Enhancing PEFT Efficiency by Adaptively Reducing Low-Rank Modules

Anonymous ACL submission

Abstract

As a prominent Parameter-Efficient Fine-Tuning (PEFT) method, LoRA is widely used for efficiently fine-tuning large language models (LLMs). However, LoRA's uniform insertion of trainable modules to target modules across all layers often results in redundancy in the number of trainable modules, and we contend that reducing the number of these modules can further enhance the efficiency of PEFT. To address this issue, we propose Gradient-Guided Redundancy Reduction ($\mathcal{G}^2\mathcal{R}^2$), a novel module-level approach that adaptively prunes redundant LoRA modules, which boosts fine-tuning efficiency while preserving or even improving performance. Specifically, $\mathcal{G}^2\mathcal{R}^2$ evaluates the contribution and redundancy of trainable modules using a Gradient-Based Redundancy Evaluation score, which leverages gradient information to achieve this. Based on this score, $\mathcal{G}^2\mathcal{R}^2$ progressively eliminates redundant LoRA modules through a Three-Stage Redundancy Reduction Strategy. Extensive experiments demonstrate that $\mathcal{G}^2\mathcal{R}^2$ not only boosts fine-tuning efficiency but also maintains or even surpasses state-of-the-art methods across commonsense reasoning and natural language understanding tasks.

1 Introduction

Fine-tuning large language models (LLMs) is essential for adapting them to diverse applications, including instruction tuning, preference alignment, and domain adaptation. However, full fine-tuning requires updating all parameters, leading to substantial memory and computational costs. To address this, Parameter-Efficient Fine-Tuning (PEFT) methods (Vucetic et al., 2022; Houlsby et al., 2019; Zi et al., 2023; Zhang et al., 2024; Yang et al., 2023; Ding et al., 2023; Chen et al., 2023) freeze most model parameters and introduce a small number of trainable ones into specific modules, significantly reducing resource consumption. LoRA (Hu et al., 2022) is the most widely used method to reduce parameters by reparameterizing ΔW. And then, many methods seek to further reduce trainable prameters by refining reparameterization strategies, e.g. VeRA (Kopiczko et al., 2023), DoRA (Liu et al., 2024a), AdaLoRA (Zhang et al., 2023) and HRA (Yuan et al., 2024).

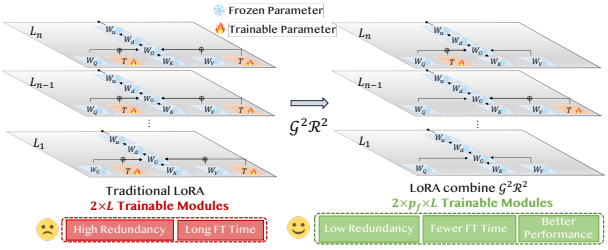

Figure 1: Traditional LoRA (left) suffers from high redundancy and longer fine-tuning time. $\mathcal{G}^2\mathcal{R}^2$ (right) reduces redundancy, prunes redundant LoRA modules, and accelerates fine-tuning while improving performance. Where $p_f < 1$.

However, LoRA's strategy of uniformly inserting trainable parameters into all target modules across each layer frequently leads to redundancy in the number of trainable module (refer to the left figure in Figure 1). We contend that reducing the number of these trainable modules not only decreases the total trainable parameters but also further enhances the efficiency of PEFT. To address these issues, we propose a novel parameter reduction method, **G**radient-**G**uided **R**edundancy **R**eduction Method ($\mathcal{G}^2\mathcal{R}^2$), which adaptively reduces redundant training parameters at the module level while maintaining, or even improving training quality (refer to the right figure in Figure 1).

Our $\mathcal{G}^2\mathcal{R}^2$ is built on two key components: Gradient-Based Redundancy Evaluation Scores and a module-level Three-Stage Redundancy Reduction Strategy, which ensure both the stability and accuracy of the pruning process. The Gradient-Based Redundancy Evaluation score exploit gra-

dient information to quantify the contribution of each LoRA module, enbling a precise measure of module-level redundancy. Modules that exhibit high scores are identified as redundant and targeted for pruning. Building on this evaluation, the Three-Stage Redundancy Reduction Strategy systematically reduces redundancy in three phases: Warmup Stage, Progressive Pruning Stage and Fine-Tuning Refinement Stage. In first stage, all LoRA modules are initially assessed to identify potentially less contributive ones. In second stage, redundant modules are progressively pruned over several steps, ensuring that the reduction process is gradual and does not disrupt convergence. In third stage, remaining modules are further optimized to maximize their effectiveness on the target task. By focusing on module-level pruning instead of element-level parameter removal, $\mathcal{G}^2\mathcal{R}^2$ achieves a more efficient reduction in trainable parameters while maintaining robust convergence and performance—a critical advantage for large-scale model fine-tuning.

We conduct extensive experiments across diverse tasks and model architectures to thoroughly evaluate the effectiveness and efficiency of $\mathcal{G}^2\mathcal{R}^2$. Specifically, we benchmark our method on commonsense reasoning using LLaMA2-7B and LLaMA3-8B, and on natural language understanding (GLUE) with DeBERTaV3-base. The experimental results demonstrate that $\mathcal{G}^2\mathcal{R}^2$ not only achieves state-of-the-art performance (**+0.4** on LLaMA2-7B, **+7.1** on LLaMA3-8B and **+0.44** on DeBERTaV3) but also significantly improves the efficiency of parameter-efficient fine-tuning by reducing training time (**-11.6%**) and decreasing memory usage (**-22.1%**). The contributions of our work are as follows:

- We introduce a gradient-driven metric to assess the redundancy of LoRA modules. By focusing on module contributions inferred through gradient information, our approach enables precise identification of less critical components, laying a solid foundation for efficient module-level pruning.
- We design a module-level pruning framework encompassing three phases, Warmup, Progressive Pruning, and Fine-Tuning Refinement, that incrementally remove redundant modules without causing catastrophic failure. This strategy preserves stable convergence, reduces overall training time, and avoids significant memory overhead.
- We have conducted extensive experiments on commonsense reasoning and natural language understanding, and the results show that our $\mathcal{G}^2\mathcal{R}^2$ achieves state-of-the-art performance with higher fine-tuning efficiency.

2 Related Work

Parameter Efficient Fine-Tuning. Parameter-Efficient Fine-Tuning (PEFT) methods have garnered considerable attention. Houlsby et al. (2019) introduced bottleneck-shaped modules inserted after attention and FFN layers, and Pfeiffer et al. (2021b) found that a single adapter after the self-attention layer can achieve performance comparable to placing two adapters per Transformer block. Prompt-based methods, such as Lester et al. (2021), prepend trainable "soft prompts" to the model's input embeddings while keeping the main weights frozen. Some approaches fine-tune only specific subsets of existing parameters (e.g., biases in BitFit (Zaken et al., 2022), or crucial parameters as in DiffPruning (Guo et al., 2021) and FAR (Vucetic et al., 2022)). Aghajanyan et al. (2021) introduced the concept of intrinsic dimensionality, showing that larger models can be tuned in lower-dimensional subspaces. Building on this insight, LoRA (Hu et al., 2022) factorizes weight updates into low-rank matrices. Later extensions include MAM Adapter (He et al., 2022), which combines scaling parallel adapters with soft prompts; AdaLoRA (Zhang et al., 2023), allocating parameter budgets based on importance scores; DyLoRA (Valipour et al., 2023), training a range of ranks simultaneously; and LongLoRA (Chen et al., 2023), extending context sizes for large language models. Currently, many studies aim to devise more parameter-efficient low-rank decomposition strategies to further reduce the total number of parameters, e.g. DoRA (Liu et al., 2024a), VeRA (Kopiczko et al., 2023), BOFT (Liu et al., 2024b) and HRA (Yuan et al., 2024).

Parameter Pruning. Pruning methods typically begin by evaluating the importance of each parameter, then removing those deemed less critical. While a common metric is parameter magnitude (Li et al., 2018; Lee et al., 2021; Han et al., 2015; Paganini and Forde, 2020; Zafrir et al., 2021a), large weights are not always crucial, and small weights can be indispensable. An alternative is *sensitivity-based* scoring, which measures the change in loss

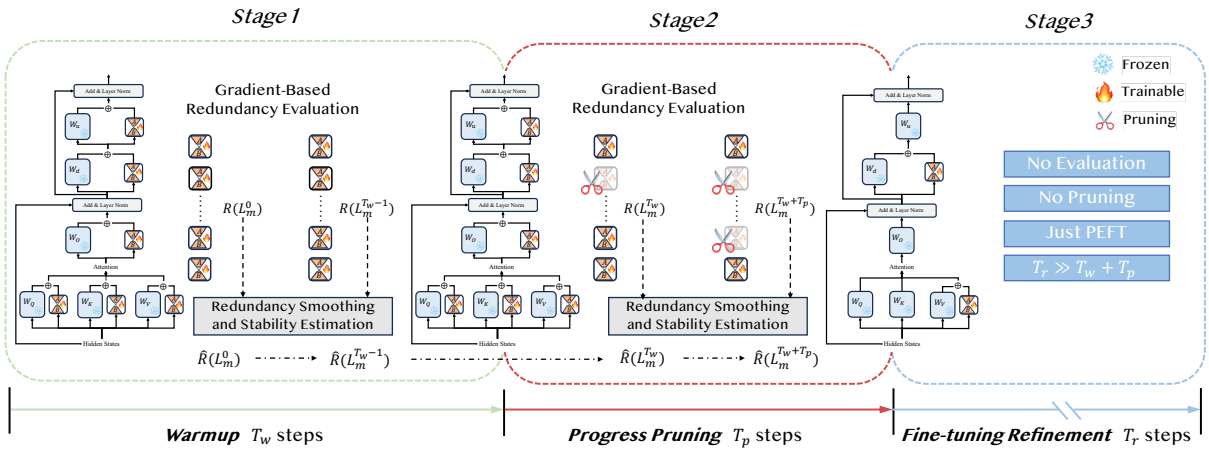

Figure 2: Overview of the Three-Stage Redundancy Reduction Strategy. Warmup: All LoRA modules are inserted to target modules, and redundancy is evaluated with Smoothing and Stability Estimation. Progressive Pruning: High-redundancy modules are gradually pruned. Fine-Tuning Refinement: Only low-redundancy modules remain for final tuning, ensuring efficiency without compromising performance.

resulting from pruning a parameter (Molchanov et al., 2019; Sanh et al., 2020; Liang et al., 2021; Zhang et al., 2022; Ma et al., 2023). Molchanov et al. (2019) approximate the pruning error using a first-order Taylor expansion to avoid computing the Hessian, while Zhang et al. (2022) propose PLATON, which accounts for both sensitivity and estimation uncertainty to stabilize training. In the context of large language models, Ma et al. (2023) introduce LLM-Pruner, extending pruning techniques to these massive architectures.

3 Method

In this section, we introduce our Gradient-Guided Redundancy Reduction Method ($\mathcal{G}^2\mathcal{R}^2$), designed to address the redundant allocation of trainable parameters and improve efficiency of PEFT. ($\mathcal{G}^2\mathcal{R}^2$) achieve this by leveraging a Three-Stage Redundancy Reduction Strategy: Warmup, Progressive Pruning, and Fine-Tuning Refinement (see Figure 2). In the Warmup stage, LoRA modules are inserted at all target positions and their module-level redundancy is measured using our Gradient-Based Redundancy Evaluation Score. Next, during the Progressive Pruning stage, modules with high redundancy are gradually pruned based on a dynamic threshold. Finally, in the Fine-Tuning Refinement stage, only the retained modules are updated, reducing computational cost and resource usage compared to the original LoRA method. For more details, please refer to the following sections.

3.1 Low-Rank Adaptations

LoRA (Hu et al., 2022) proposes to adapt a pre-trained model by injecting low-rank updates into certain weight matrices, thereby reducing the number of trainable parameters. Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, instead of updating W_0 during fine-tuning, LoRA keeps W_0 frozen and learns a low-rank update ΔW such that:

$$\Delta W = AB, \tag{1}$$

where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$, with $r \ll \min(d, k)$. The adapted weight is then given by:

$$W = W_0 + \Delta W = W_0 + AB. \tag{2}$$

For a given input x, the forward pass is computed as:

$$h = xW_0 + s \cdot xAB, \tag{3}$$

where $s \geq 1$ is a tunable scalar hyperparameter. During training, only the low-rank matrices A and B are updated, while W_0 remains fixed. In this work, we focus on this simplest form of low-rank adaptation as proposed in LoRA.

3.2 Gradient-Based Redundancy Evaluation

To accurately evaluate the redundancy of low-rank modules, we draw inspiration from the importance assessment techniques in parameter pruning and propose Gradient-Based Redundancy Evaluation. We consider a LoRA module defined as $L = AB$, where $L \in \mathbb{R}^{d\times k}$, $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$. Suppose that given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$, where

N is the number of samples. We denote that l_{ij} is the element in the *i-th* row and *j-th* column of matrix L. Before evaluating the redundancy of a LoRA module, we first measure the sensitivity of each parameter within the module to the loss (LeCun et al., 1989; Molchanov et al., 2019). The sensitivity of l_{ij} is approximated by the change in loss when l_{ij} is zeroed out. According to this, the change in loss can be formulated as:

$$S(l_{ij}) = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{l_{ij}}(\mathcal{D}) - \mathcal{L}_{l_{ij}=0}(\mathcal{D})| \quad (4)$$

Where $\mathcal{L}$ represents the loss function. Calculating $S(l_{ij})$ for each parameter directly using Formula (4) is computationally expensice since for each module, an extra computation is needed to compute the network's loss after setting that parameter to zero. We can utilize first-order Taylor expansion to approximate the calculation of $S(l_{ij})$, which will effectively reduce the computational complexity.

$$S(l_{ij}) = |\frac{\partial\mathcal{L}(\mathcal{D})}{\partial l_{ij}} l_{ij} + \mathcal{O}\left(\|l_{ij}\|^2\right)| \quad (5)$$

Now all we need to do is calculate the gradient of l_{ij}. Given that $l_{ij} = \sum_{m=1}^{r} A_{im}B_{mj}$, the subscripts indicate the corresponding rows and columns. On the basis of the existing deep learning framework, e.g., Pytorch, it is easy for us to obtain $\frac{\partial\mathcal{L}(\mathcal{D})}{\partial A}$ and $\frac{\partial\mathcal{L}(\mathcal{D})}{\partial B}$. Based on the rules of differentiation, we can get that:

$$\frac{\partial\mathcal{L}(\mathcal{D})}{\partial l_{ij}} = \sum_{m=1}^{r} \frac{1}{B_{mj}} \frac{\partial\mathcal{L}(\mathcal{D})}{\partial A_{im}} = \sum_{m=1}^{r} \frac{1}{A_{im}} \frac{\partial\mathcal{L}(\mathcal{D})}{\partial B_{mj}} \quad (6)$$

By substituting (6) into (5), we obtain $S(l_{ij})$ for each parameter in the LoRA module. We then define a module-level redundancy measure:

$$R(L) = \frac{d \times k}{\sum_{i=1}^{d}\sum_{j=1}^{k} S(l_{ij})} \quad (7)$$

as the redundancy of LoRA module L. If the average sensitivity of the elements within a LoRA module to the loss is low, we infer that the module contributes minimally to the model's adaptation for the specific downstream task. Such modules can be pruned with minimal impact on the model's overall performance.

3.3 Three-Stage Redundancy Reduction Strategy

To ensure the stability of the pruning process and avoid catastrophic failure during fine-tuning, we propose a Three-Stage Redundancy Reduction Strategy. Our Three-Stage Redundancy Reduction Strategy systematically eliminates redundant LoRA modules while preserving those with lower redundancy, thereby enhancing fine-tuning efficiency. This process unfolds in three stages: Warmup, Progressive Pruning, and Fine-Tuning Refinement.

3.3.1 Warmup Stage

In the initial Warmup phase, it can gather sufficient gradient information on each module's contribution to the task, enabling more informed pruning decisions later. At each training step t, we compute a module-level redundancy measure $R(L_m^t)$ for each LoRA module L_m, using the method described in Section 3.2. However, due to gradient descent randomness, Dropout, and other stochastic factors, $R(L_m^t)$ may fluctuate considerably (Zhang et al., 2022).

To stabilize $R(L_m^{(t)})$, we follow (Zhang et al., 2022) and maintain an exponential moving average of the redundancy measure to achieve Redundancy Smoothing

$$\overline{R}(L_m^t) = \beta_1 \overline{R}(L_m^{t-1}) + (1-\beta_1) R(L_m^t), \quad (8)$$

and we also apply Redundancy Stability Estimation to quantify fluctutations between $R(L_m^t)$ and its smoothed value $\overline{R}(L_m^t)$

$$\begin{aligned} U(L_m^t) &= \left| R(L_m^t) - \overline{R}(L_m^t) \right|, \\ \overline{U}(L_m^t) &= \beta_2 \overline{U}(L_m^{t-1}) + (1-\beta_2) U(L_m^t), \end{aligned} \quad (9)$$

where $\beta_2 \in (0,1)$ governs the impact of past stabilities.

We then combine the smoothed redundancy measure with its stability estimation to obtain the *final* module-level redundancy at step t:

$$\widehat{R}(L_m^t) = \overline{R}(L_m^t) \cdot \overline{U}(L_m^t). \quad (10)$$

Modules with higher values of $\widehat{R}(L_m^t)$ are regarded as more redundant, whereas modules with lower values are likely more beneficial and thus retained.

3.3.2 Progressive Pruning

After the Warmup phase, we have stabilized redundancy measures $\widehat{R}(L_m^t)$. In the Progressive Pruning stage, we gradually remove the most redundant LoRA modules using a *threshold scheduler* to ensure stability. Let p^t denote the retention threshold at training step t. We initialize $p^0 = p_0$, keeping

most modules active at the start. Following a *Sparsity Step Schedule* (Zhu and Gupta, 2018; Zafrir et al., 2021b), p^t decreases from p_0 to p_f, determining the fraction of modules to retain. At each step, we rank modules in ascending order of $\widehat{R}(L_m^t)$ (i.e., from less redundant to more redundant) and retain only the lowest $p^t\%$. The remaining modules are pruned by zeroing out their trainable parameters.

Concretely, let A_m^t and B_m^t be the trainable parameters of L_m^t at step t. The update rule is:

$$[A_m^{t+1}, B_m^{t+1}] = \mathcal{T}(L_m^t, \widehat{R}(L_m^t)) = \begin{cases} [A_m^t - \eta\nabla_A\mathcal{L}, B_m^t - \eta\nabla_B\mathcal{L}] & \text{Lowest } p^t\%, \\ [\mathbf{0}, \mathbf{0}] & \text{otherwise.} \end{cases} \tag{11}$$

Once both A_m^{t+1} and B_m^{t+1} are zero, the module L_m is effectively pruned at timestep t. By gradually reducing p^t, the model adapts smoothly to the removal of redundant modules, preserving convergence stability. At the end of this stage (which means that p^t reach to p_f), we set the parameters' *requires_grad=False* in redundant LoRA module and delete them. This allows the subsequent fine-tuning stage to involve fewer trainable parameters and a lower memory usage, thus increasing the efficiency of fine-tuning.

3.3.3 Fine-Tuning Refinement

Finally, once p^t reaches p_f, the Fine-Tuning Refinement stage begins. Only the LoRA modules with the lowest redundancy remain, typically constituting $p_f\%$ of the original module count. In this stage, there is no additional computation compared with LoRA, which notably reduces both GPU memory usage and running time compared to original LoRA method. We continue training retained modules to refine key parameters and preserve or improve performance. By the end of Fine-Tuning Refinement, our $\mathcal{G}^2\mathcal{R}^2$ framework yields a more compact model that maintains robust performance with significantly fewer trainable parameters.

3.4 Memory Usage and Time Complexity Analysis

We analyze the additional computational cost and memory requirements introduced by $\mathcal{G}^2\mathcal{R}^2$ compared to LoRA, and how these are offset by efficiency gains in the Fine-Tuning Refinement stage.

Computational Overhead vs. Savings. Our method introduces extra computations in the Warmup and Progressive Pruning stages. Specifically, computing the redundancy measure $R(L)$ for each module requires calculating element-wise sensitivity $I(l_{ij})$, costing $\mathcal{O}(rdk)$ per module. With M LoRA modules, the total overhead across $T_w + T_p$ iterations is $\mathcal{O}((T_w + T_p)Mrdk)$. Additionally, module ranking in Progressive Pruning has a cost of $\mathcal{O}(T_p M \log M)$, which is negligible when $M \ll d, k$.

In the Fine-Tuning Refinement stage, only a fraction p_f of LoRA modules remain active. Given that a full LoRA update per iteration costs $\mathcal{O}(Mrdk)$, pruning reduces this to $\mathcal{O}(p_f Mrdk)$, yielding total savings of $\mathcal{O}(T_r(1 - p_f)Mrdk)$. Thus, our method reduces overall computational cost compared to LoRA when $T_r(1 - p_f) > T_w + T_p$. This condition implies that if the Fine-Tuning Refinement stage dominates the total training iterations and a significant portion of modules are pruned, our $\mathcal{G}^2\mathcal{R}^2$ framework achieves a net reduction in computational cost compared to maintaining all modules in the original LoRA method.

Memory Usage Analysis. $\mathcal{G}^2\mathcal{R}^2$ introduces minimal memory overhead. Throughout training, we store only three scalar redundancy values per module: R, $\overline{R}$, and $\widehat{R}$, which do not significantly increase memory usage. The main additional memory usage arises during sensitivity computation, where intermediate values $S(l_{ij})$ must be temporarily stored. However, since these are computed per module and discarded afterward, they do not persistently impact overall memory consumption. By pruning redundant modules, our method further reduces memory usage in the Fine-Tuning Refinement stage, leading to lower VRAM requirements

Algorithm 1 $\mathcal{G}^2\mathcal{R}^2$

Input: LoRA modules $\{L_m\}_{m=1}^{m=M}$; thresholds $p_0 \to p_f$; three stage iterations T_w, T_p, T_r
Output: Fine-tuned model with pruned LoRA modules
Stage1: Warmup
for $t = 0 \to T_w$ **do**
  Compute $R(L_m^t)$, smooth & combine for $\widehat{R}(L_m^t)$
end for
Stage2: Progressive Pruning
for $t = T_w + 1 \to T_w + T_p$ **do**
  Compute p^t from schedule
  Rank modules by $\widehat{R}(L_m^t)$ (ascending), keep top $p^t\%$, prune rest
end for
Stage3: Fine-Tuning Refinement
for $t = T_w + T_p + 1 \to T_w + T_p + T_r$ **do**
  No further pruning; train only retained modules
end for
return Fine-tuned model

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
LLaMA2-7B	LoRA	0.83	65.3	73.6	77.5	**83.6**	**84.3**	77.4	61.8	71.6	74.4
	DoRA	0.84	69.9	77.6	**77.6**	82.4	80.7	80.2	64.6	**80.0**	76.6
	$\mathcal{G}^2\mathcal{R}^2_{0.2}$	0.83→0.17	**70.4**	**80.0**	77.5	79.7	81.3	**82.3**	**67.2**	77.6	**77.0**
LLaMA3-8B	LoRA	0.70	70.2	81.7	78.7	88.8	83.7	83.9	71.9	79.6	79.8
	DoRA	0.71	70.5	83.4	75.2	79.3	79.3	84.0	66.5	79.8	77.3
	$\mathcal{G}^2\mathcal{R}^2_{0.2}$	0.70→0.14	**73.8**	**87.5**	**79.8**	**94.6**	**85.1**	**90.1**	**79.0**	**85.6**	**84.4**

Table 1: Accuracy comparison of LLaMA2-7B, and LLaMA3-8B with various PEFT methods on eight commonsense reasoning datasets. The best results on each dataset are shown in **bold**.

compared to standard LoRA fine-tuning.

4 Experiments

We conduct a variety of experiments to showcase the efficacy of $\mathcal{G}^2\mathcal{R}^2$ on various tasks including Commonsense Reasoning and Natural Language Understanding (NLU) (We also conduct experiments on Question Answering and Natural Language Generation, please refer to A.3 and A.2). Firstly, we evaluate $\mathcal{G}^2\mathcal{R}^2$ against *LoRA* (Hu et al., 2022) and *DoRA* (Liu et al., 2024a) on commonsense reasoning task. Subsequently, we compare $\mathcal{G}^2\mathcal{R}^2$ with *Full fine-tuning*, *BitFit* (Zaken et al., 2022), *HAdapter* (Houlsby et al., 2019), *PAdapter* (Pfeiffer et al., 2021a), *LoRA* (Hu et al., 2022), *AdaLoRA* (Zhang et al., 2023) and *PiSSA* (Meng et al., 2024) on natural language understanding task. Following this, we explore the fine-tuning efficiency of $\mathcal{G}^2\mathcal{R}^2$ through comparing time comsumption and memory usage with *LoRA* (Hu et al., 2022). Finally, we perform a series of ablation studies to demonstrate the effectiveness of two components of $\mathcal{G}^2\mathcal{R}^2$.

Implementation Details. We integrate $\mathcal{G}^2\mathcal{R}^2$ with LoRA (Hu et al., 2022) to enhance its efficiency by selectively reducing redundant trainable modules. All methods are implemented using *PyTorch* (Paszke et al., 2019) and the *Huggingface PEFT* library (Mangrulkar et al., 2022). Experiments are conducted on a cluster of 8× NVIDIA RTX 3090 GPUs and a cluster of 8× NVIDIA A800 GPUs. For optimization, we use AdamW (Loshchilov and Hutter, 2019) with a weight decay of 0.05 for commonsense reasoning and 0.01 for NLU. We initialize $p_0 = 1.0$ for all tasks and $p_f = 0.2$ for commonsense reasoning tasks, $p_f \in \{0.2, 0.7\}$ for NLU. The exponential moving average parameters are set as $\beta_1 = 0.85$ and $\beta_2 = 0.95$ for all experiments. For more training details, please refer to A.7.

4.1 Commonsense Reasoning

Models and Datasets. We evaluate $\mathcal{G}^2\mathcal{R}^2$ on LLaMA2-7B and LLaMA3-8B using commonsense reasoning tasks. The commonsense reasoning tasks comprise 8 sub-tasks, each with a predefined training and testing set. We follow the setting of (Hu et al., 2023) and amalgamate the training datasets from all 8 tasks to create the final training dataset and conduct evaluations on the individual testing dataset for each task. We set batch size to 16 for all methods, and fine-tune 1 epoch. And then we evaluate the fine-tuned model on 8 reasoning tasks.

Main Results. Table 1 presents the accuracy comparison of LoRA, DoRA, and our proposed $\mathcal{G}^2\mathcal{R}^2$ across multiple commonsense reasoning benchmarks using LLaMA2-7B and LLaMA3-8B. The results demonstrate that $\mathcal{G}^2\mathcal{R}^2$ consistently outperforms both LoRA and DoRA while utilizing fewer trainable parameters. Specifically, for LLaMA2-7B, our method achieves an average accuracy of 77.0%, representing a +2.6% improvement over LoRA (74.4%) and a +0.4% gain over DoRA (76.6%). Similarly, for LLaMA3-8B, our method significantly boosts the average accuracy to 84.4%, yielding a +4.6% increase compared to LoRA (79.8%) and a +7.1% increase over DoRA (77.3%). These improvements highlight the effectiveness of our redundancy-aware fine-tuning strategy in enhancing model performance while reducing the number of trainable parameters.

Notably, these gains are obtained after 1 training epoch, demonstrating that $\mathcal{G}^2\mathcal{R}^2$ accelerates convergence while enhancing performance. This aligns with intuition: after the first two stages, $\mathcal{G}^2\mathcal{R}^2$ significantly reduces the number of trainable parameters (e.g., from 0.83% → 0.17% in LLaMA2-7B), making the model easier to optimize and getting good performance. As a result, our method can achieve comparable or superior performance in significantly less training time.

Method	# Params(%)	MNLI	SST-2	CoLA	QQP	QNLI	RTE	MRPC	STS-B	All
Full FT	100	89.90	95.63	69.19	92.40	94.03	83.75	89.46	91.60	88.25
BitFit	0.05	89.37	94.84	66.96	88.41	92.24	78.70	87.75	91.35	86.22
HAdapter	0.66	90.13	95.53	68.64	91.91	94.11	84.48	89.95	91.48	88.28
PAdapter	0.64	90.33	95.61	68.77	92.04	94.29	85.20	89.46	91.54	88.41
LoRA$_{r=8}$	0.72	90.65	94.95	69.82	91.99	93.87	85.20	89.95	91.60	88.50
AdaLoRA	0.69	**90.76**	96.10	71.45	92.23	94.55	88.09	90.69	91.84	89.46
PiSSA	0.72	90.43	95.87	**72.64**	92.26	94.29	87.00	91.67	91.88	89.50
$\mathcal{G}^2\mathcal{R}^2{}_{0.2}$	0.72→0.15	90.32	**96.44**	71.64	**92.65**	**94.60**	88.45	91.42	**92.17**	89.71
$\mathcal{G}^2\mathcal{R}^2{}_{0.7}$	0.72→0.51	90.43	96.33	72.24	92.36	94.38	**89.17**	**92.60**	92.06	**89.94**

Table 2: Results with DeBERTaV3-base on GLUE development set. The best results on each dataset are shown in **bold**. We report the average correlation for STS-B and accuracy for MRPC. We report mean of 5 runs using different random seeds.

Method	Total Time (s)	Peak / Final Mem (MB)
LoRA	638.35	7156/7156
$\mathcal{G}^2\mathcal{R}^2_{0.2}$	564.14	7156/5578
Δ	**−11.6%**	**+0%/−22.1%**

Table 3: Comparison of fine-tuning time and peak memory usage between LoRA and $\mathcal{G}^2\mathcal{R}^2$ on the MRPC dataset using DeBERTaV3-base.

4.2 Natural Language Understanding

Model and Datasets. We evaluate the fine-tuning performance of DeBERTaV3-base (He et al., 2021) using $\mathcal{G}^2\mathcal{R}^2$ algorithm. We conduct experiments on the General Language Understanding Evaluation (GLUE (Wang et al., 2019)) benchmark. The benchmark includes two single-sentence classification tasks, three similarity and paraphrase tasks and four natural language inference tasks.

Main Results. Table 2 compares $\mathcal{G}^2\mathcal{R}^2$ with baseline methods across different trainable parameter scales on the GLUE development set. Our approach consistently achieves superior or competitive performance across all datasets. For instance, with only 0.15% trainable parameters, $\mathcal{G}^2\mathcal{R}^2$ outperforms PiSSA (0.72% parameters) on 5 out of 8 datasets, achieving a 0.21% improvement on average scores of all 8 tasks. Moreover, as the number of trainable parameters increases, $\mathcal{G}^2\mathcal{R}^2$ maintains its advantage, demonstrating stable and efficient parameter utilization. For example, when fine-tuning with 0.51% parameters, our method surpasses PiSSA (0.72% parameters) with notable gains of +0.44% on average scores of all 8 tasks.

4.3 Efficiency Analysis

To evaluate the efficiency of our proposed method, we conduct a comparative analysis of fine-tuning time and memory consumption. We fine-tune the DeBERTaV3 model on the MRPC dataset for 20 epochs using both $\mathcal{G}^2\mathcal{R}^2$ and LoRA, measuring per-epoch time consumption, memory usage, and validation performance. The experimental results are presented in Figure 3 and Table 3.

Time comsuption. As shown in Figure 3, during the Warmup and Progressive Pruning stages, the per-epoch time consumption of $\mathcal{G}^2\mathcal{R}^2$ is slightly higher than that of LoRA due to redundancy evaluation and module selection. However, upon entering the Fine-Tuning Refinement stage, the substantial reduction in trainable parameters leads to a significant drop in per-epoch time consumption. Given our experimental setting of $T_w + T_p = 5$ and $T_r = 15$, the total fine-tuning time for $\mathcal{G}^2\mathcal{R}^2$ is notably lower than that of LoRA (11.6% fine-tuning time saved), as reflected in Table 3.

Memory Usage. The bar plots in Figure 3 represent memory consumption per epoch. In the first two stages, although redundancy evaluation introduces additional computations, the memory overhead remains minimal since the majority of stored values are scalars. When entering the Fine-Tuning Refinement stage, the drastic reduction in trainable parameters results in a substantially lower memory footprint compared to LoRA. From Table 3, we observe that in this stage, $\mathcal{G}^2\mathcal{R}^2$ reduces memory usage by approximately 22.1% relative to LoRA.

Furthermore, in terms of validation accuracy per epoch, $\mathcal{G}^2\mathcal{R}^2$ consistently outperforms LoRA throughout training. Given that our method achieves better accuracy while requiring less total training time and memory, these results strongly validate the effectiveness of $\mathcal{G}^2\mathcal{R}^2$ in enhancing fine-tuning efficiency without sacrificing and even improving task performance.

	MRPC				RTE			
p_f	0.1	0.2	0.4	0.7	0.1	0.2	0.4	0.7
Random	68.38	68.38	68.38	68.38	62.09	63.17	60.06	60.29
$R(L) = R(A) \times R(B)$	90.44	91.18	91.19	89.46	87.73	88.09	87.00	85.92
$\mathcal{G}^2\mathcal{R}^2$	90.93	91.42	92.15	92.60	88.45	88.45	88.08	89.17

Table 4: Comparison of different importance metrics for $\mathcal{G}^2\mathcal{R}^2$.

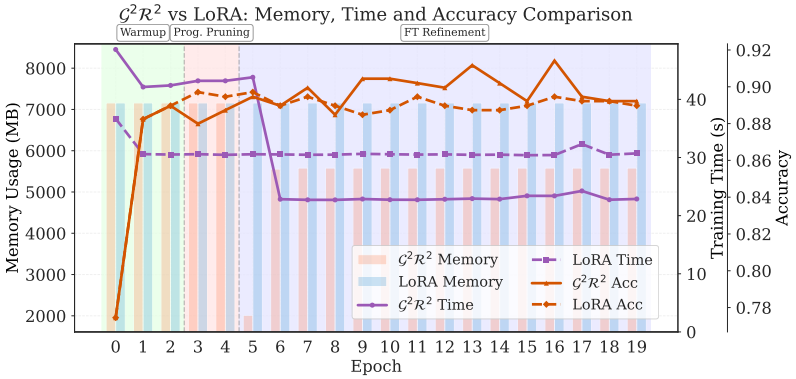

Figure 3: Memory usage, fine-tuning time per epoch, and accuracy comparison between $\mathcal{G}^2\mathcal{R}^2$ and LoRA on the MRPC dataset using DeBERTa. The background is segmented into three stages of $\mathcal{G}^2\mathcal{R}^2$: Warmup, Progressive Pruning, and Fine-Tuning Refinement.

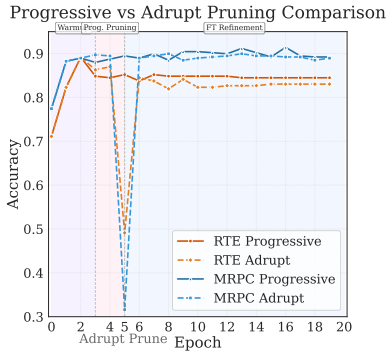

Figure 4: Comparison of progressive pruning and adrupt pruning on the MRPC and RTE datasets.

4.4 Ablation Study

Effectiveness of Gradient-Based Redundancy Evaluation. To validate the effectiveness of our proposed Gradient-Based Redundancy Evaluation, we compare two alternative redundancy calculation methods in Table 4: (1) $R(L) = Random$, where redundancy scores are randomly assigned; (2) $R(L) = R(A) \times R(B)$, where redundancy is computed as the product of the individual redundancy scores of A and B. Results on the MRPC and RTE datasets show that our method achieves the highest accuracy and exhibits the most stable performance across different final thresholds p_f. The random redundancy assignment completely fails on MRPC, while the alternative method results in a noticeable performance drop, highlighting the necessity of our redundancy evaluation strategy.

Effectiveness of Three-Stage Redundancy Reduction Strategy. To validate the effectiveness of our three-stage pruning method, we compare it with an adrupt pruning strategy on the MRPC and RTE datasets useing DeBERTaV3-base model. In our $\mathcal{G}^2\mathcal{R}^2$, redundancy reduction is completed at $epoch5$, so for a fair comparison, we apply adrupt pruning to the baseline method at the same epoch. The experimental results are presented in Figure 4. As shown in Figure 4, our progressive pruning strategy ensures a smooth training process without causing sudden drops in performance. In contrast, adrupt pruning results in a sharp performance degradation (catastrophic failure) at the pruning step. Although the baseline method gradually recovers in later epochs, its final performance remains consistently lower than that achieved by $\mathcal{G}^2\mathcal{R}^2$. This demonstrates that our pruning strategy effectively mitigates the instability introduced by parameter reduction, leading to superior overall fine-tuning stability and final performance.

5 Conclusion

We propose $\mathcal{G}^2\mathcal{R}^2$, a novel approach designed to mitigate redundancy in LoRA-based PEFT. By evaluating module-level redundancy with a Gradient-Based Redundancy Evaluation score and pruning less critical modules through a Three-Stage Redundancy Reduction Strategy, $\mathcal{G}^2\mathcal{R}^2$ effectively enhancing the efficiency of PEFT while preserving or improving performance. $\mathcal{G}^2\mathcal{R}^2$ accelerate convergence by running fewer fine-tuning steps, achieving competitive performance and further improving fine-tuning efficiency. Our extensive experiments on commonsense reasoning and natural language understanding tasks underscore the method's efficiency and robustness, outperforming existing baselines.

6 Limitations

While $\mathcal{G}^2\mathcal{R}^2$ improves fine-tuning efficiency, it introduces additional computations in early training stages. It is specifically designed for LoRA and may require adaptation for other PEFT methods. Moreover, its effectiveness depends on the accuracy of redundancy estimation, which could be further optimized. Future work will focus on enhancing efficiency and extending applicability.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7319–7328. Association for Computational Linguistics.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*, pages 1–14. Association for Computational Linguistics.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *CoRR*, abs/2309.12307.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4133–4145. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Demi Guo, Alexander M. Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4884–4896. Association for Computational Linguistics.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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 *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5254–5276. Association for Computational Linguistics.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *CoRR*, abs/2310.11454.

Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. 2021. Layer-adaptive sparsity for the magnitude-based pruning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, pages 3045–3059. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Guiying Li, Chao Qian, Chunhui Jiang, Xiaofen Lu, and Ke Tang. 2018. Optimization based layer-wise magnitude-based pruning for DNN compression. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 2383–2389. ijcai.org.

Chen Liang, Simiao Zuo, Minshuo Chen, Haoming Jiang, Xiaodong Liu, Pengcheng He, Tuo Zhao, and Weizhu Chen. 2021. Super tickets in pre-trained language models: From model compression to improving generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 6524–6538. Association for Computational Linguistics.

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 *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. 2024b. Parameter-efficient orthogonal finetuning via butterfly factorization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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

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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 11264–11272. Computer Vision Foundation / IEEE.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1797–1807. Association for Computational Linguistics.

Michela Paganini and Jessica Zosa Forde. 2020. On iterative neural network pruning, reinitialization, and the similarity of masks. *CoRR*, abs/2001.05050.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021a. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 487–503. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021b. Adapterfusion: Non-destructive task composition for transfer learning. *Preprint*, arXiv:2005.00247.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018a. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics,*

ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers, pages 784–789. Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018b. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

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, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 3266–3279. Association for Computational Linguistics.

Danilo Vucetic, Mohammadreza Tayaranian, Maryam Ziaeefard, James J. Clark, Brett H. Meyer, and Warren J. Gross. 2022. Efficient fine-tuning of BERT models on the edge. In *IEEE International Symposium on Circuits and Systems, ISCAS 2022, Austin, TX, USA, May 27 - June 1, 2022*, pages 1838–1842. IEEE.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Trans. Assoc. Comput. Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Adam X. Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. 2023. Bayesian low-rank adaptation for large language models. *CoRR*, abs/2308.13111.

Shen Yuan, Haotian Liu, and Hongteng Xu. 2024. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021a. Prune once for all: Sparse pre-trained language models. *CoRR*, abs/2111.05754.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021b. Prune once for all: Sparse pre-trained language models. *CoRR*, abs/2111.05754.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1–9. Association for Computational Linguistics.

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

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022. PLATON: pruning large transformer models with upper confidence bound of weight importance. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 26809–26823. PMLR.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *CoRR*, abs/2403.09113.

Michael Zhu and Suyog Gupta. 2018. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

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

A Appendix

A.1 Sparsity Step Schedule

Sparsity Step Scheduling dynamically evolves the value of threshold $p^{(t)}$ based on the relationship between the current time t and the total time T. The exact formula is as follows:

$$p^{(t)} = \begin{cases} p_0 & 0 \le t < t_i, \\ p_f + (p_0 - p_f)(1 - \frac{t - t_i - t_f}{T - t_i - t_f})^3 & t_i \le t < T - t_f, \\ p_f & \text{o.w.} \end{cases} \tag{12}$$

Where t_i and t_f are our hyperparameters to control each fine-tuning eopchs of three stages.

A.2 GLUE Benchmark

GLUE benchmark (Wang et al., 2019) is a wide-ranging collection of natural language understanding tasks. It includes MNLI (Williams et al., 2018) (inference), SST-2 (Socher et al., 2013) (sentiment analysis), MRPC (Dolan and Brockett, 2005) (paraphrase detection), CoLA (Warstadt et al., 2019) (linguistic acceptability), QNLI (Rajpurkar et al., 2018a) (inference), QQP(question-answering), RTE (inference), and STS-B (Cer et al., 2017) (textual similarity). Please refer to Table 5 for details.

A.3 Question Answering

Model and Datasets. We evaluate $\mathcal{G}^2\mathcal{R}^2$ on two question answering benchmarks: SQuADv1.1 (Rajpurkar et al., 2016) and SQuADv2.0 (Rajpurkar et al., 2018b), using DeBERTaV3-base (He et al., 2021) as the base model. Both tasks are formulated as sequence labeling problems, where the model predicts the start and end positions of the answer span.

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. SQuAD2.0 combines the 100,000 questions in SQuAD1.1 with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. The statistics of question answering datasets are summarized in Table 7.

Main Results. Table 6 presents the results of fine-tuning DeBERTaV3-base with $\mathcal{G}^2\mathcal{R}^2$ at different trainable parameter scales. Our method consistently outperforms existing approaches in both Exact Match (EM) and F1 scores across most parameter scales on both datasets. Notably, $\mathcal{G}^2\mathcal{R}^2$ maintains strong performance even with fewer trainable parameters, often matching or surpassing baselines with larger parameter budgets. These results demonstrate the effectiveness of our redundancy-aware module selection, which optimally allocates trainable parameters across different tasks, maximizing fine-tuning efficiency.

A.4 Natural Language Generation

Model and Datasets. Having demonstrated that $\mathcal{G}^2\mathcal{R}^2$ achieves state-of-the-art performance on NLU and QA tasks, we further investigate its effectiveness on natural language generation (NLG). To this end, we conduct experiments on the XSum (Narayan et al., 2018) dataset, a benchmark for abstractive single-document summarization, using BART-large (Lewis et al., 2020) as the base model.

The Extreme Summarization (XSum) dataset is a dataset for evaluation of abstractive single-document summarization systems. The goal is to create a short, one-sentence new summary answering the question "What is the article about?". The dataset consists of 226,711 news articles accompanied with a one-sentence summary. The articles are collected from BBC articles (2010 to 2017) and cover a wide variety of domains (e.g., News, Politics, Sports, Weather, Business, Technology, Science, Health, Family, Education, Entertainment and Arts). The official random split contains 204,045 (90%), 11,332 (5%) and 11,334 (5%) documents in training, validation and test sets, respectively.

Main Results. Table 8 presents the experimental results across different trainable parameter scales. $\mathcal{G}^2\mathcal{R}^2$ consistently outperforms all baseline methods, with its advantage becoming more pronounced as the trainable parameter budget decreases. For instance, at a trainable parameter scale of 0.13%, our

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST	Sentiment	67k	872	1.8k	2	Accuracy
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr

Table 5: Summary of the GLUE benchmark.

	SQuADv1.1				SQuADv2.0			
Full FT	86.0 / 92.7				85.4 / 88.4			
# Params	0.08%	0.16%	0.32%	0.65%	0.08%	0.16%	0.32%	0.65%
HAdapter	84.4/91.5	85.3/92.1	86.1/92.7	86.7/92.9	83.4/86.6	84.3/87.3	84.9/87.9	85.4/88.3
PAdapter	84.4/91.7	85.9/92.5	86.2/92.8	86.6/93.0	84.2/87.2	84.5/87.6	84.9/87.8	84.5/87.5
LoRA	86.4/92.8	86.6/92.9	86.7/93.1	86.7/93.1	84.7/87.5	83.6/86.7	84.5/87.4	85.0/88.0
AdaLoRA	87.2/93.4	87.5/93.6	87.5/93.7	87.6/93.7	85.6/88.7	85.7/88.8	**85.5/88.6**	86.0/88.9
$\mathcal{G}^2\mathcal{R}^2$	**88.0/93.7**	**88.4/94**	**87.9/93.7**	**88.0/93.7**	**85.8/88.8**	**85.8/88.8**	85.4/88.5	**86.1/88.9**

Table 6: Results with DeBERTaV3-base on SQuAD v1.1 and SQuADv2.0. Here *# Params* is the number of trainable parameters relative to that in full fine-tuning. We report EM/F1. The best results in each setting are shown in **bold**.

	# Train	# Validation
SQuAD v1.1	87,599	10,570
SQuAD v2.0	130,319	11,873

Table 7: Statistics of the SQuAD dataset.

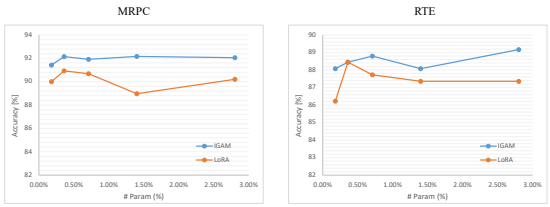

Figure 5: Stability performance under different number of learnable parameters.

Method	# Params	Rouge-1	Rouge-2	Rouge-L
Full FT	**100%**	**45.49**	**22.33**	**37.26**
LoRA	**2.20%**	43.95	20.72	35.68
AdaLoRA		44.72	21.46	36.46
$\mathcal{G}^2\mathcal{R}^2$		**44.74**	**21.51**	**36.57**
LoRA	**1.10%**	43.40	20.20	35.20
AdaLoRA		44.35	21.13	36.13
$\mathcal{G}^2\mathcal{R}^2$		**44.43**	**21.20**	**36.26**
LoRA	**0.26%**	43.18	19.89	34.92
AdaLoRA		43.55	20.17	35.20
$\mathcal{G}^2\mathcal{R}^2$		**44.23**	**21.12**	**36.17**
LoRA	**0.13%**	42.81	19.68	34.73
AdaLoRA		43.29	19.95	35.04
$\mathcal{G}^2\mathcal{R}^2$		**44.35**	**21.07**	**36.17**

Table 8: Results with BART-large on XSum. Here *# Params* is the number of trainable parameters relative to that in full fine-tuning. We report R-1/2/L. The best results are shown in **bold**.

method improves ROUGE-1/2/L scores by 1.06, 1.12, and 1.13 points, respectively, over AdaLoRA. Moreover, $\mathcal{G}^2\mathcal{R}^2$ exhibits greater stability across different parameter scales. We attribute this to the fact that only a few insertion positions significantly impact fine-tuning performance for summarization tasks. Once LoRA modules are placed at these key positions, additional insertions or removals at other positions have minimal effect on overall performance.

A.5 Robustness of Different p_f

Figure 5 compares the performance of $\mathcal{G}^2\mathcal{R}^2$ and LoRA across different trainable parameter scales on the MRPC and RTE datasets. Even when the trainable parameter ratio varies significantly from 0.18% to 2.8%, our method maintains consistently superior and stable performance. In contrast,

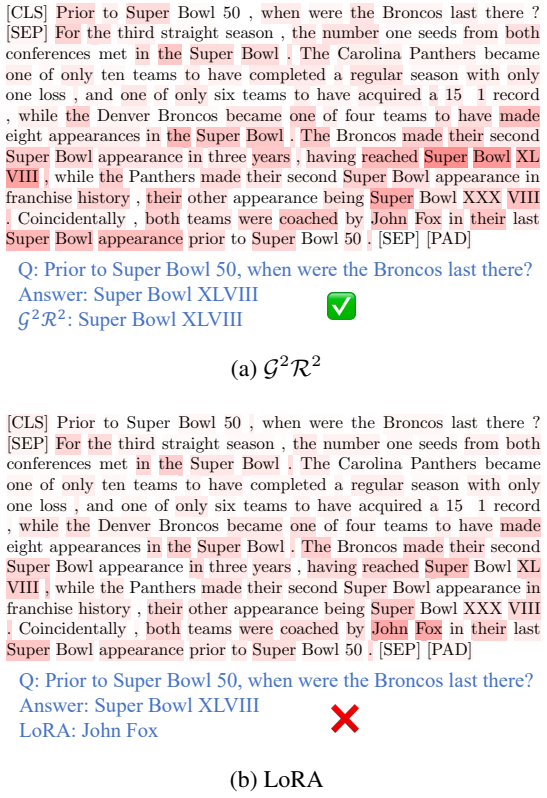

Figure 6: Visualization of some results. The shades of red indicate the degree of emphasis that the fine-tuned model places on different words.

LoRA exhibits higher variance, confirming that our redundancy-aware selection mechanism not only enhances fine-tuning efficiency but also ensures robust performance across different parameter budgets.

A.6 Visualization of Results.

We further analyze the prediction performance of $\mathcal{G}^2\mathcal{R}^2$ compared to LoRA on the SQuADv1 dataset. Out of 10,756 test samples, our method correctly predicts 496 more samples than LoRA. As illustrated in Figure 6, our importance-guided adaptation better captures semantic relevance in complex text environments. For instance, in the question "Prior to Super Bowl 50, when were the Broncos last there?", our method correctly identifies "Super Bowl XLVIII" as the answer, whereas LoRA incorrectly assigns the highest attention to "John Fox," leading to an incorrect prediction. This demonstrates that $\mathcal{G}^2\mathcal{R}^2$ effectively selects insertion positions that enhance semantic representation, improving the model's ability to capture critical information for downstream tasks.

A.7 Experiments Hyperparameters

A.7.1 Hyperparameters for Commonsense Reasoning Tasks

We have detailed the hyperparameters required for fine-tuning LLaMA2-7B and LLaMA3-8B using $\mathcal{G}^2\mathcal{R}^2$ on the commonsense reasoning tasks in Table 10.

Hyperparameters	LLaMA2-7B		LLaMA3-8B	
Rank r	16	32	16	32
α	32	64	32	64
Dropout	0.05			
Optimizer	AdamW			
LR	2e-4	2e-4	1e-4	1e-4
LR Scheduler	Linear			
Batch size	16			
Micro batch size	4			
Warmup Steps T_w	4000			
Prog. Pruning Steps T_p	4000			
Refinement Steps T_r	34570			
β_1	0.85			
β_2	0.95			
Epochs	1			
Total Steps	34570			
Where	Q,K,V,Up,Down			

Table 9: Hyperparameter configurations of $\mathcal{G}^2\mathcal{R}^2$ for LLaMA2-7B and LLaMA3-8B on the commonsense reasoning tasks.

A.7.2 Hyperparameters for NLU Tasks

We have detailed the hyperparameters required for fine-tuning DeBERTa-V3 (He et al., 2021) using $\mathcal{G}^2\mathcal{R}^2$ on the GLUE benchmark in Table 10. For the GLUE benchmark, $p_f = 0.2$ corresponds to 0.15% trainable parameters, and $p_f = 0.7$ corresponds to 0.51% trainable parameters.

A.8 Hyperparameter for QA and NLG Tasks

We have detailed the hyperparameters required for fine-tuning DeBERTa-V3 (He et al., 2021) using $\mathcal{G}^2\mathcal{R}^2$ on the QA dataset in Table 11, as well as the hyperparameters required for fine-tuning BART-large (Lewis et al., 2020) using $\mathcal{G}^2\mathcal{R}^2$ on the NLG dataset.

Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Optimizer	AdamW							
Warmup Ratio	0.06							
LR Schedule	Linear							
Batch Size	8	8	16	16	16	16	16	16
# Epochs	10	30	30	40	10	10	40	40
Learning Rate	1E-04	1.5E-04	2E-04	3E-04	2E-04	1.2E-04	4E-04	2E-04
LoRA Module Dim.	8							
Max Seq. Len.	512							
Initial Threshold	1.0							
Warmup Epochs T_w	1	3	3	4	1	1	4	4
Prog.Pruning Epochs T_p	1	3	3	4	1	1	4	4
Refinement Epochs T_r	8	24	24	32	8	8	32	32
β_1	0.85							
β_2	0.95							

Table 10: The hyperparameters we used for fine-tuning DeBERTa-V3 with $\mathcal{G}^2\mathcal{R}^2$ on the GLUE benchmark.

Dataset	SQuADv1.1	SQuADv2.0	XSum
Optimizer	AdamW		
Warmup Ratio	0.06		
LR Schedule	Linear		
Batch Size	16	16	8
# Epochs	15	15	20
Learning Rate	1.5E-04	1.5E-04	8E-5
LoRA Module Dim.	8		
Max Seq. Len.	384		
Initial Threshold	1.0		
Warmup Epochs T_w	2	2	3
Prog.Pruning Epochs T_p	2	2	3
Refinement Epochs T_r	11	11	14
β_1	0.85		
β_2	0.95		

Table 11: The hyperparameters we used for fine-tuning DeBERTa-V3 with $\mathcal{G}^2\mathcal{R}^2$ on the QA tasks and fine-tuning BART-large with $\mathcal{G}^2\mathcal{R}^2$ on the NLG task.