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]
[1]*National Taiwan University* [2]*IBM Research* [3]*Academia Sinica*

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, architectural modifications, and optimization adjustments, 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 over learning rate, batch size, rank, and training duration. 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 eigenvalue of the loss Hessian, aligning with classical learning theories. Our code is publicly available at https://github.com/yuang-lee/lr-matters-lora.

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 [1, 2]. 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. Inspired by the low intrinsic dimensionality observed in pretrained models [3], Hu et al. [4] proposed Low-Rank Adaptation (LoRA), hypothesizing that task-specific parameter updates can be well approximated by low-dimensional structures. In particular, LoRA injects pairs of trainable low-rank 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 additional inference latency. Even with its popularity, LoRA has been shown to underperform Full FT on challenging tasks in programming and mathematics [5]. This gap has in turn spurred recent efforts toward advanced LoRA variants [6], with promising performance improvements reported [7–9]. On Llama [10], for example, PiSSA [7] presented around a 10% accuracy improvement on GSM8K [11] by modifying LoRA initialization strategies.

© Y.-A. Lee, C.-Y. Ko, P.-Y. Chen & M.-Y. Yeh.

Yet, the results in a majority of work along this line were obtained with hyperparameters directly inherited from prior studies, or only tuned in a narrow range. To be specific, in Figure 1, we collect 64 LoRA publications to investigate whether their training involved tuning key hyperparameters—namely, learning rate, batch size, and rank. The statistics 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. These findings raise 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 [12, 13]. To address this concern, we select nine LoRA variants spanning ***initialization variants*** (OLoRA [14], PiSSA [7], MiLoRA [15], Init[AB] [16], LoRA-GA [17]), ***architectural modifications*** (DoRA [8], GraLoRA [18], RandLoRA [19]), and ***optimization adjustments*** (LoFT [9]) (details in Appendix B) and conduct a large-scale hyperparameter search over learning rate, batch size, LoRA rank, and training duration, 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 vanilla LoRA.

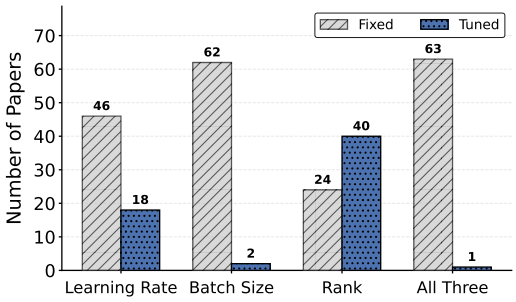


Figure 1: Frequency of advanced LoRA-based PEFT studies, categorized by whether learning rate or batch size tuning was applied and whether comparisons with vanilla LoRA across different ranks were conducted. Refer to Appendix Sec. A for detailed data counts.

By delving into the fundamentals of learning theories [20, 21], we provide an explanation for the importance of tuning learning rates during LoRA fine-tuning and uncover the reasons behind different desirable learning rate ranges among various LoRA methods. Specifically, we demonstrate that PiSSA [7], OLoRA [14], and LoRA-GA [17] 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* (***I–V***) for hyperparameter tuning in LoRA-based methods, which aim to help practitioners with limited computational resources avoid unnecessarily exhaustive hyperparameter searches[1].

2. Learning Rate Matters, Really

2.1. 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 the 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 the convergence rate. 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 [20], the optimal learning rate η^* for efficient learning is intrinsically tied to the curvature of

1. Due to space constraints, we defer discussion on related work to Appendix Sec. C

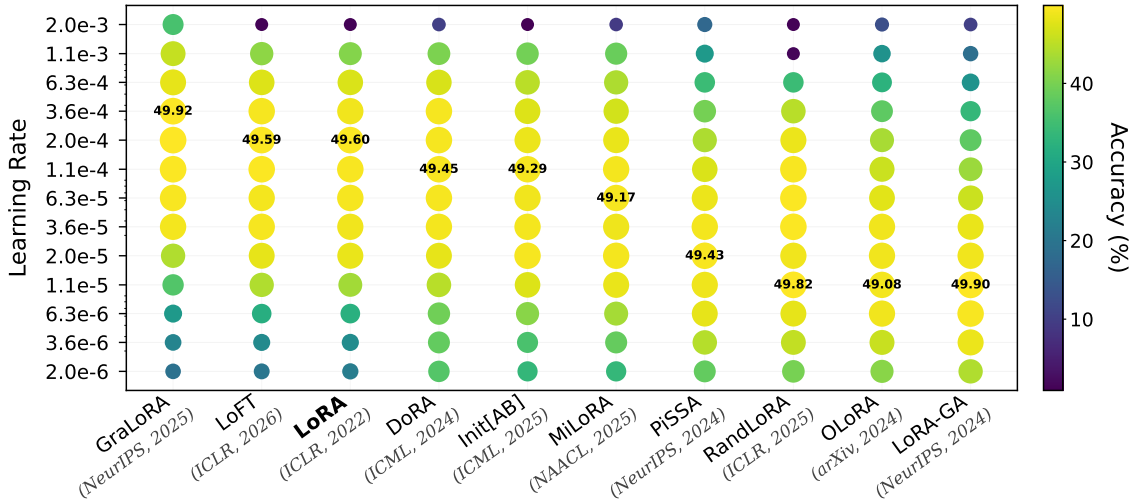


Figure 2: Performance of Qwen3-0.6B fine-tuned on mathematical reasoning tasks under rank 128 and batch size 64 across learning rates. Different methods reach a similar performance level once the learning rate is properly tuned. Each point is averaged over three training runs. We annotate the peak accuracy 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. 2.3 and Appendix Sec. D.

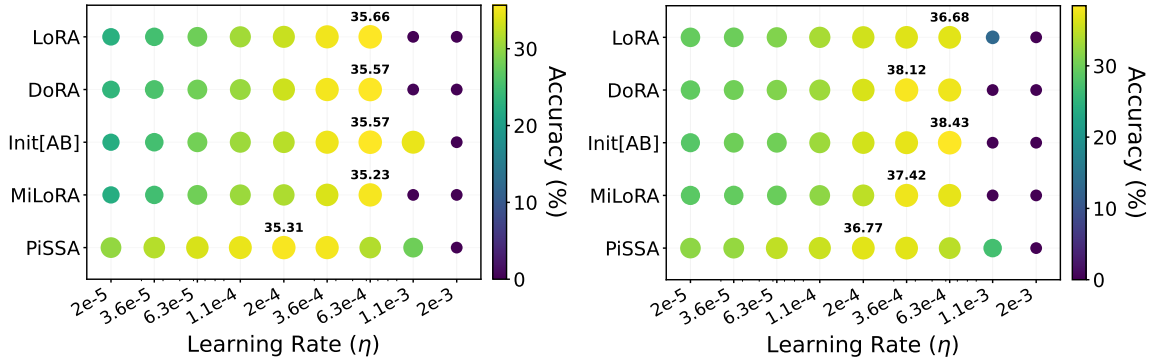
the loss landscape at $\boldsymbol{\theta}$, typically scaling inversely with the Hessian's maximum eigenvalue:

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

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 optimization 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.

2.2. Experimental Setup

Since model choices, hyperparameters, and dataset partitioning vary across papers, we establish a unified experimental framework that accommodates all methods fairly. Specifically, we consider four LLMs spanning diverse scales: Qwen3-0.6B [22], Gemma-3-1B [23], Llama-2-7B, and Llama-2-13B [10]. We train these models on four canonical tasks: commonsense reasoning, mathematical reasoning, code generation, and instruction following. For hyperparameters, 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 tuned uniformly on a logarithmic scale from 10^{-6} to 10^{-3}, 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. More implementation details are provided in Appendix Sec. E.



(a) Mathematical Reasoning

(b) Code Generation

Figure 3: Performance of Llama-2-7B on mathematical reasoning and code generation tasks across learning rates ($r = 128$, $B = 128$). Results scaling up to 13B are provided in Appendix Figure 6.

2.3. Results and Observations

In Figure 2, we fine-tune Qwen3 [22] 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 0.84% range. Moreover, different methods operate within disparate learning rate ranges (e.g., a 10× difference between PiSSA and LoRA in Figure 2), suggesting that success under a single training configuration cannot be taken as evidence of robust and reliable improvements. This phenomenon is not isolated; we consistently observe such performance parity across four task types over diverse model–task combinations (Figure 3 and Appendix Sec. D.1), LoRA ranks (Figure 7 and Appendix Sec. D.2), and training durations (Appendix Sec. D.3).

Note that in Appendix Sec. D.4, 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 the best performance in both LoRA and its variants, consistent with early findings for neural networks [13]. Moreover, we observe that the optimal learning rate scales proportionally with batch size, aligning with the "scaling rule" established in SGD literature [24, 25]. This offers several *practical heuristics* for LoRA hyperparameter tuning, where we discuss in Appendix Sec. F. More numerical results and example model responses are provided in Appendix G, Sec. H, respectively.

3. Understanding the Optimal Learning Rate via Hessian Analysis

3.1. 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, commonly referred to as sharpness [26–28]. This metric is closely linked to the optimal learning rate, a connection that originates from the Gauss-Newton method for convex optimization and was further elucidated by LeCun et al. [20] 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. [21] identified a "catapult" learning regime characterized by $2/\lambda_{\max} \leq \eta^* \leq 12/\lambda_{\max}$, in which modern architectures achieve optimal performance.

3.2. Sharpness Analysis in LoRA

For our LoRA fine-tuning problem, we leverage the downstream MetaMathQA [29] dataset to compute the Hessian matrix of the loss function and focus exclusively on the trainable LoRA parameters [30–32]. Instead of concatenating LoRA parameters across all layers, we follow standard LLM practices to estimate $\lambda_{\max}$ in a block-wise manner [33–35] at the initialization point. Formally, we calculate the layer-wise metric as $\lambda_{\max}^l = \lambda_{\max}(\mathbf{H}^l)$, where $\mathbf{H}^l$ represents the Hessian corresponding to parameters $\theta^l = \{B_0^l, A_0^l\}$, with l indexing matrix types and Transformer layers. The Lanczos algorithm [36] 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 optimization 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 $\mathbf{H}_t^{Q,i}$, where t indexes different LoRA methods. We further denote their corresponding maximum eigenvalues by $\lambda_{\max,t}^{Q,i}$. Then, we normalize the maximum eigenvalues from LoRA initialization variants by that from LoRA, and plot the distribution across layers in Figure 4, i.e. $\lambda_{\max,t}^{Q,i}/\lambda_{\max,\text{LoRA}}^{Q,i}$ 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 2. 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 2, 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.

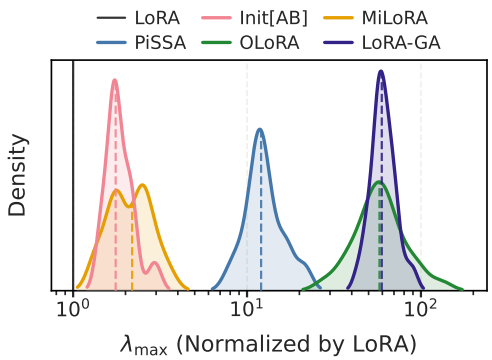


Figure 4: Distributions of the ratios of the top loss Hessian eigenvalues relative to LoRA for Query projection matrices across Transformer layers on Qwen3 ($r = 128$). Dashed lines indicate the medians.

4. 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 ten LoRA PEFT methods. Based on the comprehensive hyperparameter experiments, we conclude that **improper learning rates give a false sense of LoRA advancements**. From a high-dimensional learning dynamics perspective, our results contribute a concrete instantiation of loss landscape geometry governing practical LoRA fine-tuning: the sharpness of the loss landscape at initialization, which varies systematically across LoRA variants, reliably explains the differing optimal learning rate regimes. We acknowledge that this paper is subject to several limitations, primarily due to computational constraints, which we discuss in detail in Appendix J.

References

[1] DM Anisuzzaman, Jeffrey G Malins, Paul A Friedman, and Zachi I Attia. Fine-tuning large language models for specialized use cases. *Mayo Clinic Proceedings: Digital Health*, 3(1): 100184, 2025.

[2] Prudence Djagba and Abdelkader Y Saley. Exploring large language models for financial applications: Techniques, performance, and challenges with finma. *arXiv preprint arXiv:2510.05151*, 2025.

[3] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

[4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[5] Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

[6] Mengyao Zhu and Phuc Huu Nguyen. A survey of lora algorithm variations for language models. In *International Conference on Applications of Natural Language to Information Systems*, pages 275–290. Springer, 2025.

[7] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

[8] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

[9] Nurbek Tastan, Stefanos Laskaridis, Martin Takác, Karthik Nandakumar, and Samuel Horváth. Loft: Low-rank adaptation that behaves like full fine-tuning. *arXiv preprint arXiv:2505.21289*, 2025.

[10] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

[12] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

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

[14] Kerim Büyükakyüz. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.

[15] Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. *arXiv preprint arXiv:2406.09044*, 2024.

[16] Shiwei Li, Xiandi Luo, Xing Tang, Haozhao Wang, Hao Chen, Weihong Luo, Yuhua Li, Xiuqiang He, and Ruixuan Li. Beyond zero initialization: Investigating the impact of non-zero initialization on lora fine-tuning dynamics. *arXiv preprint arXiv:2505.23194*, 2025.

[17] Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

[18] Yeonjoon Jung, Daehyun Ahn, Hyungjun Kim, Taesu Kim, and Eunhyeok Park. 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.

[19] Paul Albert, Frederic Z Zhang, Hemanth Saratchandran, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. Randlora: Full-rank parameter-efficient fine-tuning of large models. *arXiv preprint arXiv:2502.00987*, 2025.

[20] Yann LeCun, Patrice Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. *Advances in neural information processing systems*, 5, 1992.

[21] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

[22] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[23] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[24] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[25] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

[26] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

[27] Kaifeng Lyu, Zhiyuan Li, and Sanjeev Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. *Advances in Neural Information Processing Systems*, 35:34689–34708, 2022.

[28] Haocheng Luo, Tuan Truong, Tung Pham, Mehrtash Harandi, Dinh Phung, and Trung Le. Explicit eigenvalue regularization improves sharpness-aware minimization. *Advances in Neural Information Processing Systems*, 37:4424–4453, 2024.

[29] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[30] Adam X Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. Bayesian low-rank adaptation for large language models. *arXiv preprint arXiv:2308.13111*, 2023.

[31] Yanjun Zhao, Sizhe Dang, Haishan Ye, Guang Dai, Yi Qian, and Ivor W Tsang. Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer. *arXiv preprint arXiv:2402.15173*, 2024.

[32] Xin Yu, Cong Xie, Ziyu Zhao, Tiantian Fan, Lingzhou Xue, and Zhi Zhang. Prunedlora: Robust gradient-based structured pruning for low-rank adaptation in fine-tuning, 2025. URL https://arxiv.org/abs/2510.00192.

[33] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37:131786–131823, 2024.

[34] Jinbo Wang, Mingze Wang, Zhanpeng Zhou, Junchi Yan, Lei Wu, et al. The sharpness disparity principle in transformers for accelerating language model pre-training. *arXiv preprint arXiv:2502.19002*, 2025.

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

[36] Cornelius Lanczos. 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.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171–4186, 2019.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

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

[40] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

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

[42] Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, et al. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*, 2023.

[43] Ziqing Xu, Hancheng Min, Lachlan Ewen MacDonald, Jinqi Luo, Salma Tarmoun, Enrique Mallada, and René Vidal. Understanding the learning dynamics of lora: A gradient flow perspective on low-rank adaptation in matrix factorization. *arXiv preprint arXiv:2503.06982*, 2025.

[44] Jia-Chen Zhang, Yu-Jie Xiong, Chun-Ming Xia, Dong-Hai Zhu, and Hong-Jian Zhan. Lora2: Multi-scale low-rank approximations for fine-tuning large language models. *Neurocomputing*, 650:130859, 2025.

[45] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter-efficient tuning with kronecker adapter. In *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pages 49–65. Springer, 2025.

[46] Shen Yuan, Haotian Liu, and Hongteng Xu. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. *Advances in Neural Information Processing Systems*, 37:113484–113518, 2024.

[47] Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

[48] Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10658–10671, 2023.

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

[50] Mao-Lin Luo, Zi-Hao Zhou, Yi-Lin Zhang, Yuanyu Wan, Min-Ling Zhang, and Tong Wei. KeeploRA: Continual learning with residual gradient adaptation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=T3Vc5fkTzV.

[51] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

[52] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

[53] Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

[54] Yang Li, Shaobo Han, and Shihao Ji. Vb-lora: Extreme parameter efficient fine-tuning with vector banks. *Advances in Neural Information Processing Systems*, 37:16724–16751, 2024.

[55] haonan he, Peng Ye, Yuchen Ren, 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.

[56] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

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

[58] Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*, 2023.

[59] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

[60] Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

[61] Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, et al. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024.

[62] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*, 2023.

[63] Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 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), pages 8694–8705, 2024.

[64] Yifan Yang, Jiajun Zhou, Ngai Wong, and Zheng Zhang. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. *arXiv preprint arXiv:2402.11417*, 2024.

[65] Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113*, 2024.

[66] Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*, 2024.

[67] Mahdi Nikdan, Soroush Tabesh, Elvir Crnčević, and Dan Alistarh. Rosa: Accurate parameter-efficient fine-tuning via robust adaptation. *arXiv preprint arXiv:2401.04679*, 2024.

[68] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

[69] Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. *arXiv preprint arXiv:2402.02347*, 2024.

[70] Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*, 2024.

[71] Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

[72] Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Reslora: Identity residual mapping in low-rank adaption. *arXiv preprint arXiv:2402.18039*, 2024.

[73] Xiangdi Meng, Damai Dai, Weiyao Luo, Zhe Yang, Shaoxiang Wu, Xiaochen Wang, Peiyi Wang, Qingxiu Dong, Liang Chen, and Zhifang Sui. Periodiclora: Breaking the low-rank bottleneck in lora optimization. *arXiv preprint arXiv:2402.16141*, 2024.

[74] Seyedarmin Azizi, Souvik Kundu, and Massoud Pedram. Lamda: Large model fine-tuning via spectrally decomposed low-dimensional adaptation. *arXiv preprint arXiv:2406.12832*, 2024.

[75] Yibo Yang, Xiaojie Li, Zhongzhu Zhou, Shuaiwen Song, Jianlong Wu, Liqiang Nie, and Bernard Ghanem. 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, 2024.

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

[77] Hongyun Zhou, Xiangyu Lu, Wang Xu, Conghui Zhu, Tiejun Zhao, and Muyun Yang. Loradrop: Efficient lora parameter pruning based on output evaluation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5530–5543, 2025.

[78] Qingchen Wang and Shengyu Shen. Activation-guided low-rank parameter adaptation for efficient model fine-tuning. *IEEE Access*, 2025.

[79] Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Lora-pro: Are low-rank adapters properly optimized? *arXiv preprint arXiv:2407.18242*, 2024.

[80] Chongjie Si, Zhiyi Shi, Shifan Zhang, Xiaokang Yang, Hanspeter Pfister, and Wei Shen. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations*, 2024.

[81] Fan Wang, Juyong Jiang, Chansung Park, Sunghun Kim, and Jing Tang. Kasa: Knowledge-aware singular-value adaptation of large language models. *arXiv preprint arXiv:2412.06071*, 2024.

[82] Massimo Bini, Leander Girrbach, and Zeynep Akata. Decoupling angles and strength in low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[83] Qiushi Huang, Tom Ko, Zhan Zhuang, Lilian Tang, and Yu Zhang. Hira: Parameter-efficient hadamard high-rank adaptation for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[84] Jiayang Yu, Yihang Zhang, Bin Wang, Peiqin Lin, Yongkang Liu, and Shi Feng. Ssmlora: Enhancing low-rank adaptation with state space model. *arXiv preprint arXiv:2502.04958*, 2025.

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

[86] Yuanhe Zhang, Fanghui Liu, and Yudong Chen. Lora-one: One-step full gradient could suffice for fine-tuning large language models, provably and efficiently. *arXiv preprint arXiv:2502.01235*, 2025.

[87] Yibo Zhong, Jinman Zhao, and Yao Zhou. Low-rank interconnected adaptation across layers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 17005–17029, 2025.

[88] Aochuan Chen, Jiashun Cheng, Zijing Liu, Ziqi Gao, Fugee Tsung, Yu Li, and Jia Li. Parameter-efficient fine-tuning via circular convolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2004–2019, 2025.

[89] Chenhao Ding, Jiangyang Li, Songlin Dong, Xinyuan Gao, Yuhang He, and Yihong Gong. Sulora: Subspace low-rank adaptation for parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 5334–5349, 2025.

[90] Peijia Qin, Ruiyi Zhang, and Pengtao Xie. 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.

[91] Yiding Wang, Fanxu Meng, Xuefeng Zhang, Fan Jiang, Pingzhi Tang, and Muhan Zhang. Hd-pissa: High-rank distributed orthogonal adaptation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6526–6539, 2025.

[92] Xujia Wang, Yunjia Qi, and Bin Xu. Losia: Efficient high-rank fine-tuning via subnet localization and optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6707–6726, 2025.

[93] Hao Zhang, Bo Huang, Zhenjia Li, Xi Xiao, Hui Yi Leong, Zumeng Zhang, Xinwei Long, Tianyang Wang, and Hao Xu. Sensitivity-lora: Low-load sensitivity-based fine-tuning for large language models. *arXiv preprint arXiv:2509.09119*, 2025.

[94] Xin Zhang, Guang-Ze Chen, Shuzhen Li, Zhulin Liu, CL Philip Chen, and Tong Zhang. An orthogonal high-rank adaptation for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18826–18844, 2025.

[95] Fabian Paischer, Lukas Hauzenberger, Thomas Schmied, Benedikt Alkin, Marc Peter Deisenroth, and Sepp Hochreiter. Parameter efficient fine-tuning via explained variance adaptation. *arXiv preprint arXiv:2410.07170*, 2024.

[96] Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. Aurora: Breaking low-rank bottleneck of lora with nonlinear mapping. *arXiv preprint arXiv:2505.18738*, 2025.

[97] Heming Zou, Yunliang Zang, Wutong Xu, Yao Zhu, and Xiangyang Ji. Flylora: Boosting task decoupling and parameter efficiency via implicit rank-wise mixture-of-experts. *arXiv preprint arXiv:2510.08396*, 2025.

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

[99] Qingyue Zhang, Chang Chu, Tianren Peng, Qi Li, Xiangyang Luo, Zhihao Jiang, and Shao-Lun Huang. Lora-da: Data-aware initialization for low-rank adaptation via asymptotic analysis. *arXiv preprint arXiv:2510.24561*, 2025.

[100] Dongha Lee, Jinhee Park, Minjun Kim, and Junseok Kwon. Abm-lora: Activation boundary matching for fast convergence in low-rank adaptation. *arXiv preprint arXiv:2511.19145*, 2025.

[101] Muqing Liu, Chongjie Si, and Yuheng Jia. Flexlora: Entropy-guided flexible low-rank adaptation. *arXiv preprint arXiv:2601.22905*, 2026.

[102] Yize Wu, KE GAO, Ling Li, and Yanjun Wu. 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.

[103] Jingqi Ye, haonan he, Minglei Li, Fujun Han, Tao Chen, and Peng Ye. 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.

[104] Neeraj Gangwar, Rishabh Deshmukh, Michael Shavlovsky, Hancao Li, Vivek Mittal, Lexing Ying, and Nickvash Kani. Giva: Gradient-informed bases for vector-based adaptation. *arXiv preprint arXiv:2604.21901*, 2026.

[105] Yibo Zhong, Haoxiang Jiang, Lincan Li, Ryumei Nakada, Tianci Liu, Linjun Zhang, Huaxiu Yao, and Haoyu Wang. 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*, pages 2054–2065, 2026.

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

[107] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

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

[109] David Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor. 2018.

[110] Zachary C Lipton and Jacob Steinhardt. 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.

[111] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[112] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[113] Yu-Chen Lin, Si-An Chen, Jie-Jyun Liu, and Chih-Jen Lin. Linear classifier: An often-forgotten baseline for text classification. *arXiv preprint arXiv:2306.07111*, 2023.

[114] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[115] Tillmann Rheude, Roland Eils, and Benjamin Wild. Fusion or confusion? multimodal complexity is not all you need. *arXiv preprint arXiv:2512.22991*, 2025.

[116] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[117] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[118] Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Can classic GNNs be strong baselines for graph-level tasks? simple architectures meet excellence. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ZH7YgIZ3DF.

[119] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.

[120] Joseph A Konstan and Gediminas Adomavicius. Toward identification and adoption of best practices in algorithmic recommender systems research. In *Proceedings of the international workshop on Reproducibility and replication in recommender systems evaluation*, pages 23–28, 2013.

[121] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*, pages 101–109, 2019.

[122] Michael Benigni, Maurizio Ferrari Dacrema, and Dietmar Jannach. Diffusion recommender models and the illusion of progress: A concerning study of reproducibility and a conceptual mismatch. *ACM Transactions on Recommender Systems*, 4(3):1–69, 2026.

[123] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision*, pages 681–699. Springer, 2020.

[124] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[125] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[126] Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and François Fleuret. Optimizer benchmarking needs to account for hyperparameter tuning. In *International conference on machine learning*, pages 9036–9045. PMLR, 2020.

[127] Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2021.

[128] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pages 9104–9149. PMLR, 2023.

[129] Kian Ahrabian, Xihui Lin, Barun Patra, Vishrav Chaudhary, Alon Benhaim, Jay Pujara, and Xia Song. A practical analysis of human alignment with* po. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8013–8021, 2025.

[130] Oğuz Kağan Hitit, Leander Girrbach, and Zeynep Akata. A systematic study of model merging techniques in large language models. *arXiv preprint arXiv:2511.21437*, 2025.

[131] Pau de Jorge, César Roberto de Souza, Björn Michele, Mert Bülent Sarıyıldız, Philippe Weinzaepfel, Florent Perronnin, Diane Larlus, and Yannis Kalantidis. Task alignment: A simple and effective proxy for model merging in computer vision. *arXiv preprint arXiv:2604.12935*, 2026.

[132] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[133] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple 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)*, pages 1–9, 2022.

[134] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[135] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 5254–5276, 2023.

[136] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024.

[137] Johanna Männistö, Joseph Attieh, and Jörg Tiedemann. 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)*, pages 390–396, 2025.

[138] Robert Belanec, Branislav Pecher, Ivan Srba, and Maria Bielikova. 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)*, pages 3035–3054, 2026.

[139] Lingling Xu, Haoran Xie, S Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

[140] Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. Improving lora in privacy-preserving federated learning. *arXiv preprint arXiv:2403.12313*, 2024.

[141] Grigory Malinovsky, Umberto Michieli, Hasan Abed Al Kader Hammoud, Taha Ceritli, Hayder Elesedy, Mete Ozay, and Peter Richtárik. Randomized asymmetric chain of lora: The first meaningful theoretical framework for low-rank adaptation. *arXiv preprint arXiv:2410.08305*, 2024.

[142] Xu-Hui Liu, Yali Du, Jun Wang, and Yang Yu. On the optimization landscape of low rank adaptation methods for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[143] Nan Chen, Soledad Villar, and Soufiane Hayou. Learning rate scaling across lora ranks and transfer to full finetuning. *arXiv preprint arXiv:2602.06204*, 2026.

[144] Chuyan Zhang, Kefan Wang, and Yun Gu. Beyond low-rank tuning: Model prior-guided rank allocation for effective transfer in low-data and large-gap regimes. *arXiv preprint arXiv:2507.00327*, 2025.

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

[146] Yang Lin, Xinyu Ma, Xu Chu, Yujie Jin, Zhibang Yang, Yasha Wang, and Hong Mei. Lora dropout as a sparsity regularizer for overfitting control. *arXiv preprint arXiv:2404.09610*, 2024.

[147] Sheng Wang, Liheng Chen, Jiyue Jiang, Boyang Xue, Lingpeng Kong, and Chuan Wu. Lora meets dropout under a unified framework. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1995–2008, 2024.

[148] Vlad Fomenko, Han Yu, Jongho Lee, Stanley Hsieh, and Weizhu Chen. A note on lora. *arXiv preprint arXiv:2404.05086*, 2024.

[149] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Plop: Precise lora placement for efficient fine-tuning of large models. *arXiv preprint arXiv:2506.20629*, 2025.

[150] John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

[151] Christophe Tribes, Sacha Benarroch-Lelong, Peng Lu, and Ivan Kobyzev. Hyperparameter optimization for large language model instruction-tuning. *arXiv preprint arXiv:2312.00949*, 2023.

[152] Michael Oliver and Guan Wang. Crafting efficient fine-tuning strategies for large language models. *arXiv preprint arXiv:2407.13906*, 2024.

[153] Ayan Sengupta, Vaibhav Seth, Arinjay Pathak, Aastha Verma, Natraj Raman, Sriram Gopalakrishnan, Niladri Chatterjee, and Tanmoy Chakraborty. Robust and efficient fine-tuning of llms with bayesian reparameterization of low-rank adaptation. *arXiv preprint arXiv:2411.04358*, 2024.

[154] Baek Seong-Eun, Lee Jung-Mok, Kim Sung-Bin, and Tae-Hyun Oh. Efficient hyper-parameter search for lora via language-aided bayesian optimization. *arXiv preprint arXiv:2602.11171*, 2026.

[155] Minghao Yan, Zhuang Wang, Zhen Jia, Shivaram Venkataraman, and Yida Wang. Plora: Efficient lora hyperparameter tuning for large models. *arXiv preprint arXiv:2508.02932*, 2025.

[156] Jingwei Zuo, Xinze Feng, Zien Liu, Kaijian Wang, Fanjiang Ye, Ye Cao, Zhuang Wang, and Yuke Wang. Alto: Adaptive lora tuning and orchestration for heterogeneous lora training workloads. *arXiv preprint arXiv:2604.05426*, 2026.

[157] Zicheng Zhang, Haoran Li, Yifeng Zhang, Guoqiang Gong, Jiaxing Wang, Pengzhang Liu, Qixia Jiang, and Junxing Hu. The primacy of magnitude in low-rank adaptation. *arXiv preprint arXiv:2507.06558*, 2025.

[158] Haonan He, Jingqi Ye, Minglei Li, Zhengbo Wang, Tao Chen, Lei Bai, and Peng Ye. A unified study of lora variants: Taxonomy, review, codebase, and empirical evaluation. *arXiv preprint arXiv:2601.22708*, 2026.

[159] Sangyoon Lee and Jaeho Lee. Beware of the batch size: Hyperparameter bias in evaluating lora. *arXiv preprint arXiv:2602.09492*, 2026.

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

[161] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[162] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[163] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[164] Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

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

[166] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[167] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[168] Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 4(1), 2019.

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

[170] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 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*, pages 3505–3506, 2020.

[171] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[172] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

[173] Chenghao Fan, Zhenyi Lu, Sichen Liu, Chengfeng Gu, Xiaoye Qu, Wei Wei, and Yu Cheng. Make lora great again: Boosting lora with adaptive singular values and mixture-of-experts optimization alignment. *arXiv preprint arXiv:2502.16894*, 2025.

[174] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pages 581–590. IEEE, 2020.

[175] Eamonn Cahill, Alan Irving, Christopher Johnston, James Sexton, Ukqcd Collaboration, et al. Numerical stability of lanczos methods. *Nuclear Physics B-Proceedings Supplements*, 83: 825–827, 2000.

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

[177] Gene H Golub, Richard R Underwood, and James H Wilkinson. The lanczos algorithm for the symmetric ax= λ bx problem., 1972.

[178] Juzheng Zhang, Jiacheng You, Ashwinee Panda, and Tom Goldstein. 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*, 2025. URL https://openreview.net/forum?id=aGOQYJfz6H.

[179] Yifeng Xiong and Xiaohui Xie. Oplora: Orthogonal projection lora prevents catastrophic forgetting during parameter-efficient fine-tuning. *arXiv preprint arXiv:2510.13003*, 2025.

[180] Alessio Quercia, Arya Bangun, Ira Assent, and Hanno Scharr. Least but not last: Fine-tuning intermediate principal components for better performance-forgetting trade-offs. *arXiv preprint arXiv:2602.03493*, 2026.

[181] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024.

[182] Fangcong Yin, Xi Ye, and Greg Durrett. Lofit: Localized fine-tuning on llm representations. *Advances in Neural Information Processing Systems*, 37:9474–9506, 2024.

[183] Bo Yin, Xingyi Yang, and Xinchao Wang. Don't forget the nonlinearity: Unlocking activation functions in efficient fine-tuning. *arXiv preprint arXiv:2509.13240*, 2025.

Contents

A Comprehensive Study of Hyperparameters in Prior Work **22**
A.1 Survey Criteria . . . 22
A.2 Comprehensive List of Papers . . . 23

B Details of Selected LoRA Variants **27**

C Related Work **28**
C.1 Systematic Empirical Re-evaluation of Prior Claims . . . 28
C.2 LoRA Hyperparameter Tuning . . . 28

D Additional Experiments **29**
D.1 Learning Rate Tuning on More Model–task Combinations . . . 29
D.2 Varying Adapter Ranks . . . 30
D.3 Varying Training Duration . . . 32
D.4 Joint Optimization of Learning Rate and Batch Size . . . 33
D.5 Scaling Batch Size to 512 for LoRA and PiSSA . . . 33
D.6 Fine-tuning on Instruction Following Tasks for LoRA and DoRA . . . 34

E Fine-tuning Implementation Details **34**
E.1 Models . . . 34
E.2 Dataset Setup . . . 35
E.3 Training Hyperparameter Search Ranges . . . 35
E.4 Fixed Training Hyperparameters . . . 35
E.5 Data, Code, Libraries, and Hardware . . . 36
E.6 On LoRA Scaling Factor . . . 37

F Practical Heuristics for LoRA Hyperparameter Tuning **38**

G Details of Hyperparameter Search Results **40**
G.1 Qwen3-0.6B . . . 40
G.2 Gemma-3-1B . . . 41
G.3 Llama-2-7B . . . 41

H Example Model Responses **46**

I Hessian Computation Details **50**
I.1 Lanczos Algorithm Implementation Details . . . 50
I.2 Hessian Results on Gemma and Llama . . . 54
I.3 Detailed $\lambda_{\max}$ Values . . . 54

J Limitations and Future Work **55**

Appendix A. Comprehensive Study of Hyperparameters in Prior Work

A.1. Survey Criteria

To generate the statistics presented in Figure 1, 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 [37], encoder-decoder [38] architectures, Vision Transformers [39], and Vision-Language Models [40], as these lie outside the scope of this work. Consequently, papers lacking experiments on decoder-only LLMs are excluded from our statistics (e.g., [41–46]). Moreover, we exclude papers focusing on objectives other than standard PEFT efficiency, such as parameter-efficient pretraining [47], continual learning [48–50], and quantization [51, 52].

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 [17] (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., [46, 53–55]).

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 [56]). 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.

A.2. Comprehensive List of Papers

A comprehensive list of the reviewed papers is detailed in Table 1. 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 1: 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 [57]	2022-10	2023-05	EACL	GPT-2 Medium	NLG	✗	✗	✓
GLoRA [58]	2023-06	–	arXiv	Llama-1-7B Llama-2-7B	NLG	✗	✗	✗
LoRA-FA [59]	2023-08	–	arXiv	Llama-1-7B Llama-2-7B	Commonsense	✗	✗	✗
Laplace-LoRA [30]	2023-08	2024-05	ICLR	Llama-1-7B Llama-2-7B	Commonsense	✗	✗	✗
VeRA [60]	2023-10	2024-05	ICLR	GPT-2 Medium/Large Llama-1-7B/13B Llama-2-7B/13B	NLG Instruction Following	✓	✗	✓
BoFT [42]	2023-11	2024-05	ICLR	Llama-2-7B	Instruction Following Math	✗	✗	✓
MoRA [61]	2024-05	–	arXiv	Llama-2-7B/13B	UUID Math Instruction Following	✓	✗	✓
Delta-LoRA [62]	2023-09	–	arXiv	GPT-2 Medium	NLG	✗	✗	✗
Tied-LoRA [63]	2023-11	2024-06	NAACL	GPT-2B-001 Llama-2-7B	NLG Commonsense Math	✗	✗	✓
LoRETTA [64]	2024-02	2024-06	NAACL	Llama-2-7B/13B/70B	NLG GLUE	✗	✗	✗
AutoLoRA [65]	2024-03	2024-06	NAACL	GPT-2 Medium	NLG	✗	✗	✗
ALoRA [66]	2024-05	2024-06	NAACL	GPT2-Large Llama-2-7B	NLG GLUE Instruction Following	✗	✗	✓
RoSA [67]	2024-01	2024-07	ICML	Llama-2-7B	NLG Math Code Instruction Following	✓	✗	✗
LoRA+ [68]	2024-02	2024-07	ICML	GPT-2 Llama-1-7B	GLUE Instruction Following	✓	✗	✗

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
scaled AdamW [69]	2024-02	2024-07	ICML	GPT-2 Medium Mistral-7B-V0.1	NLG GLUE	✓	✗	✓
DoRA [8]	2024-02	2024-07	ICML	Llama-1-7B/13B Llama-2-7B Llama-3-8B	Commonsense	✗	✗	✓
FLORA [70]	2024-02	2024-07	ICML	GPT-2 -base/XL	Summarization Translation	✓	✗	✓
FourierFT [71]	2024-05	2024-07	ICML	GPT-2 Medium/Large Llama-1-7B/13B Llama2-7B/13B	NLG Instruction Following	✓	✓	✓
ResLoRA [72]	2024-02	2024-08	ACL	Llama-2-7B	Math Commonsense	✗	✗	✓
PLoRA [73]	2024-02	–	arXiv	Llama-1-7B	Instruction Following Math	✓	✗	✓
OLoRA [14]	2024-06	–	arXiv	Mistral-7B LLaMA-2-7B Tiny Llama-1.1B Gemma-2B OPT-1.3B	Commonsense Instruction Following	✗	✗	✓
LamDA [74]	2024-06	2024-11	EMNLP	Llama-2-7B	NLG Math Commonsense	✗	✗	✓
PiSSA [7]	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 [54]	2024-05	2024-12	NeurIPS	GPT-2 Medium/Large Llama-2-7B/13B Mistral-7B-v0.1 Gemma-7B	NLG Math Instruction Following	✗	✗	✗
HRA [46]	2024-05	2024-12	NeurIPS	Llama-2-7B	Math	✗	✗	✗
CorDA [75]	2024-06	2024-12	NeurIPS	Llama-2-7B/13B Llama-3-8B Gemma-2-9B	Math Code Instruction Following World Knowledge	✗	✗	✓
LoRA-GA [17]	2024-07	2024-12	NeurIPS	Llama-2-7B	Math Code Instruction Following	✗	✗	✗
RoAd [76]	2024-09	2024-12	NeurIPS	Llama-1-7B/13B Llama-2-7B Llama-3-8B	Math Commonsense	✗	✗	✓
LoRA-drop [77]	2024-02	2025-01	COLING	Llama-2-7B	NLG Summarization GLUE Math	✗	✗	✗
AG-LoRA [78]	–	2025-01	IEEE Access	Llama-1-7B	Commonsense	✗	✗	✓

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
LoRA-Pro [79]	2024-07	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Llama-3.1-8B	Math Code Instruction Following	✓	✗	✓
LoRA-Dash [80]	2024-09	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Qwen2.5-7B	GLUE Commonsense	✗	✗	✓
KaSA [81]	2024-09	2025-04	ICLR	Llama-1-7B Llama-2-7B Llama-3-8B Qwen2.5-7B	GLUE Commonsense Instruction Following	✗	✗	✓
RandLoRA [19]	2025-02	2025-04	ICLR	GPT-2 Medium Qwen2-0.5B Phi3-3B Llama3-8B	NLG Commonsense	✗	✗	✓
DeLoRA [82]	2025-03	2025-04	ICLR	Llama-2-7B Llama3-8B	Commonsense	✓	✗	✓
HiRA [83]	–	2025-04	ICLR	Llama-2-7B Llama-3-8B	Math Commonsense Dialogue Generation	✗	✗	✓
MiLoRA [15]	2024-06	2025-04	NAACL	Llama-2-7B Llama-3-8B Qwen2.5-7B	Math Commonsense Instruction Following	✗	✗	✗
SSMLoRA [84]	2025-02	2025-04	NAACL	GPT-2 Llama-2-7B/13B	GLUE	✓	✗	✓
MiSS [85]	2024-09	2025-07	ICML∗	Llama2-7B/13B Mistral-7B Qwen3-4B Llama-3.2-3B	Math Code Instruction Following	✗	✗	✓
LoRA-One [86]	2025-02	2025-07	ICML	Llama-2-7B	Math Code Instruction Following	✓	✓	✗
Init[AB] [16]	2025-05	2025-07	ICML	Llama-3-8B	Arithmetic Commonsense	✓	✗	✗
Lily [87]	2024-07	2025-07	ACL	Llama-3-8B	Commonsense	✗	✗	✗
C3A [88]	2024-07	2025-07	ACL	Llama-2-7B Llama-3-8B/70B Mistral-7B Mistral-8x7B	Math Code Commonsense	✗	✗	✗
SuLoRA [89]	–	2025-07	ACL	Llama-2-7B	Instruction Following	✗	✗	✓
BiDoRA [90]	2024-10	2025-08	TMLR	GPT-2 Medium	NLG	✗	✗	✗
HD-PiSSA [91]	2025-05	2025-11	EMNLP	Llama-2-7B Llama-3-8B Mistral-7b-v0.1	Math Code	✗	✗	✓
LoSiA [92]	2025-07	2025-11	EMNLP	Gemma 2B Llama-2-7B/13B	Math Code Commonsense Instruction Following	✓	✗	✓

Method	arXiv Date	Pub. Date	Venue	Decoder-only LLM	Fine-tuned Task	LR	BS	Rank
Sensitivity-LoRA [93]	2025-09	2025-11	EMNLP	GPT-2 Large Qwen2.5-7B/32B Llama-3.1-8B	NLG Instruction Following	✗	✗	✗
OHoRA [94]	–	2025-11	EMNLP	Llama-2-7B Llama-3-8B Gemma-7B Llama-3.1-8B-Inst	Math Code Commonsense Instruction Following	✗	✗	✓
EVA [95]	2024-10	2025-12	NeurIPS	Llama-2-7B Gemma-2-70B Llama-3.1-8B/70B	Math Code Commonsense	✓	✗	✓
GoRA [55]	2024-10	2025-12	NeurIPS	Llama-3.1-8B Llama-2-7B	Math Code Instruction Following	✗	✗	✗
AuroRA [96]	2025-05	2025-12	NeurIPS	Llama-3-8B	Commonsense	✗	✗	✓
GraLoRA [18]	2025-05	2025-12	NeurIPS	Llama-3.1-8B/70B Llama-3.2-3B Qwen-2.5-1B/7B	Math Code Commonsense	✗	✗	✓
FlyLoRA [97]	2025-10	2025-12	NeurIPS	Llama-3.1-8B Qwen-2.5-7B/14B	MMLU Science Math Code	✗	✗	✓
DropLoRA [98]	2025-08	–	arXiv	Llama-2-7B Llama-3-8B	Math Code Commonsense Instruction Following	✗	✗	✓
PrunedLoRA [32]	2025-09	–	arXiv	Llama-3-8B	Math Commonsense	✓	✗	✓
LoRA-DA [99]	2025-10	–	arXiv	Llama-2-7B	Math Commonsense	✗	✗	✓
ABM-LoRA [100]	2025-11	–	arXiv	Llama-2-7B	Instruction Following	✗	✗	✓
LoFT [9]	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 [101]	2026-01	2026-04	ICLR	Llama-3-8B	Commonsense	✗	✗	✗
Stable-LoRA [102]	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 [103]	–	2026-04	ICLR	LLaMA-3.1-8B	Math Code Instruction Following	✗	✗	✓
GiVA [104]	2026-04	2026-05	AISTATS	Qwen-2-0.5B OLMo-2-7B Phi-3-3.8B Mistral-7B	Math Code Commonsense Instruction Following	✓	✗	✗
PEANuT [105]	2024-10	2026-08	KDD	Llama-2-7B Llama-3-8B Qwen-3-8B	Math Commonsense	✗	✗	✗

Appendix B. Details of Selected LoRA Variants

Initialization Variants. This category comprises methods that explore improved initialization strategies for LoRA [7, 14–17, 75, 86, 95]. 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 [14] applies QR decomposition to W_{pre} to initialize A and B using the first-r columns of Q and the first-r rows of R, respectively. PiSSA [7] and MiLoRA [15], 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 [16, 43, 106]. In particular, Init[AB] [16] showed that randomly initializing both LoRA matrices using Kaiming initialization can be more advantageous by balancing stability, training efficiency, and hyperparameter robustness. Turning to the *data-informed* subcategory, LoRA-GA [17] 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:

$$h = W_{\text{res}}x + \gamma_r BAx. \tag{2}$$

Architectural Modifications. Besides investigating initialization strategies, a large body of literature has also focused on architecture-level improvements, e.g., [18, 19, 42, 105**?**] available in the PEFT library [107]. By modifying vanilla LoRA's forward design, (i.e., Eq. 2), these methods improve fine-tuning effectiveness either by sustaining performance with greater parameter efficiency [53, 54, 64, 71**?**] or by achieving higher accuracy under a similar trainable-parameter budget [8, 18, 19, 61, 83]. We focus on methods in the latter category, as large differences in trainable-parameter counts relative to LoRA make 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 [42] and PEANuT [105], which involve multiple architectural choices, namely (m, b) and depth/activation function, respectively, and could therefore rapidly expand the hyperparameter search space.

Optimization Adjustments. More recent studies have started to improve LoRA by directly adjusting its optimization dynamics. For example, LoRA+ [68] assigns different learning rates to A and B, while *scaled AdamW* [69] introduces a small preconditioner into each gradient step. LoRA-Pro [79] further adjusts the gradients of LoRA so that the induced update better approximates the full fine-tuning gradient. More recently, LoFT [9] aligns the optimizer's internal dynamics with full fine-tuning by projecting Adam [108]'s first- and second-moment estimates into the same low-rank subspace, narrowing the performance gap between LoRA and full fine-tuning.

Appendix C. Related Work

C.1. Systematic Empirical Re-evaluation of Prior Claims

Incomplete performance evaluation remains a persistent concern [109, 110]. For instance, Melis et al. [111] revealed that two published improvements to the vanilla LSTM [112], 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. [113] pointed out that simple baselines such as linear SVM [114] are competitive with BERT [37]-based methods for text classification, sometimes even outperforming them with a clear gap. More recently, Rheude et al. [115] reported a striking finding in the multimodal learning literature: although new multimodal architectures are continually proposed with claims that they outperform prior baselines, they often fail to surpass standard unimodal approaches given fair comparisons and statistical correctness. Such systematic empirical re-evaluation has also been done across diverse machine learning subfields. Examples include not only traditional topics such as image classification [116], graph neural networks [117, 118], generative adversarial networks [119], recommender systems [120–122], metric learning [123], and neural network pruning [124], but also more recent areas like optimizers [125–127], reinforcement learning [128], preference optimization [129], and model merging [130, 131].

While empirical studies benchmarking LoRA with other PEFT methods such as prefix tuning [132] and BitFit [133] exist [134–139], few prior 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.

C.2. LoRA Hyperparameter Tuning

Theories regarding LoRA's lack of Lipschitz smoothness [140, 141] and its spurious loss landscape [142] point toward its intrinsic sensitivity to hyperparameter variations. Consequently, many research efforts have been invested in finding optimal training setups, such as learning rate [68, 143], rank [144], initializations [106], scaling factor [145], dropout [146, 147], and adapter placements [148, 149]. 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 [5, 150]. Addressing the computational bottleneck of such extensive searches, another line of work has focused on improving the efficiency of LoRA hyperparameter tuning, either by developing hyperparameter optimization algorithms [151–154] or by adopting system-level approaches that optimize hardware resources to maximize training throughput [155, 156]. Although prior work has optimized LoRA hyperparameters, only a few concurrent studies have begun to examine whether LoRA and its variants require distinct hyperparameter settings [157–159]. These studies, however, differ from ours in scope. Specifically, Zhang et al. [157] noted that LoRA and two of its initialization variants, PiSSA [7] and MiLoRA [15], exhibit performance shifts across two learning rates ($2e^{-4}$ and $2e^{-5}$). He et al. [158] also concluded that, with proper learning rate tuning, vanilla LoRA consistently matches or surpasses most of its variants under a single LLM scale with fixed rank and training duration. Meanwhile, Lee et al. [159] demonstrated that the reported advantages of PiSSA and MiLoRA could be driven by batch size bias. Our paper differs from these studies

by expanding the investigation to a broader set of recent LoRA variants and conducting comprehensive multivariate hyperparameter tuning to identify the best-performing configuration for each method. Moreover, our work compares methods across varying ranks and training durations, and further leverages Hessian analysis to provide theoretical insight into the underlying factors driving the observed performance differences and trends.

Appendix D. Additional Experiments

In this section, we present fine-tuning results for various LoRA methods under additional model–task combinations (Sec. D.1), different LoRA ranks (Sec. D.2), and diverse training durations (Sec. D.3). Moreover, we jointly optimize learning rate and batch size in Sec. D.4 and Sec. D.5 to examine their interaction, and include instruction-following tasks in Sec. D.6 to cover diverse task types.

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

Analogously to Figure 2, where we present all ten LoRA methods for Qwen3-0.6B on mathematical reasoning tasks, Figure 5 presents the corresponding comparison for Gemma-3-1B on 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 2, 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 2 and Figure 5. 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 leverage 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 tuning as in our study or re-estimating Hessian using new downstream task samples as in Sec. 3.

Similarly to Figure 3, where we present the performance of Llama-2-7B for LoRA, DoRA, Init[AB], MiLoRA, and PiSSA, Figure 6 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 $r = 128$.

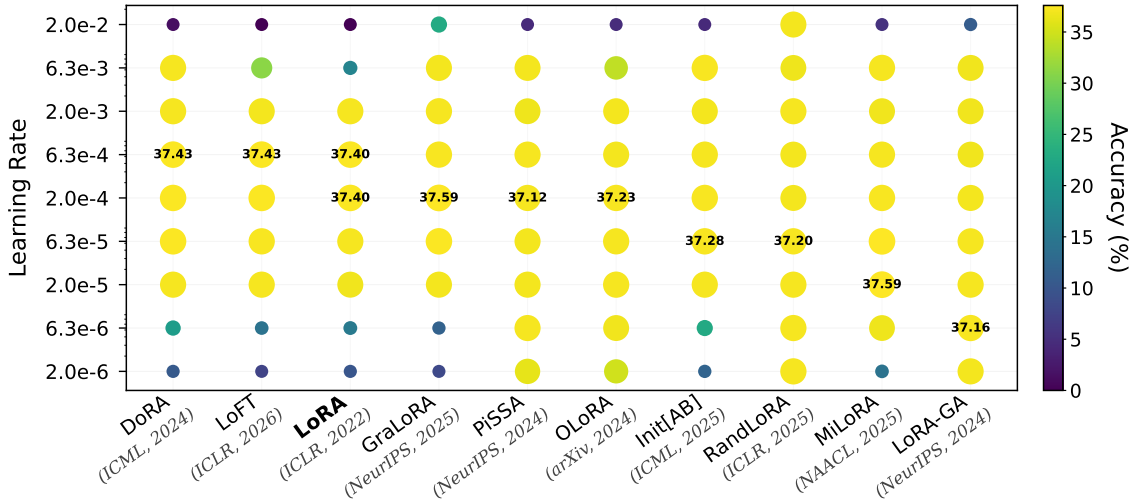
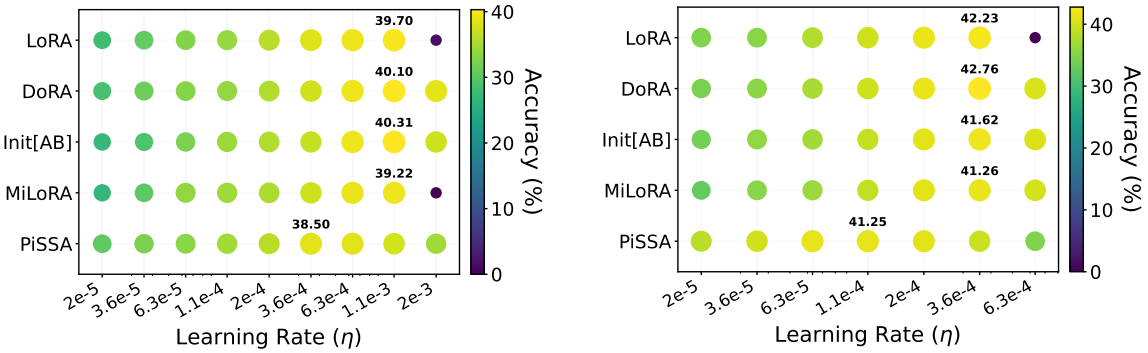


Figure 5: Performance of Gemma-3-1B fine-tuned on commonsense reasoning tasks across learning rates ($r = 4, B = 64$). Different methods reach a similar performance level once the learning rate is properly tuned. Each point is averaged over three independent training runs, and methods are sorted by their optimal learning rate ranges.



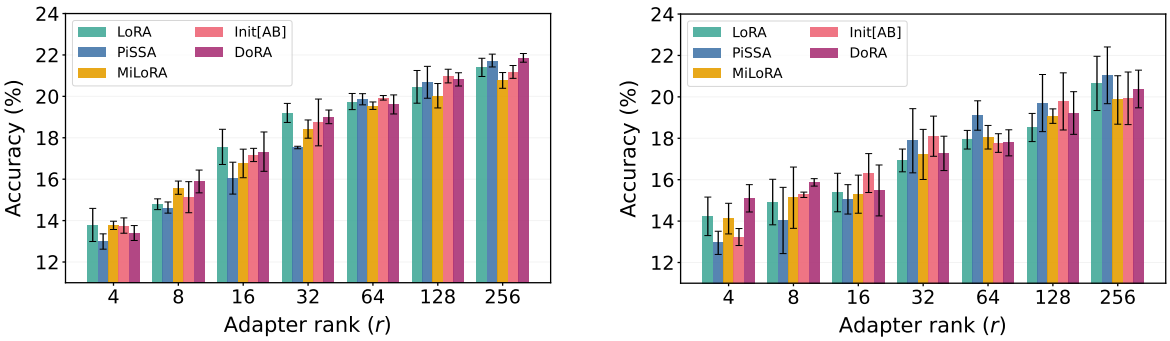
(*a*) $r = 8$ (*b*) $r = 128$

Figure 6: Performance of Llama-2-13B on mathematical reasoning tasks across varying learning rates and ranks $r \in \{8, 128\}$ ($B = 64$).

D.2. Varying Adapter Ranks

In this section, we extend our analysis by varying adapter ranks for Gemma-3-1B and Llama-2-7B, as shown in Figure 7 and 8, respectively. The results indicate that the previously observed performance parity persists across a wide range of rank settings. For example, in Figure 7, the maximum performance differences among methods are only 1.67% (Math, $r = 32$) and 2.15% (Code, $r = 4$). Interestingly, however, we observe that the relative performance of variants compared to LoRA fluctuates across different ranks within these margins.

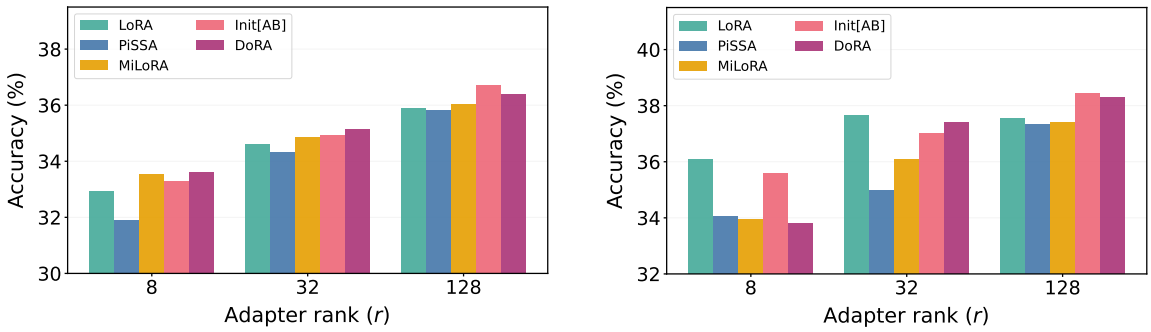
In particular, PiSSA initially underperforms vanilla LoRA before gradually overtaking it as the rank increases. Taking the math task as an example (Figure 7 (a)), PiSSA exhibits performance deficits of up to 1.67% at low ranks ($r \leq 32$), but narrows the gap to within 0.11% at $r = 64$ and shifts 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 7 (b) indicates that these 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 $r = 8$.



(a) Mathematical Reasoning

(b) Code Generation

Figure 7: 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.



(a) Mathematical Reasoning

(b) Code Generation

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

D.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 training duration increases, with their performance generally falling within one another's standard deviation range.

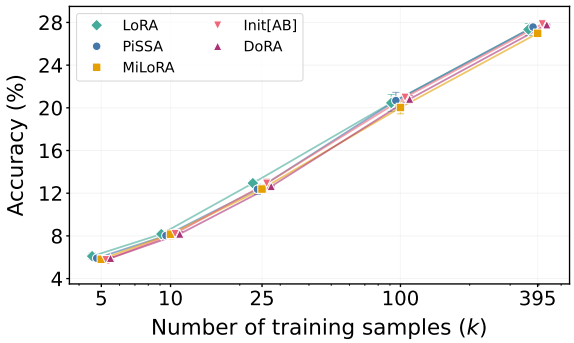


Figure 9: Best achievable performance of LoRA and its variants across different training sample sizes on mathematical reasoning with Gemma-3-1B ($r = 128, B = 64$). Once the learning rate is properly tuned, all methods exhibit nearly identical improvement trends as the number of training samples increases.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

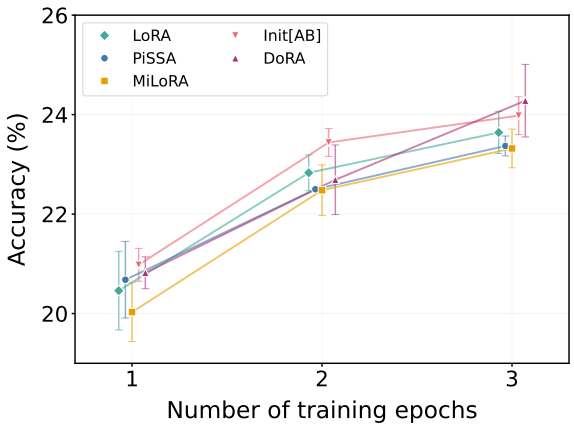


Figure 10: Best achievable performance of LoRA and its variants across different numbers of training epochs on mathematical reasoning with Gemma-3-1B ($r = 128, B = 64$). Results are reported with means and standard deviations over three independent runs. Once the learning rate is properly tuned, all 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.

D.4. Joint Optimization of Learning Rate and Batch Size

In Table 2, 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 the best performance in both LoRA and its variants, consistent with early findings for neural networks [13]. 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%.

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 2: 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 ( ).

D.5. Scaling Batch Size to 512 for LoRA and PiSSA

In Table 2, 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***.

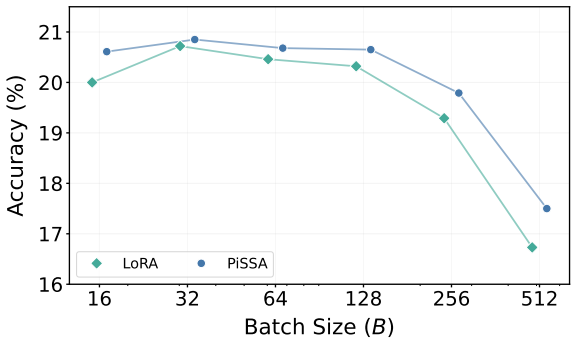


Figure 11: Best achievable performance of LoRA and PiSSA across different batch sizes on mathematical reasoning tasks with Gemma-3-1B ($r = 128$). While at $B \leq 128$, both LoRA and PiSSA can reach $\approx 20\%$ accuracy with proper learning rate tuning, the performance upper bound gradually decays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

D.6. 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 [160], while evaluating models' prompt-level strict accuracy under the IFEval [161] 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 3: 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.

Appendix E. Fine-tuning Implementation Details

E.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. Specifically, we utilize the official checkpoints hosted on Hugging Face [162]: `Qwen3-0.6B-Base`[2], `gemma-3-1b-pt`[3], `Llama-2-7b-hf`[4], and `Llama-2-13b-hf`[5].

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. https://huggingface.co/meta-llama/Llama-2-13b-hf

E.2. Dataset Setup

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. [135], which comprise eight general question-answering subtasks. For mathematical reasoning, we use 100k subsampled training examples from MetaMathQA [29] and evaluate models on GSM8K [11] and MATH [163]. For code generation, we use 104k subsampled training examples from CodeFeedback [164] and evaluate models on HumanEval [165] and MBPP [166]. For instruction following, we train models on 52k Alpaca [160] examples and evaluate them using the IFEval framework [161]. Unless otherwise specified, we report mean accuracy over the testing datasets.

E.3. Training 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.

E.4. 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.

Configuration	Value
Epoch	1
α	r
Optimizer	AdamW [167]
LR scheduler	Cosine annealing with warmup
Warmup ratio	3%
Dropout	None
Weight Decay	None
Adapter placement	All linear layers (except for the LM head)
Base model precision	BFloat16 [168][6]
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 except for the final language modeling head (i.e., `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`).

E.5. Data, Code, Libraries, and Hardware

We use the PiSSA codebase [7] 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 [107]. For MiLoRA, Init[AB], and LoFT, we refer to their official codebases[7] 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. [7, Table 2]). Additionally, while Li et al. [16] 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. [135][8]. For mathematical reasoning and code generation tasks, we used the preprocessed dataset released by Meng et al. [7][9]. For instruction-following tasks, we trained models on Alpaca-cleaned, which is publicly available on HuggingFace[10].

PyTorch [169] 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 [170]

6. 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.
7. https://github.com/sufenlp/MiLoRA
https://github.com/Leopold1423/non_zero_lora-icml25
https://github.com/tnurbek/loft
8. https://github.com/AGI-Edgerunners/LLM-Adapters/blob/main/ft-training_set/commonsense_15k.json
9. https://huggingface.co/datasets/fxmeng/pissa-dataset
10. https://huggingface.co/datasets/yahma/alpaca-cleaned

for parallel training and vLLM [171] for parallel inference. During inference, we apply greedy decoding (i.e., temperature set to 0), and utilize the EvalPlus [172] 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.6. 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 core 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 [145] 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. [5] 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. [5, Appendix B.2, Figure S3]). Notably, Zhang et al. [157] 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 [173]. This further validates our decision to fix the scaling factor in our experiments.

Appendix F. Practical Heuristics for LoRA Hyperparameter Tuning

Based on the 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 (Appendix Tables 2, 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. D.5). 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 2). In practice, however, practitioners should still account for the scaling relationship between batch size and learning rate (discussed in Appendix Sec. D.4), 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. 3.2 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, Hessian estimation for LoRA adapters of a single layer 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.

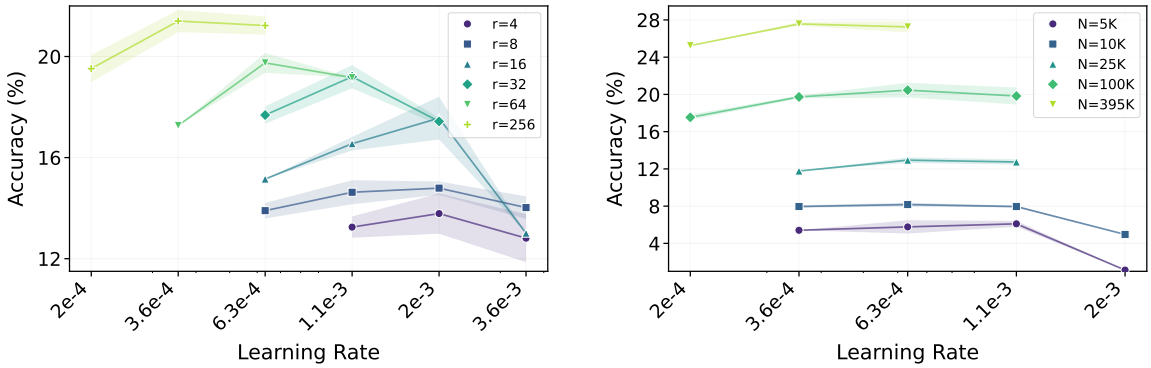
We also note that, based on our broad experiments, a given variant typically exhibits a stable relationship in optimal learning rate relative to LoRA across different model–task combinations, in terms of being either higher or lower (e.g., as discussed in Sec. D.1 for Figure 2 and 5). Practitioners may 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 7 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, Appendix 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 in Figure 12 (a), with $r = 4$ requiring a $5.6\times$ larger learning rate than $r = 256$. 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 [8, Figure 5], LoFT [9, Figure 3], GraLoRA [18, Table 2]).

V. Prolong Training Duration. If practitioners have computational resources to further improve 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. D.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 12 (b). 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 12: Vanilla LoRA performance across learning rates with (a) varying ranks and (b) varying training-set sizes on mathematical reasoning with Gemma-3-1B ($B = 64$). Note that panels (a) and (b) re-plot the data behind Figures 7 (a) and 9, respectively, with the learning rate on the x-axis.

Appendix G. Details of Hyperparameter Search Results

G.1. Qwen3-0.6B

G.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**.

G.2. Gemma-3-1B

G.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**.

G.3. Llama-2-7B

G.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**.

G.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**.

Appendix H. 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 13 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}. These qualitative findings correspond to our discussion in Sec. 2.3.

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 13: 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 $\binom{8}{2} = \frac{8!}{2!6!} = \frac{8 \cdot 7}{2 \cdot 1} = \boxed{16}$ 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 $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!2!} = \frac{8 \cdot 7}{3 \cdot 2} = \boxed{14}$ 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 $\frac{8}{3} \times \frac{6}{2} = \frac{8 \times 6}{3 \times 2} = \frac{48}{6} = \boxed{8}$. 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 $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} = \frac{8 \cdot 7}{3 \cdot 2} = 28$ ways. Therefore, there are $\boxed{28}$ quintets possible. The answer is: 28 ✗ **Incorrect**

LR (η)	LoRA Response	PiSSA Response
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**
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 $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!4!} = \frac{8\cdot7\cdot6}{3\cdot2\cdot1} = \frac{216}{6} = \boxed{36}$. 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 $\binom{8}{3} \cdot \binom{6}{2} = \frac{8!}{3!3!} \cdot \frac{6!}{2!2!} = \frac{8\cdot7\cdot6}{3\cdot2\cdot1} = 56$ 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**

LR (η)	LoRA Response	PiSSA Response
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
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...(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

LR (η)	LoRA Response	PiSSA Response
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**

Appendix 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[11] [174] and LLM-Hessian[12] [35], 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[13]. 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 [175], we use Float32 precision for both the base model and adapters and incorporate re-orthogonalization steps [176, 177].

11. https://github.com/amirgholami/PyHessian
12. https://github.com/vectozavr/llm-hessian
13. This is the default way of calculating loss during LLM supervised fine-tuning; see https://github.com/huggingface/transformers/issues/34510

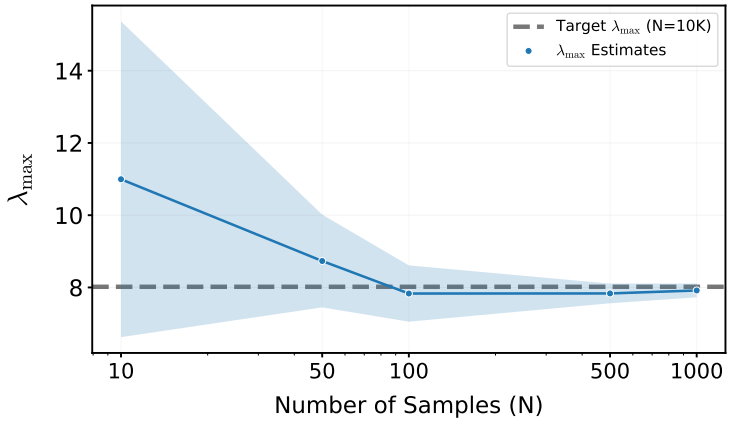


Figure 14: Approximately 500 training samples are sufficient for stable Hessian estimation. The figure reports the estimated $\lambda_{\max}$ for PiSSA on the first Query projection matrix of Qwen3-0.6B ($r = 128$). We track these estimates across varying sample sizes (N) from the MetaMathQA dataset,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

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. 3.2, Figure 4 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 2. 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 mathematical reasoning tasks under rank $r = 128$ are reported earlier in Table 2 and Figure 3 (a), respectively.

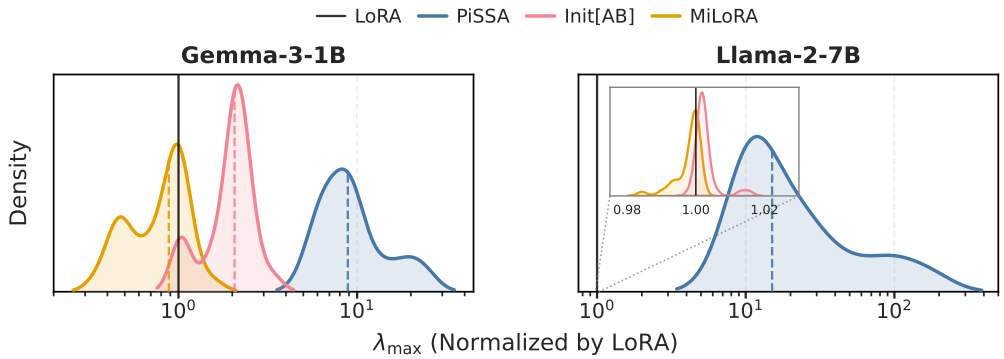


Figure 15: Distributions of the ratios of the top loss Hessian eigenvalues relative to LoRA for Query projection matrices across Transformer layers ($r = 128$). Dashed lines indicate the medians.

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–$2\times$ smaller learning rates in Table 2 and Figure 3 (a). 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 explains their similar optimal learning rate ranges to vanilla LoRA. 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 4. 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.

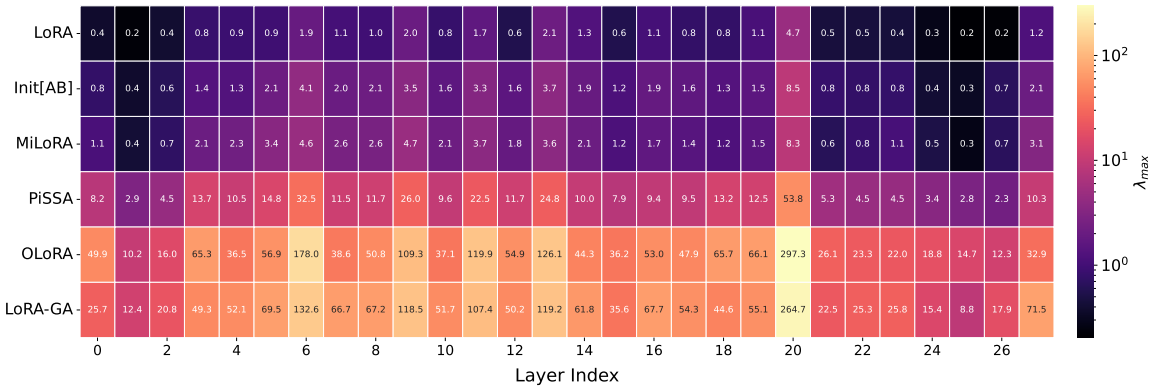


Figure 16: Heatmap of the top eigenvalues of the **Query projection matrix** across Transformer layers, i.e., $\lambda_{\max}^{Q,i}$ for $i = 1, \ldots, L$, for Qwen3-0.6B on MetaMathQA ($r = 128$). All methods exhibit similar distributional patterns across layers, with PiSSA, OLoRA and LoRA-GA consistently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

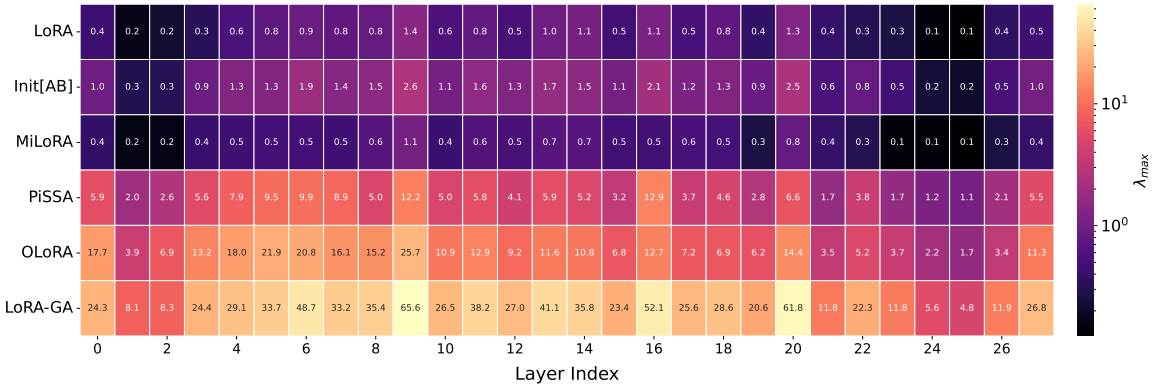


Figure 17: Heatmap of the top eigenvalues of the **Key projection matrix** across Transformer layers, i.e., $\lambda_{\max}^{K,i}$ for $i = 1, \ldots, L$, for Qwen3-0.6B on MetaMathQA ($r = 128$). All methods exhibit similar distributional patterns across layers, with PiSSA, OLoRA and LoRA-GA consistently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

Appendix 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 [37], Vision Transformers [39], and Vision-Language Models [40]) 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 [86], which initializes adapters via the SVD of the one-step full gradient, with $\approx 2\%$ performance improvement on Llama (cf. Zhang et al. [86] 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 [5, 15, 75, 178–180].

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 [181, 182] and approaches that adapt non-linear functions within layers [183]. We leave the exploration of these orthogonal paradigms as future work.