

LoRA-RG: Residual Gradient-Guided Rank Initialization and Expansion

Dayoung Kwon¹ Jaeseong Lee² Seung-won Hwang¹ *

¹ Computer Science and Engineering, Seoul National University

² Snowflake AI Research

{ekdud1660, seungwonh}@snu.ac.kr jaeseong.lee@snowflake.com

Abstract

We study how to optimally allocate ranks in Low-Rank Adaptation (LoRA). Despite importance, ranks are typically chosen manually and fixed across all modules. We argue its suboptimality, risking either under-parameterization (hurting performance) or over-parameterization (wasting memory and compute). To address this, we propose **LoRA-RG**, a gradient-guided framework that adjusts LoRA ranks through two stages: rank initialization and rank expansion. In the rank initialization stage, LoRA-RG estimates a minimal rank for each layer based on gradient information prior to training. During training, the rank expansion stage further increases ranks only when the current LoRA parameterization is insufficient to capture the remaining update demand. This design aligns LoRA ranks with the underlying optimization needs dynamically. We evaluate LoRA-RG on reasoning, mathematical, and code generation benchmarks, and show that it consistently achieves higher accuracy compared to fixed-rank LoRA and recent rank initialization and adaptation baselines under the same parameter budget. ¹

1 Introduction

Parameter-efficient fine-tuning (PEFT) methods (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2022) have become a practical approach for adapting large pretrained models under constrained computational and memory budgets. Among them, LoRA (Hu et al., 2022) is widely adopted due to its simplicity and efficiency. It reparameterizes weight updates as $\Delta W = AB$, where A and B are low-rank matrices to reduce the number of trainable parameters during fine-tuning. One challenge in LoRA is choosing the rank r for each layer (Biderman et al., 2024). Standard LoRA uses the same

fixed rank for all layers. In practice, layers play different roles and require different update capacity, so uniform allocation often wastes parameters.

To overcome this, recent studies have explored more flexible rank allocation strategies. One line of work determines layer-wise ranks prior to training, using heuristic rules or search-based strategies to obtain a predefined rank configuration (Zhang et al., 2024; Benedek and Wolf, 2024). Although these methods avoid training-time rank adaptation, they do not directly reflect the update structure induced by data and optimization during fine-tuning. Another line of work introduces adaptive rank allocation methods that adjust ranks during fine-tuning based on layer importance or update statistics (Zhang et al., 2023b; Ding et al., 2023b; Zhang et al., 2023a; Gu et al., 2025). However, existing dynamic rank allocation methods still share two key limitations. They start from a large uniform initial rank across layers and adjust ranks during training without directly evaluating whether the allocated capacity is sufficient for each layer.

Despite their differences, existing approaches target one common question: *What rank is sufficient for each layer’s update demand?* Current rank selection criteria are typically based on heuristic rules (Valipour et al., 2023; Zhang et al., 2024), layer-wise importance scores (Zhang et al., 2023b; Liu et al., 2024), or indirect update statistics derived from optimization signals (He et al., 2025; Gu et al., 2025). As a result, rank allocation is often performed without a direct criterion for determining whether the chosen rank is sufficient to represent the actual training updates at each layer. This limitation becomes more pronounced over training because gradient structure and scale keep changing. Without direct sufficiency checks, ranks can remain too small and degrade performance, or grow too large and waste memory and computation.

To address these limitations, we reconsider rank allocation in LoRA from the perspective of gradi-

*Corresponding author.

¹Code is available at <https://github.com/allzero-kwon/LoRA-RG>.

ent approximation. From this viewpoint, the rank directly determines how accurately a low-rank update can represent the full fine-tuning gradient at each layer. When the rank is insufficient, the mismatch between the full gradient and its low-rank approximation shows how much of the required update remains uncaptured. We refer to this mismatch as the residual gradient, and treat it as a direct indicator of rank insufficiency. This view motivates both rank initialization and rank adaptation.

Based on this view, we propose **LoRA-RG**, a gradient-guided framework for rank initialization and expansion in LoRA. LoRA-RG operates in two stages. First, before fine-tuning, it initializes layer-wise ranks by estimating the minimal rank required to approximate early gradient signals. Second, during training, it expands ranks only when the residual gradient indicates that the current rank is insufficient, thereby adjusting ranks in response to evolving update requirements.

Our main contributions are summarized below:

- We propose LoRA-RG, a gradient-guided framework that initializes and expands ranks using an explicit sufficiency criterion.
- We formulate rank allocation as a gradient approximation problem, and introduce the Rank Sufficiency Score (RSS), a residual-based criterion that underlies both stages.
- It consistently outperforms fixed-rank LoRA and recent rank allocation baselines across reasoning, math, and code-generation benchmarks under the same parameter budget.

2 Related Work

2.1 Rank Allocation Methods

Existing approaches to LoRA rank allocation can be broadly categorized into two paradigms based on when and how ranks are determined.

Pre-defined Rank Allocation Pre-defined rank allocation methods (Benedek and Wolf, 2024; Zhang et al., 2024; Meng et al., 2024) determine the rank configuration prior to training. PRILoRA (Benedek and Wolf, 2024) assigns ranks under a fixed parameter budget by following a deterministic rule, allocating larger ranks to higher layers. AutoLoRA (Zhang et al., 2024) formulates rank allocation as an architecture search problem and identifies suitable rank configurations via Neural Architecture Search (NAS) prior to training.

PiSSA (Meng et al., 2024) initializes LoRA factors via SVD of the pretrained weight matrix prior to training. These methods have the advantage of zero additional overhead during training, but they do not account for the actual update dynamics induced by data and optimization.

Adaptive Rank Allocation Adaptive rank allocation methods (Zhang et al., 2023b; Ding et al., 2023b; Valipour et al., 2023), on the other hand, adjust ranks dynamically during fine-tuning to better match layer-wise update requirements. AdaLoRA (Zhang et al., 2023b) starts from a large uniform rank and progressively prunes or re-allocates ranks based on importance scores computed from singular values. IncreLoRA (Zhang et al., 2023a) follows an opposite strategy, gradually increasing ranks for layers deemed under-parameterized during training. While adaptive methods can better respond to layer-specific demands, they typically involve iterative rank reallocation, additional computation, and increased training complexity.

Distinction Prior methods rely on predefined rules or search for initialization, and importance-based reallocation. LoRA-RG instead uses residual gradient as a direct sufficiency signal, estimating the initial rank from early gradients and expanding it only when the residual indicates insufficient capacity.

2.2 Gradient-Based LoRA Optimization

Recent studies (Wang et al., 2024; He et al., 2025; Zhang et al., 2025) show that gradient information provides useful signals for LoRA fine-tuning. These methods differ in how gradient information is used. LoRA-One (Zhang et al., 2025) and LoRA-GA (Wang et al., 2024) use early full-gradient information mainly for LoRA parameter initialization. They apply SVD to an early gradient and use the resulting singular directions to initialize the LoRA factors under a given rank. Thus, gradients are used to improve the starting point of fixed-rank LoRA fine-tuning, while the rank itself remains pre-specified. To use gradients beyond parameter initialization, GoRA (He et al., 2025) introduces a gradient-based rank allocation criterion. It also leverages gradient information, but adopts a different approach by estimating parameter-wise sensitivity based on the element-wise product of parameters and gradients, and allocating ranks accordingly.

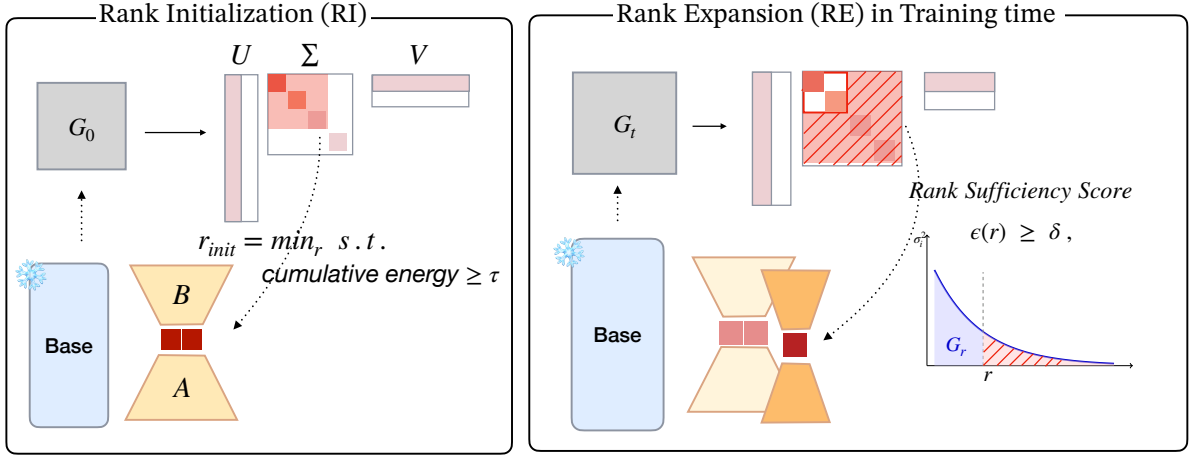


Figure 1: Overview of LoRA-RG. (a) Rank Initialization estimates a minimal layer-wise rank based on early gradient signals. (b) Rank Expansion increases LoRA ranks only when the current low-rank structure becomes insufficient, and initializes newly added parameters using dominant residual-gradient directions to avoid ineffective zero or random initialization.

Distinction While prior methods mainly use gradients for fixed-rank LoRA parameter initialization or importance-based rank allocation, LoRA-RG uses residual gradient error both to estimate the initial rank before training and to decide during training whether additional ranks are needed.

3 Problem Setting

We reformulate LoRA rank selection as a gradient approximation problem. In full fine-tuning, model parameters are updated using the full gradient

$$\Delta W_{\text{full}} = -\eta G,$$

where $G = \nabla_W \mathcal{L}(W)$ denotes the gradient of the loss.

In contrast, LoRA restricts parameter updates to a low-rank subspace. Under this view, a rank- r LoRA update corresponds to a rank- r approximation of the full gradient,

$$\Delta W_{\text{LoRA}} \approx P_r(G),$$

where $P_r(\cdot)$ denotes the optimal rank- r approximation under the Frobenius norm.

The quality of this approximation is characterized by the residual

$$\|G - G_r\|_F, \quad \text{where } G_r = P_r(G),$$

which captures gradient components not representable under the imposed rank constraint, providing a direct criterion for rank sufficiency.

4 Method

4.1 Overview

In this section, we introduce **LoRA-RG**, a gradient-guided rank allocation framework for LoRA. LoRA-RG allocates ranks in two stages. (i) Before training, it estimates layer-wise initial ranks from early gradients. (ii) During training, it expands rank only when the current rank becomes insufficient. The overall workflow is illustrated in Figure 1.

4.2 Gradient-Guided Rank Initialization

Before fine-tuning, we estimate the minimal rank required for each layer from the low-rank structure of its gradient matrix. This initializes LoRA ranks to approximate early updates under a global parameter budget.

Gradient-Based Rank Estimation Given a gradient matrix $G \in \mathbb{R}^{d \times k}$, we perform singular value decomposition (SVD),

$$G = U \Sigma V^\top$$

where $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_k)$ with $\sigma_1 \geq \dots \geq \sigma_k$. By the Eckart–Young–Mirsky theorem, the best rank- r approximation is given by $G_r = U_r \Sigma_r V_r^\top$. We define the retained energy ratio at rank r as

$$R(r) = \frac{\|G_r\|_F^2}{\|G\|_F^2} = \frac{\sum_{i=1}^r \sigma_i^2}{\sum_{i=1}^k \sigma_i^2}.$$

Given a target retained energy ratio τ , we define the required rank as

$$r_\tau = \min\{r \mid R(r) \geq \tau\}.$$

This criterion selects the smallest rank that retains sufficient gradient energy.

Budget-Adaptive Rank Allocation While r_τ is computed independently for each layer, the resulting total rank may exceed a global budget R_{budget} . To enforce the budget constraint, we gradually decrease τ and select the largest value τ^* such that

$$\sum_i r_{\tau^*,i} \leq R_{\text{budget}}.$$

The resulting $r_{\text{init}} = r_{\tau^*}$ is used to initialize LoRA ranks across layers. The full algorithm is described in Appendix A.1.

4.3 Measuring Rank Sufficiency

While gradient-based initialization provides a strong starting point, the required rank can change as training progresses. We therefore measure whether the current rank remains sufficient during training.

Residual Gradients Given the full fine-tuning gradient G and its rank- r approximation G_r , we define the residual gradient as

$$G_{\text{res}} = G - G_r$$

where G_{res} captures gradient components not represented by the current low-rank subspace.

Rank Sufficiency Score (RSS) We quantify rank sufficiency using the *Rank Sufficiency Score (RSS)*, defined as

$$\epsilon(r) = \frac{\|G_{\text{res}}\|_F}{\|G\|_F} \quad (1)$$

A small RSS indicates that the current rank adequately captures the gradient update, while a large score indicates a large residual ratio and signals the need for rank expansion. As illustrated in Figure 2, residual gradients saturate when the rank is insufficient, which motivates using RSS as a direct signal of rank sufficiency.

4.4 Gradient-Guided Rank Adaptation

We adapt LoRA ranks during training only when the current rank becomes insufficient. This adaptation is guided by the Rank Sufficiency Score (RSS), which directly measures how well the low-rank update approximates the full fine-tuning gradient.

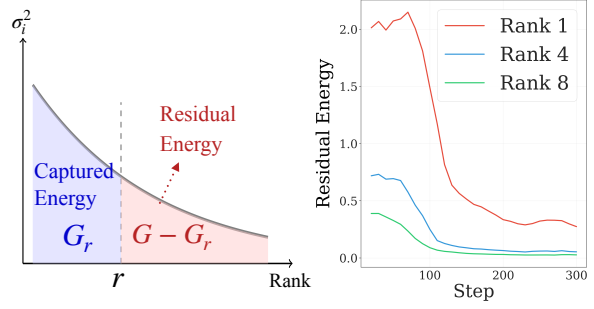


Figure 2: (Left) Conceptual illustration of residual gradient energy not captured by a rank- r subspace. (Right) Evolution of residual gradients during fine-tuning, showing saturation when the rank is insufficient.

When to Expand Ranks We periodically track the exponential moving average (EMA) of RSS for each LoRA layer. When this EMA exceeds a pre-defined threshold θ , indicating a persistently high residual ratio, we expand the LoRA rank by adding one rank-1 component. Unless otherwise stated, we set $\theta = 0.5$, which is robust across tasks. Implementation details are provided in Appendix A.2.

Residual Gradient Initialization

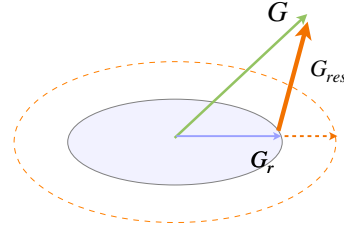


Figure 3: Residual gradient initialization during rank expansion. Newly added LoRA components are initialized to capture $G_{\text{res}} = G - G_r$.

Stable Initialization for Rank Expansion RSS decides when to expand rank, but it does not specify how to initialize new components. Since rank expansion changes the trainable parameter space during training, naive zero or random initialization can introduce optimization instability. To stabilize expansion, we introduce **Residual-Gradient Initialization**, which reuses the residual G_{res} from the sufficiency score to initialize newly added rank components: This residual directly captures update directions not captured at the current rank. We initialize new components from this residual rather than from random or zero values.

When adding Δr new rank components, we com-

pute the rank- Δr truncated SVD:

$$G_{\text{res}} \approx U_{\Delta r} \Sigma_{\Delta r} V_{\Delta r}^\top$$

We initialize the newly added LoRA parameters as

$$\Delta A = \frac{1}{\sqrt{s}} \Sigma_{\Delta r}^{1/2} V_{\Delta r}^\top, \quad \Delta B = \frac{1}{\sqrt{s}} U_{\Delta r} \Sigma_{\Delta r}^{1/2},$$

where s is the LoRA scaling factor.

We further adopt advance learning from IncreLoRA (Zhang et al., 2023a), which pre-trains upcoming rank components before activation to reduce optimizer-state discontinuity. Figure 3 illustrates this residual-gradient initialization process during rank expansion.

5 Experiments

5.1 Datasets and Models

We conducted experiments on Qwen3-4B (Qwen et al., 2025) and LLaMA-3.1-8B (Grattafiori et al., 2024) across commonsense reasoning, mathematics, code generation, and instruction-following tasks.

Commonsense Reasoning We used OpenBookQA (Mihaylov et al., 2018), SIQA (Sap et al., 2019), and ARC-Challenge/Easy (Clark et al., 2018) with predefined training and test splits following LLM-Adapters (Hu et al., 2023).

Mathematics We evaluated on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021).

Code Generation We trained on 30K samples from CodeFeedback (Zheng et al., 2024) and evaluated on the HumanEval (Chen et al., 2021) benchmark.

Instruction Following We fine-tuned the models on 50K samples from UltraChat (Ding et al., 2023a) and evaluated them on the IFEval (Zhou et al., 2023) benchmark.

5.2 Baselines

We compare our approach with LoRA (Hu et al., 2022) and state-of-the-art LoRA variants. We categorize baselines according to when and how LoRA ranks are determined, reflecting the design choices addressed by LoRA-RG.

Rank Initialization Methods

- PRILoRA (Benedek and Wolf, 2024) assigns linearly increasing ranks across layers according to layer depth. To isolate the effect of rank initialization, we use only this initial rank allocation scheme.
- PiSSA (Meng et al., 2024) initializes LoRA weights using the principal singular components of the pretrained weights.

Adaptive Rank Allocation Methods

- AdaLoRA (Zhang et al., 2023b) prunes singular values based on importance scores.
- IncreLoRA (Zhang et al., 2023a) incrementally increases ranks of important modules, where module importance is estimated from the average magnitude of the element-wise product between the LoRA update and its gradient.

Gradient-informed Allocation Methods

- LoRA-GA (Wang et al., 2024) initializes weights via gradient SVD.
- GoRA (He et al., 2025) allocates LoRA ranks based on element-wise weight-gradient magnitude.

5.3 Implementation Details

To ensure a fair comparison, all methods are trained using identical optimization settings, including the learning rate, batch size, number of training steps, and optimizer configuration. Unless otherwise specified, we follow the default hyperparameter settings used in prior LoRA studies.

For adaptive rank allocation methods, we controlled the total number of trainable parameters by matching the same average LoRA rank budget across methods. In particular, we set the vanilla LoRA rank to $r = 4, 8$ and used the same average rank budget for PRILoRA, IncreLoRA, AdaLoRA, and LoRA-RG. The resulting average rank may vary slightly due to differences in rank distribution across layers and modules. We therefore report the final average rank for all methods and include strictly parameter-matched comparisons in Appendix D.

We evaluated commonsense reasoning tasks following the settings of LLM-Adapters (Hu et al.,

Method	Avg. Rank	Mathematics		Code	Commonsense Reasoning				Chat
		GSM8K	MATH500	HumanEval	SIQA	OBQA	ARC-e	ARC-c	IFEval
Qwen3-4B ($r = 4$)									
LoRA	4.0	78.4 $\pm$ 0.8	47.6 $\pm$ 0.2	64.0 $\pm$ 0.3	73.6 $\pm$ 0.1	78.4 $\pm$ 0.2	92.5 $\pm$ 0.0	82.7 $\pm$ 0.0	70.6 $\pm$ 0.1
Rank Initialization Methods									
PRILoRA	4.0	79.9 $\pm$ 0.3	<u>50.9 $\pm$ 0.3</u>	67.7 $\pm$ 0.2	74.9 $\pm$ 0.1	81.6 $\pm$ 0.8	92.6 $\pm$ 0.0	83.7 $\pm$ 0.4	71.8 $\pm$ 0.0
PiSSA	4.0	79.9 $\pm$ 0.4	47.4 $\pm$ 0.5	66.5 $\pm$ 0.2	78.6 $\pm$ 0.7	<u>88.1 $\pm$ 1.0</u>	89.1 $\pm$ 0.4	86.7 $\pm$ 0.0	71.2 $\pm$ 0.6
Adaptive Rank Allocation Methods									
AdaLoRA	4.0	84.9 $\pm$ 0.2	49.9 $\pm$ 0.7	<u>69.5 $\pm$ 0.8</u>	71.7 $\pm$ 3.3	72.9 $\pm$ 0.2	86.1 $\pm$ 0.3	78.6 $\pm$ 0.2	73.7 $\pm$ 0.1
IncreLoRA	4.0	82.9 $\pm$ 0.9	51.2 $\pm$ 0.3	67.2 $\pm$ 0.8	71.9 $\pm$ 1.1	78.9 $\pm$ 0.2	92.1 $\pm$ 0.1	82.4 $\pm$ 0.2	73.0 $\pm$ 0.1
Gradient-based LoRA Rank & Weight Allocation									
GoRA	4.0	<u>85.8 $\pm$ 0.3</u>	47.6 $\pm$ 0.6	68.7 $\pm$ 0.9	74.3 $\pm$ 0.5	79.1 $\pm$ 0.3	<u>93.6 $\pm$ 0.4</u>	85.8 $\pm$ 0.8	71.1 $\pm$ 0.4
LoRA-GA	4.0	77.8 $\pm$ 0.5	47.9 $\pm$ 0.3	66.2 $\pm$ 1.1	74.2 $\pm$ 0.3	78.4 $\pm$ 0.7	91.6 $\pm$ 1.1	82.6 $\pm$ 0.8	71.8 $\pm$ 0.8
LoRA-RG (RI)	3.7	82.1 $\pm$ 0.1	49.9 $\pm$ 0.7	68.9 $\pm$ 0.4	<u>75.8 $\pm$ 0.8</u>	79.0 $\pm$ 0.5	93.5 $\pm$ 0.7	88.4 $\pm$ 0.5	74.2 $\pm$ 0.7
LoRA-RG (RI+RE)	4.0	87.9 $\pm$ 1.8	49.2 $\pm$ 0.5	70.1 $\pm$ 0.5	78.6 $\pm$ 0.9	89.6 $\pm$ 1.6	95.2 $\pm$ 0.7	<u>87.7 $\pm$ 0.3</u>	<u>73.9 $\pm$ 0.2</u>
Qwen3-4B ($r = 8$)									
LoRA	8.0	80.6 $\pm$ 0.3	46.3 $\pm$ 0.1	68.5 $\pm$ 0.8	74.5 $\pm$ 0.1	80.3 $\pm$ 0.0	93.6 $\pm$ 0.0	84.2 $\pm$ 0.1	72.0 $\pm$ 0.1
Rank Initialization Methods									
PRILoRA	8.0	80.4 $\pm$ 0.2	47.2 $\pm$ 1.0	<u>68.9 $\pm$ 0.1</u>	74.1 $\pm$ 0.0	79.1 $\pm$ 0.4	93.5 $\pm$ 0.0	84.8 $\pm$ 0.0	73.5 $\pm$ 0.1
PiSSA	8.0	81.5 $\pm$ 0.0	48.5 $\pm$ 0.8	63.6 $\pm$ 1.8	74.4 $\pm$ 0.7	82.4 $\pm$ 1.2	<u>93.9 $\pm$ 0.1</u>	84.5 $\pm$ 0.3	71.7 $\pm$ 0.3
Adaptive Rank Allocation Methods									
AdaLoRA	8.0	<u>84.8 $\pm$ 1.9</u>	50.6 $\pm$ 0.2	67.0 $\pm$ 0.3	74.1 $\pm$ 0.2	72.3 $\pm$ 0.5	90.4 $\pm$ 1.0	82.5 $\pm$ 0.4	72.6 $\pm$ 0.0
IncreLoRA	8.0	80.5 $\pm$ 0.6	45.8 $\pm$ 1.8	<u>68.9 $\pm$ 0.4</u>	74.8 $\pm$ 0.1	80.1 $\pm$ 0.1	93.5 $\pm$ 0.0	93.7 $\pm$ 0.1	71.5 $\pm$ 1.2
Gradient-based LoRA Rank & Weight Allocation									
GoRA	8.0	80.2 $\pm$ 0.6	47.5 $\pm$ 0.1	67.7 $\pm$ 0.7	73.1 $\pm$ 0.6	77.3 $\pm$ 2.1	93.4 $\pm$ 2.5	85.9 $\pm$ 0.0	71.1 $\pm$ 0.3
LoRA-GA	8.0	81.1 $\pm$ 0.4	47.8 $\pm$ 0.1	68.2 $\pm$ 0.0	78.4 $\pm$ 0.7	90.6 $\pm$ 0.2	83.2 $\pm$ 0.3	88.6 $\pm$ 0.3	73.0 $\pm$ 0.2
LoRA-RG (RI)	7.1	80.7 $\pm$ 0.4	<u>50.2 $\pm$ 0.1</u>	70.1 $\pm$ 0.1	<u>78.2 $\pm$ 0.6</u>	<u>89.8 $\pm$ 1.5</u>	95.0 $\pm$ 0.0	<u>93.2 $\pm$ 0.3</u>	73.1 $\pm$ 0.4
LoRA-RG (RI+RE)	7.4	88.2 $\pm$ 0.3	49.8 $\pm$ 0.2	68.2 $\pm$ 0.4	74.2 $\pm$ 1.3	88.5 $\pm$ 0.1	93.6 $\pm$ 0.5	90.4 $\pm$ 0.2	<u>73.1 $\pm$ 0.1</u>

Table 1: Performance comparison on Qwen3-4B under different LoRA ranks. **Bold** values indicate the best result within each rank block, and underlined values indicate the second-best result. For LoRA-RG, **RI** denotes the initialization-only variant, while **RI+RE** denotes the full method with additional rank expansion during training.

2023), replacing the original Alpaca-style instruction format with the model’s chat template. Mathematics, code generation, and instruction-following tasks were evaluated using lm-evaluation-harness (Gao et al., 2021).

All experiments are conducted on two NVIDIA A6000 $\times$ 2 GPUs. Further details of implementation, including hyperparameters are provided in Appendix B.

6 Main Results

We evaluate LoRA-RG through the following research questions.

- **RQ1: Is LoRA-RG an effective starting point for LoRA fine-tuning?**
- **RQ2: Does gradient-guided rank expansion improve performance during training?**
- **RQ3: Does LoRA-RG achieve a better performance–efficiency trade-off under the same parameter budget?**

6.1 RQ1: Is LoRA-RG an Effective Starting Point for LoRA Fine-Tuning?

We first evaluate whether LoRA-RG provides an effective starting point for LoRA fine-tuning. Table 1 compares LoRA-RG (RI) with fixed-rank LoRA and rank initialization baselines under the same rank budget.

Across reasoning, mathematical, code generation, and instruction-following benchmarks, RI consistently outperforms fixed-rank LoRA. For example, with Qwen3-4B at rank 4, RI improves GSM8K accuracy from 78.4 to 82.1. At rank 8, the GSM8K gain is smaller (80.6 to 80.7), but it clearly improves OBQA (80.3 to 89.8) and ARC-c (84.2 to 87.4). This indicates that gradient-guided rank initialization can allocate ranks more effectively according to layer-wise update demand, with the small GSM8K gain at $r = 8$ suggesting that this rank already provides sufficient capacity for the task.

Comparison with Rank Initialization Methods
Among rank initialization methods, LoRA-RG con-

sistently achieves the strongest performance across most tasks.

PRILoRA assigns larger ranks to deeper layers and performs well on MATH500 and IFEval. This suggests that tasks requiring more task-specific transformation, such as multi-step reasoning or precise instruction following, benefit from additional capacity in deeper layers. PiSSA initializes weights from the top singular vectors of the pre-trained weights, reusing their principal directions. This helps on SIQA and OBQA, where pretrained knowledge alone already covers much of the task.

In contrast, LoRA-RG performs consistently well across task types at both ranks. For example, it improves GSM8K accuracy to 82.1 at rank 4, and reaches 89.8 on OBQA at rank 8 while PRILoRA reaches only 79.1.

Overall, these results suggest that structural initialization rules tailored to specific task types are less effective than gradient-guided rank estimation, which adapts to the update demand of each task. We further analyze the resulting rank patterns in Appendix C.

The Effect of Gradient-Based Initialization

Gradient-based initialization methods generally outperform heuristic or structure-driven rank initialization, indicating that early gradients carry informative signals for LoRA adaptation. For example, with Qwen3-4B at $r = 4$, gradient-based methods such as GoRA and LoRA-RG outperform PRILoRA on several tasks, including GSM8K and ARC-c.

Although both use gradients, LoRA-RG differs from prior gradient-based methods in how it uses them. GoRA and LoRA-GA use gradients for importance estimation or weight initialization, but do not explicitly evaluate rank sufficiency. LoRA-RG instead examines the low-rank approximability of the gradient itself to identify a minimal yet sufficient rank, giving it a stronger initialization than other gradient-based methods, particularly on reasoning tasks.

6.2 RQ2: Does Gradient-Guided Rank Expansion Improve Performance During Training?

We next test whether rank expansion adds gains beyond initialization. As shown in Table 1, at $r = 4$, RI+RE improves most tasks over RI, including GSM8K and OBQA. This shows that expansion helps when the initial rank budget is small and

capacity is limited.

At rank 8, the pattern differs. RI+RE improves GSM8K substantially but leaves most other tasks, including ARC-e and ARC-c, largely unchanged. This suggests that when the initial rank already provides enough capacity, RI captures most of the gain and little expansion is triggered. GSM8K improves because it still lacks capacity even at $r=8$.

Overall, rank expansion is most effective when the initial rank is insufficient, and adds extra capacity once the rank is already sufficient. For rank 8, the average rank grows by only 4.2% over RI (7.1 to 7.4), compared to 8.1% at rank 4 (3.7 to 4.0). This shows that it adds capacity only where needed, without wasting memory. These results show that rank expansion helps when the initial low-rank subspace is not enough. By expanding ranks only when needed, LoRA-RG adds capacity without unnecessary growth. We analyze the expansion threshold further in Appendix G.

Table 2 shows the same trend under a higher base rank ($r = 8$) on LLaMA-3.1-8B-Instruct: RI improves over LoRA, and RI+RE gives the best overall results.

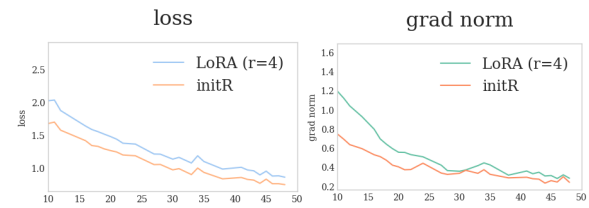


Figure 4: Convergence of training and gradient norm in training for GSM8K.

6.3 RQ3: Does LoRA-RG Achieve a Better Performance-Efficiency Trade-off under the Same Parameter Budget?

We evaluate whether LoRA-RG improves the trade-off between performance and efficiency under the same parameter budget.

As shown in Table 1, LoRA-RG (RI) uses a lower average rank than the target budget (3.7 at $r = 4$, 7.1 at $r = 8$), yet still outperforms fixed-rank LoRA. In other words, it achieves better results with fewer parameters. As shown in Table 3, RI requires an extra gradient computation step like LoRA-GA and GoRA, but its lower rank keeps training time comparable to these methods. LoRA-RG also converges faster than fixed-rank LoRA, reaching lower training loss earlier (Figure 4).

Method	GSM8K	MATH500	HumanEval	OBQA	SIQA	ARC-e	ARC-c	IFEval
LoRA ($r = 8$)	81.9	35.4	59.7	65.3	78.6	79.1	64.8	63.9
LoRA-RG (RI)	82.3	37.2	61.5	66.0	79.7	81.1	65.1	72.1
LoRA-RG (RI+RE)	83.4	39.2	63.5	66.8	80.6	81.1	66.2	68.3

Table 2: Performance comparison on LLaMA-3.1-8B-Instruct ($r = 8$).

Method	Training Time (s, $\times$ LoRA)	Peak Memory (GB, $\times$ LoRA)
LoRA ($r = 4$)	$\times 1.00$	$\times 1.00$
PRILoRA	$\times 1.01$	$\times 1.18$
PiSSA	$\times 0.99$	$\times 1.20$
AdaLoRA	$\times 1.50$	$\times 1.18$
IncreLoRA	$\times 1.0$	$\times 1.20$
GoRA	$\times 1.13$	$\times 1.20$
LoRA-GA	$\times 1.52$	$\times 1.05$
LoRA-RG (RI)	$\times 1.3$	$\times 1.03$
LoRA-RG (RI+RE)	$\times 1.54$	$\times 1.17$

Table 3: Training efficiency on GSM8K using QWEN3-4B.

Stage	Step	Time	Memory (MB)
Init	LoRA-GA	55.42s	9274MB
	LoRA-RG	19.8s	7938MB
Expansion	Score Comp.	0.21ms	–
	Rank-1 Exp.	4-7ms	0.05MB
	Δ FLOPs	0.0002%	–

Table 4: Time and memory overhead of LoRA-RG measured on Qwen3-4B. Rank expansion costs are reported per single rank increment ($\Delta r = 1$).

LoRA-RG (RI+RE) increases both training time and average rank due to expansion, but clearly improves performance further. Memory usage also rises over RI, since RE periodically recomputes RSS, which requires extra gradient computation and a larger rank, remaining comparable to other gradient-based methods such as LoRA-GA and GoRA. This shows that gradient-guided rank allocation offers a favorable performance-efficiency trade-off with fewer parameters.

7 Ablation Study

Time and Memory Overhead of LoRA-RG We analyze the computational overhead introduced by LoRA-RG during both the initialization stage and the adaptive rank expansion stage in Table 4.

During initialization, LoRA-RG analyzes the initial gradient using a single batch, incurring a one-

time SVD cost of 19.8 s and 7.9 GB of additional memory. In contrast, LoRA-GA incurs substantially higher overhead (55.4 s, 9.3 GB), as it aggregates gradients over multiple iterations (32 in our setup). Importantly, this additional memory is released after initialization and does not affect the training-time memory footprint.

During rank expansion, we measure the cost of computing the rank saturation score and the overhead of increasing the LoRA rank. The rank saturation score computation requires only 0.21 ms per step. A single rank-1 expansion introduces 0.05 MB of additional memory and increases FLOPs by $2 \times 10^{-4}\%$ relative to the full model. Overall, these results show that dynamic rank expansion in LoRA-RG introduces negligible computational overhead.

Method	GSM8K	MATH	HumanEval
LoRA ($r = 4$)	78.4	47.5	64.0
LoRA-RG (RE + zero-init)	87.1	48.9	69.1
LoRA-RG (RE + residual-init)	87.9	49.6	70.8

Table 5: Effect of rank expansion and rank initialization strategy. We compare fixed-rank LoRA with LoRA-RG using RSS based rank expansion (RE), where newly added ranks are initialized either with zeros or using residual gradient directions. All methods start from the same fixed initial rank.

Effect of Residual Gradient Initialization We analyze the effect of residual gradient initialization in adaptive rank expansion. Disabling gradient-based rank initialization, we first apply RSS-based rank expansion from a fixed rank. As shown in Table 5, rank expansion improves over fixed-rank LoRA, and residual-gradient initialization further improves over zero initialization by giving newly added ranks a more informative starting point.

Effectiveness of RSS-based Rank Expansion. To verify that RSS provides a useful signal for rank expansion, we compare LoRA-RG with and without RSS-based rank expansion on GSM8K with Qwen3-4B. As shown in Table 6, rank expansion reduces the mean RSS and improves accuracy. This

Setting	Avg. RSS	GSM8K Acc.
No-RE	0.54	82.7
RE	0.47	87.1

Table 6: Effect of RSS-based rank expansion on GSM8K with Qwen3-4B.

shows that RSS-based expansion reduces residual gradient error and provides additional capacity when the current rank is insufficient.

8 Conclusion

We propose LoRA-RG, a gradient-guided framework that addresses both rank initialization (RI) and rank expansion (RE) for LoRA. RI estimates a sufficient per-layer rank from warm-up gradients, achieving better performance than fixed-rank LoRA, rank initialization baselines, and other gradient-based methods while using a lower average rank. RE further expands rank selectively during training based on RSS, improving performance when the initial rank is insufficient without adding unnecessary parameters. Experiments across reasoning, math, code generation, and instruction-following benchmarks show that LoRA-RG achieves a better accuracy-efficiency trade-off than existing baselines at comparable time and memory cost.

Limitations

While LoRA-RG introduces only moderate overhead in our experiments, computing residual gradients and truncated SVD may incur additional cost for extremely large models or long training schedules.

Rank expansion also has an implementation constraint. Increasing a LoRA rank during training requires adding new parameters to the optimizer, which is difficult under frameworks such as DeepSpeed and the HuggingFace Trainer, since their optimizer state and parameter partitioning are set up at initialization. We therefore implement rank expansion, as well as IncreLoRA and GoRA, by pre-allocating parameters for the full rank budget and activating them progressively, following AdaLoRA.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) under the Leading Generative

AI Human Resources Development(IITP-2026-RS-2026-25543726) grant funded by the Korea government(MSIT), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) [NO.RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)].

References

- Nadav Benedek and Lior Wolf. 2024. PRILoRA: Pruned and Rank-Increasing Low-Rank Adaptation. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 252–263.
- 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 P. Cunningham. 2024. [LoRA Learns Less and Forgets Less](#). *Preprint*, arXiv:2405.09673.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating Large Language Models Trained on Code](#). *Preprint*, arXiv:2107.03374.
- 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](#). *Preprint*, arXiv:1803.05457.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training Verifiers to Solve Math Word Problems](#). *Preprint*, arXiv:2110.14168.
- Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023a. [Enhancing chat language models by scaling high-quality instructional conversations](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore. Association for Computational Linguistics.
- Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023b. [Sparse Low-rank Adaptation of Pre-trained Language Models](#).
- Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite,

- Ben Wang, Kevin Wang, and Andy Zou. 2021. [A framework for few-shot language model evaluation](#). Zenodo.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The Llama 3 Herd of Models](#). *Preprint*, arXiv:2407.21783.
- Naibin Gu, Zhenyu Zhang, Xiyu Liu, Peng Fu, Zheng Lin, Shuohuan Wang, Yu Sun, Hua Wu, Weiping Wang, and Haifeng Wang. 2025. [BeamLoRA: Beam-Constraint Low-Rank Adaptation](#). *Preprint*, arXiv:2502.13604.
- Haonan He, Peng Ye, Yuchen Ren, Yuan Yuan, Luyang Zhou, Shucun Ju, and Lei Chen. 2025. [GoRA: Gradient-driven Adaptive Low Rank Adaptation](#). In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS 2025)*. Accepted at NeurIPS 2025.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring Mathematical Problem Solving With the MATH Dataset](#). In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.
- 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, ICML 2019*, arXiv:1902.00751, pages 2790—2799.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi 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](#). *Preprint*, arXiv:2304.01933.
- Xiang Lisa Li and Percy Liang. 2021. [Prefix-tuning: Optimizing continuous prompts for generation](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4582–4597.
- Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. 2024. [ALoRA: Allocating Low-Rank Adaptation for Fine-tuning Large Language Models](#). *Preprint*, arXiv:2403.16187.
- Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. [PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models](#). In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS 2024), Spotlight*. Accepted as NeurIPS 2024 Spotlight.
- 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.
- Qwen, An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, and 42 others. 2025. [Qwen3 Technical Report](#). *Preprint*, arXiv:2505.09388.
- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. [SocialIQA: Commonsense Reasoning about Social Interactions](#). *Preprint*, arXiv:1904.09728.
- Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. [DyLoRA: Parameter-Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation](#). In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287.
- Shaowen Wang, Linxi Yu, and Jian Li. 2024. [LoRA-GA: Low-Rank Adaptation with Gradient Approximation](#). *Preprint*, arXiv:2407.05000.
- Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. [InCreLoRA: Incremental Parameter Allocation Method for Parameter-Efficient Fine-tuning](#). *Preprint*, arXiv:2308.12043.
- Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. [Adaptive budget allocation for parameter-efficient fine-tuning](#). In *The Eleventh International Conference on Learning Representations*.
- Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. [AutoLoRA: Automatically Tuning Matrix Ranks in Low-Rank Adaptation Based on Meta Learning](#). 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 5048–5060, Mexico City, Mexico. Association for Computational Linguistics.
- Yuanhe Zhang, Fanghui Liu, and Yudong Chen. 2025. [LoRA-One: One-Step Full Gradient Could Suffice for Fine-Tuning Large Language Models, Provably and Efficiently](#). In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*. Comment: Accepted by ICML 2025 (Oral).

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhui Chen, and Xiang Yue. 2024. [OpenCodeInterpreter: Integrating code generation with execution and refinement](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12834–12859, Bangkok, Thailand. Association for Computational Linguistics.

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

Appendix

A Algorithm Details

This section presents the full algorithm for our proposed LoRA-RG Rank Initialization and Rank Expansion method.

A.1 Rank Initialization Algorithm

The procedure estimates per-layer intrinsic adaptation ranks based on the gradient energy distribution obtained from a single warm-up step. *This warm-up step consists of a single forward–backward pass used solely to compute gradients, without performing any parameter updates.*

The algorithm proceeds as follows:

1. Compute the warmup-step gradient for all LoRA layers.
2. Determine the minimum rank r_τ that preserves a target fraction τ of total gradient energy.
3. Select the largest τ^* that satisfies the global parameter budget.

Algorithm 1 Gradient-Guided Rank Initialization

```
1:  $r_\tau[\ell, \tau] \leftarrow 0, r_{\text{init}}[\ell] \leftarrow 0$  for all layers  $\ell$ 
2:  $\mathcal{T} \leftarrow [0.99, 0.95, 0.90, \dots, 0.50]$ 
3:  $\text{loss} \leftarrow \text{FORWARD\_BACKWARD}(M, \mathcal{D}_{\text{warmup}})$ 
4: for each layer  $\ell \in \mathcal{L}$  do
5:    $G \leftarrow \nabla(\text{base\_weight}_\ell)$ 
6:    $\sigma \leftarrow \text{SVD}(G)$ 
7:    $\text{total\_energy} \leftarrow \sum_i \sigma_i^2$ 
8:   for each  $\tau \in \mathcal{T}$  do
9:      $S_i \leftarrow \text{cumsum}(\sigma^2) / \text{sum}(\sigma^2)$ 
10:     $r_\tau[\ell, \tau] \leftarrow \min\{i \mid S_i \geq \tau\}$  ▷ Smallest rank preserving  $\tau$  fraction of energy
11:   end for
12: end for
13: for  $\tau \in \mathcal{T}$  (high  $\rightarrow$  low) do
14:    $R_{\text{total}}(\tau) \leftarrow \sum_\ell r_\tau[\ell, \tau]$ 
15:   if  $R_{\text{total}}(\tau) \leq R_{\text{budget}}$  then
16:      $\tau^* \leftarrow \tau$ ; break ▷ Select the largest  $\tau$  satisfying budget constraint
17:   end if
18: end for
19: for each  $\ell \in \mathcal{L}$  do
20:    $r_{\text{init}}[\ell] \leftarrow r_{\tau^*}[\ell, \tau^*]$  ▷ Assign final initialization rank for each module
21: end for
22: return  $r_{\text{init}}$ 
```

A.2 Training-Time Rank Adaptation Algorithm

Algorithm 2 LoRA-RG: Training-Time Rank Adaptation via Residual Gradients

```

1:  $G^\ell \leftarrow 0, n^\ell \leftarrow 0$  for all layers  $\ell$  ▷ per-round gradient accumulator
2: for  $t = 1, \dots, T$  do
3:   Sample mini-batch  $\mathcal{B}_t$ ; forward/backward pass
4:   if  $t_{\text{init}} < t < t_{\text{final}}$  then
5:     for each LoRA layer  $\ell$  do
6:        $g_t^\ell \leftarrow \nabla_{W_\ell} \mathcal{L}(\mathcal{B}_t)$  ▷ gradient w.r.t. the frozen base weight
7:        $n^\ell \leftarrow n^\ell + 1; \quad G^\ell \leftarrow G^\ell + (g_t^\ell - G^\ell)/n^\ell$  ▷ running mean over the round
8:     end for
9:   end if
10:  if  $t \bmod \Delta T = 0$  and  $t_{\text{init}} < t < t_{\text{final}}$  then
11:    for each LoRA layer  $\ell$  do
12:       $[U_r, \Sigma_r, V_r] \leftarrow \text{SVD}_{r_\ell}(G^\ell); \quad G_r^\ell \leftarrow U_r \Sigma_r V_r^\top$ 
13:       $G_{\text{res}}^\ell \leftarrow G^\ell - G_r^\ell; \quad \epsilon^\ell \leftarrow \|G_{\text{res}}^\ell\|_F / \|G^\ell\|_F$ 
14:    end for
15:     $\mathcal{C} \leftarrow \{\ell : \epsilon^\ell > \theta \text{ and } \ell \text{ not in cooldown}\}$  ▷ candidate layers
16:    for each  $\ell \in \mathcal{C}$  do
17:       $[U_{\Delta r}^\ell, \Sigma_{\Delta r}^\ell, V_{\Delta r}^\ell] \leftarrow \text{SVD}_{\Delta r}(G_{\text{res}}^\ell)$ 
18:       $\text{eff}^\ell \leftarrow \|\Sigma_{\Delta r}^\ell\|_2^2 / (\Delta r \cdot \|G^\ell\|_F^2)$  ▷ energy captured per added rank
19:    end for
20:     $\mathcal{S} \leftarrow \text{top-}h \text{ layers in } \mathcal{C} \text{ ranked by } \text{eff}^\ell$ 
21:    for each  $\ell \in \mathcal{S}$  do
22:       $\Delta A_\ell \leftarrow (\Sigma_{\Delta r}^\ell)^{1/2} (V_{\Delta r}^\ell)^\top / \sqrt{s}; \quad \Delta B_\ell \leftarrow U_{\Delta r}^\ell (\Sigma_{\Delta r}^\ell)^{1/2} / \sqrt{s}$ 
23:       $A_\ell \leftarrow [A_\ell; \Delta A_\ell]; \quad B_\ell \leftarrow [B_\ell, \Delta B_\ell]; \quad r_\ell \leftarrow r_\ell + \Delta r$ 
24:      Start cooldown for  $\ell$ 
25:    end for
26:     $G^\ell \leftarrow 0, n^\ell \leftarrow 0$  for all  $\ell$  ▷ reset accumulator for next round
27:  end if
28: end for

```

B Implementation Details

B.1 Dataset

We report the fine-tuning setup for each benchmark used in our experiments, including commonsense reasoning, mathematical reasoning, and code generation tasks. All experiments share similar hyperparameters as shown in Table 7. Additionally, results are reported as the mean and standard deviation over five runs with different random seeds.

Dataset	Num Updates	Epochs	Batch Size	Learning Rate
GSM8K	5	1	32	2e-4
CodeFeedback-30K	10	1	32	2e-4
OBQA	5	2	32	2e-4
SIQA	5	1	32	2e-4
ARC-c	5	7	32	2e-4
ARC-e	5	3	32	2e-4
UltraChat-50K	10	1	32	2e-4

Table 7: Implementation details for all evaluation benchmarks.

B.2 Hyperparameters

Since the number of training steps varies across datasets, we set the update interval ΔT between 5 and 20 so that each interval contains at least 20 training steps.

- PRILoRA (Benedek and Wolf, 2024): Adopting only the rank initialization method from the paper, we configured the rank to increase linearly to achieve an average target rank.
- PiSSA (Meng et al., 2024): To reduce computation time, we use power iteration for SVD with 32 iteration steps.
- AdaLoRA: We set the initial warmup steps (t_{init}) to 20% of total training steps and final fine-tuning steps (t_{final}) to 70% (fixing the rank for the last 30%). The number of gradient updates is set to 5-20 (ΔT), with an initial rank of $1.5 \times r$ as recommended in the original paper. We report the detailed interval and gradient-update settings in Table 7.
- IncreLoRA (Zhang et al., 2023a): Similar to AdaLoRA, we set $t_{init} = 20\%$ and $t_{final} = 70\%$, and gradually increase the rank from an initial rank of 1 to the target rank r . We report the detailed interval and gradient-update settings in Table 7.
- LoRA-GA (Wang et al., 2024): We used 32 gradient estimation steps for initialization.
- GoRA (He et al., 2025): We set the gradient estimation steps to 32. The rank is dynamically adjusted between a minimum of 4 and a maximum of 32, with a dedicated learning rate of 5e-2.
- LoRA-RG: We set the threshold to 0.5. The rank adjustment starts after 20% of training steps (t_{init}) and concludes at 70% (t_{final}), performing a total of 5-20 update intervals (ΔT) same with AdaLoRA and IncreLoRA for fair comparisons.

C Rank Initialization Pattern of LoRA-RG

Figure 5 illustrates the layer-wise rank allocation produced by LoRA-RG under a fixed energy threshold ($\tau = 0.99$) across different tasks. The patterns reveal substantial task-dependent differences in how adaptation capacity is distributed across layers.

In code generation tasks such as CODEFEEDBACK, higher ranks are concentrated in deeper layers, whereas in commonsense and reasoning tasks (e.g., OPENBOOKQA, SIQA), ranks are more evenly distributed. This analysis shows that the layers requiring adaptation vary across tasks.

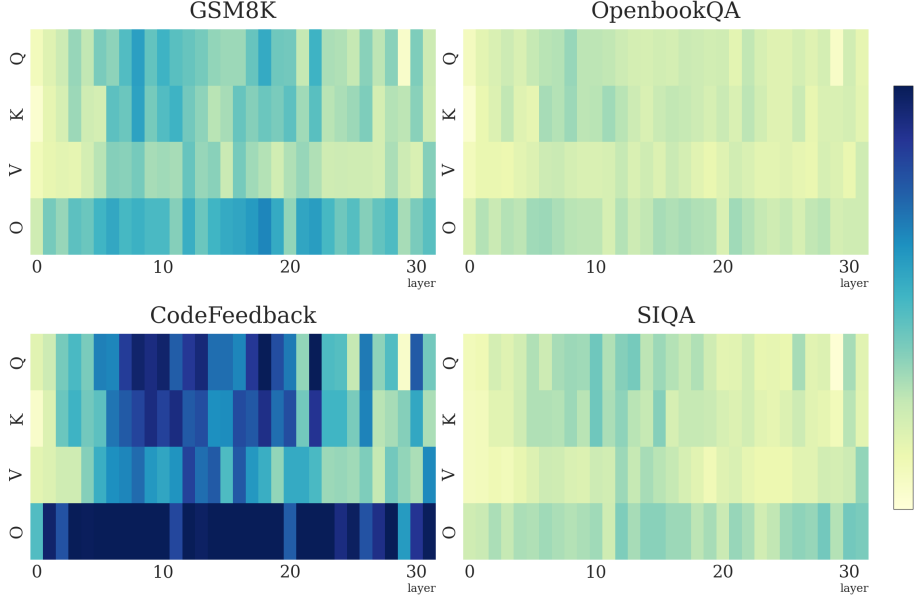


Figure 5

D Comparison under Matched Trainable Parameters

The RI stage searches the threshold τ with a coarse grid, so the realized average rank (and parameter count) can be slightly below the target used in the main experiments. To verify this does not explain our gains, we binary-search τ to exactly match LoRA’s parameter count, denoted LoRA-RG (param-matched).

Method	Avg. Rank	GSM8K
LoRA	4.0	78.5
LoRA-RG (RI)	3.8	82.2
LoRA-RG (param-matched)	4.0	82.4

Table 8: Comparison under matched trainable parameter budgets on GSM8K with Qwen3-4B.

As shown in Table 8, LoRA-RG still outperforms LoRA even under an exactly matched parameter count, confirming the gain comes from better rank allocation rather than extra parameters. Since the exact match requires a more expensive search over τ , we use the coarse grid as our default and report this result only as a check.

E QwQ-32B Results

Method	GSM8K	HumanEval
LoRA	81.3	78.0
PriLoRA	84.5	82.9
RI	88.5	83.5
RI + RE	90.3	84.7

Table 9: Results on QwQ-32B ($r = 8$).

We further evaluated on QwQ-32B to examine whether the proposed rank allocation strategy remains effective for larger reasoning models. As shown in Table 9, LoRA-RG consistently improves over LoRA and PriLoRA on both GSM8K and HumanEval, with RI+RE achieving the best performance. We used 2 H100 GPUs for the experiments to keep same hyperparameter settings as in Qwen-3B.

F Sensitivity to Probing LR/Steps in RI

We examine the sensitivity of the rank initialization (RI) stage to probing learning rate and steps. A single probing step gives noisy gradients and degrades performance. A moderate number of steps removes this noise, and larger K brings no further gain. The probing learning rate shows the same pattern. RI is robust to both once they reach a reasonable value.

Task	$K=1$	$K=8$	$K=16$	$K=32$
GSM8K	82.41	81.65	82.26	81.65
CodeFeedback	67.68	67.68	68.90	68.90
ARC-Challenge	82.73	86.35	85.86	85.69
ARC-Easy	92.68	93.30	93.30	93.17

Table 10: Sensitivity of RI to the number of probing steps K .

Task	No Probe LR	Probe LR=2e-4	Probe LR=1e-5
GSM8K	81.65	81.58	81.96
CodeFeedback	67.68	70.12	70.73
ARC-Challenge	86.35	85.94	85.61
ARC-Easy	93.30	93.34	93.30

Table 11: Sensitivity of RI to the probing learning rate.

G Sensitivity to RSS Threshold

RSS measures rank insufficiency, remaining small when the rank is adequate and growing large otherwise. As shown in Table 12, a larger initial rank ($r=8$) keeps the average RSS low, so expansion rarely triggers and performance stays stable. A smaller initial rank ($r=4$) raises the average RSS and makes results more sensitive to the threshold. Too low a value causes unnecessary expansion, too high a value blocks expansion when needed, and a moderate value works best.

Rank 4	Avg. RSS	RSS Threshold θ			
		0.1	0.3	0.5	0.7
GSM8K	0.54	81.18	84.13	87.70	87.72
ARC-Easy	0.48	93.38	93.46	93.50	93.40
Rank 8	Avg. RSS	0.3	0.5	0.7	0.9
GSM8K	0.39	88.62	88.70	88.90	88.62
ARC-Easy	0.40	92.16	93.26	93.01	92.76

Table 12: Sensitivity analysis of the RSS threshold θ for rank 4 and rank 8. Performance remains stable across a wide range of threshold values.

H Scheduling Hyperparameters in Rank Expansion

We study the sensitivity of rank-expansion schedules (t_{init} , t_{final} , and the number of updates). Table 13 shows that more updates improve accuracy up to 84.7 but raise runtime, and too many updates lower

accuracy since new ranks converge too little, as seen at 40 updates. We use a moderate schedule to balance both.

t_{init}	t_{final}	# Updates	Accuracy	Time (s)
10%	70%	20	82.8	608
40%	90%	20	80.4	612
40%	70%	1	80.9	592
40%	70%	10	82.8	612
40%	70%	20	83.4	618
40%	70%	30	84.7	640
40%	70%	40	54.7	640

Table 13: Sensitivity to rank-expansion scheduling hyperparameters on GSM8K (LLaMA-3.1-8B-Instruct).