WeightLoRA: Keep Only Necessary Adapters

Andrey Veprikov[1,2,3], Vladimir Solodkin[1,2], Alexander Zyl[3], Andrey Savchenko[3,4], Aleksandr Beznosikov[1,2,5]
[1]Moscow Independent Research Institute of Artificial Intelligence (MIRAI),
[2]Basic Research of Artificial Intelligence Laboratory (BRAIn Lab),
[3]Sber AI Lab, [4]HSE University, [5]Innopolis University
veprikov.as.18@gmail.com

Abstract

The widespread utilization of language models in modern applications is inconceivable without Parameter-Efficient Fine-Tuning techniques, such as low-rank adaptation (LoRA), which adds trainable adapters to selected layers. Although LoRA may obtain accurate solutions, it requires significant memory to train large models and intuition on which layers to add adapters. In this paper, we propose a novel method, WeightLoRA, which overcomes this issue by adaptive selection of the most critical LoRA heads throughout the optimization process. As a result, we can significantly reduce the number of trainable parameters while maintaining the capability to obtain consistent or even superior metric values. We conduct experiments for a series of competitive benchmarks and DeBERTa, BART, Llama and Qwen models, comparing our method with different adaptive approaches. The experimental results demonstrate the efficacy of WeightLoRA and the superior performance of WeightLoRA+ in almost all cases. The source code is available at https://github.com/brain-lab-research/WLoRA.

1 Introduction

The emergence of Large Language Models (LLMs) has transformed numerous domains, from natural language processing to code generation (Dubey et al., 2024; He et al., 2020). Despite the undebatable performance, their fine-tuning remains computationally intensive and memory-demanding. Parameter-Efficient Fine-Tuning (PEFT) (Ding et al., 2023; Han et al., 2024), such as Low-Rank Adaptation (LoRA) (Hu et al., 2021), has significantly alleviated these challenges by reducing the number of trainable parameters while maintaining competitive accuracy.

However, LoRA still faces inherent limitations, including substantial memory requirements for its rank-specific adapters and the challenge of selecting appropriate hyperparameters, particularly the rank and position of the adapters, which significantly influence performance and resource consumption (Liu et al., 2024b; Zhou et al., 2025). Over the years, several modifications of LoRA have been introduced to deal with its disadvantages and improve efficiency (Liu et al., 2024a,b; Zhang et al., 2023b; Zhou et al., 2025). Although these modifications are generally effective in addressing certain deficiencies, empirical evidence shows that, for general staging applications, the conventional LoRA approach is often more advantageous due to the inherent complexity in configuring specific methods (see Section 3). Details about various LoRA modifications are presented in Section 4.

In this paper, we propose WeightLoRA (Figure 1), a novel approach to fine-tuning LLMs that adaptively optimizes and selects LoRA adapters based on their contribution to model performance. Using an alternating minimization algorithm (Karlin, 2018), we manage to keep only the top-K adapters with the highest impact on fine-tuning. Our dual-phase approach significantly reduces the number of trainable parameters while maintaining high performance, making WeightLoRA a scalable solution for fine-tuning LLMs in resource-constrained environments. To evaluate the effectiveness of WeightLoRA, we conduct extensive experiments using natural language processing (Wang, 2018), LLM-benchmarks (Cobbe et al., 2021; Yu et al., 2023), question answering (Rajpurkar, 2016), and natural language generation (Narayan et al., 2018) benchmarks with state-of-the-art low-budget LLMs, including DeBERTaV3-base (He et al., 2021), BART-large (Lewis, 2019), Llama3-7B (AI@Meta, 2024) and Qwen3-8B (Yang et al., 2025).

Our Contribution

- We introduce WeightLoRA (Section 2), a lightweight extension of the standard LoRA framework that augments each adapter with

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12419–12437

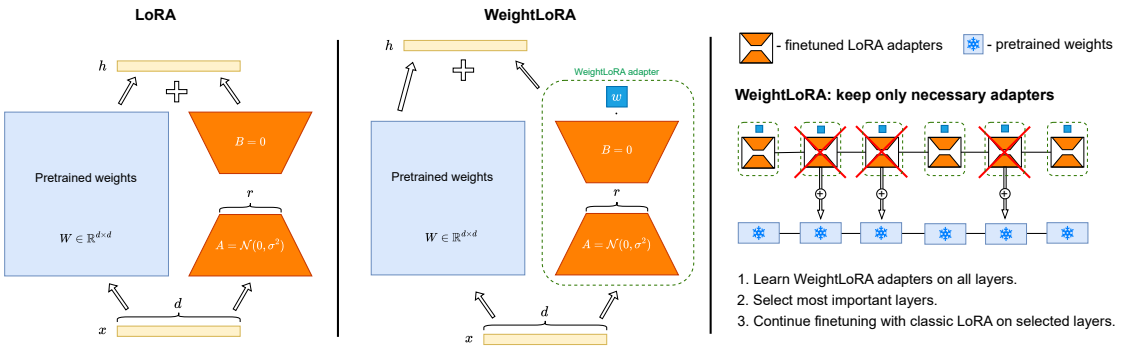

Figure 1: Comparison between LoRA (left) and the proposed WeightLoRA framework (right). The core idea of WeightLoRA is to add weights to the LoRA adapters, choose the most important ones, and train only this small subset.

a scalar, trainable importance weight. Unlike prior approaches that modify the adapter parameterization, dynamically adjust ranks, or introduce task-dependent routing mechanisms, WeightLoRA performs adapter selection only during a short warm-up phase. After this phase, the method reduces exactly to standard LoRA fine-tuning on a fixed subset of selected adapters. This design allows WeightLoRA to significantly reduce the number of trainable parameters while preserving compatibility with existing LoRA variants and maintaining competitive performance.

- We introduce WeightLoRA+ (Section 2.2), which advances the concept of WeightLoRA by adaptively increasing the rank of selected adapters.

- We validate our framework through extensive experiments across diverse architectures and fine-tuning tasks (Sections 3.1-3.4), complemented by ablation analyses (Section 3.5) that isolate the effect of our design choices.

2 WeightLoRA Framework

In this section, we introduce the selection process for the most valuable LoRA adapters. Let $\mathcal{W}_0 := \{W_0^i\}_{i=1}^n$ denote the collection of pretrained weight matrices to which LoRA adapters are attached. In the WeightLoRA framework, each such matrix is modified as

$$W^i = W_0^i + \omega_i A^i B^i, \tag{1}$$

where A^i and B^i are the low-rank adapter matrices, and ω_i represents the importance weight of the i-th adapter. Throughout this paper, an *adapter* refers to the low-rank update $A^i B^i$ applied to a single weight matrix (e.g., a query, key, or value projection), not to an entire transformer block. For a model with m linear projections per layer, each projection is treated as an independent selection unit, therefore $n = m \cdot L$ where L is the number of layers. The weights are composed into a vector $\omega = (\omega_1, \ldots, \omega_n)^T$, which is then optimized throughout the fine-tuning process. To control the number of active adapters, we propose conditioning the l_0-"norm" of ω: $\|\omega\|_0 \stackrel{\text{def}}{=} \sum_{i=1}^n \mathbb{I}(\omega_i \neq 0)$ (Nguyen et al., 2014).

That is, if one uses the classical LoRA framework (see Appendix A for details), the optimization problem to be solved is:

$$\min_{\{A^i B^i\}} \mathcal{L}(A^1 B^1, \ldots, A^n B^n),$$

where $\mathcal{L}$ denotes the fine-tuning loss. In WeightLoRA, the optimization problem transforms into:

$$\min_{\|\omega\|_0 \leq K} \min_{\{A^i B^i\}} \mathcal{L}(\omega_1 A^1 B^1, \ldots, \omega_n A^n B^n), \tag{2}$$

where K is a positive constant defining the number of adapters to keep. We next describe the two stages of the procedure.

Phase 1: Warm-up (selection). At the beginning of fine-tuning, we jointly optimize the adapter weights $\{A^i, B^i\}_{i=1}^n$ and the importance vector ω under the sparsity constraint $\|\omega\|_0 \leq K$ for T warmup steps. In this stage, all adapters are active, and the role of ω is to identify the few heads that contribute most to the loss reduction. We update ω using the classical StoIHT algorithm (Nguyen et al., 2014), which performs a gradient step followed by hard thresholding that keeps only the Top-K entries and sets the rest to zero. A short overview

of StoIHT and related ℓ_0-constrained optimization methods is provided in Appendix B.

It is important to note that the number of warm-up steps T should be kept small to avoid increasing computational overhead. One can take $T \lesssim 0.5\%$ of the total training iterations. The sparsity level K is another hyperparameter of the method; by default, we recommend $K = n/2$. In Section 2.2, where we introduce a modification of the WeightLoRA algorithm, we provide an additional motivation for choosing $K = n/2$. Finally, the ablation study in Section 3.5 confirms that $T \lesssim 0.5\%$ and $K = n/2$ are sufficient in practice.

Phase 2: Disabling adapters. After T steps, we fix ω and disable all adapters with zero weights, keeping only the K most important ones. For the selected adapters ($\omega_i > 0$), we reinitialize the corresponding low-rank matrices A^i and B^i as in the standard LoRA setup (random initialization for A^i and zeros for B^i), and discard all remaining adapters. The effective weight matrix of the i-th layer then becomes

$$W^i = \begin{cases} W_0^i + A^i B^i, & \text{if } \omega_i > 0, \\ W_0^i, & \text{otherwise.} \end{cases}$$

Discussion. Instead of optimizing a large number of matrices A^i, B^i across all layers, WeightLoRA first moves the structural decision into the sparse vector ω and only then continues fine-tuning on the selected $K \leq n$ adapters. This two-stage procedure makes the cost of adapter selection negligible, reduces the number of trainable parameters, and allows, after reaching the timestamp T, to increase the batch size because fewer parameters need to be stored.

A more detailed analysis of GPU memory savings as a function of the number and placement of LoRA adapters is provided in Appendix C.

2.1 Motivating Example

We now provide a simple example that illustrates why different adapters can have very different impacts on loss and why it is reasonable to explicitly select only a small subset of them. Under the classical LoRA setup, we solve

$$\min_{W^1,\dots,W^n} \mathcal{L}(W^1, \dots, W^n). \tag{3}$$

We consider a scaled variant in which the contribution of each adapter is controlled by a fixed vector $\omega = (\omega_1, \dots, \omega_n)^\top \in \mathbb{R}^n$. The effective weight matrix of the i-th layer is $W^i = W_0^i + \omega_i A^i B^i$. Using a first-order Taylor expansion of $\mathcal{L}$ around the pretrained model $\mathcal{W}_0$ we obtain:

$$\begin{aligned} \mathcal{L}(W^1, \dots, W^n) \approx \mathcal{L}(\mathcal{W}_0) + \\ + \omega_1 \langle \nabla_{W^1} \mathcal{L}(\mathcal{W}_0), A^1 B^1 \rangle + \dots \\ + \omega_n \langle \nabla_{W^n} \mathcal{L}(\mathcal{W}_0), A^n B^n \rangle. \end{aligned}$$

Substituting this approximation into (3) and dropping the constant term, the problem decouples across layers:

$$\begin{aligned} (3) \Leftrightarrow \omega_1 \min_{A^1,B^1} \left\langle \nabla_{W^1} \mathcal{L}(\mathcal{W}_0), A^1 B^1 \right\rangle + \cdots \\ + \omega_n \min_{A^n,B^n} \left\langle \nabla_{W^n} \mathcal{L}(\mathcal{W}_0), A^n B^n \right\rangle. \end{aligned} \tag{4}$$

This perspective reveals that different layers (and their corresponding adapters) can significantly reduce losses, motivating the adapter-selection mechanism used in WeightLoRA. For clarity, we stress that this is a motivating first-order view: our method does not require full gradient computation during training, and the Taylor expansion is used only to build intuition.

In the general case, the minima in (4) are distinct and have different magnitudes, reflecting heterogeneous tuning difficulty across adapters. Some $A^i B^i$ may require substantially more updates to reach their optimum. In contrast, others may not need tuning at all (e.g., when $\nabla_{W^i} \mathcal{L}(\mathcal{W}_0) = 0$, indicating no benefit from adapting layer i).

To validate these considerations empirically, we examine how the minima in (4) differ in practice. We take a DeBERTa model (He et al., 2021) with $n = 36$ self-attention layers, attach a single adapter to the i-th layer, fine-tune for several epochs, then compute the full gradient $\nabla_{W^i} \mathcal{L}(\mathcal{W}_0)$ and the dot product between the adapter and this gradient. We repeat this procedure for each layer i on the same subset of the GLUE benchmark (Wang, 2018). The results are shown in Figure 2.

We observe that the value adapters in layers 8, 10, 11, as well as the key/query adapters in layer 11, exhibit significantly larger absolute scalar products, indicating that their optimization is more involved. Interestingly, when we later apply WeightLoRA on the same setup (star points in Figure 2), the set of selected adapters largely overlaps with these "high-dot-product" ones (except for 11-th key). This agreement is non-trivial: WeightLoRA follows an entirely different, data-driven selection process based on sparse weight optimization rather than

Table 1: Results of fine-tuning the DeBERTaV3-base model on GLUE benchmark. The best results are shown in **bold**, and the second-best are underlined. All experiment details are provided in Appendix E.1.

Method	# Params	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	ALL Avg
Full Fine-Tuning	184M (100%)	0.8910	0.9541	0.6806	0.8962/0.8644	0.9383	0.8376	0.8848/0.9165	**0.8689**
LoRA (Hu et al., 2021)	442K (0.24%)	0.8797	0.9450	0.6913	0.8802/0.8437	0.9301	0.8448	0.8897/0.9220	**0.8655**
IncreLoRA (Zhang et al., 2023a)	589K (0.32%)	0.8046	0.8303	0.6266	0.8206/0.7938	0.8556	0.7426	0.7598/0.8435	0.7812
AdaLoRA (Zhang et al., 2023b)	664K (0.36%)	0.8590	0.9392	0.6222	0.8534/0.8083	0.8999	0.7834	0.7059/0.8225	0.8141
EVA (Paischer et al., 2024)	664K (0.36%)	0.8617	0.9396	0.6252	0.8664/0.8212	0.9018	0.7841	0.7191/0.8333	0.8189
MELoRA (Ren et al., 2024)	110K (0.06%)	0.7986	0.9358	0.6188	0.7599/0.8342	0.9170	0.7527	0.8138/0.8701	0.8088
Dynamic LoRA (Mao et al., 2024)	626K (0.34%)	0.8545	0.9258	0.6490	0.8961/0.8643	0.9215	0.7315	0.8172/0.8701	0.8295
ALoRA (Liu et al., 2024b)	664K (0.36%)	0.8301	0.9559	0.6419	0.8408/0.8098	0.8833	0.7644	0.8572/0.9086	0.8263
FlexLoRA (Wei et al., 2025)	442K (0.24%)	0.7776	0.8204	0.6165	0.7959/0.7510	0.8383	0.7246	0.7187/0.8019	0.7587
LoRA-drop (Zhou et al., 2025)	221K (0.12%)	0.7659	0.9358	0.5903	0.8186/0.7701	0.8480	0.7668	0.7605/0.8283	0.7851
PrunedLoRA (Yu et al., 2025)	442K (0.24%)	0.8809	0.9453	0.6377	0.8850/0.8239	0.9247	0.7917	0.7193/0.8377	0.8305
WLoRA$_{K=1}$	**12.3K (0.007%)**	0.7912	0.8876	0.5624	0.8307/0.7813	0.8724	0.7271	0.8309/0.8825	0.7862
WLoRA$_{K=5}$	**61.5K (0.03%)**	0.8638	0.9427	0.6491	0.8685/0.8292	0.9136	0.7560	0.8799/0.9148	0.8388
WLoRA$_{K=10}$	123K (0.07%)	0.8706	0.9473	0.6703	0.8728/0.8362	0.9167	0.7643	0.8775/0.9101	0.8454

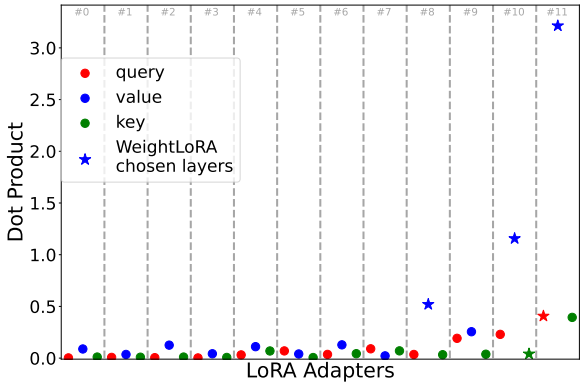

Figure 2: Comparison of the absolute values of the scalar products $\langle \nabla_{W^i} \mathcal{L}(\mathcal{W}_0), A^i B^i \rangle$ from (4) for all layers $i \in \{1, 2, ..., 36\}$. The adapters selected through our WeightLoRA framework are starred.

explicit gradient dot product probing. The fact that these two independent criteria identify almost the same set of adapters provides additional empirical support for the underlying intuition. A consistent qualitative pattern emerges across all evaluated architectures: deeper layers are systematically prioritized over shallower ones, and value projections are selected more frequently than query or key projections. This aligns with the intuition that value projections must capture task-specific information and therefore require greater adaptation, while attention patterns encoded in query and key projections tend to be more transferable from pretraining.

2.2 WeightLoRA+

In this section, we extend the idea of adapter selection to the fixed-budget setting and introduce WeightLoRA+, which uses the same total parameter budget as standard LoRA but allocates it preferentially to more important adapters.

Phase 1: Warm-up (selection). We run WeightLoRA for T warm-up steps with a fixed common rank r and a sparsity constraint $\|\omega\|_0 \leq n/2$. Thus, only half of the layers are allowed to keep non-zero importance weights. After the warm-up, we freeze ω and disable all adapters with $\omega_i = 0$.

Phase 2: Expanded adapter training. For each selected adapter ($\omega_i > 0$), we double its rank and discard all remaining adapters. Formally, the effective rank allocation after the warm-up phase is given by

$$r_i = \begin{cases} 2r, & \text{if } \omega_i > 0, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{with} \quad \sum_{i=1}^{n} r_i = nr,$$

which ensures that the total number of trainable parameters matches that of standard LoRA.

Discussion. WeightLoRA+ can be viewed as a budget-preserving variant of WeightLoRA: it uses the same parameters number as standard LoRA. However, it reallocates capacity from less useful adapters to more important ones. Importantly, we eliminate the hyperparameter K by fixing $K = n/2$, which is a natural choice given that LoRA ranks are typically powers of two, and we reuse the same warm-up steps T as in WeightLoRA. As shown in the ablation study on hyperparameters (Section 3.5), the method is robust to these choices and consistently outperforms uniform parameter allocation across layers.

3 Experiments

In this section, we provide a comprehensive experimental validation of our approach. We evaluate the fine-tuning performance of WeightLoRA

Table 2: Results of fine-tuning the DeBERTaV3-base model on GLUE benchmark with different ranks, and applying adapters to all attention layers. All experiment details are provided in Appendix E.1.

Rank	Method	# Params	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	ALL Avg
$r=1$	LoRA	0.07%	0.8912	0.9488	0.6558	0.8905/0.8544	0.9250	0.7943	0.8905/0.9213	0.8562
	WLoRA	**0.03%**	0.8740	0.9431	0.6465	0.8813/0.8490	0.9254	0.7764	0.8882/0.9210	0.8479
	WLoRA+	0.07%	0.8977	0.9495	0.6620	0.8841/0.8501	0.9246	0.8255	0.8873/0.9190	**0.8612**
$r=2$	LoRA	0.13%	0.9084	0.9413	0.6600	0.9077/0.8631	0.9369	0.8068	0.9003/0.9193	0.8641
	WLoRA	**0.06%**	0.9089	0.9518	0.6728	0.8848/0.8733	0.9048	0.7823	0.9004/0.9276	0.8591
	WLoRA+	0.13%	0.8960	0.9524	0.6848	0.8898/0.8425	0.9149	0.8567	0.8910/0.9009	**0.8667**
$r=4$	LoRA	0.26%	0.8780	0.9562	0.6809	0.9153/0.8866	0.9582	0.7816	0.9207/0.9471	0.8700
	WLoRA	**0.13%**	0.8600	0.9614	0.6824	0.9005/0.8762	0.9200	0.8637	0.8901/0.9099	0.8680
	WLoRA+	0.26%	0.9241	0.9714	0.6907	0.9032/0.9133	0.9057	0.8434	0.8615/0.8912	**0.8743**
$r=8$	LoRA	0.51%	0.9030	0.9641	0.6965	0.9092/0.8723	0.9449	0.8537	0.9064/0.9127	0.8804
	WLoRA	**0.25%**	0.9109	0.9653	0.6898	0.9130/0.8744	0.9467	0.7955	0.9061/0.9315	0.8744
	WLoRA+	0.51%	0.9074	0.9644	0.6934	0.9212/0.8926	0.9466	0.8531	0.9023/0.9356	**0.8844**

and WeightLoRA+ on DeBERTaV3-base (He et al., 2021), BART-large (Lewis, 2019), Llama3-7B (AI@Meta, 2024) and Qwen3-8B (Yang et al., 2025) models. Our experiments cover a wide range of tasks, including natural language understanding with the GLUE benchmark (Wang, 2018), question answering with SQuADv1 (Rajpurkar, 2016) and SQuADv2 (Rajpurkar et al., 2018), natural language generation with XSum (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets, classical LLM benchmarks: MathQA (Yu et al., 2023), GSM8K (Cobbe et al., 2021), HellaSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019) and ARC-Challenge (Clark et al., 2018). We provide the full experimental setup, additional details on hyperparameters, and the fine-tuning process in Appendix E.

3.1 Natural Language Understanding

Dynamic rank comparison. Table 1 reports a comparative analysis against LoRA-inspired methodologies that incorporate dynamic rank or capacity allocation. In particular, we consider methods such as MELoRA (Ren et al., 2024), AdaLoRA (Zhang et al., 2023b), FlexLoRA (Wei et al., 2025), EVA (Paischer et al., 2024), IncreLoRA (Zhang et al., 2023a), Dynamic LoRA (Mao et al., 2024), ALoRA (Liu et al., 2024b), LoRA-drop (Zhou et al., 2025), and PrunedLoRA (Yu et al., 2025). For a detailed description of these baselines and their design principles (dynamic rank allocation, SVD-based initialization, score-based pruning, and routing), see Section 4. For all methods, we set LoRA rank to 8.

We employ the DeBERTaV3-base (He et al., 2021) model, comprising 184M parameters. Following the original setup, LoRA adapters are applied exclusively to the self-attention weights, resulting in $n = 36$ fine-tuned layers. This constrained choice enables us to explicitly monitor the importance of individual adapters and clearly observe the impact of disabling them in adaptive selection methods. The goal of this experiment is to evaluate existing dynamic layer and rank selection methods in a unified setting with minimal learning-rate tuning (see Appendix E.1 for further experimental details).

The results indicate that all existing adaptive layer selection methods require extensive hyperparameter tuning and tend to perform suboptimally "out of the box". In contrast, our proposed WeightLoRA algorithm demonstrates competitive performance compared to standard LoRA and full fine-tuning baselines, highlighting its robustness and simplicity. This improved stability can be attributed to the fact that, throughout almost the entire training process, WeightLoRA effectively performs regular LoRA fine-tuning, with adapter selection occurring only in a short warm-up phase, thereby reducing sensitivity to hyperparameters.

Comparison with tuned LoRA baselines. Table 2 provides a focused evaluation under well-understood conditions. We limit our further comparisons to the classical LoRA method. In this experiment, we maintain the same DeBERTaV3-base model but extend the adapters to all linear layers, resulting in $n = 72$ layers. We consider four adapter rank values to cover a range of capacities and perform comprehensive hyperparameter

Table 3: Results of fine-tuning the Llama3 8B model on GLUE benchmark. For every rank r, the best results on each task are shown in **bold**. All experiment details are provided in Appendix E.1.

Rank	Method	# Params	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	STS-B Corr	ALL Avg
$r=1$	LoRA	0.07%	0.9213	0.9703	0.6867	0.9001/0.8438	0.9208	0.8808	0.9406/0.9578	0.9212	**0.8903**
	WLoRA	**0.03%**	0.8913	0.9604	0.6275	0.9001/0.8438	0.9208	0.8810	0.9406/0.9578	0.9653	0.8834
	WLoRA+	0.07%	0.9012	0.9682	0.6365	0.9100/0.8522	0.9304	0.8897	0.9491/0.9657	0.9737	**0.8923**
$r=2$	LoRA	0.13%	0.9222	0.9703	0.6867	0.9001/0.8485	0.9209	0.8911	0.9409/0.9598	0.9220	0.8922
	WLoRA	**0.07%**	0.9012	0.9703	0.6475	0.9001/0.8485	0.9209	0.9109	0.9408/0.9593	0.9687	0.8930
	WLoRA+	0.13%	0.9133	0.9703	0.6587	0.9118/0.8601	0.9330	0.9234	0.9510/0.9695	0.9806	**0.9046**
$r=4$	LoRA	0.26%	0.9301	0.9802	0.6913	0.9101/0.8585	0.9408	0.9109	0.9506/0.9678	0.9223	0.9024
	WLoRA	**0.13%**	0.9111	0.9802	0.6784	0.9208/0.8752	0.9506	0.9110	0.9507/0.9696	0.9687	0.9073
	WLoRA+	0.26%	0.9213	0.9802	0.6887	0.9312/0.8873	0.9607	0.9223	0.9621/0.9804	0.9810	**0.9168**
$r=8$	LoRA	0.52%	0.9447	0.9802	0.7015	0.9111/0.8658	0.9507	0.9110	0.9557/0.9617	0.9290	0.9080
	WLoRA	**0.26%**	0.9201	0.9802	0.6885	0.9208/0.8757	0.9604	0.9208	0.9604/0.9718	0.9696	0.9130
	WLoRA+	0.52%	0.9342	0.9802	0.7057	0.9324/0.8912	0.9771	0.9312	0.9740/0.9879	0.9854	**0.9258**

tuning, including grid search over learning rates and warmup steps in the linear scheduler (see Appendix E.1 for details).

These results show that our proposed rank extension method, WeightLoRA+, performs better on average than LoRA for all ranks. At the same time, the adapter deactivation method, WeightLoRA, achieves results comparable in quality to the LoRA baseline while using approximately three times fewer parameters. We note that small absolute improvements are standard for GLUE in the PEFT literature, where the primary goal is to achieve parity with, or slightly improve upon, LoRA under a reduced parameter budget. For reference, LoRA-drop (Zhou et al., 2025) reports gains of 0.1–0.5% over LoRA on individual GLUE tasks, and AdaLoRA (Zhang et al., 2023b) typically shows improvements of less than 1% on MNLI or QQP. The advantages of our approach become significantly more pronounced when scaling to larger models and more complex reasoning tasks, as demonstrated in Section 3.2.

LLama3 8B experiment. To demonstrate the scalability of our approach, we evaluate the larger Llama3-7B model on the same GLUE benchmark. This model comprises $n = 224$ adapter layers, substantially increasing the parameter budget compared to previous experiments. We compare the classical LoRA method with both WeightLoRA and WeightLoRA+. Hyperparameter tuning follows the same protocol as before, with a longer training schedule to accommodate the increased model size (see Appendix E.1 for details). The quantitative outcomes and detailed comparisons are summarized in Table 3.

Table 4: Fine-tuning results of Qwen3-8B on classical LLM benchmarks. All experiment details are provided in Appendix E.2.

Rank	Method	# Params	MQA Acc	GSM8K Acc	HS Acc	BQ Acc	ARC-C Acc	ALL Avg
$r=2$	LoRA	0.14%	57.3	31.1	78.7	68.9	86.9	64.58
	WLoRA	**0.07%**	56.9	30.8	78.2	68.0	86.1	64.00
	WLoRA+	0.14%	**60.1**	**31.7**	**79.5**	**69.1**	**87.4**	**65.56**
$r=4$	LoRA	0.28%	58.1	31.4	78.8	69.7	87.6	65.12
	WLoRA	**0.14%**	58.0	31.2	78.5	68.9	87.5	64.82
	WLoRA+	0.28%	**60.5**	**32.0**	**79.8**	**70.4**	**88.2**	**66.18**
$r=8$	LoRA	0.57%	59.9	31.6	79.5	70.8	88.1	65.98
	WLoRA	**0.28%**	59.1	31.4	79.1	69.7	87.7	65.40
	WLoRA+	0.57%	**61.6**	**32.4**	**80.1**	**72.4**	**88.7**	**67.04**
$r=16$	LoRA	1.14%	60.7	31.8	80.0	74.4	89.3	67.24
	WLoRA	**0.57%**	59.2	31.5	79.4	74.4	88.1	66.52
	WLoRA+	1.14%	**63.5**	**32.5**	**81.9**	**74.7**	**89.7**	**68.46**

The experimental results in this section demonstrate the broad applicability of the WeightLoRA method across models with different numbers of trainable parameters: for large-scale models, selectively deactivating less important adapters not only reduces resource usage but can also lead to improved performance, especially at higher adapter ranks ($r = 4, 8$).

3.2 Evaluation on Classical LLM Benchmarks

To verify the effectiveness of our methods on large language models and diverse reasoning tasks, we conduct experiments with the Qwen3-8B model (Yang et al., 2025) on a set of classical LLM benchmarks: MathQA (MQA) (Yu et al., 2023), GSM8K (Cobbe et al., 2021), HellaSwag (HS) (Zellers et al., 2019), BoolQ (BQ) (Clark et al., 2019), and ARC-Challenge (ARC-C) (Clark et al., 2018). We compare LoRA, WeightLoRA and WeightLoRA+ across multiple adapter ranks (see details in Appendix E.2).

The results in Table 4 demonstrate that WeightLoRA+ consistently outperforms standard LoRA across all adapter ranks and benchmarks, confirming that reallocating capacity to more important adapters improves performance under a fixed parameter budget. At the same time, WeightLoRA attains competitive results while using approximately three times fewer trainable parameters, validating the effectiveness of our adapter selection strategy on large-scale models and diverse reasoning tasks.

Scalability to 70B models. We evaluate on GSM8K using Llama-3.1 70B (Figure 3). WeightLoRA+ outperforms LoRA at every rank, reaching 54.71% vs. 51.25% at $r = 16$ ($+7\%$ relative). This gap is larger than on the 8B models (Table 4), suggesting that rank reallocation scales with model capacity. WeightLoRA stays within 0.5% of LoRA while using half the parameters.

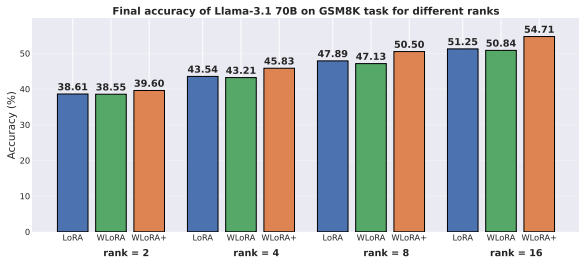

Figure 3: GSM8K accuracy on Llama-3.1 70B. WeightLoRA+ outperforms LoRA at every rank under the same parameter budget; the gap grows with rank.

3.3 Question Answering

In this experiment, we consider the more challenging question-answering task on the SQuAD v1.1 and v2.0 datasets (Rajpurkar, 2016; Rajpurkar et al., 2018). We refer to the Appendix E.3 for a detailed description of the experimental setup. The results are summarized in Table 5.

Analysis of these results indicates that the proposed WeightLoRA+ method outperforms LoRA approach for every rank selection. At the same time, the WeightLoRA method, with a three times lower number of trainable parameters, demonstrates competitive performance compared to both approaches.

3.4 Natural Language Generation

In this section, we address the task of natural language generation. We apply the WeightLoRA method to fine-tune a BART-large model (Lewis, 2019) on widely used NLG benchmarks: XSum (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) (see details in Appendix E.3).

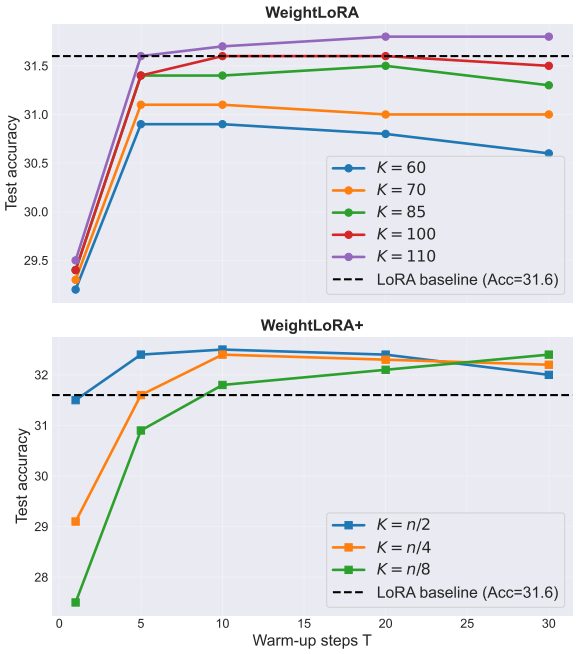

Figure 4: Ablation study on the warm-up length T and sparsity level K for WeightLoRA (top) and WeightLoRA+ (bottom) on the GSM8K benchmark with Qwen3-8B ($r = 8$). Both methods demonstrate robust performance across a wide range of hyperparameters.

The experimental results (Table 6) demonstrate that WeightLoRA+ consistently achieves superior performance compared to the standard LoRA method. Second-best results are typically found to be obtained by WeightLoRA, although the number of its trainable parameters is significantly smaller.

3.5 Ablation Studies

Sensitivity to K and T. Figure 4 presents an ablation study of the warm-up length T and the sparsity level K for both WeightLoRA and WeightLoRA+. Overall, the results indicate that our framework is highly robust, as test accuracy remains close to or above the LoRA baseline across a wide range of K and T values, with only mild variations.

For WeightLoRA, we observe that increasing the warm-up horizon T can lead to a slight degradation in performance when K is small. Intuitively, if only a few adapters are allowed to remain active, a longer warm-up phase provides the sparse weight optimization with more opportunities to overfit. In contrast, WeightLoRA+ does not exhibit this behavior: when K decreases, the rank of the remaining adapters is increased, therefore the effective capacity of the selected subset grows rather than shrinks.

These robustness patterns are consistent across all datasets and architectures considered in our ex-

Table 5: Results of fine-tuning DeBERTaV3-base on SQuAD v1.1 and v2.0 datasets. All experiment details are provided in Appendix E.3.

Method	Metric	SQuADv1.1				SQuADv2.0			
		$r=1$	$r=2$	$r=4$	$r=8$	$r=1$	$r=2$	$r=4$	$r=8$
LoRA	F1-score	90.01	92.05	92.17	92.29	80.19	82.98	84.40	85.19
	# Params	0.07%	0.13%	0.26%	0.51%	0.07%	0.13%	0.26%	0.51%
WLoRA	F1-score	89.12	89.12	92.08	92.07	80.37	82.73	84.23	85.19
	# Params	**0.04%**	**0.07%**	**0.13%**	**0.25%**	**0.04%**	**0.07%**	**0.13%**	**0.25%**
WLoRA+	F1-score	**91.37**	**92.20**	**92.48**	**92.61**	**80.65**	**83.13**	**84.86**	**85.49**
	# Params	0.07%	0.13%	0.26%	0.51%	0.07%	0.13%	0.26%	0.51%

Table 6: Results of fine-tuning BART-large on XSum and CNN/DailyMail datasets. All experiment details are provided in Appendix E.3.

Method	Metric	XSum				CNN/DailyMail			
		$r=1$	$r=2$	$r=4$	$r=8$	$r=1$	$r=2$	$r=4$	$r=8$
LoRA	Rogue1	35.23	35.59	35.96	36.07	24.43	24.50	25.16	25.28
	# Params	0.13%	0.27%	0.53%	1.05%	0.13%	0.27%	0.53%	1.05%
WLoRA	Rogue1	35.63	36.11	36.27	36.56	24.43	24.43	24.67	24.96
	# Params	**0.07%**	**0.14%**	**0.26%**	**0.52%**	**0.07%**	**0.14%**	**0.26%**	**0.52%**
WLoRA+	Rogue1	**35.82**	**36.37**	**36.45**	**36.74**	**24.45**	**24.72**	**25.31**	**25.72**
	# Params	0.13%	0.27%	0.53%	1.05%	0.13%	0.27%	0.53%	1.05%

periments. In practice, one can adopt a simple default configuration without per-task tuning. For both WeightLoRA and WeightLoRA+, one can set the warm-up length to $T = 0.5\%$ of the total training iterations, and the sparsity level can be chosen as $K = n/2$. These default settings consistently deliver strong performance throughout our experiments and can be used as reliable, off-the-shelf hyperparameters.

Selection stability. The Jaccard similarity of the top-K adapter sets across 5 random seeds on DeBERTaV3-base / SST-2 with $K = 10$ is shown in Figure 5. Most seed pairs achieve a similarity of 1.0, and the minimum is 0.82, a difference of 1 adapter out of 10. This selection identifies structurally important layers rather than reflecting noise.

Ablation summary. We perform a series of ablations to validate our design choices (due to space constraints, results are reported in Appendix D):

- **Optimized selection:** Random adapters disabling is consistently worse than WeightLoRA, showing the necessity of optimizing ω.
- **Memory efficiency:** WeightLoRA reduces GPU memory relative to LoRA, while WeightLoRA+ matches LoRA.
- **Training time:** Both methods match LoRA in wall-clock time, adding negligible overhead for adapter selection.

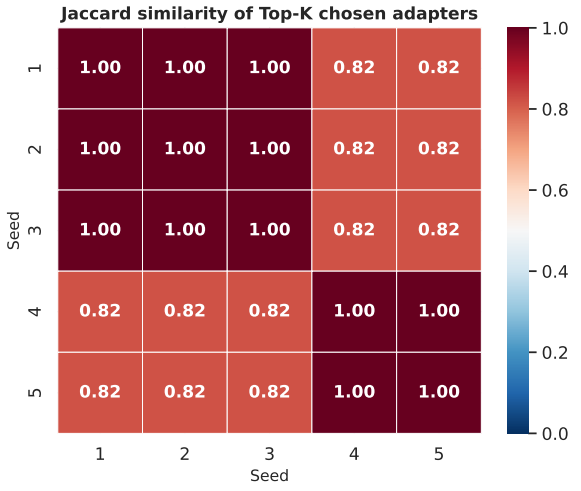

Figure 5: Jaccard similarity of selected adapter sets across 5 seeds (DeBERTaV3-base, SST-2, $K = 10$). Most pairs select identical sets; the minimum similarity is 0.82.

4 Related Work on Dynamic LoRA Rank

We focus here on the methods that are close to our approach: dynamic choice of the LoRA adapters and provide an overview of other PEFT approaches in Appendix A.1.

SVD-based approaches. AdaLoRA (Zhang et al., 2023b) performs rank selection during training using SVD to update weight matrices. As a result, it better distributes the trainable weights over lay-

ers. FlexLoRA (Wei et al., 2025) also employs the SVD approach, but it operates in a federated setting. The algorithm sends truncated decompositions to each client based on their budget, allowing for resource-based rank allocation. Another SVD-based approach, EVA (Paischer et al., 2024), focuses on initializing LoRA adapters from activations. It performs SVD on intermediate representations and selects the Top-K singular vectors, using them to initialize the low-rank subspace.

Such SVD-based methods introduce additional computational overhead and couple adapter allocation to decomposition-specific heuristics.

Mini-ensemble approaches. MELoRA (Ren et al., 2024) freezes the pretrained weights and trains a set of mini LoRA adapters, leveraging their diversity to improve generalization while reducing memory usage. Related approaches follow a similar ensemble-style philosophy, either by maintaining multiple parallel adapters or by tying and reusing low-rank components across layers, e.g., Tied-LoRA (Renduchintala et al., 2024).

These approaches require training and maintaining multiple adapters throughout fine-tuning, thereby increasing optimization complexity and departing from the single-adapter LoRA setup.

Score-based approaches. IncreLoRA (Zhang et al., 2023a) increases ranks based on the importance scores computed as an average of the scores of all parameters of the update matrix. This method also adds regularization terms that force these matrices to be orthogonal Dynamic LoRA (Mao et al., 2024), removes adapters based on their importance to specific tasks during training. It also treats high-rank layers as a combination of single-rank components. ALoRA (Liu et al., 2024b) utilizes the diagonal matrix of gate units for each rank. During training, the rank gates are set to zero for ranks with negative contributions, and the pruned rank budget is added to other weights. LoRA-drop (Zhou et al., 2025) evaluates the importance of LoRA based on the product of the LoRA parameter and hidden state, dropping the adapters with small output. PrunedLoRA (Yu et al., 2025) combines LoRA with structured pruning. It assigns an importance score to adapter parameters and prunes columns of the low-rank matrices based on this score.

While such methods yield compact adapters, they tightly couple fine-grained, score-based pruning with the optimization dynamics, thereby increasing sensitivity to pruning schedules and thresholds and often necessitating additional hyperparameter tuning.

Routing. A closely related line of work studies the routing of task-specific capacity across layers (Zhang et al., 2025; Gao et al., 2025). For example, D-MoLE (Ge et al., 2025) treats each layer as an expert and uses binary routing variables $I_t \in \{0, 1\}$ to decide, for each training step or task, which layers are active. Similarly, the method (Gong et al., 2025) treats not only adapter weights but also their structure as learnable, introducing differentiable gating and sparsity-inducing variables that jointly optimize adapter placement and routing.

While highly flexible, routing-based methods require maintaining and optimizing a layer-wise routing policy throughout training, thereby increasing architectural complexity and tightening the coupling between optimization and model structure.

Rank-learning approaches. ARD-LoRA (Shinwari and Usama, 2025) learns rank allocation continuously throughout training, keeping rank decisions coupled to gradient updates until convergence and requiring a fixed (typically large) batch size for the full run. Flexi-LoRA (Li et al., 2025) adjusts ranks per-input at both training and inference time. On complex inputs, it may still activate the maximum rank, so parameter savings are not guaranteed across all inputs.

Summary. Overall, existing approaches introduce additional architectural, optimization, or decomposition-specific complexity by tightly coupling adapter selection to the training dynamics. In contrast, the extant literature does not consider adapter selection in its most direct form, namely via explicit optimization-based selection during a short phase, followed by standard LoRA fine-tuning.

5 Conclusion

In this paper, we propose the WeightLoRA framework, which assigns a trainable weight to each LoRA layer and adaptively keeps only the most important ones during a short warm-up phase. Building on this idea, we introduce two methods, WeightLoRA and WeightLoRA+, that integrate seamlessly with standard LoRA and its variants. Across a wide range of models and benchmarks, we show that these methods significantly improve the parameter–performance trade-off of PEFT: WeightLoRA matches or closely tracks classical LoRA with 2–3$\times$ fewer trainable parameters, while WeightLoRA+ consistently outperforms LoRA under the same parameter budget.

Ethical Considerations

This work focuses on enhancing the parameter efficiency of fine-tuning large language models, without introducing new model architectures, training objectives, or application domains. As such, the proposed methods do not pose ethical risks beyond those already associated with the underlying pretrained models and the fine-tuning datasets.

Improved parameter efficiency may lower the computational and memory requirements for adapting large models, potentially increasing accessibility to such technologies. At the same time, this can also facilitate wider deployment of language models, including in contexts where misuse is possible. Addressing these broader societal risks remains outside the scope of this work and depends on responsible dataset curation, deployment practices, and the design of downstream applications.

We do not use sensitive personal data in our experiments, and all evaluations are conducted on established, publicly available benchmarks. Any ethical considerations related to data collection, annotation, or model deployment are inherited from these existing resources.

Limitations

While the proposed `WeightLoRA` and `WeightLoRA+` frameworks demonstrate strong empirical performance and improved parameter efficiency across a wide range of models and tasks, several limitations should be noted.

First, the adapter selection mechanism relies on a short warm-up phase during which importance weights are optimized. Although our experiments show that the method is robust to the choice of the warm-up length and sparsity level, this phase still introduces additional hyperparameters whose optimal values may depend on the model architecture or task. In highly non-stationary or extremely low-resource training regimes, the quality of early importance estimates can impact the final selection.

Second, our approach performs selection at the granularity of entire adapters attached to predefined layers. As a result, it does not capture finer-grained structure within individual adapters, such as rank-wise or parameter-level importance. Methods that operate at a more granular level may achieve further compression, albeit at the cost of increased optimization complexity.

Third, `WeightLoRA` assumes a fixed adapter placement determined before training and focuses solely on selecting among these candidates. It does not address the problem of dynamically inserting adapters into previously unmodified layers or altering the underlying model architecture, which may be beneficial in some settings.

Finally, while `WeightLoRA+` preserves the overall parameter budget of standard `LoRA` by reallocating rank capacity, it assumes that increasing rank on selected adapters is an effective way to utilize the freed capacity. Alternative forms of capacity reallocation may be more suitable for specific architectures or tasks and are not explored in this work.

We view these limitations as directions for future research rather than fundamental shortcomings of the proposed framework.

Acknowledgments

The work was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-15-2025-013, dated June 20, 2025, IGK 000000C313925P4B0002).

References

AI@Meta. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Vladimir Bogachev, Vladimir Aletov, Alexander Molozhavenko, Denis Bobkov, Vera Soboleva, Aibek Alanov, and Maxim Rakhuba. 2025. Lora meets riemannion: Muon optimizer for parametrization-independent low-rank adapters. *arXiv preprint arXiv:2507.12142*.

Peter Bühlmann and Sara Van De Geer. 2011. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media.

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

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

William de Vazelhes, Hualin Zhang, Huimin Wu, Xiaotong Yuan, and Bin Gu. 2022. Zeroth-order hard-thresholding: gradient error vs. expansivity. *Advances in Neural Information Processing Systems*, 35:22589–22601.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, and 1 others. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Ruibo Liu, Baochen Sun, Yawen Zhang, Daiyi Peng, Xiaoyuan Guo, and VS Subrahmanian. 2025. Mola: Moe lora with layer-wise expert allocation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5097–5112.

Chendi Ge, Xin Wang, Zeyang Zhang, Hong Chen, Jiapei Fan, Longtao Huang, Hui Xue, and Wenwu Zhu. 2025. Dynamic mixture of curriculum lora experts for continual multimodal instruction tuning. *arXiv preprint arXiv:2506.11672*.

Ming Gong, Yingnan Deng, Nia Qi, Yujun Zou, Zhihao Xue, and Yun Zi. 2025. Structure-learnable adapter fine-tuning for parameter-efficient large language models. In *IET Conference Proceedings CP944*, volume 2025, pages 225–230. IET.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Anna R Karlin. 2018. Alternating minimization, scaling algorithms, and the null-cone problem from invariant theory. In *Proceedings of 9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94, page 24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

Mike Lewis. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Xingguo Li, Raman Arora, Han Liu, Jarvis Haupt, and Tuo Zhao. 2016. Nonconvex sparse learning via stochastic optimization with progressive variance reduction. *arXiv preprint arXiv:1605.02711*.

Yang Li, Shaobo Han, and Shihao Ji. 2024. VB-LoRA: Extreme parameter efficient fine-tuning with vector banks. *arXiv preprint arXiv:2405.15179*.

Zongqian Li, Yixuan Su, Han Zhou, Zihao Fu, and Nigel Collier. 2025. Flexi-lora: Efficient lora finetuning with input-adaptive dynamic ranks. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

Cheng Lin, Lujun Li, Dezhi Li, Jie Zou, Wei Xue, and Yike Guo. 2024. NoRA: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv preprint arXiv:2408.10280*.

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

Zequan Liu, Jiawen Lyn, Wei Zhu, and Xing Tian. 2024b. ALoRA: Allocating low-rank adaptation for fine-tuning large language models. 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 622–641.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. 2024. DoRA: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. *arXiv preprint arXiv:2405.17357*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Nam Nguyen, Deanna Needell, and Tina Woolf. 2014. Linear convergence of stochastic iterative greedy algorithms with sparse constraints. *Preprint*, arXiv:1407.0088.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Preprint*, arXiv:1912.01703.

P Rajpurkar. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

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

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten Rijke, Zhumin Chen, and Jiahuan Pei. 2024. MELoRA: mini-ensemble low-rank adapters for parameter-efficient fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3052–3064.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2023. Tied-LoRA: Enhacing parameter efficiency of lora with weight tying. *arXiv preprint arXiv:2311.09578*.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2024. 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.

Jie Shen and Ping Li. 2018. A tight bound of hard thresholding. *Journal of Machine Learning Research*, 18(208):1–42.

Haseeb Ullah Khan Shinwari and Muhammad Usama. 2025. Ard-lora: Dynamic rank allocation for parameter-efficient fine-tuning of foundation models with heterogeneous adaptation needs. *IEEE Transactions on Artificial Intelligence*.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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

Chenxing Wei, Yao Shu, Ying Tiffany He, and Fei Yu. 2025. Flexora: Flexible low-rank adaptation for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14643–14682.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Diyuan Wu, Ionut-Vlad Modoranu, Mher Safaryan, Denis Kuznedelev, and Dan Alistarh. 2024. The iterative optimal brain surgeon: Faster sparse recovery by leveraging second-order information. *arXiv preprint arXiv:2408.17163*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Xin Yu, Cong Xie, Ziyu Zhao, Tiantian Fan, Lingzhou Xue, and Zhi Zhang. 2025. Prunedlora: Robust gradient-based structured pruning for low-rank adaptation in fine-tuning. *arXiv preprint arXiv:2510.00192*.

Xiaotong Yuan and Ping Li. 2021. Stability and risk bounds of iterative hard thresholding. In *International conference on artificial intelligence and statistics*, pages 1702–1710. PMLR.

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

Dacao Zhang, Kun Zhang, Shimao Chu, Le Wu, Xin Li, and Si Wei. 2025. More: A mixture of low-rank experts for adaptive multi-task learning. *arXiv preprint arXiv:2505.22694*.

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

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

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

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

Pan Zhou, Xiaotong Yuan, and Jiashi Feng. 2018. Efficient stochastic gradient hard thresholding. *Advances in Neural Information Processing Systems*, 31.

A LoRA Background

For completeness, we briefly recall the original LoRA framework (Hu et al., 2021) and fix notation. Given a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$, LoRA introduces a trainable low-rank update $\Delta W \in \mathbb{R}^{d\times k}$. Therefore, only ΔW is optimized during fine-tuning while W remains frozen. The update is parametrized as

$$\Delta W = AB,$$

where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$, and the rank r is chosen to be small, $r \ll \min(d, k)$, to reduce the number of trainable parameters. This low-rank decomposition enables task-specific information to be encoded in the matrices A and B without modifying the original backbone weights W.

Given an input $x \in \mathbb{R}^k$, the forward computation of the corresponding layer under LoRA becomes

$$h(x) = Wx + \Delta Wx = Wx + ABx.$$

Thus, the pre-trained transformation Wx is augmented with an additive low-rank correction ABx, learned from task-specific data.

LoRA scaling (ω). In practical implementations, LoRA introduces a scalar *scaling factor* $\omega > 0$ to control the magnitude of the low-rank update relative to the frozen backbone. A common parametrization is

$$\Delta W = \frac{\omega}{r} AB,$$

therefore

$$h(x) = Wx + \frac{\omega}{r} ABx.$$

The division by r keeps the variance of the update approximately stable as the rank changes, which simplifies hyperparameter tuning across different values of r. In the main text, we often omit the explicit factor ω/r for notational simplicity, but it is present in our implementation.

LoRA dropout. To regularize the low-rank path and mitigate overfitting, LoRA typically applies dropout to the intermediate low-rank representation. Concretely, one first computes

$$z = Bx \in \mathbb{R}^r,$$

then applies a dropout mask with probability p_{drop},

$$\tilde{z} = \text{Dropout}(z; p_{\text{drop}}),$$

and finally obtains the update

$$\Delta Wx = \frac{\omega}{r} A\tilde{z}.$$

The dropout probability p_{drop} (often referred to as `lora_dropout` in implementations) is a hyperparameter that trades off robustness and effective capacity of the low-rank adapters.

A.1 Related Work on LoRA and Its Variants

Low-Rank Adaptation (LoRA). Traditional LoRA (Hu et al., 2021) is one of the first successful applications of PEFT to LLMs. It avoids full supervised fine-tuning by introducing low-rank matrix decompositions on top of frozen backbone weights. Concretely, for a pre-trained matrix W, LoRA learns an additive low-rank update $\Delta W = AB$. Unfortunately, LoRA requires careful selection of these hyperparameters, especially the per-layer ranks. As a result, constant rank is typically used across all layers, resulting in a relatively high number of trainable parameters. However, the resulting accuracy can still lag behind that of full fine-tuning.

Parameter sharing and tied adapters. Several techniques aim to reduce the number of trainable parameters by sharing low-rank components across layers, without necessarily reducing training time. VeRA (Kopiczko et al., 2023) employs a single pair of low-rank matrices shared across all layers and learns only small, layer-specific scaling vectors. Tied-LoRA (Renduchintala et al., 2023) combines selective training with weight tying of low-rank matrices across layers. While such approaches substantially reduce the number of trainable parameters, they often lead to a noticeable drop in accuracy when the effective rank is very small, highlighting

the importance of allocating sufficient capacity to critical layers.

Another way to share parameters across modules and layers is VB-LoRA (Li et al., 2024). Inspired by a "divide-and-share" paradigm, it utilizes rank-one decompositions combined with a shared vector bank, dividing vectors into sub-vectors that are reused across layers and modules. A differentiable top-k selection mechanism then selects task-specific components from this bank. VB-LoRA achieves competitive performance across diverse tasks under tight storage budgets, but introduces additional hyperparameters (e.g., bank size, selection sparsity) and requires careful tuning to be effective.

Weight decomposition and SVD-based approaches. DoRA (Weight-Decomposed LoRA) (Liu et al., 2024a) decomposes each weight matrix into magnitude and direction components, applying low-rank adaptation only to the directional part. This design can provide better control over the magnitude of updates and often yields strong empirical performance, but it increases training-time overhead compared to standard LoRA. NoRA (Lin et al., 2024) uses singular value decomposition (SVD) and a dual-structure parameterization: an outer, low-rank layer with fixed weights and an inner layer that introduces fine-grained, task-specific adjustments. MiLoRA (Wang et al., 2024) also relies on SVD, but focuses on training only the minor singular components, effectively concentrating adaptation on directions that the pre-trained model underutilizes.

Optimization and geometry-aware LoRA. Beyond architectural modifications, several works explore improved optimization schemes for LoRA-style adapters (Zhang and Pilanci, 2024; Bogachev et al., 2025). Geometry-aware and Riemannian optimization methods have been proposed to better respect the low-rank structure of A and B, for example, by constraining updates to low-dimensional manifolds or factorized parameter spaces. Such methods can stabilize training and, in some cases, improve convergence for small ranks, but they further complicate the optimization pipeline and may increase implementation complexity.

Summary. Overall, existing LoRA variants trade off parameter count, training overhead, and robustness to hyperparameter choices. Many approaches reduce the number of trainable parameters by sharing or decomposing them, but at the cost of more complex designs or additional tuning knobs. Our methods, WeightLoRA and WeightLoRA+, take a complementary direction by keeping the basic LoRA parameterization (Appendix A) and instead focusing on principled adapter selection and rank allocation, which can be integrated into standard LoRA pipelines with minimal modification.

B Optimization of the Importance Weights

In this appendix, we briefly summarize the optimization scheme used for the importance vector ω in WeightLoRA. Recall that the outer problem in (2) can be seen as an ℓ_0-constrained minimization over a low-dimensional vector:

$$\min_{\omega \in \mathbb{R}^n} F(\omega) \quad \text{s.t.} \quad \|\omega\|_0 \leq K,$$

where $F(\omega)$ denotes the task loss after updating the adapter parameters $\{A^i, B^i\}_{i=1}^n$.

StoIHT. To handle the sparsity constraint, we employ the stochastic iterative hard thresholding (StoIHT) algorithm of Nguyen et al. (2014). In our setting, each update of ω during the warm-up phase consists of two simple steps:

1. *Stochastic gradient step:* given the current iterate ω_t, we compute a stochastic gradient g_t of $F(\omega)$ on a mini-batch and form an intermediate point

$$\tilde{\omega}_{t+1} = \omega_t - \eta_t g_t,$$

where η_t is a step size.

2. *Hard thresholding:* we then apply the hard-thresholding operator $\mathcal{H}_K$ that keeps only the K entries of largest magnitude and sets all others to zero:

$$\omega_{t+1} = \mathcal{H}_K(\tilde{\omega}_{t+1}).$$

In practice, this coincides with the "Top-K + zeroing" operation, and enforces $\|\omega_{t+1}\|_0 \leq K$ at every iteration.

Relation to other ℓ_0 methods. Iterative hard-thresholding algorithms and their variants have been extensively studied in the context of high-dimensional sparse learning, with stability and risk guarantees under suitable conditions (Bühlmann and Van De Geer, 2011; Yuan and Li, 2021;

Table 7: Ablation study about random adapters disabling. The results are reported as the average of five runs with different random seeds. The best results are shown in **bold**.

# Adapters	Method	MNLI Acc	SST-2 Acc	CoLA Mcc	QQP Acc/F1	QNLI Acc	RTE Acc	MRPC Acc/F1	ALL Avg
$K = 1$	WLoRA	0.7912	0.8876	0.5624	0.8307/0.7813	0.8724	0.5271	0.8309/0.8825	**0.7862**
	RLoRA	0.5124	0.4908	0.0115	0.6318/0.4265	0.4946	0.5271	0.3162/0.3532	0.4204
$K = 5$	WLoRA	0.8638	0.9427	0.6491	0.8685/0.8292	0.9136	0.5560	0.8799/0.9148	**0.8388**
	RLoRA	0.7260	0.9178	0.5217	0.8396/0.8001	0.8640	0.7160	0.8440/0.8905	0.7855
$K = 10$	WLoRA	0.8706	0.9473	0.6703	0.8728/0.8362	0.9167	0.5343	0.8775/0.9101	**0.8454**
	RLoRA	0.8209	0.9300	0.5368	0.8539/0.8110	0.8875	0.7316	0.8529/0.8950	0.8137

de Vazelhes et al., 2022; Wu et al., 2024). More sophisticated greedy schemes for ℓ_0-constrained problems, such as stochastic gradient matching pursuit (StoGradMP) (Nguyen et al., 2014; Li et al., 2016; Zhou et al., 2018; Shen and Li, 2018), introduce additional support-selection and debiasing steps that can improve constants in theoretical bounds at the cost of higher memory and computational overhead. Since the dimension of ω is relatively small and the warm-up phase is short, we found the simpler StoIHT updates to be sufficient in practice and did not observe consistent benefits from employing more complex sparse optimization routines.

C Memory Savings from Adapter Selection

In this appendix, we demonstrate how reducing the number of active LoRA adapters results in practical GPU memory savings.

If we look at the relationship between GPU memory utilization and the number of enabled adapters (see Figure 6), we can expect a significant reduction in the memory required for model training by disabling several LoRA heads, provided that competitive performance is maintained.

As a concrete example, consider training a DeBERTaV3-base model on a single NVIDIA V100 16GB GPU, which is a realistic setting for fine-tuning a small (or a larger, quantized) model on a downstream task. According to the dashed line in Figure 1, this configuration allows for at most twenty adapters, even for a minimal rank $r = 1$. If these adapters are spread across random layers, the probability of selecting exactly those LoRA heads that require the most extensive fine-tuning is low. Consequently, the performance of the resulting model is substantially lower than that of classical LoRA (see Section D for a detailed comparison).

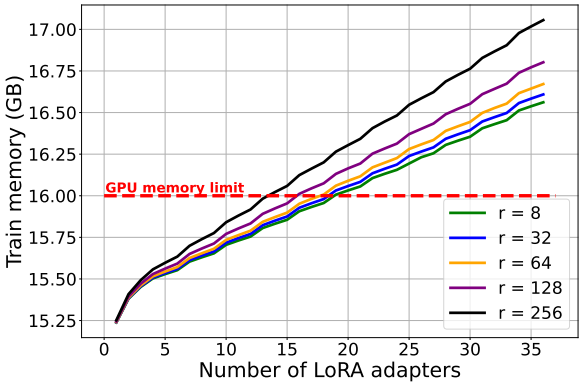

Figure 6: Dependence of the required GPU memory on the number of connected LoRA adapters. The red dashed line indicates the memory capacity of the NVIDIA V100 16GB.

Conversely, if only five adapters are placed in optimal locations (which also allows higher ranks), we can achieve gains in memory and time without sacrificing performance.

D Ablation Studies

In this section, we validate the key design choices of WeightLoRA and WeightLoRA+ through a series of ablation experiments. We focus on three main aspects: (i) the necessity of optimized adapter selection via (2), and (ii) the computational efficiency in terms of memory and training time.

Importance of optimized selection. We first show that the selection of adapters by rule (2) is essential in the WeightLoRA technique. We implement the same method as WeightLoRA, except that we do not disable the adapters by an optimized weight ω, but randomly. In this paper, we refer to this method as RLoRA. The results comparing WeightLoRA and RLoRA in an experiment similar to Section 3.1 with fixed rank $r = 8$ are summarized in Table 7.

The results of this experiment indicate that randomly disabling the adapters yields poor performance. This again confirms that optimizing the weights ω is necessary for the precise performance of our method WeightLoRA.

Memory footprint. Figure 7 reports the peak GPU memory consumption of LoRA, WeightLoRA, and WeightLoRA+ at fixed warm-up $T = 5$. As the sparsity level K decreases, WeightLoRA progressively disables more adapters and thus reduces the effective number of trainable parameters, which leads to a clear reduction in memory relative to the full LoRA baseline. WeightLoRA+ further improves the memory–performance trade-off: despite increasing the rank of the remaining adapters, its peak memory usage remains comparable to or below that of LoRA, while preserving strong accuracy. These results confirm that selective adapter activation via (2) is not only beneficial for quality, but also for memory efficiency.

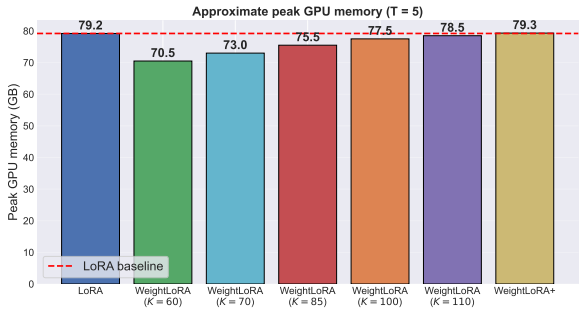

Figure 7: Peak GPU memory consumption (in GB) for LoRA, WeightLoRA, and WeightLoRA+ at fixed warm-up $T = 5$ on GSM8K with Qwen3-8B ($r = 8$). We report the memory usage of the full LoRA model and compare it with WeightLoRA for different sparsity levels K, as well as with WeightLoRA+. Both WeightLoRA and WeightLoRA+ achieve comparable or lower peak memory usage than LoRA, while maintaining competitive or better accuracy (see Figure 4). Note: WeightLoRA+ preserves the full LoRA parameter budget by design (by halving the number of adapters while doubling their rank), so its memory footprint matches that of LoRA, and the two curves overlap in the figure.

Training time. Figure 8 compares the wall-clock training time of LoRA, WeightLoRA, and WeightLoRA+ under the same training schedule with warm-up $T = 5$. The measured times are almost identical across all methods, with only minor fluctuations attributable to run-to-run variability. This suggests that the additional computations required to optimize the weights ω and select adapters in WeightLoRA and WeightLoRA+ incur negligible overhead in practice. In particular, our techniques can be used as drop-in replacements for LoRA without sacrificing training speed.

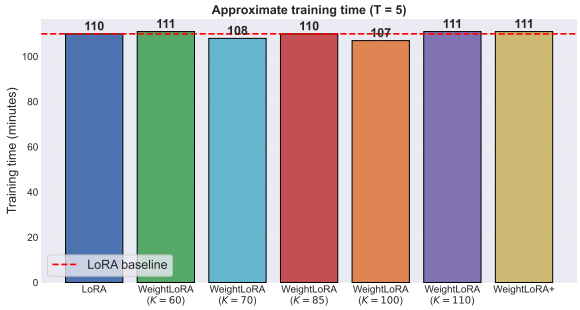

Figure 8: Training time (in minutes) for one full fine-tuning run of LoRA, WeightLoRA, and WeightLoRA+ at warm-up $T = 5$ on GSM8K with Qwen3-8B ($r = 8$). All methods have very similar wall-clock time, indicating that adapter selection in WeightLoRA and WeightLoRA+ does not introduce a noticeable overhead compared to standard LoRA.

E Experimental Details

We use PyTorch (Paszke et al., 2019) to implement our algorithms and the publicly available baselines' implementations. Our code is based on the publicly available Huggingface Transformers3 (Wolf et al., 2020) library. All the experiments are conducted on NVIDIA V100 16GB and NVIDIA A100 80GB GPUs. All the results are averaged across 5 random seeds.

E.1 Additional Details about NLU Experiment from Section 3.1

GLUE benchmark. We evaluate our methods on the General Language Understanding Evaluation (GLUE) benchmark (Wang, 2018), a standard collection of diverse natural language understanding tasks. GLUE covers a range of phenomena, including sentence-level and sentence-pair classification, entailment, paraphrase detection, and linguistic acceptability. In our experiments, we use the following tasks: MNLI (multi-genre natural language inference), SST-2 (sentiment analysis), CoLA (linguistic acceptability), QQP (paraphrase detection), QNLI (question–answer natural language inference), RTE (recognizing textual entailment), and MRPC (paraphrase detection). Performance is reported using the official metrics for each task (accuracy or Matthews correlation coefficient for CoLA).

Table 8: Hyperparameter setup for the DeBERTaV3-base model on the GLUE benchmark (self-attention adapters only) for the dynamic-rank comparison in Table 1.

Hyperparameter	**Value**
Experiment	Dynamic-rank comparison (Table 1)
Model	DeBERTaV3-base
Adapter placement	Self-attention projection matrices only
Total # adapters (n)	36
Optimizer	Adam
$\beta_1,\ \beta_2,\ \epsilon$	$0.9,\ 0.99,\ 1 \times 10^{-8}$
Weight decay	1×10^{-4}
Learning rate	Selected from $\{5 \times 10^{-5},\ 5 \times 10^{-4},\ 5 \times 10^{-3}\}$
LR scheduler	Linear
Warm-up steps	100
Max sequence length	512
LoRA precision (`dtype`)	`bfloat16`
Backbone quantization bits	32 (no quantization)
# Epochs	7 (MNLI); 10 (SST-2); 15 (CoLA); 5 (QQP, QNLI); 30 (MRPC); 50 (RTE)
Batch size	32 (MNLI, SST-2, QQP, QNLI); 64 (CoLA, MRPC, RTE)
Gradient accumulation steps	2 (MNLI, SST-2, QQP, QNLI); 1 (CoLA, MRPC, RTE)
Effective batch size	64 for all tasks
LoRA rank r	8
LoRA scaling ω	32
LoRA dropout	0.05

E.2 Additional Details about Classical LLM Benchmarks Experiment from Section 3.2

Benchmarks. To evaluate our methods on large language models and diverse reasoning tasks, we use the Qwen3-8B model (Yang et al., 2025) on a set of standard LLM benchmarks: MathQA (MQA) (Yu et al., 2023) for math word problem solving, GSM8K (Cobbe et al., 2021) for grade-school mathematical reasoning, HellaSwag (HS) (Zellers et al., 2019) for commonsense completion of narrative texts, BoolQ (BQ) (Clark et al., 2019) for yes/no question answering over passages, and ARC-Challenge (ARC-C) (Clark et al., 2018) for multiple-choice science questions.

E.3 Additional Details about QA and NLG Experiment from Sections 3.3 and 3.4

Benchmarks. For question-answering (QA), we use SQuAD v1.1 and v2.0. SQuAD v1.1 consists of extractive QA pairs where the answer is always a span in the given context paragraph. SQuAD v2.0 extends this setting by adding unanswerable questions, requiring the model to both extract spans and detect when the context supports no answer. For natural language generation (NLG), we consider two summarization benchmarks: XSum, which focuses on highly abstractive, single-sentence summaries of BBC news articles, and CNN/DailyMail, which provides longer news articles paired with multi-sentence summaries.

Use of AI Assistants

The authors utilized AI-based tools to enhance the clarity and presentation of the manuscript text and to facilitate code development. The tools were used solely for language polishing and implementation assistance. All research ideas, methodological design, experimental setup, results, and conclusions were developed, verified, and validated by the authors.

Table 9: Hyperparameter setup for the DeBERTaV3-base model on the GLUE benchmark for the comparison with tuned LoRA baselines (Table 2).

Hyperparameter	**Value**
Experiment	Comparison with tuned LoRA baselines (Table 2)
Model	DeBERTaV3-base
Adapter placement	All linear layers
Total # adapters (n)	72
Optimizer	Adam
$\beta_1,\ \beta_2,\ \epsilon$	$0.9,\ 0.99,\ 1 \times 10^{-8}$
Weight decay	1×10^{-4}
Learning rate	Selected from $\{10^{-6},\ 3 \cdot 10^{-5},\ 5 \cdot 10^{-5},\ 8 \cdot 10^{-5},\ 10^{-4},\ 3 \cdot 10^{-4},\ 5 \cdot 10^{-4},\ 8 \cdot 10^{-4},\ 10^{-3},\ 5 \cdot 10^{-3}\}$
LR scheduler	Linear
Warm-up steps	Selected from $\{0,\ 20,\ 50,\ 70,\ 100,\ 200\}$
Max sequence length	512
LoRA precision (`dtype`)	`bfloat16`
Backbone quantization bits	32 (no quantization)
# Epochs	7 (MNLI); 10 (SST-2); 15 (CoLA); 5 (QQP, QNLI); 30 (MRPC); 50 (RTE)
Batch size	32 (MNLI, SST-2, QQP, QNLI); 64 (CoLA, MRPC, RTE)
Gradient accumulation steps	6 (MNLI, SST-2, QQP, QNLI); 3 (CoLA, MRPC, RTE)
Effective batch size	192 for all tasks
LoRA rank r	$1, 2, 4, 8$
LoRA scaling ω	16
LoRA dropout	0.1

Table 10: Hyperparameter setup for the Llama 3 8B model on the GLUE benchmark for the experiment in Table 3.

Hyperparameter	**Value**
Experiment	Llama 3 7B GLUE comparison (Table 3)
Model	Llama 3 7B
Adapter placement	All linear layers
Total # adapters (n)	224
Optimizer	Adam
$\beta_1,\ \beta_2,\ \epsilon$	$0.9,\ 0.99,\ 1 \times 10^{-8}$
Weight decay	1×10^{-4}
Learning rate	Same grid as in Table 9: $\{10^{-6},\ 3 \cdot 10^{-5},\ 5 \cdot 10^{-5},\ 8 \cdot 10^{-5},\ 10^{-4},\ 3 \cdot 10^{-4},\ 5 \cdot 10^{-4},\ 8 \cdot 10^{-4},\ 10^{-3},\ 5 \cdot 10^{-3}\}$
LR scheduler	Linear
Warm-up steps	Selected from $\{10,\ 25,\ 30,\ 50\}$
Max sequence length	512
LoRA precision (`dtype`)	`bfloat16`
Backbone quantization bits	8
# Training steps	256 (MNLI, QNLI, RTE, STS-B); 128 (SST-2); 512 (MRPC); 1024 (CoLA, QQP)
Batch size	16 (MNLI, CoLA, QNLI, RTE, STS-B); 32 (SST-2, MRPC); 8 (QQP)
Gradient accumulation steps	2 (MNLI, CoLA, QNLI, RTE, STS-B); 1 (SST-2, MRPC); 4 (QQP)
Effective batch size	32 for all tasks
LoRA rank r	$1, 2, 4, 8$
LoRA scaling ω	32
LoRA dropout	0.05

Table 11: Hyperparameter setup for the Qwen3-8B model on classical LLM benchmarks experiment from Section 3.2.

Hyperparameter	**Value**
Experiment	Evaluation on Classical LLM Benchmarks (Table 4)
Model	Qwen3-8B
Adapter placement	All linear
Total # adapters (n)	253
Optimizer	Adam
$\beta_1,\ \beta_2,\ \epsilon$	$0.9,\ 0.99,\ 1 \times 10^{-8}$
Weight decay	1×10^{-4}
Learning rate	Selected from $\{5 \cdot 10^{-5},\ 10^{-4},\ 2 \cdot 10^{-4},\ 3 \cdot 10^{-4},\ 4 \cdot 10^{-4},\ 5 \cdot 10^{-4},\ 10^{-3}\}$
LR scheduler	Linear
Warm-up steps	500
Max train steps	5000
Max sequence length	512
Backbone precision (`dtype`)	`bfloat16`
Backbone quantization bits	8
Batch size	32
Gradient accumulation steps	1
Effective batch size	32
LoRA rank r	$2, 4, 8, 16$
LoRA scaling ω	32
LoRA dropout	0.05

Table 12: Hyperparameter setup for SQuAD v1.1, SQuAD v2.0, XSum, and CNN/DailyMail benchmarks (DeBERTa-based QA and NLG experiments from Sections 3.3 and 3.4).

Hyperparameter	**Value**
Experiment	QA and NLG experiments (Tables 5 and 6)
Model	DeBERTaV3-base (QA) ; BART-large (NLG)
Adapter placement	All linear layers
Total # adapters (n)	72
Optimizer	Adam
$\beta_1,\ \beta_2,\ \epsilon$	$0.9,\ 0.99,\ 1 \times 10^{-8}$
Weight decay	1×10^{-4}
Learning rate	Selected from $\{10^{-6},\ 3 \cdot 10^{-5},\ 5 \cdot 10^{-5},\ 8 \cdot 10^{-5},\ 10^{-4},$ $3 \cdot 10^{-4},\ 5 \cdot 10^{-4},\ 8 \cdot 10^{-4},\ 10^{-3}\}$
LR scheduler	Linear
Warm-up steps	Selected from $\{0, 20, 50, 70, 100, 200\}$
Max sequence length	1024 (QA) ; 2048 (NLG)
Backbone precision (`dtype`)	`bfloat16`
Backbone quantization bits	32 (no quantization)
# Epochs	10 (SQuAD v1.1); 15 (SQuAD v2.0); 35 (XSum); 25 (CNN/DailyMail)
Batch size	32 (SQuAD v1.1, SQuAD v2.0); 8 (XSum, CNN/DailyMail)
Gradient accumulation steps	2 (SQuAD v1.1, SQuAD v2.0); 16 (XSum, CNN/DailyMail)
Effective batch size	64 (all datasets)
LoRA rank r	$1, 2, 4, 8$
LoRA scaling ω	32
LoRA dropout	0.05