G-LoRA: Global-Local Decoupled Low-Rank Adaptation

Jiahao Xiong[1], Yihong Huang[1], Yihe Liu[1], Xianming Hu[1], Hongbo Zhao[1], Kai Zhang[1]*

[1]School of Computer Science and Technology, East China Normal University

Correspondence: xiongjiahao@stu.ecnu.edu.cn, kzhang@cs.ecnu.edu.cn

Abstract

Low-Rank Adaptation (LoRA) has achieved remarkable progress in improving the fine-tuning efficiency and downstream performance of large language models (LLMs). Although prior work has recognized that different weight update matrices $\Delta\mathbf{W}$ exhibit varying importance and therefore should be allocated different ranks, parameters within the same update matrix are still typically constrained to a uniform rank configuration, neglecting fine-grained parameter-level heterogeneity. To address this limitation, we propose G-LoRA (Global-Local Decoupled LoRA), which decomposes each update matrix into global and local adapters. The key idea is to reorganize the rows and columns of the update matrix using a first-order Taylor approximation of parameter importance, such that highly influential parameters are clustered into a local sub-block of $\Delta\mathbf{W}$. During training, the local adapter then focuses on this high-importance sub-region and is allocated a higher rank, whereas the global adapter captures the residual updates for the entire update matrix with relatively lower rank. By allocating higher representational capacity to more critical parameters, G-LoRA enables more efficient utilization of model resources. Extensive evaluations on benchmarks spanning commonsense reasoning, mathematical reasoning, and code generation demonstrate that G-LoRA achieves up to 2.7% absolute accuracy improvement over LoRA and its variants, validating its effectiveness for LLM fine-tuning.

1 Introduction

As large language models (LLMs) continue to expand in real-world applications, adapting them to specialized downstream tasks via fine-tuning has increasingly become the mainstream approach in industrial practice (Achiam et al., 2023; Grattafiori et al., 2024; Yang et al., 2025). However, full-parameter fine-tuning requires substantial computational resources, making parameter-efficient fine-tuning (PEFT) methods (Xu et al., 2023) a promising alternative. Among them, low-rank adaptation approaches are particularly representative, with LoRA (Hu et al., 2021) being a prime example. These methods are based on the modeling assumption that weight update matrices possess low intrinsic rank and can therefore be approximated by the product of two low-rank matrices, significantly reducing the number of trainable parameters.

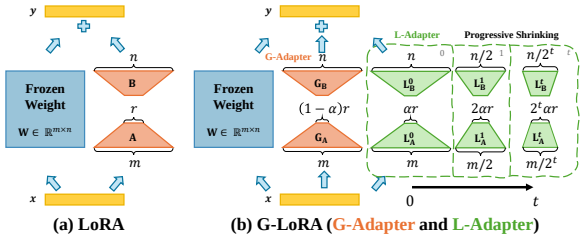

Figure 1: An illustration of G-LoRA compared with LoRA. G-LoRA progressively narrows the trainable region of local adapters during training, enabling them to focus on locally critical sub-regions.

Building on LoRA, recent works have explored dynamic parameter allocation strategies to further improve both fine-tuning efficiency and downstream task performance. Adaptive LoRA methods (Zhang et al., 2023) recognize that different weight update matrices $\Delta\mathbf{W}$ contribute unequally to downstream performance, and therefore allocate different rank budgets r across matrices. Similarly, sparse fine-tuning methods (Ding et al., 2023) reduce overall computational cost by selectively updating subsets of matrix weights that are most important to the target task. However, these methods could still be limited in catering to more fine-grained parameter capacity allocation. For example, adaptive LoRA methods focusing on differences across update matrices may not explicitly model the heterogeneity of parameter importance

*Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 20125–20141

within a single update matrix. Sparse fine-tuning methods, on the other hand, can capture importance differences within a single update matrix but rely on an additional warm-up phase to obtain gradient-based importance estimates, leading to peak memory consumption comparable to full-parameter fine-tuning. This naturally raises the following question: *Is it possible to allocate rank budgets differentially within a single update matrix while avoiding excessive computational overhead?*

To address this limitation, we investigate whether the update matrix can be reorganized to highlight a high-importance local sub-region, while still maintaining a global adaptation over the entire matrix. Specifically, we propose G-LoRA, a global–local decoupled low-rank adaptation framework that enables heterogeneous rank allocation within a single update matrix. G-LoRA dynamically identifies a locally critical sub-region during training and assigns it a higher-rank configuration, implemented as the local adapter (L-Adapter). In parallel, the global adapter (G-Adapter) performs low-rank updates over the entire matrix, capturing overall parameter interactions. Under a fixed parameter budget, this design enhances the representational capacity of LoRA by giving more resources to critical parameter sub-blocks while preserving global coverage.

We have derived both the upper and lower bounds of the rank attainable by G-LoRA. Our analysis shows that decoupling global and local low-rank adaptations allows the L-Adapter to concentrate its parameters in a local region, substantially enhancing the expressive power of the update matrix $\Delta\mathbf{W}$. For instance, when the G-Adapter is applied to the entire weight matrix and the L-Adapter is restricted to a local region covering one-sixteenth of the weight matrix, G-LoRA achieves an effective rank approximately $1.5-2.5\times$ higher than standard LoRA under the same parameter budget. Notably, this gain increases further as the L-Adapter's local region shrinks. Extensive experiments on commonsense reasoning, mathematical reasoning, and code generation show that G-LoRA consistently outperforms LoRA and its variants, achieving average accuracy gains of up to 2.7%. Moreover, these advantages remain consistent across diverse tasks and model scales, demonstrating that G-LoRA effectively strengthens task performance without compromising the parameter efficiency of standard LoRA.

Our main contributions are summarized below:

- We propose G-LoRA, which introduces a global–local decoupled low-rank adaptation. By dynamically focusing capacity on a locally critical region under a fixed parameter budget, G-LoRA enables more expressive adaptation while preserving parameter efficiency.
- We derive theoretical bounds on the effective rank of G-LoRA, demonstrating that its global–local decoupled structure admits a higher effective rank than LoRA under the same parameter budget, thereby improving overall parameter efficiency.
- We validate the effectiveness and generalization of G-LoRA across diverse NLP benchmarks and model scales, and it consistently outperforms LoRA and its variants on commonsense reasoning, mathematical reasoning, and code generation tasks.

2 Related Work

2.1 LoRA and Its Variants

With the rapid advancement of large language models (LLMs), parameter-efficient fine-tuning (PEFT) techniques (Huang et al., 2025b; Zou et al., 2025; Zhang et al., 2025; Li et al., 2025b) have attracted increasing attention. LoRA (Hu et al., 2021) decomposes the weight update matrix $\Delta\mathbf{W}$ into two low-rank matrices, $\mathbf{A}$ and $\mathbf{B}$. Moreover, LoRA enables merging the low-rank updates back into the original weight matrix $\mathbf{W}$ during inference, introducing no additional inference latency. Owing to its efficiency and flexibility, an increasing number of studies have been devoted to enhancing LoRA's update capability and representation power.

Specifically, AdaLoRA (Zhang et al., 2023) parameterizes incremental updates in a singular-value form and prunes unimportant singular values to adaptively manage parameter budgets. PiSSA (Meng et al., 2024) applies singular value decomposition to the original weights and initializes $\mathbf{A}$ and $\mathbf{B}$ with the top-r singular vectors, leading to improved training stability and performance. DoRA (Liu et al., 2024a) decomposes the update into two independent components: magnitude and direction, where the direction part retains the LoRA structure while the magnitude is modeled using a 1-dimensional vector. To relax the constraint of a fixed rank, RaSA (He et al., 2025) introduces a shared rank pool with layer-wise dynamic rank allocation, enabling flexible rank distribution

across layers. To address structural bottlenecks and gradient entanglement in high-rank settings, GraLoRA (Jung et al., 2025) adopts a fine-grained block decomposition to increase expressiveness.

Beyond additive update enhancements, several methods explore alternative update mechanisms. HiRA (Huang et al., 2025a) leverages the Hadamard product to realize high-rank updates with very few additional parameters, achieving stronger expressive capacity. RoSA (Hameed et al., 2024) periodically reinitializes and merges weight updates to adapt subspaces of arbitrarily large dimension, better approximating full-finetuning. TopLoRA (Li et al., 2025a) introduces lightweight token-wise projections, allowing different tokens to obtain independent input–output mappings, thereby enhancing semantic discrimination and flexibility.

2.2 Other PEFT methods

Beyond LoRA-based approaches, other PEFT methods also demonstrate strong effectiveness. Adapter-based methods (Houlsby et al., 2019; Dong et al., 2025) insert lightweight trainable modules into transformer layers, allowing efficient knowledge injection without modifying the original weights. Prompt-based (Lester et al., 2021; Liu et al., 2024b; Li and Liang, 2021) methods optimize a small set of task-dependent prompts, enabling parameter-efficient steering of model behavior. Selection-based methods (Ben-Zaken et al., 2022; Pan et al., 2024; Lai et al., 2025) identify and update only the most influential weights, while sparsity-based methods (Lingam et al., 2024; Liu et al., 2025; Miao et al., 2025) impose structural constraints to reduce parameter usage and training cost. These diverse strategies collectively provide complementary avenues for efficient adaptation of LLMs under different deployment constraints.

3 Methodology

We begin by reviewing the preliminaries of low-rank adaptation to provide context for an overview of G-LoRA. We then provide detailed descriptions of both global and local adapters of G-LoRA.

3.1 Preliminaries of Low-Rank Adaptation

LoRA (Hu et al., 2021) is a widely adopted PEFT method for LLMs, motivated by the observation that fine-tuning updates lie in a low-rank subspace. Accordingly, LoRA parameterizes the update as $\Delta\mathbf{W} = \mathbf{AB}$, where $\mathbf{A} \in \mathbb{R}^{m\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times n}$, and $r \ll \min(m, n)$ controls the number of trainable parameters. The forward computation is then:

$$\mathbf{y} = \mathbf{x}\mathbf{W}_0 + \mathbf{x}\mathbf{A}\mathbf{B}, \tag{1}$$

where $\mathbf{W}_0$ is frozen and only $\mathbf{A}$, $\mathbf{B}$ are trainable. To maintain the behavior of the pre-trained model, LoRA initializes $\mathbf{A}$ using Kaiming-uniform (He et al., 2015) values and sets $\mathbf{B}$ to zero.

3.2 G-LoRA

To more effectively allocate the limited parameter budget, G-LoRA decomposes the update matrix into two complementary components, namely the global adapter (G-Adapter) and the local adapter (L-Adapter), and expresses the output as

$$\mathbf{y} = \mathbf{x}\mathbf{W}_{\text{res}} + \underbrace{\mathbf{x}\mathbf{G}_\mathbf{A}\mathbf{G}_\mathbf{B}}_{\text{Global Adapter}} + \underbrace{\sum_{i=0}^{T} \left(\mathbf{x}[:, \boldsymbol{\pi}_r^i]\mathbf{L}_\mathbf{A}^i\mathbf{L}_\mathbf{B}^i\right)[:, (\boldsymbol{\pi}_c^i)^{-1}]}_{\text{Local Adapter}}. \tag{2}$$

Here, $\mathbf{W}_{\text{res}}$ is the frozen residual weight matrix obtained by subtracting the initialized portions of the G-Adapter and L-Adapter from $\mathbf{W}_0$. The G-Adapter is parameterized by two trainable low-rank factors $\mathbf{G}_\mathbf{A} \in \mathbb{R}^{m\times(1-\alpha)r}$ and $\mathbf{G}_\mathbf{B} \in \mathbb{R}^{(1-\alpha)r\times n}$, whose dimensions remain fixed throughout training. Then, the L-Adapter uses factors $\mathbf{L}_\mathbf{A}^i \in \mathbb{R}^{\frac{m}{2^i}\times 2^i\alpha r}$ and $\mathbf{L}_\mathbf{B}^i \in \mathbb{R}^{2^i\alpha r\times\frac{n}{2^i}}$, focusing on increasingly smaller sub-regions at each iteration i. The row and column permutations $\boldsymbol{\pi}_r^i$ and $\boldsymbol{\pi}_c^i$, obtained via importance-based reordering, identify these regions, while $(\boldsymbol{\pi}_*)^{-1}$ restores the original ordering.

In G-LoRA, the G-Adapter contributes a global low-rank update over the entire weight matrix, while the L-Adapter progressively shrinks its target region while increasing the rank allocated to that region, focusing its expressive capacity on the most important parameters during training. Together, these components enable G-LoRA to preserve significant parameter efficiency while notably enhancing its ability to model both global structures and localized patterns.

3.2.1 Global Adapter Component

In G-LoRA, the G-Adapter is designed to provide a global low-rank update across the entire weight matrix. Accordingly, it is parameterized in a standard low-rank form as

$$\Delta\mathbf{W}_{\text{global}} = \mathbf{G}_\mathbf{A}\mathbf{G}_\mathbf{B}. \tag{3}$$

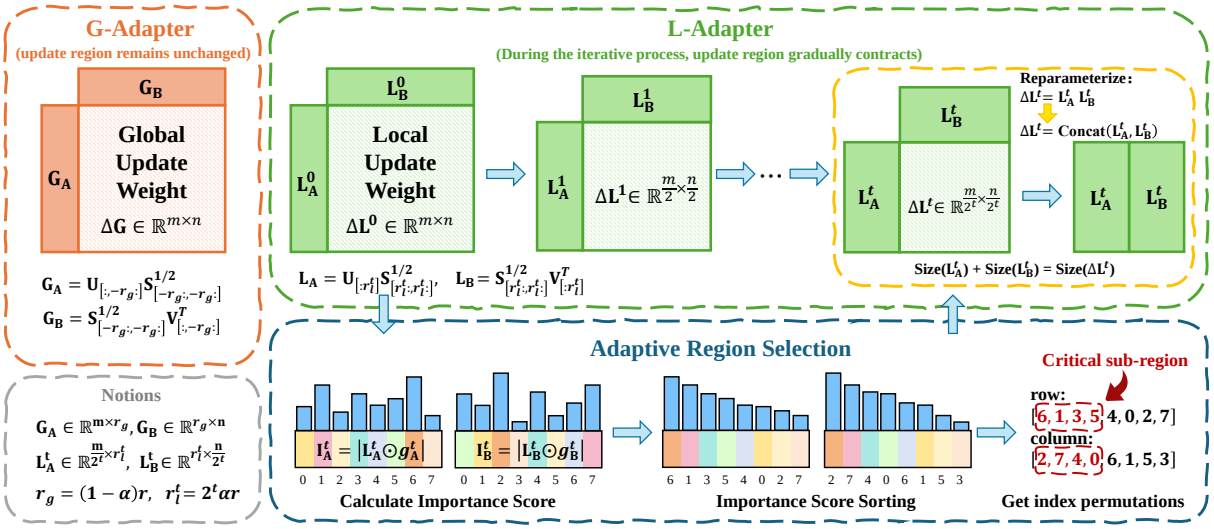

Figure 2: G-LoRA consists of two complementary components: the G-Adapter provides a global update over the weight space, while the L-Adapter concentrates on locally critical regions and allocates higher rank to these regions.

Because the G-Adapter operates on the full parameter space, an unconstrained initialization may interfere with the dominant spectral components of the pretrained weight matrix $\mathbf{W}_0$. To explicitly distinguish between high- and low-energy knowledge directions, we perform singular value decomposition (SVD) on $\mathbf{W}_0$:

$$\mathbf{W}_0 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top. \tag{4}$$

The G-Adapter is initialized using the trailing singular components indexed by $\mathcal{I}_{\text{global}} = \{k - r_g + 1, \ldots, k\}$, and $\mathbf{W}_{res}$ is defined as:

$$\begin{aligned} \mathbf{G_A} &= \mathbf{U}_{[:,\mathcal{I}_{\text{global}}]}\mathbf{\Sigma}^{1/2}_{[\mathcal{I}_{\text{global}}]}, \\ \mathbf{G_B} &= \mathbf{\Sigma}^{1/2}_{[\mathcal{I}_{\text{global}}]}\mathbf{V}^\top_{[\mathcal{I}_{\text{global}},:]}, \\ \mathbf{W}_{res} &= \mathbf{W}_0 - \mathbf{G_A}\mathbf{G_B}, \end{aligned} \tag{5}$$

where $k = \min(m, n)$ and $r_g = (1 - \alpha)r$ is the rank allocated to the global adapter.

By initializing the G-Adapter with trailing singular components corresponding to low-energy directions, this design mitigates interference with high-energy spectral components while preserving a global update throughout training.

3.2.2 Local Adapter Component

The L-Adapter allocates its parameter budget to critical sub-regions (sub-blocks) of the update matrix. To identify such regions, we estimate parameter importance using a first-order Taylor expansion (Molchanov et al., 2019) of the loss function:

$$\mathcal{L}(\mathbf{W} + \delta\mathbf{W}) \approx \mathcal{L}(\mathbf{W}) + \langle \nabla_{\mathbf{W}}\mathcal{L}, \delta\mathbf{W} \rangle. \tag{6}$$

For an individual parameter $\mathbf{W}_{jk}$, its importance is approximated by the change in loss incurred when the parameter is removed (i.e., set to zero). This operation can be modeled as a virtual perturbation $\delta\mathbf{W}_{jk} = -\mathbf{W}_{jk}$. Under the first-order Taylor approximation, the change in loss is given by

$$\Delta\mathcal{L}_{jk} \approx -\nabla_{\mathbf{W}_{jk}}\mathcal{L} \cdot \mathbf{W}_{jk}. \tag{7}$$

Accordingly, the parameter importance based on the first-order Taylor approximation is defined as

$$\mathrm{I}_{\text{taylor}}(j, k) = |\nabla_{\mathbf{W}_{jk}}\mathcal{L} \cdot \mathbf{W}_{jk}|. \tag{8}$$

This importance formulation can be interpreted as jointly capturing two complementary aspects: (1) the gradient term reflects the sensitivity of the loss to infinitesimal changes in the parameter. (2) the weight magnitude reflects the scale of the parameter's contribution to the model output. Taken together, these properties make Eq. 8 well suited for identifying important parameters in the model.

In practice, we do not have direct access to the explicit gradient of the full update matrix, i.e., the $\nabla_{\mathbf{W}_{ij}}\mathcal{L}$ term in Eq. 8, which can be computationally too expensive. Therefore, we provide an approximate estimate of the importance of the low-rank factors $\mathbf{L}^i_{\mathbf{A}}$ (row-wise) and $\mathbf{L}^i_{\mathbf{B}}$ (column-wise) as follows. Here, i denotes the iteration index, where the size of the influential sub-block is progressively reduced by a factor of 2 at each step.

First, we perform the entry-wise product of the low-rank factors and their gradients, as

$$\mathbf{I}^i_{\mathbf{A}} = |\mathbf{L}^i_{\mathbf{A}} \odot g^i_{\mathbf{A}}|, \quad \mathbf{I}^i_{\mathbf{B}} = |\mathbf{L}^i_{\mathbf{B}} \odot g^i_{\mathbf{B}}|, \tag{9}$$

where $g^i_{\mathbf{A}}$ and $g^i_{\mathbf{B}}$ denote the gradients of $\mathbf{L}^i_{\mathbf{A}}$ and $\mathbf{L}^i_{\mathbf{B}}$, respectively.

Then we use the ℓ_2 norms of the rows of $\mathbf{I}^i_{\mathbf{A}}$ and the columns of $\mathbf{I}^i_{\mathbf{B}}$ as an indicator of their respective importance, and sort them according to their magnitudes. This procedure yields the row and column importance–based permutation indices for the $(i+1)$-th iteration:

$$\begin{aligned} \boldsymbol{\pi}_r^{i+1} &= \operatorname{argsort}\left(\left\|\mathbf{I}^i_{\mathbf{A}}\right\|_{2,\text{row}}\right), \\ \boldsymbol{\pi}_c^{i+1} &= \operatorname{argsort}\left(\left\|\mathbf{I}^i_{\mathbf{B}}\right\|_{2,\text{col}}\right). \end{aligned} \tag{10}$$

After determining the row and column orderings for the next iteration based on the current update, the update matrix $\Delta\mathbf{W}^i_{\mathbf{L}}$ produced by the local adapter factors $\mathbf{L}^i_{\mathbf{A}}$ and $\mathbf{L}^i_{\mathbf{B}}$ is merged back into the frozen residual weight matrix $\mathbf{W}_{\text{res}}$. To ensure structural consistency with the original weight structure, $\Delta\mathbf{W}^i_{\mathbf{L}}$ is reordered according to the inverse permutation indices of the current iteration, namely $(\boldsymbol{\pi}^i_r)^{-1}$ and $(\boldsymbol{\pi}^i_c)^{-1}$:

$$\begin{aligned} \Delta\mathbf{W}^i_{\mathbf{L}} &= (\mathbf{L}^i_{\mathbf{A}}\mathbf{L}^i_{\mathbf{B}})[(\boldsymbol{\pi}^i_r)^{-1}, :][:, (\boldsymbol{\pi}^i_c)^{-1}], \\ \mathbf{W}^i_{\text{merge}} &= \mathbf{W}_{\text{res}} + \Delta\mathbf{W}^i_{\mathbf{L}}. \end{aligned} \tag{11}$$

Subsequently, the merged weight matrix is reordered according to the row and column permutations obtained for the $(i+1)$-th iteration, $\boldsymbol{\pi}^{i+1}_r$ and $\boldsymbol{\pi}^{i+1}_c$, producing $\mathbf{W}^{i+1}_{\text{init}}$ in which the locally critical sub-region is concentrated in the upper-left region:

$$\mathbf{W}^{i+1}_{\text{init}} = \mathbf{W}^i_{\text{merge}}[\boldsymbol{\pi}^{i+1}_r, :][:, \boldsymbol{\pi}^{i+1}_c]. \tag{12}$$

For the locally critical sub-region, we extract its dominant directions by applying SVD to the corresponding sub-matrix of $\mathbf{W}^{i+1}_{\text{init}}$. As training progresses, this local region is progressively narrowed, allowing the L-Adapter to concentrate its capacity on critical parameter subspaces:

$$\mathbf{W}^{i+1}_{\text{init}}[: \tfrac{m}{2^{i+1}}, : \tfrac{n}{2^{i+1}}] = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top. \tag{13}$$

To initialize the low-rank factors of the next local adapter, we retain only the high-energy directions associated with the leading singular values.

Specifically, we define the index set of high-energy directions as $\mathcal{I}^{i+1}_{\text{local}} = \{1, \ldots, r^{i+1}_\ell\}$, where $r^{i+1}_\ell = 2^{i+1}\alpha r$ denotes the rank allocated to the L-Adapter. Thus, the L-Adapter is initialized as

$$\begin{aligned} \mathbf{L}^{i+1}_{\mathbf{A}} &= \mathbf{U}_{[:, \mathcal{I}^{i+1}_{\text{local}}]}\boldsymbol{\Sigma}^{1/2}_{[\mathcal{I}^{i+1}_{\text{local}}]}, \\ \mathbf{L}^{i+1}_{\mathbf{B}} &= \boldsymbol{\Sigma}^{1/2}_{[\mathcal{I}^{i+1}_{\text{local}}]}\mathbf{V}^\top_{[:, \mathcal{I}^{i+1}_{\text{local}}]}, \\ \mathbf{W}^{i+1}_{\text{frozen}} &= \mathbf{W}^{i+1}_{\text{init}} - \mathbf{L}^{i+1}_{\mathbf{A}}\mathbf{L}^{i+1}_{\mathbf{B}}, \end{aligned} \tag{14}$$

This initialization aligns the update direction of the L-Adapter with the leading singular vectors of the locally critical sub-region, thereby constraining the update to its dominant spectral directions. After initializing $\mathbf{L}^{i+1}_{\mathbf{A}}$ and $\mathbf{L}^{i+1}_{\mathbf{B}}$, we apply the inverse permutations to restore the original row and column orders of $\mathbf{W}_{\text{res}}$ prior to reordering:

$$\mathbf{W}_{\text{res}} = \mathbf{W}^{i+1}_{\text{frozen}}[(\boldsymbol{\pi}^{i+1}_r)^{-1}, :][:, (\boldsymbol{\pi}^{i+1}_c)^{-1}], \tag{15}$$

where $\mathbf{W}_{\text{res}}$ denotes the frozen residual weight matrix for the next iteration, used in Eq. 2 and Eq. 11.

During the initial training stage, the L-Adapter treats the entire weight matrix as its trainable region. Subsequently, it employs the gradient–weight product derived from the first-order Taylor approximation to dynamically estimate the importance of each parameter. Guided by this distribution, the L-Adapter then concentrates iteratively on the most important parameters, reducing the trainable region to **a quarter of its original size** at each stage. Once the number of parameters required to update the selected region equals or exceeds the number required for full fine-tuning of the region, the L-Adapter switches to full fine-tuning and stops shrinking the region any further. During training, the L-Adapter progressively focuses its trainable parameters on critical sub-regions, enabling efficient updates within a limited parameter budget.

Since L-Adapter requires repeatedly constructing low-rank bases over dynamically changing regions, exact SVD can be computationally expensive. To reduce the overhead of constructing local low-rank bases further, G-LoRA also provides a **randomized SVD variant** (Halko et al., 2011), which approximates the principal directions via randomized projections and achieves substantially faster re-initialization while maintaining comparable fine-tuning quality.

3.3 Analysis of Rank Upper/Lower Bounds

Proposition 1 (Rank bounds for the sum of a global low-rank matrix and a zero-padded local high-rank sub-matrix)**.** *Assume that the global update matrix* $\mathbf{G} \in \mathbb{R}^{m\times n}$ *satisfies* $\operatorname{rank}(\mathbf{G}) = r$*. Similarly, let the local update matrix* $\mathbf{L} \in \mathbb{R}^{\frac{m}{4}\times\frac{n}{4}}$ *satisfy* $\operatorname{rank}(\mathbf{L}) = 4r$*, which requires* $\frac{m}{4} \geq 4r$ *and* $\frac{n}{4} \geq 4r$*, i.e.,* $m, n \geq 16r$*. To make the two matrices compatible for addition,* $\mathbf{L}$ *is zero-padded to match the dimensions of* $\mathbf{G}$*, yielding* $\mathbf{L}_{new} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$*. Since zero-padding does not alter matrix rank, we have* $\operatorname{rank}(\mathbf{L}_{new}) = 4r$*. The re-*

Base Models	Methods	Params	BoolQ	PIQA	SIQA	ARC-C	ARC-E	OBQA	HellaSwag	Winogrande	Avg
LLaMA2 (7B)	LoRA	8.39M	68.7%	80.0%	74.6%	**74.5%**	85.1%	79.2%	88.1%	72.4%	77.8%
	DoRA	8.91M	69.4%	79.6%	74.2%	73.0%	84.7%	80.8%	87.5%	70.6%	77.5%
	PiSSA	8.39M	68.6%	81.2%	75.4%	74.0%	86.5%	81.0%	88.0%	74.8%	78.7%
	RoSA	8.39M	69.2%	81.3%	74.9%	74.1%	**87.2%**	80.8%	88.6%	72.8%	78.6%
	RaSA	8.39M	68.4%	80.9%	74.3%	72.4%	85.4%	80.4%	86.5%	69.9%	77.3%
	GraLoRA	8.39M	68.5%	81.3%	76.0%	73.2%	85.9%	79.8%	88.1%	73.4%	78.3%
	MiLoRA	8.39M	69.4%	80.7%	74.4%	73.6%	85.4%	78.8%	87.4%	71.8%	77.7%
	DenseLoRA	4.16M	69.7%	81.0%	75.1%	73.3%	85.9%	78.8%	87.9%	75.1%	78.4%
	G-LoRA$_{svd}$	8.39M	**71.0%**	81.5%	75.8%	73.5%	87.1%	**83.4%**	**90.0%**	74.6%	**79.6%**
	G-LoRA$_{randomsvd}$	8.39M	69.8%	**81.9%**	**76.7%**	74.1%	86.7%	81.6%	89.3%	**75.5%**	79.5%
LLaMA2 (13B)	LoRA	13.11M	71.4%	83.3%	77.1%	78.8%	89.1%	81.8%	90.0%	76.8%	81.0%
	DoRA	13.93M	71.2%	83.4%	77.9%	79.2%	89.3%	82.2%	90.3%	76.6%	81.2%
	PiSSA	13.11M	**72.8%**	82.4%	77.2%	78.0%	89.6%	82.6%	90.5%	78.5%	81.5%
	RoSA	13.11M	70.5%	83.9%	76.7%	78.8%	89.4%	83.0%	91.1%	78.2%	81.4%
	RaSA	13.11M	70.8%	82.6%	76.6%	78.1%	89.3%	82.6%	89.2%	77.4%	80.8%
	GraLoRA	13.11M	71.0%	84.1%	77.7%	79.3%	89.6%	83.4%	90.7%	**79.2%**	81.8%
	MiLoRA	13.11M	71.5%	83.7%	76.8%	79.2%	89.2%	83.4%	91.1%	77.7%	81.6%
	DenseLoRA	5.21M	72.1%	83.5%	77.4%	78.7%	89.0%	83.0%	89.1%	78.3%	81.4%
	G-LoRA$_{svd}$	13.11M	**72.8%**	**84.5%**	**78.8%**	**80.7%**	**90.8%**	83.6%	**92.0%**	78.8%	**82.7%**
	G-LoRA$_{randomsvd}$	13.11M	71.3%	84.2%	78.7%	79.8%	89.6%	**84.0%**	91.8%	**79.2%**	82.3%
LLaMA3 (8B)	LoRA	6.82M	71.9%	86.9%	76.2%	81.5%	92.1%	84.8%	91.2%	80.6%	83.1%
	DoRA	7.14M	71.8%	86.9%	76.4%	81.1%	91.4%	85.2%	91.5%	80.5%	83.1%
	PiSSA	6.82M	71.1%	86.0%	76.7%	80.6%	92.4%	84.4%	91.1%	70.0%	82.7%
	RoSA	6.82M	72.0%	86.5%	76.5%	81.5%	91.6%	84.4%	91.4%	79.7%	82.9%
	RaSA	6.82M	71.5%	87.2%	76.4%	81.4%	91.5%	86.2%	90.9%	77.8%	82.9%
	GraLoRA	6.82M	72.7%	86.2%	77.0%	81.3%	92.4%	85.0%	91.4%	80.4%	83.3%
	MiLoRA	6.82M	72.6%	87.2%	76.0%	82.1%	91.4%	85.8%	91.2%	80.2%	83.3%
	DenseLoRA	4.19M	72.9%	87.2%	77.9%	81.5%	92.2%	84.8%	91.6%	80.8%	83.6%
	G-LoRA$_{svd}$	6.82M	72.7%	86.6%	77.9%	82.8%	**93.3%**	**87.8%**	92.7%	**81.8%**	84.4%
	G-LoRA$_{randomsvd}$	6.82M	**73.7%**	**88.2%**	**78.4%**	**83.1%**	**93.3%**	85.6%	**92.8%**	**81.8%**	**84.6%**

Table 1: Accuracy comparison of different PEFT methods across eight commonsense reasoning tasks.

sulting update matrix is given by $\Delta\mathbf{W} = \mathbf{G}+\mathbf{L}_{new}$*, whose rank satisfies* $3r \leq \mathrm{rank}(\Delta\mathbf{W}) \leq 5r$.

Proof. Proof can be found in the Appendix A. □

Remark 1. *The above bounds hold without assuming any orthogonality between the global update matrix* $\mathbf{G}$ *and the local update matrix* $\mathbf{L}$*. Orthogonality is sufficient for achieving the upper bound of* $5r$*; in contrast, the lower bound of* $3r$ *follows from standard rank inequalities and holds in the absence of additional structural assumptions.*

Compared with standard LoRA, whose update matrix has rank at most $2r$ under the same parameter budget, the decoupled global-local design of G-LoRA admits a larger effective rank $3r$–$5r$. This analysis suggests that G-LoRA can allocate representational capacity more flexibly across global and local updates, thereby improving expressiveness while maintaining parameter efficiency. The algorithmic procedure is provided in Appendix B.

4 Experiments

Following LLM-Adapters (Hu et al., 2023), we fine-tuned LLaMA2-7B/13B and LLaMA3-8B on Commonsense15K and Math10K, and evaluated their performance across 15 sub-tasks. For the code generation task, we fine-tuned LLaMA3-8B-Instruct and Qwen2.5-7B on the CodeAlpaca-20K (Chaudhary, 2023) dataset and reported results on the MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) benchmarks.

4.1 Experiments Setting

We compare G-LoRA with eight low-rank adaptation baselines, namely LoRA (Hu et al., 2021), DoRA (Liu et al., 2024a), PiSSA (Meng et al., 2024), RoSA (Hameed et al., 2024), RaSA (He et al., 2025), GraLoRA (Jung et al., 2025), MiLoRA (Wang et al., 2025), and DenseLoRA (Mu et al., 2025). Other PEFT methods discussed in Section 2.2 belong to adaptation paradigms that are fundamentally different from the LoRA family. Since these methods do not rely on low-rank parameter updates, they are not included in our experimental comparison. To maintain consistency in memory usage across all evaluated methods, we set the rank (r = 8), use a batch size of 4, and fine-tune only the Q, K, V, and O weight matrices for 10 epochs. More detailed hyperparameter configurations are provided in Appendix C.

4.2 Commonsense Reasoning

Table 1 presents the performance of G-LoRA on LLaMA2-7B/13B and LLaMA3-8B compared to

Base Models	Methods	Params	MultiArith	GSM8K	AddSub	AQuA	SingleEq	SVAMP	MAWPS	Avg
LLaMA2 (7B)	LoRA	8.39M	97.3%	40.8%	90.0%	26.0%	93.3%	55.0%	87.0%	70.0%
	DoRA	8.91M	97.3%	42.2%	89.9%	28.3%	92.3%	56.5%	87.0%	70.5%
	PiSSA	8.39M	96.3%	41.3%	90.0%	26.0%	92.1%	58.9%	86.6%	70.2%
	RoSA	8.39M	95.3%	44.4%	90.0%	25.6%	91.5%	58.1%	87.8%	70.5%
	RaSA	8.39M	96.5%	41.5%	89.9%	25.6%	89.0%	58.2%	85.3%	69.4%
	GraLoRA	8.39M	96.8%	42.5%	88.9%	**29.9**%	90.0%	56.1%	85.7%	70.0%
	MiLoRA	8.39M	96.3%	41.8%	87.3%	23.2%	91.7%	56.9%	85.3%	68.9%
	DenseLoRA	4.16M	**98.2**%	43.1%	89.1%	26.0%	**94.7**%	56.5%	**88.7**%	70.9%
	G-LoRA$_{svd}$	8.39M	97.5%	44.8%	90.1%	24.0%	91.7%	59.7%	88.2%	70.9%
	G-LoRA$_{randomsvd}$	8.39M	**98.2**%	**45.6**%	**90.4**%	28.0%	92.3%	**61.1**%	87.0%	**71.8**%
LLaMA2 (13B)	LoRA	13.11M	96.3%	53.8%	91.4%	23.6%	**95.1**%	64.4%	87.0%	73.1%
	DoRA	13.93M	96.7%	54.2%	89.9%	27.6%	**95.1**%	65.8%	87.0%	73.7%
	PiSSA	13.11M	97.3%	53.8%	91.4%	26.8%	94.7%	66.4%	87.4%	73.9%
	RoSA	13.11M	97.5%	52.2%	88.6%	27.6%	94.9%	61.8%	86.1%	72.7%
	RaSA	13.11M	97.7%	54.4%	91.1%	**29.5**%	93.7%	67.1%	89.1%	74.6%
	GraLoRA	13.11M	98.8%	53.4%	90.6%	25.2%	93.5%	66.7%	86.6%	73.6%
	MiLoRA	13.11M	97.7%	52.5%	91.1%	25.2%	94.1%	65.7%	87.0%	73.3%
	DenseLoRA	5.21M	97.2%	52.5%	91.4%	26.8%	93.5%	66.4%	86.1%	73.4%
	G-LoRA$_{svd}$	13.11M	99.2%	**56.9**%	91.6%	25.6%	94.5%	**69.9**%	**89.5**%	**75.3**%
	G-LoRA$_{randomsvd}$	13.11M	**99.3**%	56.7%	**91.9**%	24.0%	94.9%	68.7%	88.7%	74.9%
LLaMA3 (8B)	LoRA	6.82M	98.2%	65.6%	91.6%	30.3%	95.9%	73.2%	89.5%	77.8%
	DoRA	7.14M	97.8%	65.4%	92.2%	**32.3**%	**97.2**%	70.8%	89.9%	78.0%
	PiSSA	6.82M	98.2%	66.1%	93.4%	29.9%	96.5%	75.2%	91.2%	78.6%
	RoSA	6.82M	97.3%	65.8%	93.4%	30.3%	96.3%	76.1%	87.4%	78.1%
	RaSA	6.82M	97.5%	67.5%	90.1%	29.5%	96.7%	76.0%	89.9%	78.2%
	GraLoRA	6.82M	**98.7**%	66.8%	92.4%	29.5%	**97.2**%	76.8%	89.9%	78.7%
	MiLoRA	6.82M	98.0%	65.9%	91.4%	29.5%	95.9%	72.9%	89.5%	77.6%
	DenseLoRA	4.19M	98.2%	**68.3**%	93.7%	27.8%	96.3%	76.2%	90.3%	78.7%
	G-LoRA$_{svd}$	6.82M	97.5%	**68.3**%	93.9%	31.1%	96.5%	**79.2**%	**92.0**%	**79.8**%
	G-LoRA$_{randomsvd}$	6.82M	**98.7**%	67.5%	**94.9**%	31.5%	96.9%	74.9%	**92.0**%	79.5%

Table 2: Accuracy of PEFT methods on seven mathematical reasoning tasks.

several baseline methods. Under the same parameter budget, G-LoRA achieves superior or competitive results, indicating robust performance across varying model capacities and task settings. On LLaMA2-7B, G-LoRA attains an average accuracy of 79.6% with SVD initialization and 79.5% with randomized SVD initialization, representing an improvement of approximately 1.8% over LoRA. It achieves the best performance on BoolQ, OBQA, and HellaSwag. As the model scale increases to LLaMA2-13B, G-LoRA exhibits a more pronounced advantage, achieving an average accuracy of 82.7% and outperforming LoRA (81.0%). A similar trend is observed on LLaMA3-8B, where G-LoRA$_{randomsvd}$ achieves the highest average accuracy of 84.6%, improving upon LoRA by 1.5% and attaining the best performance on six tasks.

4.3 Mathematical Reasoning

Table 2 shows the performance of G-LoRA on LLaMA2-7B/13B and LLaMA3-8B across seven mathematical reasoning tasks, compared with several PEFT methods. G-LoRA demonstrates consistent improvements on most datasets and maintains stable performance across different model scales. On LLaMA2-7B, G-LoRA achieves average accuracies of 70.9% with SVD initialization and 71.8% with randomized SVD initialization, yielding a gain of 1.8% over LoRA (70.0%). The performance advantages of G-LoRA are particularly pronounced on MultiArith, GSM8K, and SVAMP. When scaling to LLaMA2-13B, G-LoRA continues to benefit from increased model capacity, reaching an average accuracy of 75.3%. A similar trend is observed for LLaMA3-8B, where G-LoRA$_{svd}$ attains an average accuracy of 79.8%, surpassing LoRA by 2.0%, and further demonstrating the robustness of G-LoRA across different base models.

4.4 Code Generation

Table 3 reports the performance of G-LoRA on the MBPP and HumanEval benchmarks using LLaMA3-8B-Instruct and Qwen2.5-7B. On LLaMA3-8B-Instruct, G-LoRA achieves an average accuracy of 62.2%, outperforming LoRA and PiSSA by 2.7% and 1.4%, respectively. When switching to Qwen2.5-7B, G-LoRA maintains consistent performance on both MBPP and HumanEval, with average accuracies of 74.1% for G-LoRA$_{svd}$ and 74.5% for G-LoRA$_{randomsvd}$.

These results suggest that, by explicitly modeling locally critical sub-regions, G-LoRA achieves a more effective utilization of the limited adaptation capacity without introducing additional parameters.

Base Models	Methods	MBPP	HumanEval	Avg
LLaMA3 (8B) Instruct	LoRA	62.3%	56.7%	59.5%
	DoRA	61.9%	57.3%	59.6%
	PiSSA	63.0%	58.5%	60.8%
	RoSA	63.0%	54.9%	59.0%
	RaSA	63.0%	55.5%	59.3%
	GraLoRA	60.3%	59.1%	59.7%
	MiLoRA	61.9%	59.1%	60.5%
	DenseLoRA	60.3%	50.6%	55.5%
	G-LoRA$_{svd}$	**64.6%**	**59.8%**	**62.2%**
	G-LoRA$_{randomsvd}$	64.2%	59.1%	61.7%
Qwen2.5 (7B)	LoRA	72.0%	71.3%	71.7%
	DoRA	72.4%	72.6%	72.3%
	PiSSA	71.2%	**76.2%**	73.7%
	RoSA	71.2%	69.5%	70.4%
	RaSA	72.0%	75.0%	73.5%
	GraLoRA	72.8%	75.0%	73.9%
	MiLoRA	72.8%	75.0%	73.9%
	DenseLoRA	71.2%	**76.2%**	73.7%
	G-LoRA$_{svd}$	**73.2%**	75.0%	74.1%
	G-LoRA$_{randomsvd}$	72.8%	**76.2%**	**74.5%**

Table 3: Pass@1 accuracy of PEFT methods on MBPP and HumanEval benchmarks.

Importance Evaluation Strategy	Avg
Gradient-averaged	70.8%
Fisher Information-based	71.6%
First-order Taylor Approximation-based	**71.8%**

Table 4: Ablation study comparing different importance evaluation strategies under identical model settings.

4.5 Ablation study

We performed all ablation studies by fine-tuning LLaMA2-7B on the Math10K and evaluating performance on eight mathematical reasoning tasks.

4.5.1 Effect of Different Importance Evaluation Strategies

To analyze the impact of different importance evaluation strategies on model performance, we compare three representative strategies under identical model settings: gradient-averaged, Fisher information–based, and first-order Taylor approximation–based. Table 4 shows that the gradient-averaged strategy yields the lowest performance, although it still surpasses the LoRA baseline (70%). The Fisher information–based strategy, which incorporates gradient statistics, achieves higher accuracy. The first-order Taylor approximation–based strategy attains the best performance, indicating that leveraging the local sensitivity of the loss to parameter perturbations more effectively identifies critical regions that influence model performance.

4.5.2 Influence of Ranks

Table 5 shows the performance of different low-rank adaptation methods under varying rank budgets. Both variants of G-LoRA outperform the baseline methods under the same rank settings. This advantage is already evident at a low rank setting (rank 8), where G-LoRA$_{svd}$ achieves a 0.9% improvement over LoRA, while G-LoRA$_{randomsvd}$ further increases the accuracy to 71.8%. As the rank increases, this performance gap remains stable and slightly widens, indicating that G-LoRA is able to exploit additional rank capacity more effectively than existing methods. Overall, these results demonstrate that G-LoRA leverages a global–local decoupled structure to allocate ranks differentially within the update matrix, achieving more effective parameter utilization under the same rank budget. In addition, by restricting re-initialization to progressively shrinking local critical regions and leveraging randomized SVD, G-LoRA incurs only a modest $\sim 8\%$ increase in training time compared to LoRA, as analyzed in Appendix E.

Method	Rank-8	Rank-16	Rank-32
LoRA	70.0%	70.1%	71.9%
PiSSA	69.6%	70.8%	71.1%
RoSA	70.5%	70.2%	71.5%
G-LoRA$_{svd}$	70.9%	71.5%	72.5%
G-LoRA$_{randomsvd}$	**71.8%**	**71.7%**	**72.7%**

Table 5: Accuracy of LoRA, PiSSA, RoSA, and G-LoRA across varying rank configurations.

G-Adapter Init	L-Adapter Init	Avg
Top	Top	70.4%
Bottom	Bottom	71.6%
Top	Bottom	70.9%
Bottom	Top	**71.8%**

Table 6: Ablation results analyzing different initialization strategies for the G-Adapter and L-Adapter.

4.5.3 Impact of Initialization Strategies on G-Adapter and L-Adapter

We explore the effect of initialization strategies on the global (G-Adapter) and local (L-Adapter) adapters, as shown in Table 6. Initializing G-Adapter with smaller singular values and L-Adapter with larger singular values (Bottom–Top) achieves the best average performance of 71.8%, while using small singular values for both adapters (Bottom–Bottom) yields a comparable performance of 71.7%. These results indicate that small singular value initialization for the G-Adapter helps preserve global stability, whereas large singular value initialization for the L-Adapter better aligns up-

Re-init freq	2	3	4	5	None
G-LoRA$_{svd}$	70.9	71.3	71.2	70.5	70.3
G-LoRA$_{randomsvd}$	71.8	70.9	71.2	71.1	70.4

Table 7: Performance under different values of *Re-initialization freq*, which denotes the number of epochs between two consecutive re-initialization operations.

dates with task-relevant directions, enabling more effective adaptation to critical sub-regions.

4.5.4 Effect of the Re-initialization Schedule

In G-LoRA, the trainable region of the local adapters is progressively refined during training through periodic re-initialization. To better understand the impact of this design choice, we evaluate several variants with different re-initialization frequencies. From Table 7, we observe that relatively frequent re-initialization tends to provide slightly better performance. In particular, settings with a re-initialization frequency of 2–4 epochs generally achieve stronger results, suggesting that periodically refining the trainable region can improve adaptation effectiveness. Notably, even when re-initialization is completely disabled, G-LoRA still achieves 70.4%, outperforming LoRA (70.0%). This confirms that the global-local decoupling architecture itself is effective, while periodic re-initialization provides further improvements.

4.5.5 Efficiency and Scalability

We analyze the computational overhead of G-LoRA, and provide detailed results in Appendix E. Overall, G-LoRA introduces only a modest additional cost compared to standard LoRA. It adds approximately one minute of training time per epoch in our single-GPU setting, mainly due to lightweight reordering operations, and remains a relatively small fraction of the overall training time. In addition, we further examine the scalability of G-LoRA and analyze its implications for distributed training, as detailed in Appendix J.

4.5.6 Effective Rank Analysis

To evaluate how different parameter-efficient fine-tuning methods utilize the available subspace, we compute the **Entropy-Based Effective Rank (EBER)** (Roy and Vetterli, 2007) of the weight updates on LLaMA2-7B with rank 8. Appendix F provides a detailed definition of EBER.

As shown in Table 8, G-LoRA significantly increases the effective rank of the update matrix, indicating that the energy of the weight updates is distributed across a significantly larger number of singular directions. In contrast, conventional low-rank methods tend to concentrate the update energy in a limited set of dominant components. These results empirically supports our theoretical analysis that the global-local decoupled design enhances subspace expressivity. By enabling more diverse directional updates, G-LoRA can better capture complex task-specific variations while maintaining parameter efficiency.

Method	Type	EBER
LoRA	Standard LoRA	5.57
DoRA	Direction-Magnitude Decoupling	5.54
PiSSA	SVD Initialization	7.78
RaSA	Shared Adapter	18.22
RoSA	Epoch-wise Merge	43.25
GraLoRA	Block Decomposition	13.14
G-LoRA	Global-Local Decoupling	**137.23**

Table 8: Entropy-Based Effective Rank (EBER) of update matrices under rank 8 on LLaMA2-7B.

5 Conclusions

In this paper, we propose G-LoRA, which introduces a global–local decoupling structure into low-rank adaptation to better focus adaptation capacity on critical sub-regions. We theoretically derive upper and lower bounds on the rank attainable by the decoupled updates, providing insights into the expressive advantages of this design. Experimental results show that G-LoRA consistently outperforms LoRA and its variants across a series of NLP benchmarks. As future work, we plan to further investigate the benefits of the global–local decoupling structure for mitigating catastrophic forgetting.

6 Limitations

The experimental evaluation in this work is limited to a single language (English) and a single modality (text). As a result, the applicability of G-LoRA to multilingual settings or to other modalities is not directly assessed and remains to be explored. In addition, the allocation ratio between the G-Adapter and the L-Adapter is determined empirically in our experiments, rather than being optimized or adapted automatically across tasks.

7 Acknowledgement

This work was partially supported by the National Natural Science Foundation of China (grants No. 62276099).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M Ponti. 2024. Scaling sparse fine-tuning to large language models. *arXiv preprint arXiv:2401.16405*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*, pages 1–9. Association for Computational Linguistics (ACL).

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code.

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:1803.05457v1*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. 2025. Aurora: Breaking low-rank bottleneck of lora with nonlinear mapping. In *Advances in Neural Information Processing Systems*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Marawan Gamal Abdel Hameed, Aristides Milios, Siva Reddy, and Guillaume Rabusseau. 2024. Rosa: Random subspace adaptation for efficient fine-tuning. *Preprint*, arXiv:2407.07802.

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 (ICCV)*, 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 *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533.

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)*, pages 2790–2799. PMLR.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5254–5276.

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

Weizhong Huang, Yuxin Zhang, Xiawu Zheng, Liuyang, Jing Lin, Yiwu Yao, and Rongrong Ji. 2025b. Dynamic low-rank sparse adaptation for large language models. In *The Thirteenth International Conference on Learning Representations*.

Yeonjoon Jung, Daehyun Ahn, Hyungjun Kim, Taesu Kim, and Eunhyeok Park. 2025. Gralora: Granular low-rank adaptation for parameter-efficient fine-tuning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of NAACL*, pages 1152–1157.

Wen Lai, Alexander Fraser, and Ivan Titov. 2025. Joint localization and activation editing for low-resource fine-tuning. In *Forty-second International Conference on Machine Learning*.

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 (EMNLP)*, pages 3045–3059.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2025a. Beyond higher rank: Token-wise input-output projections for efficient low-rank adaptation. In *Advances in Neural Information Processing Systems*.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2025b. Bora: Towards more expressive low-rank adaptation with block diversity. *Preprint*, arXiv:2508.06953.

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 (ACL)*, pages 4582–4597.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167.

Vijay Chandra Lingam, Atula Neerkaje, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Eunsol Choi, Alex Dimakis, Aleksandar Bojchevski, and Sujay Sanghavi. 2024. Svft: Parameter-efficient fine-tuning with singular vectors. volume 37, pages 41425–41446.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024a. Dora: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Zequan Liu, Yi Zhao, Ming Tan, Wei Zhu, and Aaron Xuxiang Tian. 2024b. Para: Parameter-efficient fine-tuning with prompt-aware representation adjustment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 728–737.

Zihang Liu, Tianyu Pang, Oleg Balabanov, Chaoqun Yang, Tianjin Huang, Lu Yin, Yaoqing Yang, and Shiwei Liu. 2025. LIFT the veil for the truth: Principal weights emerge after rank reduction for reasoning-focused supervised fine-tuning. In *Forty-second International Conference on Machine Learning*.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 121038–121072.

Daiye Miao, Yufang Liu, Jie Wang, Changzhi Sun, Yunke Zhang, Demei Yan, Shaokang Dong, Qi Zhang, and Yuanbin Wu. 2025. TASO: Task-aligned sparse optimization for parameter-efficient model adaptation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 22735–22747, Suzhou, China. Association for Computational Linguistics.

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.

P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. 2019. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*.

Lin Mu, Xiaoyu Wang, Li Ni, Yang Li, Zhize Wu, Peiquan Jin, and Yiwen Zhang. 2025. Denselora: Dense low-rank adaptation of large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 10198–10211.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024. Lisa: Layerwise importance sampling for memory-efficient large

language model fine-tuning. *Advances in Neural Information Processing Systems*, 37:57018–57049.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of NAACL*, pages 2080–2094.

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.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

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. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. 2025. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4823–4836.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *Preprint*, arXiv:2312.12148.

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, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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*.

Jinyang Zhang, Yue Fang, Hongxin Ding, Weibin Liao, Muyang Ye, Xu Chu, Junfeng Zhao, and Yasha Wang. 2025. Adept: Continual pretraining via adaptive expansion and dynamic decoupled tuning. *Preprint*, arXiv:2510.10071.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Heming Zou, Yunliang Zang, Wutong Xu, Yao Zhu, and Xiangyang Ji. 2025. FlyloRA: Boosting task decoupling and parameter efficiency via implicit rank-wise mixture-of-experts. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

A Proof of Proposition 1

Proposition 1. *Assume that the global update matrix* $\mathbf{G} \in \mathbb{R}^{m\times n}$ *satisfies* $\mathrm{rank}(\mathbf{G}) = r$. *Similarly, let the local update matrix* $\mathbf{L} \in \mathbb{R}^{\frac{m}{4}\times\frac{n}{4}}$ *satisfy* $\mathrm{rank}(\mathbf{L}) = 4r$, *which requires* $\frac{m}{4} \geq 4r$ *and* $\frac{n}{4} \geq 4r$, *i.e.,* $m, n \geq 16r$. *To make the two matrices compatible for addition,* $\mathbf{L}$ *is zero-padded to match the dimensions of* $\mathbf{G}$, *yielding* $\mathbf{L}_{new} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$. *Since zero-padding does not alter matrix rank, we have* $\mathrm{rank}(\mathbf{L}_{new}) = 4r$. *The resulting update matrix is given by* $\Delta\mathbf{W} = \mathbf{G}+\mathbf{L}_{new}$, *whose rank satisfies* $3r \leq \mathrm{rank}(\Delta\mathbf{W}) \leq 5r$.

Proof. We first observe that zero-padding does not change the rank of $\mathbf{L}$, and therefore $\mathrm{rank}(\mathbf{L}_{\mathrm{new}}) = \mathrm{rank}(\mathbf{L}) = 4r$.

Upper bound. For any two matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size, the subadditivity of rank implies that $\mathrm{rank}(\mathbf{A}+\mathbf{B}) \leq \mathrm{rank}(\mathbf{A}) + \mathrm{rank}(\mathbf{B})$.

Applying this to $\mathbf{A} = \mathbf{G}$ and $\mathbf{B} = \mathbf{L}_{\mathrm{new}}$ yields $\mathrm{rank}(\Delta\mathbf{W}) = \mathrm{rank}(\mathbf{G}+\mathbf{L}_{\mathrm{new}}) \leq r + 4r = 5r$, which proves the desired upper bound.

Lower bound. A standard inequality for matrix ranks also gives $\big|\mathrm{rank}(\mathbf{A}) - \mathrm{rank}(\mathbf{B})\big| \leq \mathrm{rank}(\mathbf{A}+\mathbf{B})$. Substituting $\mathbf{A} = \mathbf{G}$ and $\mathbf{B} = \mathbf{L}_{\mathrm{new}}$ yields $\mathrm{rank}(\Delta\mathbf{W}) = \mathrm{rank}(\mathbf{G}+\mathbf{L}_{\mathrm{new}}) \geq |\, r - 4r \,| = 3r$, establishing the lower bound.

Tightness of the bounds. We demonstrate that both the upper and lower bounds are achievable.

(i) Achievability of the upper bound $5r$. If the nonzero components of $\mathbf{G}$ lie in a block disjoint from the support of $\mathbf{L}_{\mathrm{new}}$, and if the row and column spaces of the two matrices are chosen to be independent, then their ranks add:

$$\mathrm{rank}(\mathbf{G}+\mathbf{L}_{\mathrm{new}}) = \mathrm{rank}(\mathbf{G})+\mathrm{rank}(\mathbf{L}_{\mathrm{new}}) = 5r.$$

(ii) Achievability of the lower bound $3r$. Within the upper-left $\frac{m}{4} \times \frac{n}{4}$ sub-matrix (the support of $\mathbf{L}$), one may choose a basis in which $\mathbf{L}$ has $4r$ independent directions. Construct $\mathbf{G}$ such that its row and column spaces are contained within r-dimensional subspace of those of $\mathbf{L}$, and choose $\mathbf{G}$ to cancel r linearly independent directions within this shared subspace.

$$\mathrm{rank}(\mathbf{G}+\mathbf{L}_{\mathrm{new}}) = 4r - r = 3r$$

Conclusion. Combining the above results yields the tight rank bounds $3r \leq \mathrm{rank}(\Delta\mathbf{W}) \leq 5r$, completing the proof. □

B Algorithm of G-LoRA Training

To facilitate reproducibility, we provide the training procedure of G-LoRA in Algorithm 1, including the optimization of the global adapter (G-Adapter) and local adapter (L-Adapter) and the iterative re-initialization of the L-Adapter to focus on locally critical sub-regions.

Algorithm 1 G-LoRA Training

Input: pretrained weight matrix $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$, global rank r_g, local rank r_l, epochs T, re-initialization frequency f_{re}
Output: residual weight matrix $\mathbf{W}_{\text{res}}$, G-Adapter low-rank update $\mathbf{G_A}$, $\mathbf{G_B}$, L-Adapter low-rank update $\mathbf{L}_\mathbf{A}^T$, $\mathbf{L}_\mathbf{B}^T$

Compute SVD of $\mathbf{W}_0 = \mathbf{U\Sigma V}^\top$
Initialize global factors:
$\mathbf{G_A} \leftarrow \mathbf{U}_{[:,-r_g:]}\mathbf{\Sigma}^{1/2}_{[-r_g:,-r_g:]}$
$\mathbf{G_B} \leftarrow \mathbf{\Sigma}^{1/2}_{[-r_g:,-r_g:]}\mathbf{V^T}_{[:,-r_g:]}$
Update Residual weight matrix:
$\mathbf{W}_{\text{res}} = \mathbf{W}_0 - \mathbf{G_A G_B}$
Set initial permutations:
$\boldsymbol{\pi}_r^0 \leftarrow (0,\ldots,m-1)$, $\boldsymbol{\pi}_c^0 \leftarrow (0,\ldots,n-1)$
for epoch $i = 0$ to T **do**
  if $i \bmod f_{\text{re}} = 0$ **then**
    if $i \neq 0$ **then**
      Merge local updates into $\mathbf{W}_{\text{res}}$
      Compute importance scores (Eq. 9)
      Derive row/col reordering $(\boldsymbol{\pi}_r^i, \boldsymbol{\pi}_c^i)$
      Reorder $\mathbf{W}_{\text{res}} \leftarrow \mathbf{W}_{\text{res}}[\boldsymbol{\pi}_r^i, :][:, \boldsymbol{\pi}_c^i]$
      $m \leftarrow \lfloor m/2 \rfloor$, $n \leftarrow \lfloor n/2 \rfloor$
    end if
    Compute SVD of $\mathbf{W}_{\text{res}}^{[:m,:n]} = \mathbf{U\Sigma V}^\top$
    Initialize local factors:
    $\mathbf{L}_\mathbf{A}^0 \leftarrow \mathbf{U}_{[:,:r_l]}\mathbf{\Sigma}^{1/2}_{[:r_l,:r_l]}$
    $\mathbf{L}_\mathbf{B}^0 \leftarrow \mathbf{\Sigma}^{1/2}_{[:r_l,:r_l]}\mathbf{V}^\top_{[:,-r_g:]}$
  end if
  for each batch $(\mathbf{x}, \hat{\mathbf{y}})$ **do**
    Reorder input features: $\mathbf{x}' = \mathbf{x}[:, \boldsymbol{\pi}_r^i]$
    Compute output according to Eq. 2
    Restore output order: $\mathbf{y} = \mathbf{y}'[:, (\boldsymbol{\pi}_c^i)^{-1}]$
    Compute loss $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
    Update $\mathbf{G}_A, \mathbf{G}_B, \mathbf{L}_A^i, \mathbf{L}_B^i$
  end for
end for
return $\mathbf{W}_{\text{res}}, \mathbf{G}_A, \mathbf{G}_B, \mathbf{L}_A^T, \mathbf{L}_B^T$

Base Model	LLaMA2-7B/13B		LLaMA3-8B	
Initialization	SVD	RandomSVD (4×)	SVD	RandomSVD (4×)
$r_g : r_l$	3 : 1			
LR	1e-4			
LR Scheduler	Linear			
Epochs	10			
Rank	8			
Optimizer	AdamW			
Batch size	4			
Warmup steps	500			
Dropout rate	0.05			
Re-initialization freq	2			

Table 9: Hyperparameter configurations of G-LoRA on LLaMA2-7B, LLaMA2-13B and LLaMA3-8B on commonsense reasoning tasks.

Base Model	LLaMA2-7B/13B		LLaMA3-8B	
Initialization	SVD	RandomSVD (4×)	SVD	RandomSVD (4×)
$r_g : r_l$	1 : 3	1 : 3	3 : 1	3 : 1
LR	3e-4			
LR Scheduler	Linear			
Epochs	10			
Rank	8			
Optimizer	AdamW			
Batch size	4			
Warmup steps	500			
Dropout rate	0.05			
Re-initialization freq	2			

Table 10: Hyperparameter configurations of G-LoRA on LLaMA2-7B, LLaMA2-13B and LLaMA3-8B on mathematical reasoning tasks.

Base Model	LLaMA3-8B-Instruct		Qwen2.5-7B	
Initialization	SVD	RandomSVD (8×)	SVD	RandomSVD (8×)
$r_g : r_l$	1 : 3	1 : 3	3 : 1	3 : 1
LR	2e-5	1e-5	1e-5	1e-5
LR Scheduler	Linear			
Epochs	10			
Rank	8			
Optimizer	AdamW			
Batch size	4			
Warmup steps	500			
Dropout rate	0.05			
Re-initialization freq	2			

Table 11: Hyperparameter configurations of G-LoRA on LLaMA3-8B-Instruct and Qwen2.5-7B on code generation tasks.

C Experiments Setting

All experiments were conducted on a single NVIDIA A6000 GPU (48GB). For each configuration, we report results from a single run with a fixed random seed of 42, following common practice in Natural Language Processing. We summarize the hyperparameter configurations of G-LoRA across different tasks and base models in Tables 9, 10, and 11. These tables report the training and structural choices adopted for G-LoRA in our experiments. The hyperparameter configurations for the compared methods are summarized in Table 12,

Task	Commonsense		Mathematical	Code		
Base Model	LLaMA2-7B / 13B / LLaMA3-8B		LLaMA2-7B / 13B / LLaMA3-8B	LLaMA3-8B-Instruct / Qwen2.5-7B		
Method	PiSSA, RoSA, GraLoRA	Other	All	MiLoRA	DenseLoRA	Other
LR	2e-4	1e-4	3e-4	3e-5	1e-5	5e-5
LR Scheduler	Linear					
Epochs	10					
Rank	8					
Optimizer	AdamW					
Batch size	4					
Warmup steps	500					
Dropout rate	0.05					

Table 12: Hyperparameter configurations for all methods across different tasks and base models.

which follow prior studies or correspond to the best results obtained through hyperparameter tuning.

For model selection, we construct a validation set by randomly sampling 1200 instances from the official training split of each benchmark task. The validation set is used solely for hyperparameter verification and checkpoint selection, and is not included in the training set. This protocol follows a similar practice adopted in prior work (Hu et al., 2023), which typically uses smaller validation subsets (e.g., 120 instances).

Across all experiments, the total low-rank adaptation budget is fixed to $r = 8$. AdamW is used as the optimizer with linear learning rate decay and 500 warmup steps, and training is performed for 10 epochs. The batch size is set to 4 (as experiments are conducted on a single GPU without gradient accumulation, this is also the effective batch size for parameter updates). To improve training stability, dropout rate is set to 0.05. In addition, the local adapter is periodically re-initialized every 2 epochs to refresh local low-rank subspaces during optimization.

Regarding the internal structure of G-LoRA, the relative rank allocation between the global adapter and the local adapter ($r_g : r_l$) is adjusted in a task-dependent manner. For commonsense reasoning tasks, a global-dominant configuration is adopted to encourage more global low-rank adaptation over the parameter space. In contrast, for mathematical reasoning and code generation tasks, either a local-dominant or a model-dependent allocation is employed, reflecting the stronger reliance of these tasks on localized representations.

Finally, we consider two strategies for initializing low-rank bases. Besides SVD-based initialization, a randomized SVD variant is used, which accelerates basis construction via an oversampling factor (denoted as $k\times$) and is well suited to the periodic re-initialization mechanism in G-LoRA.

D Dataset Statistics

We describe the datasets used for fine-tuning and evaluation across three tasks. All datasets and related scientific artifacts used in this paper comply with the licenses stated in the original papers or websites and are used solely for research purposes.

D.1 Commonsense Reasoning

Commonsense15K (Hu et al., 2023) is a reduced-scale subset of Commonsense170K provided by the original benchmark, comprising 15K training instances. Commonsense170K is constructed by formatting the training sets of multiple benchmarks—including BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, ARC-e, ARC-c, and OpenBookQA—into unified instruction-style templates.

Dataset	#Examples	Answer Type
BoolQ (Clark et al., 2019)	3270	Yes/No
PIQA (Bisk et al., 2020)	1830	Option
Social IQA (Sap et al., 2019)	1954	Option
ARC-C (Clark et al., 2018)	1172	Option
ARC-E (Clark et al., 2018)	2376	Option
OpenbookQA (Mihaylov et al., 2018)	500	Option
HellaSwag (Zellers et al., 2019)	10042	Option
Winogrande (Sakaguchi et al., 2021)	1267	Option

Table 13: Statistics of evaluation datasets for commonsense reasoning tasks.

D.2 Mathematical Reasoning

For mathematical reasoning, we use Math10K (Hu et al., 2023), which includes 10K mathematical training instances collected from GSM8K, MAWPS, and MAWPS-single, as well as 1,000 examples from AQuA. As the original datasets primarily provide equations with final answers, Math10K

augments these sources with step-by-step reasoning annotations generated via zero-shot chain-of-thought prompting with ChatGPT.

Dataset	#Examples	Answer Type
MultiArith (Roy and Roth, 2016)	600	Number
GSM8K (Cobbe et al., 2021)	1319	Number
AddSub (Hosseini et al., 2014)	395	Number
AQuA (Ling et al., 2017)	254	Option
SingleEq (Koncel-Kedziorski et al., 2015)	508	Number
SVAMP (Patel et al., 2021)	1000	Number
MAWPS (Koncel-Kedziorski et al., 2016)	238	Number

Table 14: Evaluation benchmarks and data statistics for mathematical reasoning tasks.

D.3 Code Generation

We further incorporate CodeAlpaca-20K (Chaudhary, 2023) for code generation tasks. This dataset consists of approximately 20K synthetically generated instruction–response pairs, covering a wide range of programming scenarios and providing supervision for instruction tuning in the code domain.

Dataset	#Examples	Answer Type
MBPP (Austin et al., 2021)	257	Code
HumanEval (Chen et al., 2021)	164	Code

Table 15: Dataset statistics for code generation tasks.

E Computational Cost Analysis

Based on Table 16, we provide a stage-wise analysis of the computational cost of different low-rank adaptation methods, focusing on initialization, re-initialization, and standard training.

G-LoRA, PiSSA, and RoSA all rely on singular value decomposition of the pretrained weights to construct their initial low-rank subspaces, which leads to comparable initialization costs across these methods. Since this operation is performed only once at the beginning of training, its contribution to the overall training time remains limited. The main differences arise during re-initialization. LoRA and PiSSA do not perform re-initialization and therefore incur no additional cost at this stage. In contrast, RoSA repeatedly applies singular value decomposition to full or near-full matrices across multiple training phases, causing the re-initialization cost to accumulate with training iterations. G-LoRA differs in that re-initialization is conducted only on progressively shrinking local subspaces. As training proceeds, the size of the matrices involved in the decomposition is gradually reduced, which substantially lowers the computational cost of each re-initialization step compared to RoSA. When the randomized SVD variant is adopted, the re-initialization overhead can be further reduced to an almost negligible level. In addition, G-LoRA introduces a modest additional cost during training due to the reordering operations applied to the inputs and outputs. This results in an overhead of approximately one extra minute over the entire training process. From an overall cost perspective, this additional overhead remains minor and does not introduce a noticeable computational burden compared to the dominant training cost.

Overall, the global–local decoupling in G-LoRA improves model performance while keeping the computational cost largely unchanged. By separating global and local adaptations, the method enables more effective use of model capacity without introducing substantial additional overhead.

F Entropy-Based Effective Rank

To quantify how the update energy is distributed across different singular directions, we adopt the Entropy-Based Effective Rank (EBER) (Roy and Vetterli, 2007) to evaluate the update matrix.

Given the singular values $\{\sigma_i\}$ of the update matrix $\Delta\mathbf{W}$, we define a normalized energy distribution based on squared singular values $p_i = \frac{\sigma_i^2}{\sum_j \sigma_j^2}$. The entropy of this distribution is $H = -\sum_i p_i \log p_i$, and the effective rank is defined as:

$$\mathrm{EBER}(\Delta\mathbf{W}) = \exp(H).$$

Intuitively, this metric measures how widely the update energy is distributed across singular directions. A higher effective rank indicates that the update utilizes more meaningful directions rather than concentrating the energy in only a few dominant components.

G Robustness under Unified Hyperparameter Configuration

To examine whether the observed improvements depend on task-specific tuning of internal parameters, we conduct an additional experiment in which the configuration of G-LoRA is fixed across all tasks. Specifically, we adopt a unified setting with a global-to-local ratio of $g : l = 3 : 1$ for all benchmarks and models, without any per-task adjustments. This experiment evaluates whether the proposed architecture remains effective under a fully shared configuration. As shown in Table 17,

Method	Initialization	Re-initialization	Training	Total Time
LoRA	7s	0s	27 min * 10	274 min
PiSSA	6 min	0s	28 min * 10	283 min
RoSA	6 min	6 min * 4	28 min * 10	312 min
G-LoRA$_{svd}$	6 min	2 min * 4	29 min * 10	306 min
G-LoRA$_{randomsvd}$	6 min	1.2s * 4	29 min * 10	297 min

Table 16: Ablation results for different initialization strategies of G-Adapter and L-Adapter.

even under this unified configuration, G-LoRA consistently outperforms LoRA and PiSSA across commonsense reasoning, mathematical reasoning, and code generation tasks. These results suggest that the performance improvements primarily stem from the architectural design of G-LoRA rather than from task-specific hyperparameter tuning.

Base Model	Method	Commonsense	Math	Code
LLaMA2 (7B)	LoRA	77.8	70.0	–
	PiSSA	78.7	70.2	–
	G-LoRA	**78.3**	**71.5**	–
LLaMA2 (13B)	LoRA	81.0	73.1	–
	PiSSA	81.5	73.9	–
	G-LoRA	**82.7**	**74.8**	–
LLaMA3 (8B)	LoRA	83.1	77.8	–
	PiSSA	82.7	78.6	–
	G-LoRA	**84.6**	**79.8**	–
LLaMA3 (8B) Instruct	LoRA	–	–	59.5
	PiSSA	–	–	60.8
	G-LoRA	–	–	**62.2**
Qwen2.5 (7B)	LoRA	–	–	71.7
	PiSSA	–	–	73.7
	G-LoRA	–	–	**74.6**

Table 17: Performance under a unified hyperparameter configuration ($g : l = 3 : 1$) without task-specific hyperparameter tuning.

H Comparison with Sparse Fine-Tuning

Since the proposed method updates selected subregions after reordering, it may appear structurally similar to sparse fine-tuning methods that restrict updates to a subset of weights. To provide a clearer comparison, we compare G-LoRA with a representative sparse fine-tuning method, SpIEL (Ansell et al., 2024), under a comparable trainable parameter budget. We conduct the experiment on a mathematical reasoning task using LLaMA2-7B, and report the results in Table 18.

Although restricting updates to reordered subblocks may resemble structured sparsity, the underlying parameterization differs fundamentally. Sparse fine-tuning limits updates to a subset of parameters, whereas G-LoRA constrains updates to a low-rank subspace through factorized matrices. In G-LoRA, both global and local components are represented in a low-rank parameterization, enabling dense yet subspace-structured updates within selected regions. Under comparable parameter budgets, this design yields a +2.2% accuracy improvement over SpIEL on the evaluated task.

Method	Setting	Avg. Acc.
SpIEL-AG	density = 0.3%	69.6
G-LoRA (ours)	$r = 8$	**71.8**

Table 18: Comparison with sparse fine-tuning under comparable parameter budgets.

I Effect of Batch Size

To further examine whether batch size affects the relative performance across methods, we increase the effective global batch size to 16 via gradient accumulation while keeping all other training configurations unchanged. As shown in Table 19, increasing the batch size leads to only minor performance changes, while the relative ranking among methods remains unchanged. In particular, G-LoRA continues to outperform both LoRA and PiSSA under the larger effective batch size. These results suggest that the performance gains of G-LoRA remain stable across different batch size settings.

Batch Size = 16	LoRA	PiSSA	**G-LoRA (ours)**
Avg. Acc.	69.6	69.1	**70.9**

Table 19: Performance comparison under a larger effective batch size.

J Scalability and Distributed Training Considerations

We discuss the distributed training aspects of G-LoRA, focusing on the periodic SVD operations and weight reordering steps.

J.1 Computation Overhead

All experiments reported in this work were conducted on a single NVIDIA A6000 GPU. The additional runtime overhead (approximately one minute) is incurred in this single-device setting, and detailed wall-clock statistics are provided in Appendix E. Importantly, the periodic SVD and reordering operations are applied only to the **L-Adapter**, rather than the frozen backbone weights or the **G-Adapter**. Although the formulation in the main text presents reordering over the full weight matrix for conceptual clarity, this is mathematically equivalent to permuting only the low-rank factors of the L-Adapter.

J.2 Distributed Training Feasibility

In a distributed setting with sharded backbone weights, the method can be implemented without modifying large-scale model parameters. A practical implementation strategy is as follows:

- The backbone weights remain frozen and fully sharded across devices.
- The G-Adapter follows the same structure as standard LoRA and introduces no additional communication overhead beyond standard LoRA.
- The SVD and permutation steps are applied only to the L-Adapter parameters, which account for approximately 0.4% of the model parameters.

Under this design, both computation and communication costs scale with the adapter size rather than the backbone size. Consequently, the method does not require global SVD or permutation over large-scale model weights, making it compatible with large distributed training setups.

J.3 Optimization Stability

Another concern is that the dynamically updated local region may introduce a non-stationary optimization objective. In practice, the optimized region is *monotonically shrinking*. At each reordering step, the active local region forms a nested subset of the previous one, i.e.,

$$\mathcal{R}_{t+1} \subseteq \mathcal{R}_t.$$

Parameters outside the current region remain unchanged. Therefore, the optimization process does not repeatedly switch between unrelated parameter subsets. Since the loss function itself remains unchanged and the updates occur within progressively refined subspaces, the overall optimization landscape is not fundamentally altered.

J.4 Inference Compatibility

Although the algorithm description includes permutation operations for clarity, these operations are applied only to the L-Adapter parameters in practice. During inference, the learned low-rank updates can be merged into the backbone weights in the same manner as standard LoRA:

$$W' = W + AB.$$

As a result, the final inference graph is identical to that of LoRA, maintaining compatibility with optimized inference engines such as the vLLM engine.

K Sensitivity to Training Epochs

The number of training epochs can influence model performance. While our main experiments follow prior work and adopt 10 epochs (with a re-initialization frequency of 2), we additionally examine a shorter training schedule of 3 epochs (with a re-initialization frequency of 1), as commonly used in standard fine-tuning.

As shown in Table 20, increasing the number of training epochs is associated with slight improvements in absolute performance across all methods. More importantly, the relative ranking among methods remains consistent under both settings, with G-LoRA consistently outperforming LoRA and PiSSA. Notably, the performance gap becomes more pronounced under the longer training schedule, suggesting that additional training may allow G-LoRA to better leverage its advantages.

Method	3 epochs	10 epochs
LoRA	69.9	70.0
PiSSA	68.9	70.2
G-LoRA	**70.2**	**71.8**

Table 20: Performance under different training epochs.