AROMA: Autonomous Rank-one Matrix Adaptation

Anonymous ACL submission

Abstract

As large language models continue to grow in size, parameter-efficient fine-tuning (PEFT) has become increasingly crucial. While low-rank adaptation (LoRA) offers a solution through low-rank updates, its static rank allocation may yield suboptimal results. Adaptive low-rank adaptation (AdaLoRA) improves this with dynamic allocation but remains sensitive to initial and target rank configurations. We introduce AROMA, a framework that automatically constructs layer-specific updates by iteratively building up rank-one components with very few trainable parameters that gradually diminish to zero. Unlike existing methods that employ rank reduction mechanisms, AROMA introduces a dual-loop architecture for rank growth. The inner loop extracts information from each rank-one subspace, while the outer loop determines the number of rank-one subspaces, i.e., the optimal rank. We reset optimizer states to maintain subspace independence. AROMA significantly reduces parameters compared to LoRA and AdaLoRA while achieving superior performance on natural language understanding and commonsense reasoning tasks, offering new insights into adaptive PEFT.

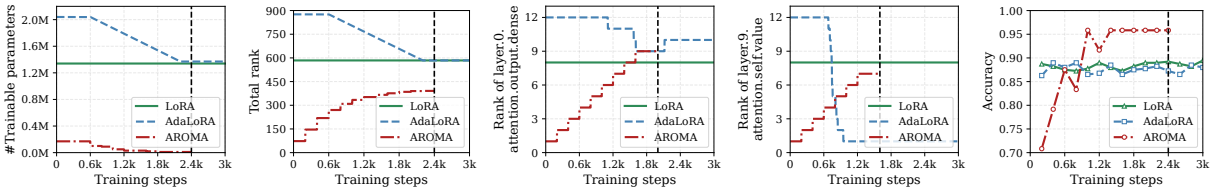
1 Introduction

The emergence of large language models (LLMs) (Devlin et al., 2019; OpenAI, 2023; Meta, 2024a; Liu et al., 2024a) has revolutionized the field of natural language processing (NLP), yet their full potential is often limited by the substantial computational demands of fine-tuning. Traditional full-parameter tuning, while effective, becomes prohibitively expensive as model sizes escalate into hundreds of billions of parameters (Lester et al., 2021; Meng et al., 2024). For instance, LLaMA3 series boasts models with up to 400B parameters (Meta, 2024b), and DeepSeek-V3 encompasses 671B total parameters due to its mixture-of-experts architecture (Liu et al., 2024a). This challenge has driven the development of parameter-efficient fine-tuning (PEFT) methods, such as prompt-tuning (Lester et al., 2021), prefix-tuning (Li and Liang, 2021), and adapter tuning (Pfeiffer et al., 2021; Houlsby et al., 2019). Besides these, low-rank adaptation (LoRA) (Hu et al., 2022) stands out as a particularly promising approach for its simplicity and strong theoretical foundation.

LoRA learns incremental low-rank update $\Delta \boldsymbol{W}$ to pretrained model $\boldsymbol{W}_0$, without altering the model architecture or introducing additional inference latency (Hu et al., 2022). While attaining impressive parameter efficiency (typically less than 1% of full fine-runing), conventional LoRA implementations impose uniform rank allocation across all layers. This might be suboptimal, as different components of the network exhibit varying sensitivities to parameter perturbations (Zhang et al., 2023a). Moreover, determining the optimal ranks remains an empirical process that often necessitates extensive trial-and-error experimentation.

As a modified version, adaptive low-rank adaptation (AdaLoRA) (Zhang et al., 2023a) adopts dynamic rank allocation through singular value decomposition (SVD)-based importance scoring. While it improves the flexibility upon static configurations like LoRA, it still faces several limitations: 1) the need to prespecify both the initial and target rank budgets; 2) substantial computational overhead caused by relaxed SVD; and 3) rank redundancy stemming from a low effective rank proportion. Consequently, the fundamental tension between adaptive rank adjustment and computational efficiency remains an open question.

In this work, we present **A**utonomous **R**ank-**O**ne **M**atrix **A**daptation (AROMA), a novel rank-growing low-rank adaptation method that reconsiders the dynamics of rank allocation. Experimental results demonstrate that AROMA significantly outperforms both LoRA and AdaLoRA



(a) #Parameter (b) Total rank (c) Specific rank (d) Specific rank (e) Accuracy

Figure 1: Results for LoRA$_{r=8}$, AdaLoRA$_{r=8}$, and AROMA (ours) include the number of trainable parameters, total rank, rank of a specific layer and evaluation accuracy versus training step for RoBERTa-base on MRPC task. For AROMA, training of "layer.0.attention.output.dense" and "layer.9.attention.self.value" automatically terminates at 2000 and 1600 steps, respectively, while the overall training automatically stops at 2400 steps.

when applied to the RoBERTa-base (Liu et al., 2019) on the GLUE benchmark (Wang et al., 2018) and the LLaMA3-8B (Meta, 2024a) on the commonsense170K dataset (Hu et al., 2023). Notably, AROMA achieves this enhanced performance only using <10% of the parameters required by LoRA$_{r=8}$ and AdaLoRA$_{r=8}$ without prespecified rank. Main contributions are summarized as follows:

- **Adaptive Rank Growth** We propose a structure that progressively establishes layer-specific ranks with minimal and decreasing trainable parameters. Unlike AdaLoRA's pruning-based strategy, AROMA initiates with zero rank and incrementally incorporates rank-one components until convergence criteria are met. This bottom-up structure ensures high parameter efficiency without loss of informative subspaces.
- **Automatic Rank Convergence** AROMA features a dual-loop architecture for automatic rank control. Each module operates with an inner loop that extracts information from individual rank-one subspace, and an outer loop determines the number of these subspaces, i.e., the optimal rank. We design a convergence criterion for both loops, enabling each module to autonomously determine the appropriate rank without the need to predefine it.
- **Independent Subspace** We introduce a training strategy termed *Check & Merge & Reinit & Reset*, which includes convergence checking, merging converged rank-one updates, periodic optimizer resets alongside learning rate warmup. After each inner loop, the optimizer states are reset while preserving the knowledge accumulated in the weights. This facilitates subspace switching, leading to high effective rank proportion and a continuous flow of new domain knowledge.

2 Background and Motivation

LoRA (Hu et al., 2022) fine-tunes the pretrained model $\boldsymbol{W}_0 \in \mathbb{R}^{m\times n}$ by incorporating a low-rank decomposition, namely:

$$\boldsymbol{W} = \boldsymbol{W}_0 + \frac{\alpha}{r}\Delta\boldsymbol{W},\ \Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A} \tag{1}$$

where $\boldsymbol{B} \in \mathbb{R}^{m\times r}$, $\boldsymbol{A} \in \mathbb{R}^{r\times n}$ with $r \ll \min\{m, n\}$, and scaling factor α secures consistent output magnitude across different rank values. However, this approach requires careful selection of r and imposes uniform rank across all layers, potentially not optimal.

AdaLoRA (Zhang et al., 2023a) addresses these static allocation limitations by parameterizing the incremental matrix as $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{Q}$, mimicking SVD while enforcing orthogonality:

$$\begin{aligned} \Delta\boldsymbol{W} &= \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{Q}, \\ \text{s.t. } \boldsymbol{P}^T\boldsymbol{P} &= \boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}_r \end{aligned} \tag{2}$$

where $\boldsymbol{P} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Q} \in \mathbb{R}^{r\times n}$ represent left and right singular vectors while $\boldsymbol{\Lambda} \in \mathbb{R}^{r\times r}$ stores singular values. AdaLoRA begins with a high initial total rank budget and gradually reduces it at certain intervals. Specifically, singular values across all layers are sorted in descending order based on the importance score, with only the top $b^{(t)}$ retained, ultimately converging to a target rank budget. Since these singular values belong to different module weights, this mechanism enables adaptive rank allocation across modules. Nevertheless, AdaLoRA exhibits several limitations:

- Like LoRA, AdaLoRA's performance remains sensitive to the initial and target total rank configurations. Optimal rank selection is task-dependent and architecture-specific, complicating deployment in empirical scenarios.

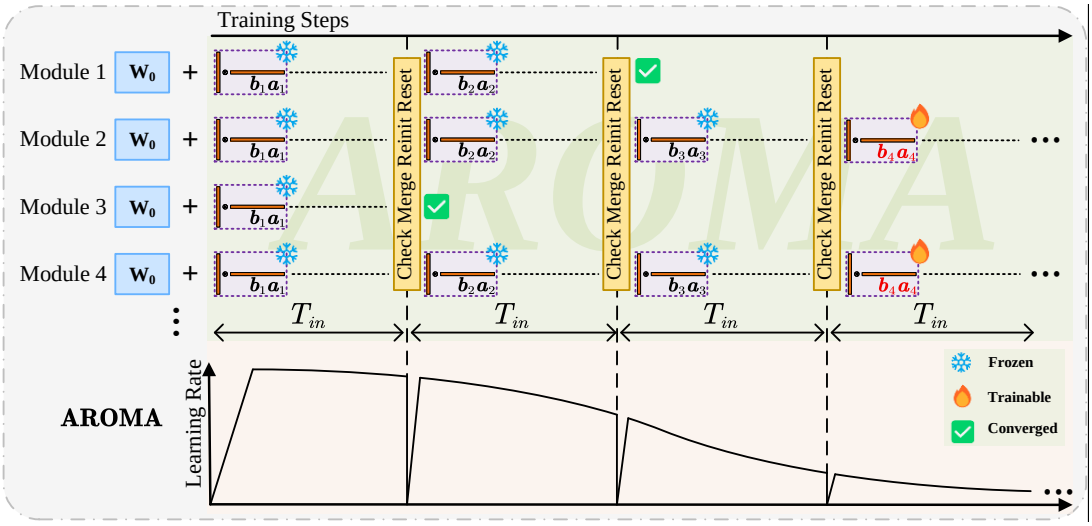


Figure 2: Workflow of ***AROMA***. For each module, AROMA trains rank-one matrices sequentially with a dual-loop architecture. In the inner loop, a rank-one LoRA, $\boldsymbol{ba}$, is updated, whose convergence is assessed by the inner stopping criterion. Prior to heading to next outer loop step, we check outer convergence by outer stopping criterion. If not converged, the computed rank-one components are merged and frozen, and new $\boldsymbol{b}$ and $\boldsymbol{a}$ are initialized for training with reset learning rate and optimizer states. For simplicity, we illustrate the length of inner loop to T_{in}, though in practice, it is determined by both T_{in} and the inner convergence criterion.

- Computing the relaxed SVD in AdaLoRA introduces substantial complexity that scales linearly with layer dimensions, creating computational bottlenecks for very large models.
- The higher initial ranks demand substantial memory allocation during early training phases, imposing practical limitations in resource-constrained environments.

Against these backdrops, we devise an automatic and adaptive rank-growing scheme inspired by rank-one matching pursuit (Wang et al., 2014, 2015). This approach leverages the principle that any rank-r matrix $\boldsymbol{L}$ can be decomposed into a sum of r rank-one matrices:

$$\boldsymbol{L} = \sum_{p=1}^{r} \boldsymbol{b}_p \boldsymbol{a}_p \tag{3}$$

where $\boldsymbol{b}_p \in \mathbb{R}^{m\times 1}$ and $\boldsymbol{a}_p \in \mathbb{R}^{1\times n}$. Building on this idea, we develop our novel framework.

3 Methodology

This section outlines two crucial aspects of AROMA: 1) the adaptive rank-growing mechanism, featuring both inner and outer stopping criteria; and 2) the training strategy known as Check & Merge & Reinit & Reset. Figure 2 depicts the AROMA framework, and Algorithm 1 in Appendix A provides the detailed steps.

3.1 Adaptive Rank Growth

Unlike AdaLoRA that truncates singular values with low important scores, we propose a rank-growing scheme which introduces a dual-loop training structure: the inner loop computes individual rank-one matrix, while the outer loop determines the quantity of these matrices. For the pth outer loop step, $\Delta\boldsymbol{W}$ is parameterized as:

$$\begin{aligned}\Delta\boldsymbol{W} &= \boldsymbol{b}_1\boldsymbol{a}_1 + \boldsymbol{b}_2\boldsymbol{a}_2 + \cdots + \boldsymbol{b}_{p-1}\boldsymbol{a}_{p-1} + \boldsymbol{b}_p\boldsymbol{a}_p \\ &= \begin{bmatrix}\boldsymbol{B}_{p-1} & \boldsymbol{b}_p\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_{p-1} \\ \boldsymbol{a}_p\end{bmatrix}\end{aligned} \tag{4}$$

where $\boldsymbol{B} \in \mathbb{R}^{m\times p}$ and $\boldsymbol{A} \in \mathbb{R}^{p\times n}$.

AROMA learns a series of rank-one LoRAs. At the beginning of the pth outer iteration, a new rank-one LoRA $\boldsymbol{b}_p\boldsymbol{a}_p$ is activated for training, while previously calculated $\boldsymbol{b}_1\boldsymbol{a}_1, \boldsymbol{b}_2\boldsymbol{a}_2, \cdots, \boldsymbol{b}_{p-1}\boldsymbol{a}_{p-1}$ are frozen and merged as a single matrix $\boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1}$.

Next, $\boldsymbol{b}_p^{(0)}$ and $\boldsymbol{a}_p^{(0)}$ enter the inner loop. Here we denote the update in the tth inner loop step as $\boldsymbol{b}_p^{(t)}$ and $\boldsymbol{a}_p^{(t)}$. They update until t reaches the maximum inner steps T_{in} or the inner stopping criterion is met:

$$\frac{\left\|\boldsymbol{b}_p^{(t)}\boldsymbol{a}_p^{(t)}\right\|_F - \left\|\boldsymbol{b}_p^{(t-\Delta T_{\text{in}})}\boldsymbol{a}_p^{(t-\Delta T_{\text{in}})}\right\|_F}{\left\|\boldsymbol{b}_p^{(t-\Delta T_{\text{in}})}\boldsymbol{a}_p^{(t-\Delta T_{\text{in}})}\right\|_F} < \varepsilon_{\text{in}} \tag{5}$$

where ε_{in} denotes the inner convergence tolerance, and ΔT_{in} is the inner checking interval. We evaluate (5) every ΔT_{in} steps, and if it is satisfied, the inner loop terminates, and the training of $\boldsymbol{b}_p\boldsymbol{a}_p$, viz., current rank-one LoRA, is completed.

When to stop? Once the inner loop ends, we check for outer loop convergence before proceeding to the next outer loop step. Here we use a relative weight change criterion between the $(p-1)$th and the pth outer steps defined as:

$$\frac{\|(\boldsymbol{W}_0 + \alpha\boldsymbol{B}_p\boldsymbol{A}_p) - (\boldsymbol{W}_0 + \alpha\boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1})\|_F}{\|\boldsymbol{W}_0 + \alpha\boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1}\|_F} = \frac{\|\alpha\boldsymbol{b}_p\boldsymbol{a}_p\|_F}{\|\boldsymbol{W}_0 + \alpha\boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1}\|_F} < \varepsilon_{\text{out}} \tag{6}$$

where ε_{out} denotes the outer convergence tolerance. If (6) is satisfied, the outer loop will terminate, viz., training of $\Delta\boldsymbol{W}$ is completed.

Since we only leverage rank-one updates, each update can be regarded as a basis spanning a rank-one matrix subspace, which encompasses different domain knowledge. In AROMA, the inner loop exploits each subspace, yielding a rank-one basis $\boldsymbol{b}_p^{(t)}\boldsymbol{a}_p^{(t)}$, while the outer loop continuously pursues new subspaces and determines the appropriate number of subspaces. This rank-growing strategy allows for continuously extraction new information while keeping only one rank-one matrix trainable at a time, securing high parameter efficiency.

Furthermore, we implement AROMA across all modules, and train them in parallel (see Figure 2). For the inner loop, each module has its own inner convergence label and advances to the next outer step when all modules have either converged or reach T_{in}. In particular, the module that converges will continue training while waiting for the others to catch up prior to proceeding together to the next outer step. Apart from facilitating rank allocation, this approach helps prevent premature termination, ensuring a more comprehensive subspace exploration.

On the other hand, each module also possesses an outer convergence label, and once a module is determined as converged according to (6), it is immediately frozen and the latest rank-one component will not be merged into it, while training continues for the remaining modules. The overall training process finishes when all modules converge or reach the maximum total training steps T. This design allows each module to determine the optimal rank independently and autonomously, enabling adaptive rank growth with a gradually reduced trainable parameters. We list the time complexity of LoRA, AdaLoRA and AROMA in Table 1, where $\tilde{r}$ denotes the current rank for AdaLoRA. Typically, we have $\mathcal{O}_{\text{AdaLoRA}} > \mathcal{O}_{\text{LoRA}} \geq \mathcal{O}_{\text{AROMA}}$. Detailed analyses and experimental verification are presented in Appendix B and Section 5.2, respectively.

Scheme	LoRA	AdaLoRA	***AROMA***
Complexity	$\mathcal{O}((m+n)r)$	$\mathcal{O}((m+n)\tilde{r})$	$\mathcal{O}((m+n)p)$

Table 1: Per-step complexity comparison

3.2 Check & Merge & Reinit & Reset

We further design a training strategy known as *Check & Merge & Reinit & Reset*. As its name implies, there are four components.

Check involves the inner and outer convergence criteria described in (5) and (6). The inner checks occur every ΔT_{in} steps, while the outer checks take place when the inner loop finishes.

Merge & Reinit where *Reinit* stands for reinitialize. As mentioned before, if (6) is met, we terminate the outer loop. Otherwise, the previously computed $\boldsymbol{b}_p\boldsymbol{a}_p$ is merged into $\boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1}$, and the training progresses to the next outer step. At this point, a new rank-one LoRA $\boldsymbol{b}_{p+1}\boldsymbol{a}_{p+1}$ is introduced, with Kaiming initialization (He et al., 2015) for $\boldsymbol{a}_{p+1}^{(0)}$ and zero for $\boldsymbol{b}_{p+1}^{(0)}$.

Reset represents optimizer state reset. With momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, Adam optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2019) tends to follow established optimization paths, as update steps are strongly influenced by previous gradients. This means that after Merge & Reinit, the previous updates still influence current learning, causing the new LoRA update to continue exploring the learned subspaces. To circumvent this, we randomly prune 99.9% of the optimizer states following each Merge & Reinit. Such an idea of subspace switching is adopted in LLM pretraining (Lialin et al., 2024; Zhao et al., 2024) and subspace learning (Larsen et al., 2022; Gur-Ari et al., 2018).

Additionally, a warmup phase is implemented at the start of training for each LoRA update to mitigate early overfitting. While the initial warmup phase is set to hundreds of steps, subsequent quick warmup phases are limited to tens of steps. The learning rate scheduler is illustrated in Figure 2.

4 Experiments

In this section, We fine-tune two LLMs of different sizes and architectures on two downstream tasks to evaluate the efficacy of AROMA. First, for natural language understanding (NLU) tasks, we fine-tune RoBERTa-base (Liu et al., 2019) on the General Language Understanding Evaluation (GLUE) (Wang et al., 2018) benchmark. Second, for commonsense reasoning tasks, we fine-tune LLaMA3-8B (Meta, 2024a) on the Commonsense170K (Hu et al., 2023) dataset. NLU experiments are conducted on a single NVIDIA Tesla V100s-PCIE (32GB) GPU while the commonsense reasoning tasks are performed on two NVIDIA A100-SXM4 (80GB) GPUs. All the results reported in this section are averaged over multiple experiments with different random seeds.

4.1 Baselines

Full fine-tuning and six PEFT methods serves as baselines, which are categorized into three groups:
Adapter-based Methods. 1) Adapter$^{\text{H}}$ (Houlsby et al., 2019), which inserts lightweight adapter modules sequentially after transformer layers; and 2) Adapter$^{\text{P}}$ (Pfeiffer et al., 2021), which places adapters after feedforward network (FNN) and LayerNorm modules.
LoRA-based Methods. 1) LoRA; 2) AdaLoRA; and 3) ReLoRA (Lialin et al., 2024), which trains K rank-r matrices sequentially and merges them. While ReLoRA is designed for pretraining, it can be regarded as a reduced version of our method, where T_{in} and T are fixed for all modules, and (5) and (6) are omitted. Therefore, we incorporate it to highlight the effectiveness of AriLoRA's adaptability and flexibility.
Other Methods. 1) Full fine-tuning, which updates all of the model's parameters; and 2) BitFit (Zaken et al., 2023), which fine-tunes only the bias terms of a pretrained model.

4.2 Natural Language Understanding

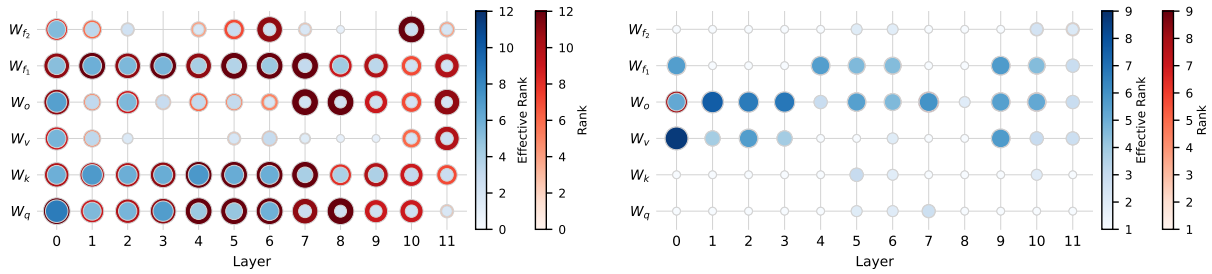
We first evaluate AROMA on NLU tasks. The model and datasets, training details are reported, followed by the results and analyses.
Model and Datasets. RoBERTa-base (125M) (Liu et al., 2019) enhances BERT (Devlin et al., 2019) by utilizing larger batches, more data, and longer sequences, resulting in a stronger language understanding capability. Eight NLU tasks in GLUE (detailed in Appendix F.1) are utilized to fine-tune RoBERTa-base, covering sentiment analysis, textual entailment, and semantic similarity.
Training Details. To secure a fair comparison, we basically follow the implementation strategy in (Zhang et al., 2023a). For each task in GLUE, we conduct a grid search for optimal hyperparameters, including the learning rate $lr \in$ [1E-4, 2E-4, 5E-4, 7E-4], inner tolerance ε_{in}=0.1, and outer tolerance $\varepsilon_{\text{out}} \in$ [1E-3, 5E-3, 6E-3]. We apply AROMA to all weight matrices, i.e., $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$,$\boldsymbol{W}_o$, $\boldsymbol{W}_{f_1}$, and $\boldsymbol{W}_{f_2}$.

LoRA and AdaLoRA are conducted using the standard HuggingFace PEFT library, and the hyperparameters are set as suggested in their original papers. We consider the rank of LoRA and the target rank of AdaLoRA across $\{1, 8, 16\}$. The corresponding AdaLoRA's initial rank is set to $\{4, 12, 24\}$. For ReLoRA, rank $r = 1$ is assigned to each LoRA to match the parameter budget. Detailed hyperparameter settings for each baseline are found in Appendix G.1.
Results and Analyses. Table 2 presents the performance of AROMA alongside its counterparts, where "#Param" refers to the number of initial trainable parameters. It is shown that both AdaLoRA and LoRA are sensitive to the rank parameter, whereas AROMA operates independently of it. AROMA achieves the highest average performance. In term of specific tasks, it surpasses other baselines on CoLA, MRPC, RTE, and SST-2, while yields comparable results on the remaining tasks. This is achieved with only 0.014% (approximately 0.17M out of 125.0M) of the trainable parameters required for full fine-tuning. In comparison to ReLoRA, a reduced version of AROMA without rank adaptability, our method demonstrates superiority on all tasks, showcasing the latter effectiveness. Particularly, AROMA shows a significant advantage in CoLA, MRPC, and RTE tasks. We will further explore MRPC and RTE to analyze the reasons behind AROMA's outstanding performance.

We plot the rank distributions for AdaLoRA and AROMA in Figs. 3 and 5, where the rank is a combination of effective rank (Roy and Vetterli, 2007) and non-effective rank. The former measures the effective dimensionality of a matrix, while the latter corresponds to dimensions with negligible contribution. Detailed description of effective rank are provided in Appendix C. It is observed that different weight matrices exhibit distinct rank characteristics, and AdaLoRA has a larger average rank

Scheme	#Param	CoLA MC	MNLI Acc	MRPC Acc	QNLI Acc	QQP Acc	RTE Acc	SST-2 Acc	STS-B PC	Avg
Full Fine-tuning	125.0M	60.26	**87.68**	88.33	92.58	90.75	78.63	94.63	90.31	85.40
BitFit$^{\sharp}$	0.10M	61.16	85.50	89.07	90.99	88.08	79.57	94.38	90.55	84.91
Adapter$^{\mathrm{H}\dagger}$	0.31M	61.76	86.31	88.64	92.52	90.16	78.56	93.54	90.88	85.30
Adapter$^{\mathrm{P}\dagger}$	0.30M	62.92	86.23	88.74	92.59	89.94	79.07	93.24	90.44	85.40
LoRA $_{r=1}$	0.17M	56.22	85.87	87.25	91.34	90.64	75.28	93.46	88.73	83.59
LoRA $_{r=8}$	1.34M	61.69	86.82	88.34	92.31	91.33	78.34	93.69	90.88	85.43
LoRA $_{r=16}$	3.27M	64.44	84.88	88.97	92.02	**91.35**	77.62	92.47	**91.18**	85.37
AdaLoRA $_{r=1}$	0.67M	57.86	87.21	88.24	92.46	89.91	76.17	93.69	89.99	84.44
AdaLoRA $_{r=8}$	2.01M	58.08	87.50	87.45	92.37	90.58	74.65	94.04	90.03	84.34
AdaLoRA $_{r=16}$	4.02M	59.35	87.67	88.73	**92.64**	90.79	77.26	93.23	90.26	84.99
ReLoRA$_{1\times 8}$	0.17M	59.91	85.61	86.11	89.13	87.20	82.54	93.44	89.20	84.14
AROMA	0.17M	**70.51**	86.96	**94.17**	91.30	89.49	**90.48**	**94.68**	90.34	**88.49**

Table 2: Comparative performance of different fine-tuning schemes for RoBERTa-base on GLUE benchmark. We report Matthew's correlation coefficient (MC) for CoLA, Pearson correlation coefficient (PC) for STS-B, and accuracy for all the remaining tasks. Higher is better for all metrics and the best results on each task are shown in **bold**. Results with "$^{\sharp}$" are retrieved from (Wang et al., 2025), and results with "†" are from (Mao et al., 2024). Note that "#Param" reflects the initial phase, and AROMA's #Param gradually descends to zero (see Figure 1a).



(a) Rank distribution of AdaLoRA

(b) Rank distribution of ***AROMA***

Figure 3: Resultant rank and effective rank distributions for RoBERTa-base fine-tuned on MRPC task by AdaLoRA$_{r=8}$ and AROMA, respectively. The x-axis represents the hidden layer index, while the y-axis refers to the weight matrix fine-tuned in each layer. The total rank is described by the red outer circle, whereas the effective rank is indicated by the blue inner circle. Experiment on RTE task is provided in Appendix D.

than AROMA. Furthermore, the rank distribution for AROMA is concentrated in the shallower layers, $\boldsymbol{W}_v$ and $\boldsymbol{W}_o$ for both MRPC and RTE tasks. In terms of effective rank, it is found that LoRA exhibits a low effective rank, just a quarter of the adapter rank (Shuttleworth et al., 2024; Biderman et al., 2024; He et al., 2025). For AdaLoRA, we see that only about half of its rank is effective (50.4% for MRPC, 49.2% for RTE), whereas AROMA exhibits an exceptionally high effective rank ratio (96.3% for MRPC and 91.7% for RTE).

Moreover, Figure 1 depicts the number of trainable parameters, total rank, ranks of specific layers and accuracy versus training step for RoBERTa-base on MRPC task. We select "layer.0.attention.output.dense" and "layer.9.attention.self.value" as illustration. It is evident that LoRA$_{r=8}$, AdaLoRA$_{r=8}$ and AROMA exhibit consistent, decreasing and growing rank behaviors, respectively. We notice that LoRA maintains nearly 1.3M trainable parameters, with a stable total rank and specific rank throughout, as it fixes the same rank for all weight matrices. AdaLoRA, on the other hand, progressively decreases the total rank and shows a fluctuating but generally declining specific rank, starting with 2.0M trainable parameters and averaging 1.62M. In contrast, AROMA necessitates only 0.17M trainable parameters initially, with an average of 0.08M. Remarkably, AROMA attains the highest accuracy among the three methods.

4.3 Commonsense Reasoning

In this section, we assess AROMA in handling a larger model and a more complex task.

Scheme	#Param	ARC-E	OBQA	SIQA	ARC-C	WinoG	PIQA	BoolQ	HellaS	Avg
ChatGPT$^{\diamondsuit}$	-	**89.7**	74.8	68.5	**79.9**	66.1	85.4	73.1	78.5	77.0
LoRA $_{r=1}$	1.77M	89.04	82.80	77.33	76.71	81.93	86.40	70.40	93.06	82.21
LoRA $_{r=8}$	14.16M	88.55	82.80	78.15	77.13	**85.71**	86.13	68.44	93.55	82.56
LoRA $_{r=16}$	28.31M	88.01	83.10	79.53	75.34	83.82	85.74	72.35	93.45	82.67
AdaLoRA $_{r=1}$	7.08M	87.58	71.00	71.14	71.16	70.09	83.95	62.17	67.33	73.05
AdaLoRA $_{r=8}$	21.23M	88.30	76.60	71.24	71.33	72.45	83.51	65.57	82.94	76.49
AdaLoRA $_{r=16}$	42.47M	88.47	75.20	71.14	72.70	71.90	84.17	62.69	84.13	76.30
AROMA $_{r=1}$	1.77M	89.31	83.70	79.12	78.50	81.85	**87.43**	71.16	93.79	83.11
AROMA $_{r=8}$	14.16M	89.48	**84.79**	**79.62**	78.76	83.98	87.22	**73.74**	**94.36**	**83.85**

Table 3: Comparative performance of different fine-tuning schemes for LLaMA3-8B on Commonsense170K dataset. We report accuracy for all tasks. Results with "$^{\diamondsuit}$" are retrieved from (Liu et al., 2024b). Note that "#Param" reflects the number of initial trainable parameters, and AROMA's average #Param is even less.

Model and Datasets. Following (Wang et al., 2025), we fine-tune LLaMA3-8B (Meta, 2024a) on the Commonsense170K dataset, which is a mixture of eight commonsense reasoning benchmarks (details provided in Appendix F.2). LLaMA3-8B model, developed by Meta, is designed for various NLP tasks, offering improved performance and efficiency over its predecessors.

Training Details. Apart from AROMA under the previous setting (denoted as AROMA$_{r=1}$), we additionally increase the rank of each LoRA update to 8 (denoted as AROMA$_{r=8}$) to accommodate this complex task. We apply AROMA to three weight matrices in the self-attention layer: $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$, and two in the FFN: $\boldsymbol{W}_{up}$, and $\boldsymbol{W}_{down}$. After fine-tuning, the resultant model is evaluated on each of the eight benchmarks in terms of accuracy. Detailed hyperparameter settings are found in Appendix G.2.

Results. Table 3 shows the comparative performance between AROMA and its counterparts, where ChatGPT (Wei et al., 2022) is also included for reference. Notably, AROMA$_{r=1}$ and AROMA$_{r=8}$ rank in the top two in terms of average accuracy. Specifically, AROMA$_{r=1}$ achieves this with approximately 0.02% of the original model's parameters, 6% of LoRA$_{r=8}$'s and 3% of AdaLoRA$_{r=8}$'s. AROMA$_{r=8}$ outpaces other baselines on three benchmarks and achieves second-best results on the remaining ones. These results validates the efficacy of our method.

5 Further Discussions

5.1 Ablation Study

We carry out ablation study on a crucial component of AROMA: *Reset*, i.e., randomly pruning 99.9% of the optimizer states after training a rank-one update, to validate its effectiveness on performance. We fine-tune RoBERTa-base on MRPC task using AROMA with and without *Reset*, respectively, with all other conditions remain unchanged. We average the results over 5 experiments with different seeds, and report the average rank and effective rank across all layers as well as accuracy.

Scheme	MRPC			RTE		
	Avg r	Eff r	Acc	Avg r	Eff r	Acc
AROMA$_{\text{w/o } Reset}$	1.43	1.39	83.33	1.42	1.30	70.48
AROMA$_{\text{w/ } Reset}$	2.78	2.68	**94.17**	3.42	3.14	**90.48**

Table 4: Comparison of AROMA with and without optimizer *Reset* for RoBERTa-base on MRPC task. "Avg r" and "Eff r" denote average rank and average effective rank, respectively.

As seen in Table 4, AROMA with the *Reset* mechanism demonstrates a larger rank than AROMA$_{\text{w/o } Reset}$ and achieves substantially higher accuracy. This suggests that *Reset* is beneficial. We interpret this as the optimizer reset allowing the new rank-one matrix to be computed from scratch, rather than relying on the previously computed rank-one matrix. This approach gives the new rank-one matrix a greater chance to explore new subspaces and learn more information. Supplementary experiment on cosine similarity in Appendix E further underscores the importance of the *Reset* mechanism.

5.2 Time Efficiency

In this subsection, we compare the efficiency of AROMA with LoRA and AdaLoRA. We unify the three methods by configuring their batch size of 64 and maximum sequence length of 256, and compute the average training time per epoch across six tasks in the GLUE benchmark on a single

NVIDIA Tesla V100s-PCIE (32GB) GPU. The results are reported in Table 5 and we see that AROMA demonstrates significant efficiency advantages in five tasks, while being comparable to LoRA in the remaining task, RTE. Particularly, its average time per epoch is 76.1% of LoRA's and 28.5% of AdaLoRA's. This superiority can be attributed to the rank-one training and unnecessity of SVD computation.

Task	LoRA	AdaLoRA	*AROMA*
CoLA	44.37	107.74	**12.43**
MRPC	17.84	45.57	**13.21**
QNLI	557.98	1547.82	**542.72**
RTE	**15.13**	31.46	20.14
SST-2	339.58	873.30	**153.47**
STS-B	30.04	73.13	**22.42**
Avg	167.50	446.50	**127.40**

Table 5: Per-epoch time comparison for RoBERTa-base on GLUE.

6 Related Work

PEFT emerges as a crucial approach for adapting LLMs to downstream tasks while minimizing computational and storage requirements. We categorize existing PEFT methods into three key paradigms (Han et al., 2024) as follows:

Additive PEFT Methods incorporate auxiliary trainable modules within transformer architectures. Serial adapter (Houlsby et al., 2019) introduces dual adapter modules positioned after self-attention and FFN layers, while (Pfeiffer et al., 2021) optimizes computational efficiency by inserting adapters exclusively after "Add & Norm" layers. Prompt-based techniques constitute another significant branch of additive PEFT. Approaches such as prefix-tuning (Li and Liang, 2021; Li et al., 2023; Zhang et al., 2023b), p-tuning (Liu et al., 2024c), and prompt-tuning (Lester et al., 2021) augment inputs or intermediate representations with trainable vectors, demonstrating particular efficacy for generative tasks and few-shot learning scenarios.

Selective PEFT Methods strategically identify and modify only the most critical subset of model parameters. BitFit (Zaken et al., 2023) achieves remarkable efficiency by exclusively fine-tuning bias terms while maintaining all other parameters frozen. Diff pruning (Guo et al., 2021) learns sparse parameter differences from pretrained weights, focusing on task-specific components. FishMask (Sung et al., 2021) leverages Fisher information to identify and update the most influential parameters for specific tasks.

Reparameterized PEFT Methods transform the parameter space to facilitate efficient updates without direct modification of original weights. $(IA)^3$ (Liu et al., 2022) and SSF (Lian et al., 2022) introduce learnable vectors that modulate activations in self-attention and FFN with low parameter overhead. LoRA (Hu et al., 2022) decomposes weight updates into low-rank matrix products, significantly reducing trainable parameters while preserving performance. AdaLoRA (Zhang et al., 2023a) enhances flexibility through SVD-like decomposition for dynamic rank allocation. DoRA (Liu et al., 2024b) decomposes the weight into magnitude and directional components. NOLA (Koohpayegani et al., 2024) and VeRA (Kopiczko et al., 2024) represent weight matrices as linear combinations of fixed random bases, optimizing only the mixture coefficients. HydraLoRA (Tian et al., 2024) maintains fixed LoRA $\boldsymbol{A}$ matrix while training multiple $\boldsymbol{B}$ matrices to accommodate multi-domain tasks. LoRA and its variants achieve state-of-the-art parameter efficiency, making them the most widely used PEFT approaches.

7 Conclusion

In this work, we propose **A**utonomous **R**ank-**O**ne **M**atrix **A**daptation (AROMA) for parameter-efficient fine-tuning. Unlike the existing adaptive rank adjustment method, AdaLoRA, which truncates singular values with low importance scores and requires both initial and target rank budgets, AROMA employs a rank-growing approach that autonomously constructs layer-specific updates with very few trainable parameters that gradually diminish to zero. We design a dual-loop architecture, featuring an inner loop that exploits each rank-one subspace to learn a LoRA update with the corresponding stopping criterion, while the outer loop determines the number of subspaces, namely, the optimal rank, guided by another stopping criterion. The learned rank-one components are merged and frozen, allowing only one rank-one LoRA to be trained at a time, thereby ensuring high parameter efficiency. Additionally, optimizer states are periodically reset to maintain subspace independence. Experimental results for NLU and commonsense reasoning tasks highlight AROMA's superiority in terms of accuracy and efficiency.

Limitations

Despite achieving promising results on NLU and commonsense reasoning benchmarks, our approach has several challenges to be tackled. It has yet to be tested in multimodal applications, a crucial area as multimodal models continue to gain prominence. Furthermore, we have not validated its scalability for extremely LLMs exceeding 100 billion parameters, where the dynamics of rank allocation may differ significantly. Future work should address these issues and explore the method's applicability across a broader range of tasks.

References

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. *Proceedings of the Second Text Analysis Conference, TAC 2009, Gaithersburg, Maryland, USA, November 16-17, 2009*, 7(8):1.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John Patrick Cunningham. 2024. LoRA learns less and forgets less. *Transactions on Machine Learning Research*. Featured Certification.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439.

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. 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 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

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*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Demi Guo, Alexander M Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. 2018. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, pages 785–794.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

Zhiwei He, Zhaopeng Tu, Xing Wang, Xingyu Chen, Zhijie Wang, Jiahao Xu, Tian Liang, Wenxiang Jiao, Zhuosheng Zhang, and Rui Wang. 2025. RaSA: Rank-sharing low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

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

Soroush Abbasi Koohpayegani, Navaneet K L, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. 2024. NOLA: Compressing LoRA using linear combination of random basis. In *The Twelfth International Conference on Learning Representations*.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*.

Brett W Larsen, Stanislav Fort, Nic Becker, and Surya Ganguli. 2022. How many degrees of freedom do we need to train deep networks: a loss landscape perspective. In *International Conference on Learning Representations*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Jonathan Li, Will Aitken, Rohan Bhambhoria, and Xiaodan Zhu. 2023. Prefix Propagation: Parameter-efficient tuning for long sequences. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1408–1419, Toronto, Canada.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. ReloRA: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. 2022. Scaling & Shifting Your Features: A new baseline for efficient model tuning. In *Advances in Neural Information Processing Systems*, volume 35, pages 109–123.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965.

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

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2024c. GPT understands, too. *AI Open*, 5:208–215.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. 2024. DoRA: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11662–11675, Bangkok, Thailand. Association for Computational Linguistics.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Meta. 2024a. LLaMA 3: Open and efficient foundation language models. `https://ai.meta.com/llama/`. Accessed: 2024-04-01.

AI Meta. 2024b. Introducing Meta LLaMA 3: The most capable openly available LLM to date. *Meta AI*, 2(5):6.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas.

Olivier Roy and Martin Vetterli. 2007. The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610. IEEE.

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

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. 2024. LoRA vs full fine-tuning: An illusion of equivalence. *arXiv preprint arXiv:2410.21228*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. Training neural networks with fixed sparse masks. In *Advances in Neural Information Processing Systems*.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 2024. HydraloRA: An asymmetric loRA architecture for efficient fine-tuning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium.

Fan Wang, Juyong Jiang, Chansung Park, Sunghun Kim, and Jing Tang. 2025. KaSA: Knowledge-aware singular-value adaptation of large language models. In *The Thirteenth International Conference on Learning Representations*.

Zheng Wang, Ming-Jun Lai, Zhaosong Lu, Wei Fan, Hasan Davulcu, and Jieping Ye. 2014. Rank-one matrix pursuit for matrix completion. In *Proceedings of the 31st International Conference on Machine Learning*, pages 91–99, Bejing, China. PMLR.

Zheng Wang, Ming-Jun Lai, Zhaosong Lu, Wei Fan, Hasan Davulcu, and Jieping Ye. 2015. Orthogonal rank-one matrix pursuit for low rank matrix completion. *SIAM Journal on Scientific Computing*, 37(1):A488–A514.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2023. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276, Singapore.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023a. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Zhen-Ru Zhang, Chuanqi Tan, Haiyang Xu, Chengyu Wang, Jun Huang, and Songfang Huang. 2023b. Towards adaptive prefix tuning for parameter-efficient language model fine-tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1239–1248, Toronto, Canada.

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

A Algorithm of AROMA

We present the details of AROMA in Algorithm 1.

B Time Complexity

We first analyze the per-step complexity to calculate $\Delta \boldsymbol{W}$ of dimensions $m \times n$. In the forward pass, considering $\boldsymbol{B} \in \mathbb{R}^{m \times r}$, $\boldsymbol{A} \in \mathbb{R}^{r \times n}$, and $\boldsymbol{x} \in \mathbb{R}^n$. LoRA costs $\mathcal{O}((m+n)r)$ time. AdaLoRA calculates $\boldsymbol{P\Lambda Q x}$, hence its complexity is $\mathcal{O}((m+n+\tilde{r})\tilde{r}) = \mathcal{O}((m+n)\tilde{r})$, where $\tilde{r}$ is the current rank. AROMA computes $\boldsymbol{B_p A_p x}$ with p being the current outer step, which requires $\mathcal{O}\left((m+n)\,p\right)$ time. Since LoRA has a consistent rank, AdaLoRA decreases rank, while AROMA increases rank, typically we have $\tilde{r} \geq r \geq p$, which leads to $\mathcal{O}^{\text{AdaLoRA}}_{\text{per-step}} > \mathcal{O}^{\text{LoRA}}_{\text{per-step}} \geq \mathcal{O}^{\text{AROMA}}_{\text{per-step}}$.

Based on this, we discuss the overall complexity. Given T as the total training steps, LoRA consumes $\mathcal{O}\left((m+n)rT\right)$ time. For AdaLoRA, we roughly denote its average rank as $\frac{r_i+r_f}{2}$ with r_i and r_f being the initial average rank and the target average rank, respectively, then its overall complexity is $\mathcal{O}\left((m+n)\frac{r_i+r_f}{2}T\right)$. For AROMA, supposing that each inner loop has T_{in} steps for simplicity, and there are P outer steps, i.e., $T = P \cdot T_{\text{in}}$, the overall complexity is $\mathcal{O}\left((m+n)T_{\text{in}}\sum_{p=1}^{P} p\right) = \mathcal{O}\left((m+n)\frac{1+P}{2}T\right)$. Typically, we have $\mathcal{O}^{\text{AdaLoRA}}_{\text{overall}} > \mathcal{O}^{\text{LoRA}}_{\text{overall}} \geq \mathcal{O}^{\text{AROMA}}_{\text{overall}}$. The above claims are listed in Table 6 and are experimentally validated in Section 5.2.

Scheme	LoRA	AdaLoRA	***AROMA***
Per-step Complexity	$\mathcal{O}((m+n)r)$	$\mathcal{O}((m+n)\tilde{r})$	$\mathcal{O}((m+n)p)$
Overall Complexity	$\mathcal{O}((m+n)rT)$	$\mathcal{O}(\frac{r_i+r_f}{2}(m+n)T)$	$\mathcal{O}\left((m+n)T\frac{1+P}{2}\right)$

Table 6: Complexity comparison

C Effective Rank

In data representation, effective rank (Roy and Vetterli, 2007) reflects the number of truly meaningful independent feature dimensions in a matrix, whose definition is given as follows. Consider a $m \times n$ matrix $\boldsymbol{W}$ with singular values:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_K \geq 0 \tag{7}$$

where $K = \min\{m, n\}$. Given $p_k = \frac{\sigma_k}{\sum_{k=1}^{K}|\sigma_k|}$, the effective rank is defined as:

$$\text{erank} = \exp\{H(p_1, p_2, \cdots, p_K)\} \tag{8}$$

where $H(p_1, p_2, \cdots, p_K)$ is the Shannon entropy:

$$H(p_1, p_2, \cdots, p_K) = -\sum_{k=1}^{K} p_k \log p_k \tag{9}$$

Effective rank is smaller than full rank as it ignores dimensions with minimal contributions.

In neural network weight matrices, effective rank indicates the number of effective feature transformations learned by that layer. Low effective rank proportion suggests redundancy or underutilized parameters (Shuttleworth et al., 2024).

D Rank Distribution for RTE Task

Figure 5 shows the rank distributions for AdaLoRA and AROMA on RTE task, and we observe a similar phenomenon to that of Figure 3.

E Cosine Similarity

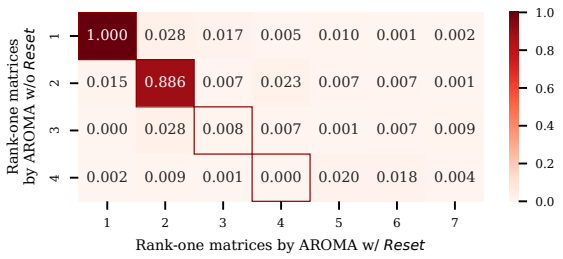


Figure 4: Cosine similarity between $\text{AROMA}_{\text{w/o } \textit{Reset}}$ and $\text{AROMA}_{\text{w/ } \textit{Reset}}$ for layer.10.attention.output.sense layer results for RoBERTa-base on MRPC task.

Figure 4 shows the cosine similarity between $\text{AROMA}_{\text{w/o } \textit{Reset}}$ and $\text{AROMA}_{\text{w/ } \textit{Reset}}$, which we only focus on values on the diagonal. It reveals that their solutions are identical initially, but increasingly diverge with each subsequent *Reset*. This finding further underscores the important role of the *Reset* mechanism.

F Dataset Details

F.1 GLUE

GLUE (Wang et al., 2018) is a collection of nine NLU benchmarks designed to evaluate the performance of LLMs across multiple dimensions of linguistic competence. This work involves eight commonly used GLUE tasks: CoLA (Warstadt et al., 2019), MNLI (Williams et al., 2018), MRPC (Dolan and Brockett, 2005), QNLI (Rajpurkar et al., 2016), QQP (Wang et al., 2018), RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007;

Bentivogli et al., 2009), SST-2 (Socher et al., 2013), STS-B (Wang et al., 2018). Their details are listed in Table 7.

Dataset	#Train	#Valid	#Test	#Label	Metric
Single-Sentence Classification					
CoLA	8.5k	1k	1k	2	MC
SST-2	67k	872	1.8k	2	Acc
Pairwise Text Classification					
MNLI	393k	20k	20k	3	Acc
RTE	2.5k	277	3k	2	Acc
QQP	364k	40k	391k	2	Acc
MRPC	3.7k	408	1.7k	2	Acc
QNLI	105k	5.5k	5.5k	2	Acc
Text Similarity					
STS-B	5.7k	1.5k	1.4k	1	PC

Table 7: Details of GLUE benchmark. "MC", "PC", and "Acc" represent Matthews correlation coefficient, Pearson correlation coefficient, and accuracy, respectively. "#Train", "#Valid", and "#Test" refer to the number of training, validation, and testing examples, respectively. "#Label" denotes the number of labels.

F.2 Commonsense170K

Commonsense170K (Hu et al., 2023) is a comprehensive benchmark collection comprising approximately 170,000 training examples and 400 validation examples across eight diverse commonsense reasoning datasets: ARC-Easy and ARC-Challenge (Clark et al., 2018), OBQA (Mihaylov et al., 2018), SIQA (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), BoolQ (Clark et al., 2019); and HellaSwag (Zellers et al., 2019). This consolidated benchmark evaluates LLMs' capabilities across multiple dimensions of commonsense knowledge, including conceptual reasoning, physical understanding, social intelligence, causal reasoning, coreference resolution, and scientific knowledge.

G Hyperparameter Settings

G.1 NLU Task

Hyperparameter setup for NLU task can be found in Table 9, where we follow the suggested setting for LoRA and AdaLoRA, and meticulously tune for AROMA, including the learning rate $lr \in$ [1E-4, 2E-4, 5E-4, 7E-4], inner tolerance $\varepsilon_{\text{in}} \in$ [0.05, 0.1], and outer tolerance $\varepsilon_{\text{out}} \in$ [1E-3, 5E-3, 6E-3]. Initial warmup is 100 and subsequent warmup is 50 for all tasks, except CoLA which uses 500 and 100 respectively. We use publicly available implementation (https://github.com/Guitaricet/relora) to run ReLoRA.

G.2 Commonsense Reasoning Task

Hyperparameter setup for commonsense reasoning task can be found in Table 8.

Scheme	Hyperparameter	Value
AROMA$_{r=1}$	r	1
	α	2
	Max Seq. Len.	256
	Batch Size	32
	Epoch	20
	Learning Rate	1E-4
	T	100,000
	T_{in}	1000
	ΔT_{in}	10
	ε_{in}	0.1
	ε_{out}	1E-3
	Eval Batch Size	8
AROMA$_{r=8}$	r	8
	α	16
	Max Seq. Len.	256
	Batch Size	32
	Epoch	15
	Learning Rate	1E-4
	T	80,000
	T_{in}	2000
	ΔT_{in}	10
	ε_{in}	0.1
	ε_{out}	1E-2
	Eval Batch Size	8

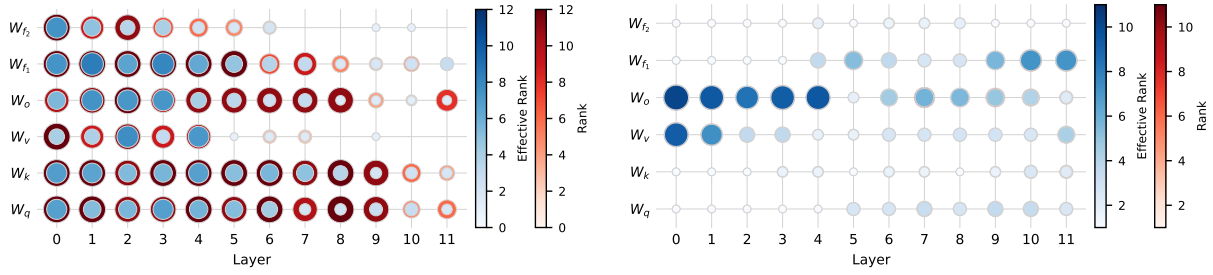
Table 8: Hyperparameter setup for LLaMA3-8B on Commonsense170k

Algorithm 1: ***AROMA***

Input: Inner and outer tolerances ε_{in} and ε_{out}, maximum inner training steps T_{in}, inner checking interval ΔT_{in}, maximum total training steps T.

for *each module in parallel*
  Initialize: $\boldsymbol{b}_1^{(0)} \leftarrow \mathbf{0}$; $\boldsymbol{a}_1^{(0)} \leftarrow$ Kaiming_init.
  Freeze $\boldsymbol{W}_0$.
  for $p = 1, 2, \cdots$ **do** `// OUTER LOOP`
    for $t = 1, 2, \cdots, T_{\text{in}}$ **do** `// INNER LOOP`
      Update $\boldsymbol{b}_p^{(t)}$, $\boldsymbol{a}_p^{(t)}$.
      if $\text{MOD}(t, \Delta T_{\text{in}}) = 0$ **then**
        inner_converged = True, **if** $\frac{\left\|\boldsymbol{b}_p^{(t)}\boldsymbol{a}_p^{(t)}\right\|_F - \left\|\boldsymbol{b}_p^{(t-\Delta T_{\text{in}})}\boldsymbol{a}_p^{(t-\Delta T_{\text{in}})}\right\|_F}{\left\|\boldsymbol{b}_p^{(t-\Delta T_{\text{in}})}\boldsymbol{a}_p^{(t-\Delta T_{\text{in}})}\right\|_F} < \varepsilon_{\text{in}}$. `// CHECK`
        Break the inner loop, **if** all modules are inner_converged.
    outer_converged = True, **if** $\frac{\|\alpha \boldsymbol{b}_p \boldsymbol{a}_p\|_F}{\|\boldsymbol{W}_0 + \alpha \boldsymbol{B}_{p-1}\boldsymbol{A}_{p-1}\|_F} < \varepsilon_{\text{out}}$. `// CHECK`
    Break the outer loop, **if** outer_converged.
    $\Delta \boldsymbol{W} = \Delta \boldsymbol{W} + \boldsymbol{b}_p^{(t)}\boldsymbol{a}_p^{(t)}$. `// MERGE`
    $\boldsymbol{b}_{p+1}^{(0)} \leftarrow \mathbf{0}$; $\boldsymbol{a}_{p+1}^{(0)} \leftarrow$ Kaiming_init. `// REINIT`
    Reset optimizer states & learning rate warmup. `// RESET`
  Finish, **if** all modules are outer_converged **or** reach T.

Output: $\Delta \boldsymbol{W}$.



(a) Rank distribution of AdaLoRA

(b) Rank distribution of ***AROMA***

Figure 5: Resultant rank and effective rank distributions for RoBERTa-base fine-tuned on RTE task by AdaLoRA$_{r=8}$ and AROMA, respectively. The x-axis represents the hidden layer index, while the y-axis refers to the weight matrix fine-tuned in each layer. The total rank is described by the red outer circle, whereas the effective rank is indicated by the blue inner circle.

Scheme	Hyperparameter	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
LoRA	Max Seq. Len.	128							
	Batch Size	64							
	Epoch	30	30	30	25	25	50	60	40
	Learning Rate	4E-4	5E-4	4E-4	4E-4	4E-4	5E-4	5E-4	4E-4
	r	8							
	α	16							
AdaLoRA	Max Seq. Len.	128							
	Batch Size	32							
	Epoch	25	7	30	5	5	52	24	26
	Learning Rate	5E-4	5E-4	1E-3	1.2E-3	5E-4	1.2E-3	8E-4	2.2E-3
	r_i	12							
	r_f	8							
	γ	0.5	0.1	0.1	0.1	0.1	0.3	0.1	0.1
	T	6500	85000	3000	15000	55000	4000	50000	4500
	t_i	800	8000	600	2000	8000	600	6000	800
	Δ_T	10	100	1	100	100	1	100	10
	t_f	3500	50000	1800	8000	25000	1800	22000	2000
	α	32							
AROMA	Max Seq. Len.	256							
	Batch Size	32	32	64	32	64	64	64	32
	Epoch	130	10	52	10	10	62	40	50
	Learning Rate	2E-4	7E-4	1E-4	2E-4	4E-4	1E-4	5E-4	5E-4
	T	35000	85000	3000	30000	55000	2400	40000	10000
	T_{in}	5000	5000	200	2000	55000	200	2500	1000
	ΔT_{in}	10							
	ε_{in}	0.1							
	ε_{out}	2E-2	5E-3	5E-3	5E-3	1E-3	6E-3	5E-3	5E-3
	α	4							

Table 9: Hyperparameter setup for RoBERTa-base on GLUE