35	29.78	31.96	34.14	**35.16**	33.26	28.92	0.55
	128	20.60	22.41	24.02	26.66	29.71	31.96	33.41	**34.25**	0.00
Init[AB]	**16**	24.06	27.68	28.71	32.52	34.10	**34.92**	34.12	27.16	0.77
	128	17.83	20.96	23.15	26.62	28.34	30.79	33.38	**34.59**	0.97
MiLoRA	**16**	24.53	27.23	29.23	31.44	33.97	**34.85**	33.85	28.35	0.62
	128	18.23	21.45	23.65	26.54	27.97	30.29	32.39	**34.59**	0.00
PiSSA	**16**	29.42	30.68	32.75	**33.66**	33.62	32.66	29.30	18.36	0.15
	128	23.98	27.02	29.44	30.01	32.89	33.42	**34.31**	32.92	0.65

Table 10. Performance of **Llama-2-7B** fine-tuned on **mathematical reasoning** with **rank=32**.

Methods	Batch Size	Learning Rate								
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03
LoRA	**16**	29.21	31.30	33.25	35.45	**35.91**	35.10	27.41	0.97	0.00
	128	22.69	24.95	27.79	30.74	32.62	34.85	**35.66**	0.00	0.00
DoRA	**16**	29.43	30.75	33.14	35.73	**36.41**	34.54	1.28	0.94	0.79
	128	23.27	25.63	27.87	30.11	33.00	35.10	**35.57**	0.38	0.00
Init[AB]	**16**	29.04	31.52	31.96	34.81	**36.72**	35.41	27.98	1.54	0.00
	128	22.01	25.03	28.11	30.47	31.80	34.78	**35.57**	34.45	0.00
MiLoRA	**16**	28.23	31.11	33.42	35.22	**36.02**	34.71	28.03	0.38	0.00
	128	22.14	24.84	27.88	30.33	31.29	33.67	**35.23**	0.00	0.00
PiSSA	**16**	33.35	35.03	35.27	**35.83**	32.89	27.90	16.75	1.45	0.00
	128	29.84	31.64	33.44	34.45	**35.31**	34.99	31.59	27.83	0.00

Table 11. Performance of **Llama-2-7B** fine-tuned on **mathematical reasoning** with **rank=128**.

F.3.2. CODE GENERATION

Methods	Batch Size	Learning Rate									
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03	3.56e-03
LoRA	**16**	27.20	29.40	27.55	30.00	31.40	33.25	**36.10**	32.65	32.30	0.00
	128	24.50	25.60	27.05	28.15	29.30	30.90	30.45	**35.35**	33.35	0.00
DoRA	**16**	28.05	28.45	28.90	31.15	31.80	33.15	**33.80**	32.00	29.50	0.00
	128	24.65	26.95	27.25	29.65	29.15	30.15	33.15	33.15	**33.35**	32.55
Init[AB]	**16**	26.25	27.30	30.30	29.90	32.90	34.15	**35.60**	32.70	31.20	0.00
	128	23.00	25.50	25.55	29.15	30.50	32.05	33.55	**34.20**	33.25	0.00
MiLoRA	**16**	26.80	28.60	27.55	28.75	31.60	**33.95**	32.25	33.30	29.85	0.00
	128	22.50	25.50	26.40	27.20	29.55	29.30	32.35	32.40	**33.35**	0.00
PiSSA	**16**	29.05	29.90	32.55	31.90	**34.05**	30.75	29.85	29.60	24.35	0.00
	128	27.45	27.70	30.15	30.45	31.45	30.10	**33.55**	31.75	30.75	29.35

Table 12. Performance of **Llama-2-7B** fine-tuned on **code generation** with **rank=8**.

Methods	Batch Size	Learning Rate								
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03
LoRA	**16**	28.30	29.10	30.55	35.30	37.05	**37.65**	36.85	28.75	0.00
	128	27.60	27.25	29.25	29.30	30.85	34.55	35.45	**35.65**	32.20
DoRA	**16**	29.00	29.10	30.30	34.75	34.20	36.40	**37.40**	29.35	0.00
	128	26.75	27.15	29.00	30.30	31.20	35.25	**35.45**	35.15	0.00
Init[AB]	**16**	27.75	29.70	31.10	34.05	**35.40**	35.20	33.05	31.25	0.00
	128	26.10	27.00	27.75	29.00	31.60	35.70	**37.00**	36.70	0.00
MiLoRA	**16**	27.15	28.55	30.55	32.75	34.10	35.90	**36.10**	27.40	0.00
	128	25.70	26.50	26.90	28.90	31.05	31.40	33.30	**34.75**	0.00
PiSSA	**16**	32.10	33.00	34.00	32.10	**34.45**	32.30	27.75	18.55	0.00
	128	29.80	30.05	31.75	32.10	33.55	**35.00**	33.20	32.60	0.00

Table 13. Performance of **Llama-2-7B** fine-tuned on **code generation** tasks with **rank = 32**.

Methods	Batch Size	Learning Rate								
		2.00e-05	3.56e-05	6.32e-05	1.12e-04	2.00e-04	3.56e-04	6.32e-04	1.12e-03	2.00e-03
LoRA	**16**	30.72	31.87	34.23	**37.55**	37.40	36.68	29.20	0.00	0.00
	128	29.37	29.82	31.18	33.40	35.70	36.48	**36.68**	13.05	0.00
DoRA	**16**	30.93	32.50	34.72	36.70	**38.30**	35.82	30.50	0.00	0.00
	128	29.12	30.17	31.22	32.80	36.12	**38.12**	37.50	0.00	0.00
Init[AB]	**16**	30.73	31.48	34.47	36.75	**38.07**	35.82	30.72	0.00	0.00
	128	28.32	30.00	30.15	32.80	35.83	36.97	**38.43**	0.00	0.00
MiLoRA	**16**	29.72	32.18	33.55	36.53	**37.08**	35.95	30.03	0.00	0.00
	128	28.67	29.20	29.72	31.93	34.67	**37.42**	37.07	0.00	0.00
PiSSA	**16**	35.47	**37.35**	35.98	36.08	33.92	28.43	17.53	0.00	0.00
	128	31.90	32.33	34.80	35.42	**36.77**	36.67	34.50	26.90	0.00

Table 14. Performance of **Llama-2-7B** fine-tuned on **code generation** tasks with **rank = 128**.

G. Example Model Responses

We examine the responses of Gemma-3-1B fine-tuned on the mathematical reasoning task using LoRA and PiSSA ($r = 128, B = 16$) under various learning rates. Figure 12 presents a randomly selected testing example from the MATH dataset, with the corresponding model responses organized in Table 15. In this example, it is easily observed that the two PEFT methods operate in distinct effective learning rate regimes. Specifically, LoRA produces correct reasoning paths within the range of 2×10^{-4} to 6.32×10^{-4}, whereas PiSSA achieves accurate results in a slightly lower range of 6.32×10^{-5} to 2×10^{-4}. Moreover, the LLM under PiSSA fine-tuning tends to continue responding reasonably at larger learning rates, as evident when $\eta = 2 \times 10^{-3}$ and 3.56×10^{-3}, where the model produces answers of "6" and "4", while under LoRA fine-tuning, the model diverges to output repetitive gibberish. These qualitative findings correspond to our discussion in Sec. 4.3.1.

Selected MATH Testing Set Input

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:
There are eight boys and six girls who are members of the trumpet section in the band. From the members of the trumpet section, a quintet is being formed. If the quintet must consist of three boys and two girls, how many quintets are possible?

Response: Let's think step by step.

Figure 12. An example input from the MATH dataset used for testing.

Table 15. Qualitative comparison of LoRA and PiSSA reasoning paths across learning rates.

LR (η)	LoRA Response	PiSSA Response
1.12×10^{-5}	We are given that there are 8 boys and 6 girls in the trumpet section. To form a quintet, we need to choose 3 boys and 2 girls. So, we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{2} = \frac{8!}{2!6!} = \frac{8\cdot7}{2\cdot1} = \boxed{16}$ ways. The answer is: 16 ✗ Incorrect	We have 8 boys and 6 girls, so we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!2!} = \frac{8\cdot7}{3\cdot2} = \boxed{14}$ ways. The answer is: 14 ✗ Incorrect
2.00×10^{-5}	We are given that there are 8 boys and 6 girls in the trumpet section. To form a quintet, we need to choose 3 boys and 2 girls. So, the number of quintets that can be formed is $\frac{8}{3} \times \frac{6}{2} = \frac{8\times6}{3\times2} = \frac{48}{6} = \boxed{8}$. The answer is: 8 ✗ Incorrect	We have 8 boys and 6 girls, so we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} = \frac{8\cdot7}{3\cdot2} = 28$ ways. Therefore, there are $\boxed{28}$ quintets possible. The answer is: 28 ✗ Incorrect
3.56×10^{-5}	We have 8 boys and 6 girls, so there are a total of 8 + 6 = 14 members in the trumpet section. To form a quintet, we need to choose 3 boys and 2 girls from the 14 members. The number of quintets that can be formed is the number of ways to choose 3 boys and 2 girls from a group of 14 members. Using the combination formula, this is $\binom{14}{3} = \frac{14!}{3!11!} = \frac{14\times13\times12}{3\times2\times1} = \boxed{105}$. The answer is: 105 ✗ Incorrect	We have 8 boys and 6 girls in the trumpet section. To form a quintet, we need to choose 3 boys and 2 girls from the 8 boys and 6 girls. This can be done in $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} = \frac{8\cdot7}{3\cdot2} = 28$ ways. Therefore, there are $\boxed{28}$ possible quintets. The answer is: 28 ✗ Incorrect

LR (η)	LoRA Response	PiSSA Response
6.32×10^{-5}	We have 8 boys and 6 girls, so we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!4!} = \frac{8 \cdot 7 \cdot 6}{3 \cdot 2 \cdot 1} = \frac{216}{6} = \boxed{36}$. The answer is: 36 ✗ Incorrect	We need to choose 3 boys from the 8 members of the trumpet section, and 2 girls from the 6 members of the trumpet section. The number of ways to choose 3 boys from 8 is $\binom{8}{3} = 56$. The number of ways to choose 2 girls from 6 is $\binom{6}{2} = 15$. Therefore, the total number of quintets is $56 \times 15 = \boxed{840}$. The answer is: 840 ✓ Correct
1.12×10^{-4}	We have 8 boys and 6 girls, so we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!2!} = \frac{8 \cdot 7 \cdot 6}{3 \cdot 2 \cdot 1} = 56$ ways. Therefore, there are $\boxed{56}$ possible quintets. The answer is: 56 ✗ Incorrect	We need to choose 3 boys from the 8 available boys, and 2 girls from the 6 available girls. The number of ways to choose 3 boys from 8 is $\binom{8}{3} = 56$. The number of ways to choose 2 girls from 6 is $\binom{6}{2} = 15$. Therefore, the total number of quintets is $56 \times 15 = \boxed{840}$. The answer is: 840 ✓ Correct
2.00×10^{-4}	We need to choose 3 boys from the 8 available, and 2 girls from the 6 available. This can be done in $\binom{8}{3} = 56$ ways. Similarly, we need to choose 2 girls from the 6 available, and 3 boys from the 3 available. This can be done in $\binom{6}{2} = 15$ ways. Therefore, there are $56 \times 15 = \boxed{840}$ possible quintets. The answer is: 840 ✓ Correct	First, we choose 3 boys from the 8 available boys. This can be done in $\binom{8}{3} = 56$ ways. Next, we choose 2 girls from the 6 available girls. This can be done in $\binom{6}{2} = 15$ ways. Therefore, the total number of quintets is $56 \times 15 = \boxed{840}$. The answer is: 840 ✓ Correct
3.56×10^{-4}	We need to choose 3 boys from the 8 available, and 2 girls from the 6 available. The number of ways to choose 3 boys from 8 is $\binom{8}{3} = 56$. The number of ways to choose 2 girls from 6 is $\binom{6}{2} = 15$. Therefore, the total number of quintets is $56 \times 15 = \boxed{840}$. The answer is: 840 ✓ Correct	For the boys, we have 8 choices for the quintet. For the girls, we have 6 choices for the quintet. So, the total number of quintets is $8 \times 6 = \boxed{48}$. The answer is: 48 ✗ Incorrect
6.32×10^{-4}	We have 8 boys and 6 girls, so we need to choose 3 boys and 2 girls. The number of ways to choose 3 boys from 8 is $\binom{8}{3} = 56$. The number of ways to choose 2 girls from 6 is $\binom{6}{2} = 15$. Therefore, the total number of quintets is $56 \times 15 = \boxed{840}$. The answer is: 840 ✓ Correct	There are 8 boys and 6 girls, so there are a total of 8 + 6 = 14 members in the trumpet section. We need to choose 3 boys from the 8 boys, so there are 8 boys. We also need to choose 2 girls from the 6 girls, so there are 6 girls. Therefore, the total number of quintets is $8 + 3 + 6 = \boxed{17}$. The answer is: 17 ✗ Incorrect

LR (η)	LoRA Response	PiSSA Response
1.12×10^{-3}	From the members of the trumpet section, there are 8 boys and 6 girls. From the members of the quintet, we need to choose 3 boys from the 8 boys and 2 girls from the 6 girls. This can be done in $\binom{8}{3} = \frac{8!}{3!5!} = \frac{8 \cdot 7 \cdot 6}{3 \cdot 2 \cdot 1} = \boxed{35}$ ways. The answer is: 35 ✗ Incorrect	There are 8 boys and 6 girls in the band, so there are a total of $8 + 6 = 14$ members in the band. We want to find the number of quintets that are possible. Since the quintet must consist of three boys and two girls, we need to choose 3 boys out of the 14 members. This can be done in $\binom{14}{3} = 46$ ways. Therefore, there are $46 - 14 = \boxed{32}$ possible quintets. The answer is: 32 ✗ Incorrect
2.00×10^{-3}	Since the number of boys is x, we can set up the equation: Number of boys = x Number of boys with the number of boys with the two boys with the two boys with the two boys with the two boys...(repetitive gibberish) ✗ Incorrect	There are 8 boys and 6 girls, so there are a total of 8 + 6 = 14 people in the band. There are 6 girls and 2 girls, so there are a total of 6 + 2 = 8 people in the band. There are 8 quintet members in the band. Therefore, the number of quintets possible is 14 - 8 = 6. #### 6 The answer is: 6 ✗ Incorrect
3.56×10^{-3}	The total number of students is 1, so the number of people is 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...(repetitive gibberish) ✗ Incorrect	There are 8 boys and 6 girls, so there are a total of $8 + 6 = 14$ possible outcomes. Out of the 14 girls, 6 are girls, so there are $14 - 6 = 18$ possible outcomes. Therefore, there are $18 - 14 = \boxed{4}$ possible outcomes. The answer is: 4 ✗ Incorrect
6.32×10^{-3}	The number of the first circle is 12, 10 = 10. The total number of the number of the number of the number of the number of the number of the number of the first two sides of the first two sides. The area of the first two sides is 10 + 1 = 120 = 100 The total number of the first two sides is 1000. The total number of the number of the first two sides is 1000, so the total number of the first 100000. The answer is 10000000000...(repetitive gibberish) ✗ Incorrect	We know that the total amount of the water is 1. We also know that the total number of eggs is 1. The total number of boys who has not the number of students is 1. We can set up the equation as follows: Number of students who has not the pool + Number of students who has not the number of students who in the pool...(repetitive gibberish) ✗ Incorrect

H. Practical Heuristics for LoRA Hyperparameter Tuning

Based on our extensive hyperparameter tuning experiments, we derive five practical heuristics for tuning LoRA-based methods. While general strategies such as early stopping unpromising learning rate ranges and using sparse grid searches before zooming into promising ranges remain useful, our LoRA-specific guidelines provide additional heuristics on how learning rate interacts with batch size, LoRA rank, training duration, and loss curvature. We summarize them below.

I. Prioritize Learning Rate Tuning. Based on the joint optimization of batch size and learning rate across diverse model–task combinations (Table 1, Appendix Tables 6, 8, 9–14), we suggest that, under limited computational resources, practitioners may prioritize learning rate tuning while fixing the batch size. Importantly, when the batch size is set too large, the best achievable performance under learning rate tuning may start to decay (Appendix Sec. A.4). We therefore suggest using a small or medium batch size as the default choice.

II. Mind Batch Size Scaling. If additional resources are available and practitioners wish to explore different batch sizes, further performance gains are likely to be marginal once the learning rate has been properly tuned for each batch size (as shown in Table 1). In practice, however, practitioners should still account for the scaling relationship between batch size and learning rate (discussed in Sec. 4.3.1), as it provides a useful initial guess for the learning rate when changing the batch size.

III. Select Learning Rate based on Hessian. As described in Sec. 5.2 and Appendix Sec. I.2, the maximum eigenvalue of the loss Hessian can serve as a useful indicator of a variant's relative operating learning rate range compared with vanilla LoRA. In Appendix Sec. I.3, we further show that Hessian trends across different matrix types and Transformer layers are typically consistent, in the sense that they are generally either larger or smaller than those of vanilla LoRA. Critically, a Hessian analysis for a single LoRA adapter requires only around 5–7 minutes on a single RTX A6000. Hence, practitioners with sufficient resources may use Hessian analysis to guide initial learning rate tuning ranges before conducting a large scale search.

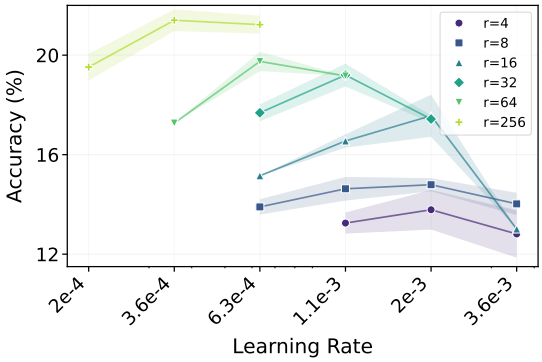
We also note that, based on our broad experiments, a given variant typically exhibits a stable relationship in optimal learning rate relative to LoRA across different settings, in terms of being either higher or lower (e.g., as discussed in Sec. A.1 for Figure 1 and 6). Practitioners may therefore estimate the Hessian once and leverage the known learning-rate range relationships of specific variants across model–task combinations, without re-running the Lanczos algorithm every time they switch to a new setting.

IV. Increase LoRA Ranks. When sufficient effort has been invested in learning rate tuning at a given rank but the resulting downstream performance remains unsatisfactory, increasing the LoRA rank can be a reliable way to further improve performance, as shown in Figures 4 and 8 for various methods. After switching to a higher rank, however, one should still perform learning rate tuning to elicit the best achievable performance at that rank.

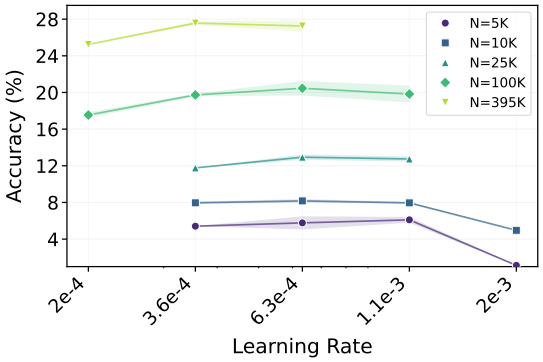
To this end, Tables 9–11 and Tables 12–14 report results for $r = \{8, 32, 128\}$ on Llama-2-7B for math and code, respectively. These results suggest that the optimal learning rate generally decreases as the rank increases. With this, we also visualize vanilla LoRA's performance trends across learning rates for varying adapter ranks behind Figure 4a in Figure 13a in the next page, with results aligning with our practical heuristics. This observation can help practitioners conduct more efficient learning rate tuning across different ranks.

Although recommending larger ranks for better performance may seem straightforward, we highlight that this trend may not be observed in practice if the learning rate is not properly configured for each rank. In fact, we find that many prior LoRA studies fail to demonstrate such a consistent performance improvement trend as the LoRA rank increases, partly because a fixed learning rate setting was applied (e.g., cf. DoRA (Liu et al., 2024a, Figure 5), LoFT (Tastan et al., 2025, Figure 3), GraLoRA (Jung et al., 2026, Table 2)).

V. Prolong Training Duration. If practitioners are computationally available to further pursue higher LoRA performance after increasing the LoRA rank with proper learning rate tuning, we suggest prolonging the training duration as a final step. In particular, one can increase the training duration by using more training samples or training epochs. In both cases, we show in Appendix Sec. A.3 that, with proper learning rate tuning, various LoRA methods typically have the capacity to further improve their performance. Interestingly, the optimal learning rate ranges behave differently from those in *practical heuristic **IV***, where they normally decrease as the LoRA rank increases. Specifically, under the same model–task combination, the optimal learning rate ranges of a specific LoRA method typically remain comparable across different training durations, as shown in Figure 13b in the next page. This suggests that, when extending training duration, practitioners may start from the learning rate range already identified under the shorter training setting, rather than restarting the search from scratch.



(a) Varying LoRA Ranks



(b) Varying Number of Training Samples

Figure 13. Performance across learning rates with (a) varying LoRA ranks and (b) varying number of training samples on mathematical reasoning with Gemma-3-1B on LoRA ($B = 64$).

I. Hessian Computation Details

We select the Lanczos algorithm over the Power Iteration method for the presented eigenvalue problem, as the latter converges to eigenvalues in descending order of magnitude, whereas our focus is on probing the algebraically largest eigenvalue of the Hessian. Sec. I.1 explains implementation details of the Lanczos algorithm, and Sec. I.2 and I.3 provide additional Hessian results for diverse model scales and matrix types, respectively.

I.1. Lanczos Algorithm Implementation Details

Our implementation is built upon several Hessian-related frameworks, such as PyHessian[10] (Yao et al., 2020) and LLM-Hessian[11] (Ilin, 2025), with several modifications to suit our custom scenario. Algorithm 1 summarizes our implementation of the Lanczos Algorithm for estimating $\lambda_{\max}(\mathbf{H})$. We set the Lanczos iterations $m = 100$ and tolerance $\epsilon = 5 \times 10^{-3}$. At each Lanczos iteration step, the Hessian-Vector Product (HVP) is applied to calculate $\mathbf{H}\mathbf{q}_k$ without explicitly forming $\mathbf{H}$ (Algorithm 2).

We strictly ensure that the loss is calculated identically to that in supervised fine-tuning, rendering the resulting curvature information meaningful. In particular, we ensure that (1) the input prompt (i.e., instruction or question) tokens are masked out from the loss calculation, and (2) the loss is averaged over each token instead of each sentence[12]. To ensure computational feasibility, a fixed subset of $N = 500$ training samples from MetaMathQA is selected for loss calculation, and a batch size of $B = 5$ is utilized. Figure 14 validates that this sample size is sufficient for reliably estimating the Hessian of the downstream task. Due to the numerical instability of the Lanczos algorithm in finite precision arithmetic (Cahill et al., 2000), we use Float32 precision for both the base model and adapters and incorporate re-orthogonalization steps (Paige, 1970; Golub et al., 1972).

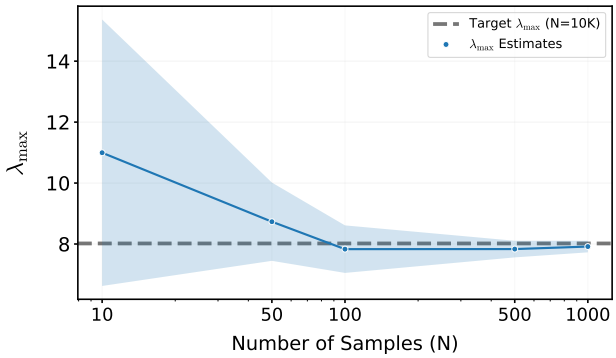


Figure 14. Approximately 500 training samples are sufficient for stable Hessian estimation. The figure reports the estimated $\lambda_{\max}$ for PiSSA on the first Query projection matrix of Qwen3-0.6B ($r = 128$). We track these estimates across varying sample sizes (N) from the MetaMathQA dataset, using $N = 10k$ as the reference baseline. Results represent the mean and standard deviation over 5 randomly selected subsets for each N.

[10] https://github.com/amirgholami/PyHessian

[11] https://github.com/vectozavr/llm-hessian

[12] This is the default way of calculating loss during LLM supervised fine-tuning; see https://github.com/huggingface/transformers/issues/34510

Algorithm 1 Estimating Maximum Eigenvalues of Hessian by Lanczos Iterations

Input: LoRA parameters $\theta = \{B_0, A_0\}$, downstream dataset $\mathcal{D}$, sample size N, iterations m, initial vector $\mathbf{b}$, tolerance ϵ.
Output: Approximation of the maximum eigenvalue $\lambda_{\max}(\mathbf{H})$.
Sampling: Sample a subset $\mathcal{S}$ of size N from $\mathcal{D}$.
Initialization:
Set $\beta_0 = 0$, $\mathbf{q}_0 = \mathbf{0}$, $\lambda_{\text{prev}} = -\infty$.
Normalize initial vector: $\mathbf{q}_1 = \mathbf{b}/\|\mathbf{b}\|_2$.
Lanczos Iteration:
for $k = 1$ **to** m **do**
  Compute Hessian-Vector Product:
  $\mathbf{v} = \text{HVP}(\theta, \mathcal{S}, \mathbf{q}_k)$ // See Algorithm 2
  Compute diagonal element of T:
  $\alpha_k = \mathbf{q}_k^\top \mathbf{v}$
  Orthogonalize (Gram-Schmidt):
  $\mathbf{v} = \mathbf{v} - \beta_{k-1}\mathbf{q}_{k-1} - \alpha_k \mathbf{q}_k$
  Compute off-diagonal element of T:
  $\beta_k = \|\mathbf{v}\|_2$
  Convergence Check:
  Construct symmetric tridiagonal matrix $T_k \in \mathbb{R}^{k\times k}$ using $\alpha_{1:k}, \beta_{1:k-1}$.
  `eig_vals` $\leftarrow$ `torch.linalg.eigvalsh`(T_k)
  $\lambda_{\text{curr}} = \max($`eig_vals`$)$
  if $|\lambda_{\text{curr}} - \lambda_{\text{prev}}| < \epsilon$ **then**
    Return λ_{curr}
  end if
  $\lambda_{\text{prev}} \leftarrow \lambda_{\text{curr}}$
  if $\beta_k \approx 0$ **then**
    Return λ_{curr} // The Krylov subspace is invariant
  end if
  Normalize: $\mathbf{q}_{k+1} = \mathbf{v}/\beta_k$
end for
Return λ_{curr}

Algorithm 2 Hessian-Vector Product (HVP) Calculation

Input: LoRA parameters $\theta = \{B_0, A_0\}$, vector $\mathbf{q}_k$, sample subset $\mathcal{S}$, batch size B.
Output: The Hessian-Vector product $\mathbf{H}\mathbf{q}_k$.
Initialization:
Set accumulator $\mathbf{u} = \mathbf{0}$.
Set total token counter $C_{\text{total}} = 0$.
Batch Processing:
for each mini-batch $\mathcal{B}$ of size B from $\mathcal{S}$ **do**
  Count supervised tokens in batch: $c_{\mathcal{B}} = \text{CountTokens}(\mathcal{B})$.
  $C_{\text{total}} \leftarrow C_{\text{total}} + c_{\mathcal{B}}$.
  Forward Pass for Loss Calculation:
  Compute sum of Cross-Entropy losses over all supervised tokens in $\mathcal{B}$:
  $\mathcal{L}_{\text{batch}}(\theta) = \sum_{(x,y)\in\mathcal{B}} \ell(f(x;\theta), y)$ // `torch.nn.CrossEntropyLoss(sum)`
  Double Backward for HVP:
  $\mathbf{g} = \nabla_\theta \mathcal{L}_{\text{batch}}$
  $s = \mathbf{g}^\top \mathbf{q_k}$
  $\mathbf{h}_{\mathcal{B}} = \nabla_\theta s$ // Implemented via `torch.autograd.functional.vhp`
  Accumulate:
  $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{h}_{\mathcal{B}}$ // Summing up batch-wise contributions
end for
Normalize:
$\mathbf{u} \leftarrow \mathbf{u}/C_{\text{total}}$ // Average over total supervised tokens
Return $\mathbf{u}$

I.2. Hessian Results on Gemma and Llama

As discussed earlier in Sec. 5.2, Figure 5 presents the distributions of the top loss Hessian eigenvalues of LoRA variants relative to vanilla LoRA on Qwen3-0.6B, providing a theoretical explanation for the optimal learning rate trends observed in Figure 1. Analogously, Figure 15 below shows the corresponding distributions on Gemma-3-1B and Llama-2-7B. The associated learning rate tuning results for these two models on MetaMath under rank $r = 128$ are reported earlier in Table 1 and Figure 3a, respectively.

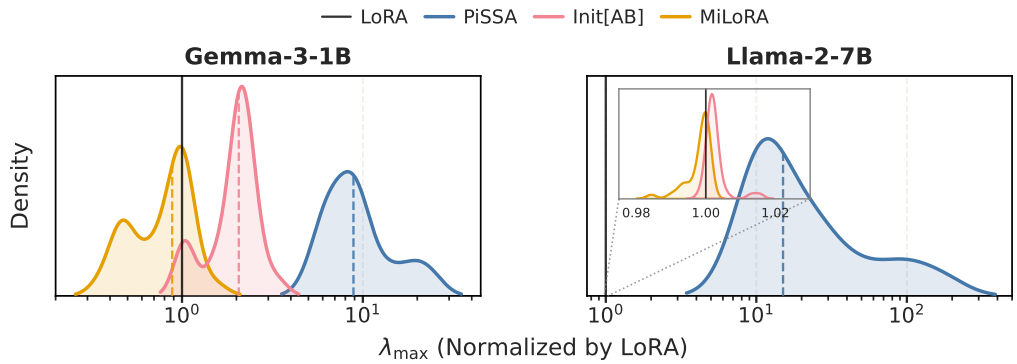


Figure 15. Distributions of the ratios of the top loss Hessian eigenvalues relative to LoRA for Query projection matrices across Transformer layers ($r = 128$). Dashed lines indicate the medians.

It is evident that the Hessian relationship presented here is, again, generally negatively correlated with the optimal learning rate of each method. Specifically, PiSSA exhibits substantially larger $\lambda_{\max}$ on both models, aligning with its requirement for 1.8–2× smaller learning rates in Table 1 and Figure 3a. MiLoRA and Init[AB], on the other hand, have $\lambda_{\max}$ values that largely overlap with those of vanilla LoRA (especially on Llama-2-7B), which also explains their similar optimal learning rate ranges to vanilla LoRA in the prior fine-tuning results. These findings further support the use of relative Hessian magnitude as a useful indicator for explaining the observed performance differences and optimal learning rate ranges across different model scales.

I.3. Detailed $\lambda_{\max}$ Values

Figure 16 presents the detailed $\lambda_{\max}$ values of the Query projection matrix for Qwen across Transformer layers, providing a layer-wise breakdown of the results shown in Figure 5. We observe intriguing patterns in which all methods tend to exhibit high or low values at similar layer locations. For example, at layer 20, $\lambda_{\max} = \{4.7, 8.5, 8.3, 53.8, 297.3, 264.7\}$ for LoRA, Init[AB], MiLoRA, PiSSA, OLoRA, and LoRA-GA, respectively, whereas at layer 26, these values drop to $\{0.2, 0.7, 0.7, 2.3, 12.3, 17.9\}$. However, at any given layer, OLoRA and LoRA-GA consistently exhibit substantially larger $\lambda_{\max}$ than LoRA, by around two orders of magnitude. PiSSA, in contrast, is larger than LoRA by roughly one order of magnitude. Similar trends for the Key projection matrix are presented in Figure 17.

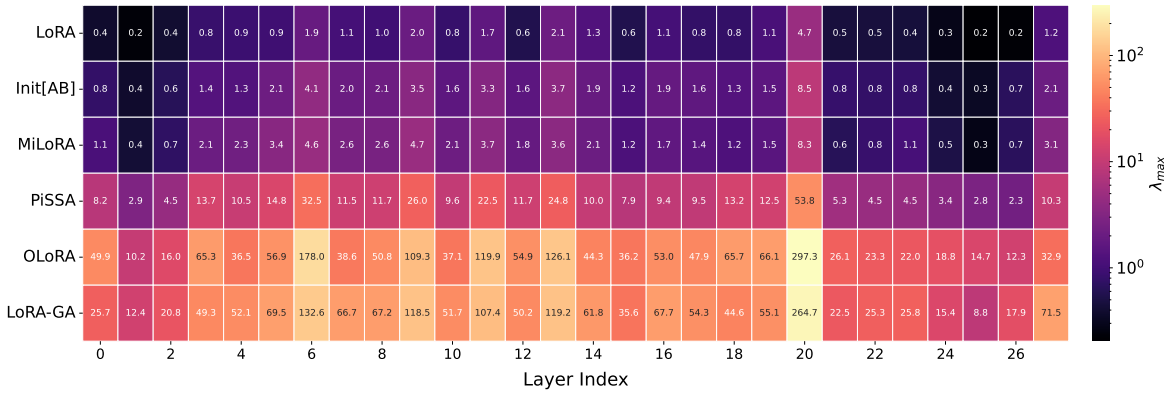


Figure 16. Heatmap of the top eigenvalues of the **Query projection matrix** across Transformer layers, i.e., $\lambda_{\max}^{Q,i}$ for $i = 1, \ldots, L$, for Qwen3-0.6B on MetaMathQA ($r = 128$). All methods exhibit similar distributional patterns across layers, with PiSSA, OLoRA and LoRA-GA consistently exhibiting significantly larger values compared to vanilla LoRA.

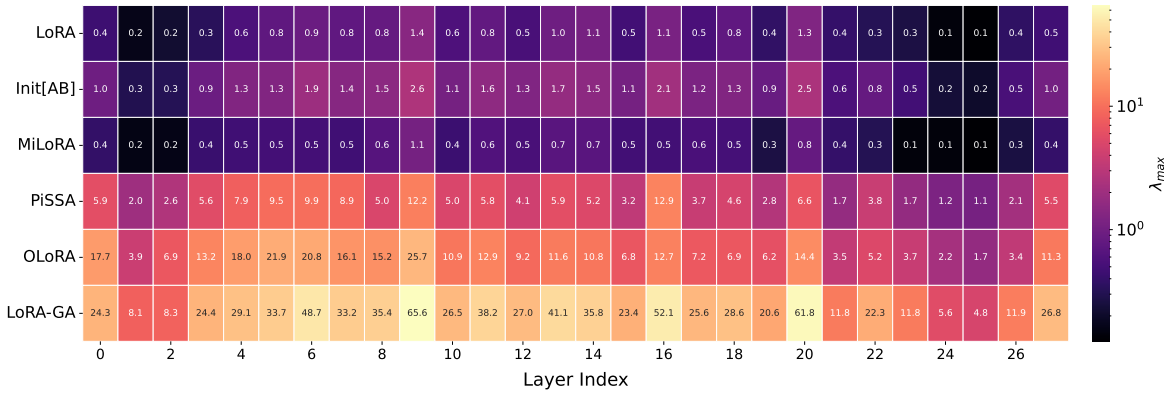


Figure 17. Heatmap of the top eigenvalues of the **Key projection matrix** across Transformer layers, i.e., $\lambda_{\max}^{K,i}$ for $i = 1, \ldots, L$, for Qwen3-0.6B on MetaMathQA ($r = 128$). All methods exhibit similar distributional patterns across layers, with PiSSA, OLoRA and LoRA-GA consistently exhibiting significantly larger values compared to vanilla LoRA.

J. Limitations and Future Work

In this paper, we focused our investigation on decoder-only LLMs from 0.6B to the 13B parameter scale. Hence, the scalability of our findings to larger foundation models remain to be verified. Additionally, the computational costs required for hyperparameter searches precluded an exhaustive search over relatively minor hyperparameters. In particular, while key hyperparameters (learning rate, batch size, rank, training durations) were tuned, other secondary training setups, such as learning rate schedulers, warmup steps, and LoRA adapter placements, remained fixed. It may be possible that fine-grained tuning of these configurations could yield further performance gains or distinct convergence behaviors.

We also highlight that our findings may not extend to untested model architectures (e.g., encoder-only LLMs (Devlin et al., 2019), Vision Transformers (Dosovitskiy, 2020), and Vision-Language Models (Alayrac et al., 2022)) or to all existing advanced LoRA variants. For instance, several methods have originally reported higher peak performance than LoRA under comprehensive learning rate sweeps—such as LoRA-One (Zhang et al., 2025g), which initializes adapters via the SVD of the one-step full gradient, with $\approx 2\%$ performance improvement on Llama (cf. Zhang et al. (2025g) Table 3). Moreover, fine-tuned accuracy on standard benchmarks is not the sole criterion for evaluating PEFT algorithms; specific variants may offer distinct advantages in other dimensions, e.g., mitigating catastrophic forgetting of pretrained knowledge (Biderman et al., 2024; Wang et al., 2024b; Yang et al., 2024a; Zhang et al., 2025c; Xiong & Xie, 2025; Quercia et al., 2026).

While the landscape of LoRA variants continues to expand, our results suggest that vanilla LoRA already suffices as a competitive baseline, potentially indicating that weight-based low-rank adaptation strategies may be approaching saturation. Looking ahead, we posit that further investigation into alternative adaptation mechanisms may unlock new dimensions of efficiency. Examples of such mechanisms include hidden representation fine-tuning (Wu et al., 2024; Yin et al., 2024) and approaches that adapt non-linear functions within layers (Yin et al., 2025). We leave the exploration of these orthogonal paradigms as future work.