DPLoRA: A Dual-Pruning Framework based on ILP Optimization and Progressive Pruning for Parameter-Efficient LoRA Fine-Tuning

Anonymous authors
Paper under double-blind review

Abstract

We propose DPLoRA (Dual-Pruning Low-Rank Adaptation), an optimized Low-Rank Adaptation (LoRA) method for parameter-efficient fine-tuning of large language models. Our approach introduces a two-stage compression framework: (1) an initial pruning stage, OPLoRA, that formulates a first ILP problem to automatically discover the optimal layer-wise LoRA rank (r) configuration before training; (2) a progressive pruning stage that formulates a second ILP problem during training, incorporating Exponential Moving Average (EMA) of layer-wise importance scores to further reduce rank (r) adaptively. On the GLUE benchmark, our first stage, OPLoRA, achieves a new state-of-the-art (SOTA) performance, surpassing all baselines. Furthermore, the full DPLoRA framework also demonstrates superior capabilities, outperforming strong PEFTs like AdaLoRA and SoRA while achieving up to an 80% reduction in trainable parameters and a 50% reduction in training time. This study offers a new direction for efficiently deploying large-scale language models in resource-constrained environments.

1 Introduction

Recent advances in large language models (LLMs) such as BERT(Devlin et al. (2019)), RoBERTa(Liu et al. (2019)), and GPT-4(Achiam et al. (2023)) have significantly enhanced the performance of natural language processing (NLP) systems. However, the immense scale of these models presents substantial challenges for task-specific fine-tuning, demanding considerable computational and memory resources that limit their practical deployment.

Parameter-Efficient Fine-Tuning (PEFT) methods have aimed to mitigate this burden. Among them, Low-Rank Adaptation (LoRA)(Hu et al. (2022)) has emerged as a prominent technique, which freezes the pre-trained model weights and injects a small number of trainable low-rank matrices. Despite its effectiveness, vanilla LoRA applies a uniform, fixed rank (r) across all layers. This one-size-fits-all approach fails to account for the heterogeneous roles and varying importance of different layers within the model architecture, leading to suboptimal parameter allocation.

To address this limitation, we propose DPLoRA (Dual-Pruning Low-Rank Adaptation), a two-stage pruning framework that introduces a structured, adaptive approach to rank allocation. Our methodology is designed to systematically identify and prune redundant ranks both before and during training. First, we perform initial rank pruning by formulating an Integer Linear Programming (ILP) problem that assigns non-zero ranks only to layers deemed important based on gradient-based metrics. This stage discovers a tailored rank configuration from the outset, achieving substantial parameter compression before training begins. Second, we apply progressive pruning, which further refines the rank allocation during training. This stage is guided by a separate ILP formulation that incorporates dynamically estimated importance scores, smoothed with Exponential Moving Averages (EMA), to adapt to the changing layer dynamics. Our use of ILP for structural configuration is inspired by methods like LQ-LoRA(Guo et al. (2024)), which applied it for quantization, whereas our work pioneers its use for guiding layer-wise rank decisions under global budget constraints.

Extensive experiments on the GLUE benchmark (using RoBERTa-base) and the Alpaca instruction-following task (using LLaMA3-8B) validate our two-stage framework. Our first stage, OPLoRA, achieves new state-of-the-art (SOTA) performance, surpassing other leading PEFTs including AdaLoRA and SoRA. Building on this, our second stage, DPLoRA, introduces progressive pruning to achieve extreme efficiency; it not only outperforms strong baselines such as SoRA and AdaLoRA but does so while reducing trainable parameters by up to 80% and accelerating training by up to 50%. These results, consistent across both encoder-only and decoder-only model architectures, underscore our framework's potential for the practical and efficient deployment of LLMs in real-world, resource-limited environments such as mobile and edge devices.

Our contributions can be summarized as follows:

- We propose DPLoRA, a novel dual-stage pruning framework that systematically allocates LoRA ranks by solving a sequence of Integer Linear Programming problems.
- We design an initial pruning stage based on gradient importance and a progressive pruning stage that dynamically adapts ranks using EMA-smoothed importance scores under a parameter budget.
- We demonstrate that DPLoRA substantially reduces parameter and computational costs while achieving state-of-the-art performance across diverse models and tasks.

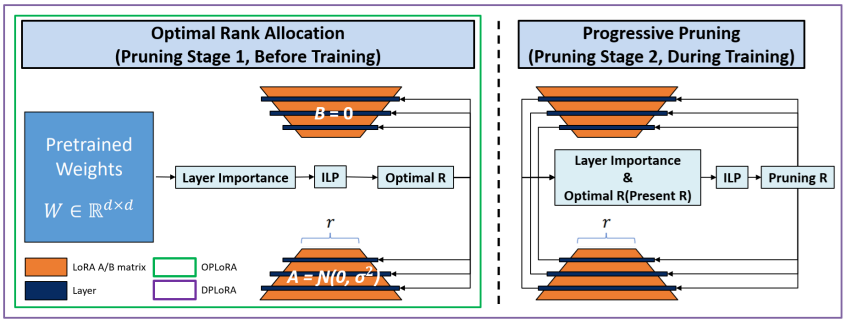

Figure 1: Overview of our proposed two-stage pruning based compression framework. Left: Pruning Stage 1 - Optimal Rank Allocation before training. Right: Pruning Stage 2 - Progressive Pruning during training.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning (PEFT)

Parameter-Efficient Fine-Tuning (PEFT) methods have become essential for adapting large language models (LLMs) with minimal computational cost. Early approaches such as Adapter Tuning(Houlsby et al. (2019)), which inserts small bottleneck layers, and Prefix-Tuning(Li and Liang (2021)) or Prompt Tuning(Lester et al. (2021)), which tune continuous prompt embeddings, all freeze the pretrained weights while introducing a small set of trainable parameters.

2.2 Refinements of the LoRA Update Mechanism

Among various PEFT methods, Low-Rank Adaptation (LoRA)(Hu et al. (2022)) is particularly prominent. It injects trainable low-rank matrices ($\Delta W = AB$) into the model, achieving performance comparable to full fine-tuning with a fraction of the parameters and no inference latency.

Building on this foundation, one line of research focuses on improving the LoRA framework by modifying the composition or training strategy of the core update matrices A and B. For instance, DoRA(Liu et al. (2024)) disentangles the LoRA update into magnitude and

direction components, stabilizing training and improving performance without additional inference cost. VeRA(Kopiczko et al. (2023)) proposes sharing a single pair of frozen, random low-rank matrices across layers and learning small scaling vectors, drastically reducing trainable parameters. In a different approach targeting memory efficiency, LoRA-FA(Zhang et al. (2023a)) freezes the randomly initialized matrix A and only trains matrix B, which significantly reduces optimizer state and activation memory requirements.

While these methods enhance the efficiency or effectiveness of the update mechanism itself, they still operate under a manually specified, fixed-rank assumption, leaving the fundamental question of optimal rank allocation unanswered.

2.3 Automated Rank Allocation via Importance-Based Strategies

A second line of research directly tackles the challenge of automating rank selection. These methods typically rely on importance-based proxies to iteratively determine or adjust layer-wise ranks.

Several approaches focus on finding a single, efficient static architecture through iterative pruning. These methods typically start with a larger rank and progressively remove less important components. For example, AdaLoRA(Zhang et al. (2023b)) dynamically allocates ranks by parameterizing weight updates in a pseudo-SVD form and pruning singular values based on gradient-based importance scores. It assigns parameter budgets to important components across layers, enabling adaptive rank selection without relying on explicit SVD or heuristic rules. SoRA(Ding et al. (2023)) dynamically prunes low-rank dimensions during training using learnable gates and proximal gradient updates, enabling adaptive rank selection without global architecture changes. LoRAPrune(Zhang et al. (2023c)) also progressively removes less important LoRA modules using importance scores estimated via first-order Taylor approximation.

In contrast to these pruning-based methods, another line of work explores more dynamic or constructive strategies. DyLoRA(Valipour et al. (2022)) enables dynamic inference across a range of ranks by training a single adapter using truncated updates. Rather than relying on rank selection or allocation, it samples ranks during training and uses a frozen-update strategy to ensure information is progressively ordered across ranks. IncreLoRA(Zhang et al. (2023d)) takes an alternative approach by incrementally increasing ranks throughout training, which can mitigate the risk of prematurely pruning important components. ElaLoRA(Chang et al. (2025)) dynamically prunes and expands ranks during fine-tuning based on gradient-derived importance scores, enabling adaptive rank allocation across layers for improved efficiency and performance. Finally, AutoLoRA(Zhang et al. (2024)) introduces a meta-learning based framework that assigns selection variables to rank-1 components in LoRA updates and optimizes them to automatically derive layer-wise optimal ranks, thereby eliminating exhaustive manual tuning.

Although these methods successfully automate rank selection, their reliance on iterative adjustments, proxy metrics, or costly search processes may not lead to a globally optimal solution. In stark contrast, our work, DPLoRA, formulates the rank allocation challenge as a principled optimization problem. By leveraging Integer Linear Programming (ILP), our approach enables a globally coordinated and structured optimization of ranks across all layers from the outset of training.

3 Methods

3.1 OPLoRA: Optimal Rank Allocation (Pruning Stage 1)

Layer Importance Estimation: The importance of each layer is measured by the absolute value of its parameter gradients. This process quantifies how sensitive the model's loss is to changes in each layer's parameters, providing a direct measure of its contribution to the learning task. Mathematically, the importance I_l of layer l is computed using Equation (1), where gradients (∇_{θ_l}) are calculated with respect to the loss function $\mathcal{L}$. To obtain a stable estimate, we calculate the absolute mean of the gradients over a small subset of training

data ($\mathcal{D}$, up to 500 samples) across a few initial batches (N, default: 5). This gradient-based score provides a reliable measure of each layer's importance and forms the foundation for our resource allocation.

$$I_l^{\text{init}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{x \sim \mathcal{D}} \left[\| \nabla_{\theta_l} \mathcal{L}(x) \| \right] \tag{1}$$

Performance Gain Estimation: The expected performance gain $G_{l,r}$ when assigning rank r to layer l is modeled using Equation (2), where I_l is the importance of layer l and $M_l = \min(d_{\text{in}}^l, d_{\text{out}}^l)$ represents the maximum possible rank for layer l. The equation's exponential component explicitly captures the diminishing returns effect — as the rank r increases, the additional performance gain gradually saturates. This formulation reflects the intuition that assigning higher rank values to more important layers yields greater performance gains, while ensuring the benefit of increasing rank decreases as the value grows.

$$G_{l,r} = I_l \cdot \left(1 - e^{-1 \cdot \frac{r}{M_l}}\right) \tag{2}$$

Objective Function: We formulate an Integer Linear Programming (ILP) problem to determine the best rank assignment per layer under a parameter budget. The decision variables are defined as $x_{l,r} \in \{0, 1\}$, where $x_{l,r} = 1$ means rank r is assigned to layer l. The objective function is given by Equation (3):

$$\max \sum_{l \in L} \sum_{r \in R} G_{l,r} \cdot x_{l,r} \tag{3}$$

Computational Cost Modeling: The computational cost $C_{l,r}$ of assigning rank r to layer l is defined in Equation (4), where d_{in}^l and d_{out}^l represent the input and output dimensions of layer l, respectively. This cost function accounts for both the forward computation cost and parameter storage of LoRA: each layer uses two low-rank matrices, $A \in \mathbb{R}^{d_{in} \times r}$ and $B \in \mathbb{R}^{r \times d_{out}}$, leading to $r \cdot (d_{in} + d_{out})$ parameters and similar computational operations. Our implementation considers these dual aspects, effectively scaling the basic parameter count. This cost is incorporated as a constraint in our ILP formulation, ensuring that the total computational resource allocation across all layers remains within the specified global budget.

$$C_{l,r} = 2 \cdot r \cdot (d_{\text{in}}^l + d_{\text{out}}^l) \tag{4}$$

3.2 Progressive Pruning (Pruning Stage 2)

Enhanced Layer Importance Estimation: In the progressive pruning stage, we employ a more refined method of importance estimation. To track changes in layer importance throughout training, we apply exponential moving average (EMA) as shown in Equation (5), where $I_l^{(t)}$ represents the importance of layer l at time t, β is the EMA decay factor(default: 0.9), and I_l^{new} is the newly computed importance. Importance is estimated using up to 5 batches from the dataloader. Gradients of both $lora\_A$ and $lora\_B$ are computed and weighted by the square root of current rank according to Equation (6). This weighting approach ensures that layers with higher ranks, which typically process more information, are properly represented in the importance calculation.

$$I_l^{(t)} = \beta \cdot I_l^{(t-1)} + (1 - \beta) \cdot I_l^{\text{update}} \tag{5}$$

$$I_l^{\text{update}} = \sqrt{\|\nabla_{A_l}\| \cdot \|\nabla_{B_l}\|} \cdot \sqrt{r_l} \tag{6}$$

Performance Loss Estimation: The expected loss when changing the rank of layer l from r_{current} to r_{new} is defined in Equation (7).

$$L_{l,r_{\text{new}}} = I_l \cdot \frac{r_{\text{current}} - r_{\text{new}}}{r_{\text{current}}} \quad \text{where } r_{\text{current}} \geq r_{\text{new}} \text{ (constraints)} \tag{7}$$

Algorithm 1 Rank Trimming Mechanism

Require: LoRA matrices $A \in \mathbb{R}^{r_{\text{current}} \times d}$, $B \in \mathbb{R}^{k \times r_{\text{current}}}$, target rank $r_{\text{new}} < r_{\text{current}}$
Ensure: Trimmed matrices $A_{\text{new}} \in \mathbb{R}^{r_{\text{new}} \times d}$, $B_{\text{new}} \in \mathbb{R}^{k \times r_{\text{new}}}$
Step 1: Compute importance for each dimension:
$I_d^A = \|A_d\|_2$ for each row d
$I_d^B = \|B_d\|_2$ for each column d
Step 2: Combine importance:
$I_d = I_d^A \cdot I_d^B$ for each dimension d
Step 3: Select top-r_{new} dimensions:
TopIndices $= \arg\max_{|S|=r_{\text{new}}} \sum_{d \in S} I_d$
Step 4: Trim matrices:
$A_{\text{new}} = A[\text{TopIndices}, :]$ {Select rows by top indices}
$B_{\text{new}} = B[:, \text{TopIndices}]$ {Select columns by top indices}
return A_{new}, B_{new}

Objective Function: The decision variables for progressive pruning are defined as $x_{l,r} \in \{0, 1\}$, where $x_{l,r} = 1$ if rank r is assigned to layer l. The objective function is given by Equation (8), where the momentum penalty term $P_{l,r}$ is defined in Equation (9). In this equation, γ represents the momentum penalty weight (default: 0.1), δ is the stability reward weight (default: 0.05), and $\mathbf{1}_{r=r^l_{\text{prev}}}$ is the indicator function that equals 1 if $r = r^l_{\text{prev}}$, and 0 otherwise.

$$\min \sum_{l \in L} \sum_{r \in R} (L_{l,r} + P_{l,r}) \cdot x_{l,r} \tag{8}$$

$$P_{l,r} = \gamma \cdot I_l \cdot |r - r^l_{prev}| - \delta \cdot I_l \cdot \mathbf{1}_{r=r^l_{prev}} \tag{9}$$

Common Constraints of Rank Allocation and Progressive Pruning: The ILP formulation incorporates several constraints to ensure practical and effective rank allocation. First, we enforce that each layer must be assigned exactly one rank value. The total cost must not exceed a given budget. Finally, to prevent assigning a rank of 0 to an entire layer type, we require that the average rank for every layer type must be greater than zero.

Bezier-based Pruning Scheduler: To ensure a smooth and controlled parameter reduction throughout the progressive pruning process, we employ a Bézier curve-based scheduling mechanism. Let N denote the total number of pruning steps. At each step t, the parameter budget is determined by Equation (10), where the reduction ratio $R(t)$ is modeled using a Bézier curve as shown in Equation (11). By setting the control points to P=[0.0,0.2,0.8,1.0], we generate an S-curve that prunes slowly in the initial and final stages, but more aggressively in the middle phases, thereby facilitating stable convergence. The timing of each pruning operation is determined by the trigger point function in Equation (12), which ensures proper spacing between pruning steps to allow for adequate model recovery and adaptation.

$$B_{\text{step}}(t) = B_{\text{initial}} \cdot (1 - R(t)) \tag{10}$$

$$R(t) = R_{\text{target}} \cdot \sum_{i=0}^{n} \binom{n}{i} P_i \cdot \left(\frac{t}{N}\right)^i \cdot \left(1 - \frac{t}{N}\right)^{n-i} \tag{11}$$

$$T_{\text{trigger}}(t) = T_{\text{delay}} + t \cdot \frac{T_{\text{total}} - T_{\text{delay}}}{N + 1} \tag{12}$$

4 Experiments

This section presents comprehensive experiments to evaluate the effectiveness of our proposed dual-pruning compression framework. We assess performance on a variety of tasks using the

GLUE benchmark for natural language understanding and Alpaca for instruction following, comparing our approach with strong baselines.

4.1 Experimental Setup

Datasets and Evaluation Metrics: We evaluate our method on the GLUE benchmark(Wang et al. (2018)), which includes tasks such as sentiment analysis, paraphrase detection, and natural language inference, with performance measured by accuracy and F1 scores. Additionally, we assess instruction-following capabilities using the Alpaca dataset(Taori et al. (2023)), where performance is evaluated based on the model's ability to generate relevant and coherent responses to diverse instructions. All experiments follow standard evaluation protocols.

Base Model and Baselines: We use RoBERTa-base (125M parameters) as our backbone model for the GLUE benchmark and Llama 3 8B for the instruction-following task on Alpaca. Our approach is compared against strong baselines specific to each task: for GLUE, these include BitFit, Adapter, Vanilla LoRA, SoRA and AdaLoRA; for Alpaca, we use Vanilla LoRA as the baseline. Our method is evaluated in two configurations: OPLoRA (Optimal Pruning LoRA, applying only stage 1) and DPLoRA (Dual-Pruning LoRA, applying both stages).

Hyperparameters and Training Setups: The results for all tasks are based on experiments repeated with three different random seeds: 42, 777, and 2025, to ensure the robustness of our findings. The detailed hyperparameters for each task were carefully tuned, and a comprehensive list of these settings is provided in the Appendix. Throughout all experiments, * indicates that a value was adopted directly from the original paper of the corresponding baseline, while † indicates runs with key hyperparameters matched for fair comparison.

4.2 Main Results

Task-wise Performance Comparison on GLUE: Table 1 presents a comprehensive comparison of performance and parameter efficiency on the GLUE benchmark. Among the baselines, Full Fine-tuning (FT) requires training all 125.0M parameters to achieve an average score of 85.1. In contrast, parameter-efficient baselines offer a strong trade-off, with Adapter* achieving the highest baseline score of 85.4 using only 0.9M parameters, and BitFit* delivering a competitive 85.2 with just 0.1M parameters.

Our proposed OPLoRA framework sets a new state-of-the-art on the GLUE benchmark by outperforming all baseline methods with an average score of 86.0. This improvement is achieved with only 1.2M trainable parameters, demonstrating superior performance to LoRA(r=16) (85.2 avg) while using less than half the number of parameters (2.7M). This result strongly validates our importance-based rank allocation strategy.

Furthermore, the DPLoRA variants showcase an exceptional performance-efficiency trade-off. For instance, DPLoRA(p=0.6) slightly outperforms LoRA(r=16) (85.3 vs 85.2) while using only 0.5M parameters—a reduction of over 80%. Even with more aggressive pruning, DPLoRA(p=0.8) maintains a strong average of 84.9 with a mere 0.2M parameters. These results confirm that our progressive pruning approach can drastically reduce the parameter budget while maintaining performance comparable to strong baselines, making it ideal for resource-constrained environments.

Computation Efficiency: Table 2 demonstrates the computational advantages of our proposed methods. Compared to the LoRA (r=16) baseline, OPLoRA reduces training time by 31%, and DPLoRA (p=0.6) achieves an even greater 43% time reduction. Most remarkably, DPLoRA (p=0.8) achieves a 50% reduction in training time, demonstrating that our approach can effectively halve the computational cost. These gains result from our progressive pruning mechanism that dynamically eliminates less important parameter dimensions during training, providing enhanced computational efficiency for resource-constrained environments.

Table 1: Performance and the number of trainable parameters (in millions) on the GLUE benchmark. We report accuracy for most tasks, Matthew's correlation for CoLA, and Pearson correlation for STS-B. Higher is better.

Method	#Trainable Params(M)	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Avg.
FT	125.0	60.7	87.5	88.6	92.7	**91.7**	75.8	93.8	90.3	85.1
LoRA(r=16)	2.7	60.0	**87.7**	88.6	92.6	91.2	77.6	94.1	89.9	85.2
LoRA(r=8)	1.3	58.3	87.6	88.2	92.6	90.8	76.4	94.2	89.9	84.7
SoRA	0.9	58.4	**87.7**	87.3	92.9	91.6	76.8	93.8	90.3	84.8
AdaLoRA	0.3	61.4	87.3	86.5	92.7	89.8	78.8	93.9	**91.0**	85.2
Adapter*	0.9	**62.6**	87.3	88.4	**93.0**	90.6	75.9	**94.7**	90.3	85.4
BitFit*	0.1	62.0	84.7	**92.7**	91.8	84.0	**81.5**	93.7	90.8	85.2
OPLoRA	1.2	**62.6**	87.4	88.8	92.8	91.0	79.7	**94.7**	90.8	**86.0**
DPLORA(p=0.6)	0.5	61.9	86.5	89.6	92.3	90.4	77.3	94.1	90.6	85.3
DPLORA(p=0.7)	0.4	60.5	86.4	89.9	92.1	90.0	76.2	93.5	90.4	84.9
DPLORA(p=0.8)	0.2	62.1	85.7	89.5	91.9	89.5	77.5	93.5	89.8	84.9

Table 2: Training runtime efficiency across GLUE tasks († Same hyperparameter matching). These experiments were conducted under a specific setting to isolate training speed; detailed hyperparameters are listed in Appendix B.

Method	Training Time (min)	Relative Time
LoRA (r=16)†	614	100%
LoRA (r=8)†	529	86%
OPLoRA (ours)	426	69%
DPLoRA (ours, p=0.6)	347	57%
DPLoRA (ours, p=0.7)	346	57%
DPLoRA (ours, p=0.8)	**307**	**50%**

4.3 Results on Instruction Following

Performance on MT-BENCH: To evaluate the model's instruction-following capabilities, we use the MT-BENCH benchmark, with scores judged by GPT-4. As shown in Table 8, our proposed framework demonstrates superior performance and efficiency.

The baseline LoRA (r=8) model achieves an MT-BENCH score of 5.56 using 21.0M trainable parameters. In contrast, our OPLoRA method not only improves performance to a state-of-the-art score of 5.65 but also does so with fewer parameters (20.0M), validating the effectiveness of our optimal rank allocation.

Furthermore, DPLoRA (p=0.8) showcases an excellent performance-efficiency trade-off. It achieves a highly competitive score of 5.54—nearly matching the baseline—while using only 4.0M parameters. This represents an 80% reduction in trainable parameters compared to OPLoRA with a minimal trade-off in performance. These results confirm that our framework offers both a path to superior performance (OPLoRA) and a method for extreme compression with graceful performance degradation (DPLoRA).

Table 3: Instruction-following performance on MT-BENCH, evaluated by GPT-4. Higher scores indicate better performance.

Method	# Trainable Params(M)	MT-BENCH
LoRA (r=8)	21.0	5.56
OPLoRA (ours)	20.0	**5.65**
DPLoRA (ours, p=0.8)	4.0	5.54

4.4 Analysis and Discussion

Dual-Pruning Mechanism Analysis: Our dual-pruning mechanism is visualized across two stages in Figure 2 and Figure 3. First, Figure 2 shows that our method assigns heterogeneous, non-uniform ranks across layers, confirming that a fixed-rank approach is suboptimal. Subsequently, Figure 3 demonstrates the effectiveness of progressive pruning: even as parameters are reduced by nearly 80% (blue line), performance peaks immediately and remains stable (red line). This illustrates that our two-stage approach achieves extreme parameter efficiency without degrading final task performance.

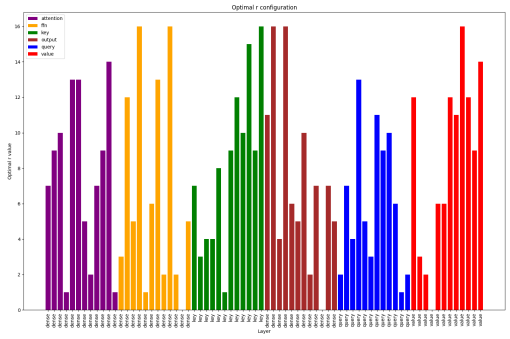

Figure 2: Initial heterogeneous rank allocation for RoBERTa-Base on SST-2.

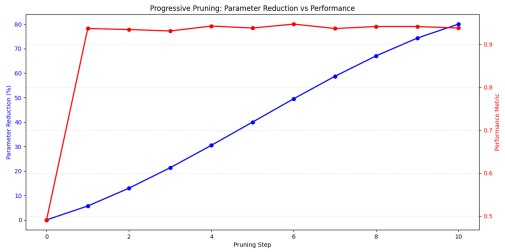

Figure 3: Progressive pruning on SST-2 (DPLoRA, p=0.8). Performance remains stable while parameter reduction reaches 80%.

ILP Computation Overhead: A potential concern with using Integer Linear Programming (ILP) is its computational overhead. Our analysis in Table 4, however, confirms that the overhead of our framework is negligible.

For a model like RoBERTa-Base, the total ILP computation time for the entire training process is less than half a second (0.46s). For the much larger Llama 3 8B, the total cumulative overhead is also remarkably low at approximately 7 seconds (6.65s). Considering that fine-tuning an 8B-parameter model typically takes several hours, this cost represents a tiny fraction of the total runtime. This demonstrates that our ILP-based optimization is highly efficient and does not introduce a practical bottleneck, even for large-scale models.

Table 4: ILP solver execution time (in seconds) for the SST-2 and Alpaca tasks. The results are from a single run with random seed 42.

Model	**Task**	**Initial Rank**	**Progressive Pruning Steps**										**Sum**
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
RoBERTa-Base	SST-2	0.02	0.04	0.03	0.06	0.06	0.03	0.10	0.04	0.02	0.01	0.05	**0.46**
Llama 3 8B	Alpaca	0.06	0.17	0.36	0.07	0.32	0.40	0.30	4.57	0.18	0.11	0.11	**6.65**

4.5 Ablation Study

Ablation Study on Pruning Schedulers: To analyze the impact of the pruning schedule, we conducted an ablation study comparing three different schedulers, with results shown in Table 5. The results indicate that the timing of the pruning is critical. Our default Bézier scheduler, which follows an S-curve (slow-fast-slow), achieved the best average performance of 84.9. This outperformed both a standard Linear scheduler (83.8) and a Early-Pruning scheduler (84.7), which applies its most aggressive pruning in the initial stages of training. These findings suggest that allowing the model a brief adaptation period before the most aggressive pruning phase is the most effective strategy.

Effect of EMA and Momentum Penalty: We evaluated our stabilization mechanisms for progressive pruning on the RTE task, as shown in Table 6. The results confirm that the combined EMA + Momentum setting achieves the highest accuracy (77.50%). Using only

Table 5: Performance comparison of different pruning schedulers for DPLoRA (p=0.8) on RoBERTa-base.

Scheduler	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Avg.
Bézier (Default)	62.1	85.7	89.5	91.9	89.5	77.5	93.5	89.8	**84.9**
Linear	55.2	85.3	88.8	91.9	89.6	76.8	93.1	89.5	83.8
Early-Pruning	62.1	85.8	88.4	92.1	89.2	76.4	93.6	90.4	84.7

the momentum penalty leads to a minimal performance drop (77.38%), while relying on EMA alone results in a more noticeable decrease (75.69%). Although not shown in the table, we observed that disabling both mechanisms caused training instabilities on larger datasets like QQP. Therefore, the combined approach is recommended as the default configuration to ensure stable and effective pruning.

Table 6: Effect of EMA and momentum penalty using DPLoRA (p=0.8) model on RTE.

Setting	ACC (%)	Relative Ratio (%)
EMA + Momentum (default)	77.50	100.0
EMA only	75.69	97.7
Momentum only	77.38	99.8
Neither	76.41	98.6

5 Conclusion

This study introduces a dual-pruning compression framework that combines optimal LoRA rank assignment with progressive pruning. Extensive experiments on the GLUE and Alpaca (MT-BENCH) benchmarks demonstrate that our proposed methods, OPLoRA and DPLoRA, not only surpass the performance of vanilla LoRA but also achieve highly superior results against various state-of-the-art PEFTs. Furthermore, our framework provides significant efficiency gains, reducing training time by up to 50% compared to vanilla LoRA. Our contribution is a two-stage pruning approach that the number of trainable LoRA parameters while simultaneously boosting performance, enabling efficient deployment in resource-constrained environments and supporting green AI initiatives.

The proposed approach has some limitations. The rank configurations generated by our method are task-specific, which currently limits their direct application to multi-task learning scenarios that require a single, generalized configuration.

Future work will focus on extending our framework to multi-task learning and applying it to diverse architectures such as GPT, T5, and Mixture-of-Experts (MoE) models. Other promising research directions include combining our method with other compression techniques like quantization and creating more efficient, self-adaptive optimization mechanisms.

6 Use of Large Language Models

We utilized large language models to support the research process. Specifically, LLMs were used for the following purposes:

- Writing Assistance: To improve the grammar, clarity, and overall readability of the manuscript.
- Code Development: To assist with the implementation and debugging of the experimental codebase.

The core scientific contributions, including the proposed methodology, experimental design, and the analysis and interpretation of the results, are entirely the original work of the authors.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

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

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Han Guo, Philip Greengard, Eric Xing, and Yoon Kim. LQ-loRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=xw29VvOMmU`.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

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

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

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023a.

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

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*, 2023.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. Loraprune: Structured pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*, 2023c.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

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

Huandong Chang, Zicheng Ma, Mingyuan Ma, Zhenting Qi, Andrew Sabot, Hong Jiang, and HT Kung. Elalora: Elastic & learnable low-rank adaptation for efficient model fine-tuning. *arXiv preprint arXiv:2504.00254*, 2025.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

A GLUE and Alpaca Results with Statistical Variability

To evaluate the robustness and consistency of our methods, we report results on both the GLUE and Alpaca benchmarks, each averaged over three independent runs with different random seeds (42, 777, 2025). Every result is presented as the mean score accompanied by the standard deviation (±), capturing the variability introduced by stochastic training dynamics and initialization. This evaluation protocol ensures that all reported findings are statistically grounded and not dependent on a single random seed. All models were trained using consistent and task-appropriate hyperparameters: configurations for GLUE are provided in Appendix B, and those for Alpaca are detailed in Appendix C.

Table 7: GLUE benchmark results (mean ± standard deviation). All results are averaged over three random seeds.

Model	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
FT	60.68±0.20	87.52±0.11	88.64±0.75	92.71±0.29	91.65±0.06	75.81±2.20	93.77±0.24	90.33±0.43
LoRA(r=16)	60.01±1.41	87.71±0.13	88.64±0.51	92.56±0.09	91.16±0.06	77.62±2.20	94.07±0.69	89.94±0.95
LoRA(r=8)	58.33±0.98	87.59±0.06	88.15±0.37	92.59±0.11	90.82±0.04	76.41±1.16	94.15±0.61	89.86±0.80
SoRA	58.35±6.76	87.65±0.09	87.25±2.34	92.85±0.01	91.55±0.08	76.77±3.41	93.81±0.80	90.32±0.30
AdaLoRA	61.36±1.50	87.29±0.18	86.52±0.49	92.73±0.07	89.84±0.14	78.82±0.55	93.85±0.59	90.97±0.19
OPLoRA	62.60±0.66	87.42±0.11	88.81±0.79	92.75±0.42	91.02±0.03	79.66±1.10	94.69±0.33	90.82±0.13
DPLORA(p=0.6)	61.91±1.10	86.46±0.30	89.62±1.74	92.32±0.33	90.39±0.18	77.26±1.08	94.07±0.57	90.60±0.07
DPLORA(p=0.7)	60.47±1.25	86.41±0.19	89.91±1.24	92.10±0.22	89.96±0.20	76.17±3.13	93.50±0.07	90.42±0.53
DPLORA(p=0.8)	62.08±1.00	85.67±0.64	89.54±1.98	91.93±0.14	89.53±0.09	77.50±1.78	93.46±1.00	89.78±0.83

Table 8: Instruction-following performance on MT-BENCH (mean ± std. dev.), evaluated by GPT-4. Higher scores are better.

Method	# Trainable Params(M)	MT-BENCH Score
LoRA (r=8)	21.0	5.56±0.03
OPLoRA (ours)	20.0	**5.65**±0.34
DPLoRA (ours, p=0.8)	4.0	5.54±0.02

B Hyperparameter Settings on GLUE

This appendix details the hyperparameter configurations used for the GLUE benchmark experiments. While certain settings were kept constant across all tasks—specifically, a LoRA Budget of 2,500,000 and a Batch Size of 32—we individually tuned other crucial hyperparameters to ensure optimal performance for every method. These task-specific parameters include the learning rate, number of training epochs, and maximum sequence length.

For published baselines, we adopted the configurations from their original papers where possible; otherwise, we performed our own search to ensure a fair comparison.

For the experiments presented in the Computation Efficiency analysis and the Additional Analysis subsection, we utilized a separate, computationally efficient configuration to manage computational resources, as these experiments primarily serve to validate design choices rather than establish optimal performance metrics. This adjustment, combined with the use of a single random seed, allowed us to conduct a comprehensive exploration of the design space while maintaining reasonable computational requirements. Additional details on reproducibility measures are provided in Appendix E (Reproducibility Settings).

C Hyperparameter Settings on Alpaca

This appendix details the hyperparameter configurations for the Alpaca instruction-following task.

Table 9: DPLoRA (p = 0.6, 0.7, 0.8) hyperparameter configurations across GLUE tasks.

Setting	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**
LoRA Budget	2.5M							
Batch Size	32							
Pruning Steps	10							
Learning Rate	9.3e-04	2e-4	2.3e-04	4.8e-04	3.3e-04	1.9e-04	4.8e-04	2.9e-04
Epochs	77	5	50	49	8	53	34	27
Weight Decay	0.378	0.15	0.015	0.283	0.023	0.006	0.378	0.015
Max Grad Norm	1.800	-	1.141	0.450	0.221	1.911	1.800	0.826
Max Sequence Length	256	320	320	128	320	256	256	256
Warmup Steps / Ratio	0.028	3000	0.036	0.022	0.082	0.022	0.060	0.044
Importance EMA Decay	0.054	0.06	0.814	0.976	0.879	0.596	0.054	0.501
Momentum Penalty Weight	0.329	0.09	0.609	0.915	0.473	0.057	0.329	0.660
Recovery Steps	100	500	100	100	100	100	100	100
Extended Recovery Steps	200	1000	200	200	200	200	200	200

Table 10: OPLoRA hyperparameter configurations across GLUE tasks.

Setting	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**
LoRA Budget	2.5M							
Batch Size	32							
Pruning Steps	10							
Learning Rate	9.3e-04	2e-4	2.3e-04	4.8e-04	3.3e-04	1.9e-04	4.8e-04	2.9e-04
Epochs	77	5	50	49	8	53	34	27
Weight Decay	0.378	0.15	0.015	0.283	0.023	0.006	0.378	0.015
Max Grad Norm	1.800	-	1.141	0.450	0.221	1.911	1.800	0.826
Max Sequence Length	256	320	320	128	320	256	256	256
Warmup Steps / Ratio	0.028	3000	0.036	0.022	0.082	0.022	0.060	0.044

Through preliminary experiments, we determined that training for 5 epochs provided the best balance between model capability and training efficiency, as longer training did not yield substantial improvements. Due to the significant memory requirements of the Llama 3 8B model, we used a physical batch size of 2 and employed 16 gradient accumulation steps to achieve an effective batch size of 32, ensuring stable training.

The full progressive pruning schedule was applied over the course of training, featuring 10 pruning steps to gradually enhance parameter efficiency. The complete list of hyperparameters used for this task is provided in Table 15.

D Computing Infrastructure

All experiments were conducted on a unified computing environment provided by the Backend.AI platform. Both model training and evaluation were performed under identical hardware configurations to ensure fairness. The system utilized a CPU with 20 cores and 20GB of RAM, along with a 0.3 FGPU (24GB VRAM) core as the standard GPU resource. Exceptions were made for the Llama 3 8B experiments, which required 0.5 FGPU (40GB VRAM). We used the PyTorch NGC container (version 24.09) with the following software stack: PyTorch 2.6.0, Python 3.10.12, CUDA 12.6 and PuLP 3.0.2. This standardized environment ensured consistency across runs and reproducibility of results. This constrained computing environment served as a practical testbed to validate the resource-efficiency of our optimization methods in real-world deployment scenarios. Despite the limited GPU memory, our progressive pruning strategy consistently demonstrated stable operation and achieved optimal parameter efficiency. These results confirm the robustness and practicality of our approach under realistic hardware constraints.

E Reproducibility Settings

To ensure reproducibility, we applied a range of deterministic configurations. Specifically, we set environment variables to fix the Python hash seed, configure the CUBLAS workspace,

Table 11: Full Fine-Tuning hyperparameter configurations across GLUE tasks.

Setting	SST-2	QQP	MNLI	QNLI	RTE	MRPC	CoLA	STS-B
Batch Size	32 (shared across all tasks)							
Learning Rate	2e-5	2e-5	2e-5	2e-5	1e-5	1e-5	1e-5	1e-5
Epochs	10	5	5	10	20	20	20	20
Weight Decay	0.1							
Max Grad Norm	-							
Max Sequence Length	128							
Warmup Steps	500	2800	3000	1000	100	100	100	100

Table 12: Hyperparameter configurations for DPLoRA Computation Efficiency and Additional Analysis. These settings were optimized for the rapid validation of design choices.

Setting	SST-2	QQP	MNLI	QNLI	RTE	MRPC	CoLA	STS-B
Batch Size	32							
Learning Rate	2e-4	2e-4	2e-4	2e-4	1e-4	1e-4	1e-4	1e-4
Pruning Steps	10	10	10	10	5	5	5	5
Budget	1250K	2250K	2500K	1750K	750K	750K	1200K	1000K
Epochs	10	5	5	10	20	20	20	20
Warmup Steps	500	2800	3000	1000	100	100	100	100
Recovery Steps	500	500	500	500	100	100	100	100
Extended Recovery Steps	1000	1000	1000	1000	200	200	200	200

and limit the number of threads used by numerical computation libraries. We also initialized consistent random seeds for both PyTorch and NumPy, and enabled deterministic algorithms in the CUDA backend.

For the ILP solver, we enforced deterministic behavior in the CBC solver by specifying the random seed, setting a strict time limit, enforcing single-threaded execution, and constraining the allowable optimality gap. All PyTorch DataLoaders were instantiated with fixed random generators, and we applied custom worker initialization functions to maintain consistency in multi-processing settings.

All experimental results presented in this paper, including main results and most ablations, were conducted using three different random seeds (42, 2025 and 777). This setup ensures that our findings are statistically meaningful and not dependent on any single initialization. The only exceptions were the ILP Computation Overhead and Additional Analysis experiments, which were conducted with a single seed to manage computational requirements.

F Practical Solvability of ILP-Based Optimization

The integer linear programming (ILP) problem formulated for optimal rank assignment is NP-hard and theoretically exhibits exponential time complexity in the worst case. However, in practice, our problem structure allows for efficient and reliable solving due to several key factors. First, the decision space is restricted, as the number of candidate rank values is typically limited. Second, the constraint matrix is mostly sparse except for the budget constraint. Third, we leverage the PuLP/CBC solver, which implements an efficient branch-and-bound algorithm tailored for structured combinatorial problems.

While we implemented conservative time limits (e.g., up to 600 seconds) as a safeguard, these were not necessary in practice. Our empirical results in Table 4 demonstrate the practical efficiency of this approach. The total ILP computation time for an entire training run was shown to be negligible for both models tested: less than half a second for RoBERTa-Base and approximately 7 seconds for the much larger Llama 3 8B. Considering the multi-hour fine-tuning process for these models, this overhead confirms that the theoretical complexity does not translate to a practical bottleneck.

Table 13: Hyperparameter configurations for OPLoRA Computation Efficiency and Additional Analysis.

Setting	**SST-2**	**QQP**	**MNLI**	**QNLI**	**RTE**	**MRPC**	**CoLA**	**STS-B**
Batch Size				32				
Learning Rate	2e-4	2e-4	2e-4	2e-4	1e-4	1e-4	1e-4	1e-4
Budget	1250K	2250K	2500K	1750K	750K	750K	1200K	1000K
Epochs	10	5	5	10	20	20	20	20
Warmup Steps	500	2800	3000	1000	100	100	100	100

Table 14: Hyperparameter configurations for LoRA(r=8, 16) Computation Efficiency and Additional Analysis.

Setting	**SST-2**	**QQP**	**MNLI**	**QNLI**	**RTE**	**MRPC**	**CoLA**	**STS-B**
Batch Size				32 (shared across all tasks)				
Learning Rate	2e-4	2e-4	2e-4	2e-4	1e-4	1e-4	1e-4	1e-4
Epochs	10	5	5	10	20	20	20	20
Warmup Steps	500	2800	3000	1000	100	100	100	100

G Additional Analysis

G.1 Comparison of Importance Estimation Methods

Our goal was to identify an importance metric that maximizes model performance while minimizing additional computational overhead. To achieve this, we first conducted a preliminary comparison of several efficient methods on the SST-2 task, using metrics derived from gradients, weights, and activations, as shown in Table 16. This initial experiment revealed that the Absolute Gradient metric provided the highest accuracy (93.8), establishing it as the most promising candidate among the low-overhead options.

Building on this result, we performed a more rigorous, large-scale comparison across the full GLUE benchmark to finalize our choice, as detailed in Table 17. This second experiment directly compared the two strongest gradient-based metrics: the absolute gradient ($\|g\|_1$) and the squared gradient ($\|g\|_2^2$). The Absolute Gradient once again demonstrated superior performance, achieving a higher average score of 84.9. These results confirm that the absolute gradient offers the best approach for maximizing task performance among the computationally efficient methods we evaluated, validating its selection as the default importance metric for our framework.

Table 15: Hyperparameter configurations for the Alpaca benchmark.

Setting	**OPLoRA**	**DPLoRA (p=0.8)**	**LoRA (r=8)**
Batch Size		2	
Learning Rate		1.0e-04	
Epochs		5	
Weight Decay		0.08	
Max Grad Norm		0.612	
Max Sequence Length		256	
Warmup Ratio		0.044	
LoRA Dropout		0.069	
Gradient Accumulation Steps		16	
LoRA Budget	40M	40M	N/A
Pruning Steps	N/A	10	N/A
Pruning Target Reduction	N/A	0.8	N/A
Importance EMA Decay	N/A	0.015	N/A
Momentum Penalty Weight	N/A	0.809	N/A
Recovery Steps	N/A	100	N/A
Extended Recovery Steps	N/A	200	N/A

Table 16: Comparison of importance scoring methods using the OPLoRA model on SST-2 (conducted on a single run with seed 42).

Method	Accuracy
Absolute Gradient (default)	93.8
Weight	93.4
Activation	93.4
Top 10 Gradient	93.6

Table 17: Comparison of importance metrics for DPLoRA (p=0.8).

Importance Metric	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**	**Avg.**
Absolute Gradient ($\|g\|_1$)	62.1	85.7	89.5	91.9	89.5	77.5	93.5	89.8	84.9
Squared Gradient ($\|g\|_2^2$)	57.1	85.6	88.3	91.7	88.7	77.6	93.0	87.5	83.7