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

Changjun Park[1], Sejong Yoon[2], Jaekwang Kim[1,3,*]

[1]Department of Applied Data Science, Sungkyunkwan University, Seoul, South Korea
[2]Department of Computer Science, The College of New Jersey, Ewing, NJ, USA
[3]Department of Applied Artificial Intelligence, Sungkyunkwan University, Seoul, South Korea
ckdwns8065@skku.edu, yoons@tcnj.edu, linux@skku.edu

Abstract

We propose DPLoRA (Dual-Pruning Low-Rank Adaptation), a framework that optimizes rank allocation via two stages: (1) an initial pruning stage (OPLoRA; Optimal Pruning LoRA) that uses Integer Linear Programming (ILP) to determine optimal layer-wise ranks without manual tuning; and (2) a progressive pruning stage that further reduces ranks adaptively during training using importance scores smoothed by Exponential Moving Average (EMA). Experiments demonstrate that OPLoRA consistently outperforms existing PEFT baselines on GLUE and instruction-following tasks, while the full DPLoRA framework establishes a new state-of-the-art among compared PEFT baselines on GLUE at high-performance settings ($p = 0.4$). At efficiency-focused settings ($p = 0.8$), our method reduces trainable parameters by over 80% and training time by 46% compared to standard LoRA, offering a highly efficient solution for deploying large-scale models in resource-constrained environments.[1]

1 Introduction

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 advanced NLP but require substantial computational resources for fine-tuning. Low-Rank Adaptation (LoRA) (Hu et al., 2022) mitigates this by freezing pre-trained weights and injecting trainable low-rank matrices.

However, standard LoRA applies a uniform rank (r) across all layers, a heuristic simplification primarily adopted to avoid the prohibitive cost of searching for optimal[2] per-layer ranks. Since modern LLMs comprise hundreds of layers, determining a heterogeneous rank configuration creates an exponentially large search space, making manual tuning or brute-force search computationally intractable. Consequently, this reliance on a fixed-rank assumption leads to suboptimal parameter efficiency: it implicitly assumes equal importance for all components, contradicting findings that layer contributions vary significantly (Zhang et al., 2023c; Valipour et al., 2023), and ultimately wasting computational resources on redundant layers while constraining the capacity of critical ones.

To overcome the limitations of heuristic rank selection, we propose DPLoRA (Dual-Pruning Low-Rank Adaptation). As illustrated in the unified framework in Figure 1, DPLoRA replaces manual tuning with a principled two-stage optimization process that systematically identifies and removes redundant ranks. The framework begins with an initial pruning stage (OPLoRA), employing Integer Linear Programming (ILP) to determine a structurally optimized rank configuration based on squared gradients prior to training. This static initialization is then refined by a progressive pruning stage, which dynamically adjusts the rank allocation during training using Exponential Moving Averages (EMA). By transitioning from a global structural decision to adaptive fine-grained adjustments, DPLoRA effectively navigates the combinatorial search space of rank allocation avoiding potential instability in purely dynamic adjustments.

To validate the effectiveness of this structured approach, we conducted extensive experiments on the GLUE benchmark (RoBERTa-base) and the Alpaca instruction-following task (LLaMA 3 family). Results demonstrate that DPLoRA successfully balances model capacity and computational efficiency across varying pruning rates p. For instance, at high-performance settings ($p = 0.4$), it establishes SOTA among compared PEFT baselines on GLUE while reducing parameters by over

*Corresponding Author

[1]https://github.com/cjpark-ai/DPLoRA

[2]In this paper, the term 'optimal' specifically denotes the mathematically exact solution to the ILP problem formulated based on our proposed layer importance metric, rather than an absolute, globally optimal parameter configuration.

Findings of the Association for Computational Linguistics: ACL 2026, pages 33892–33908

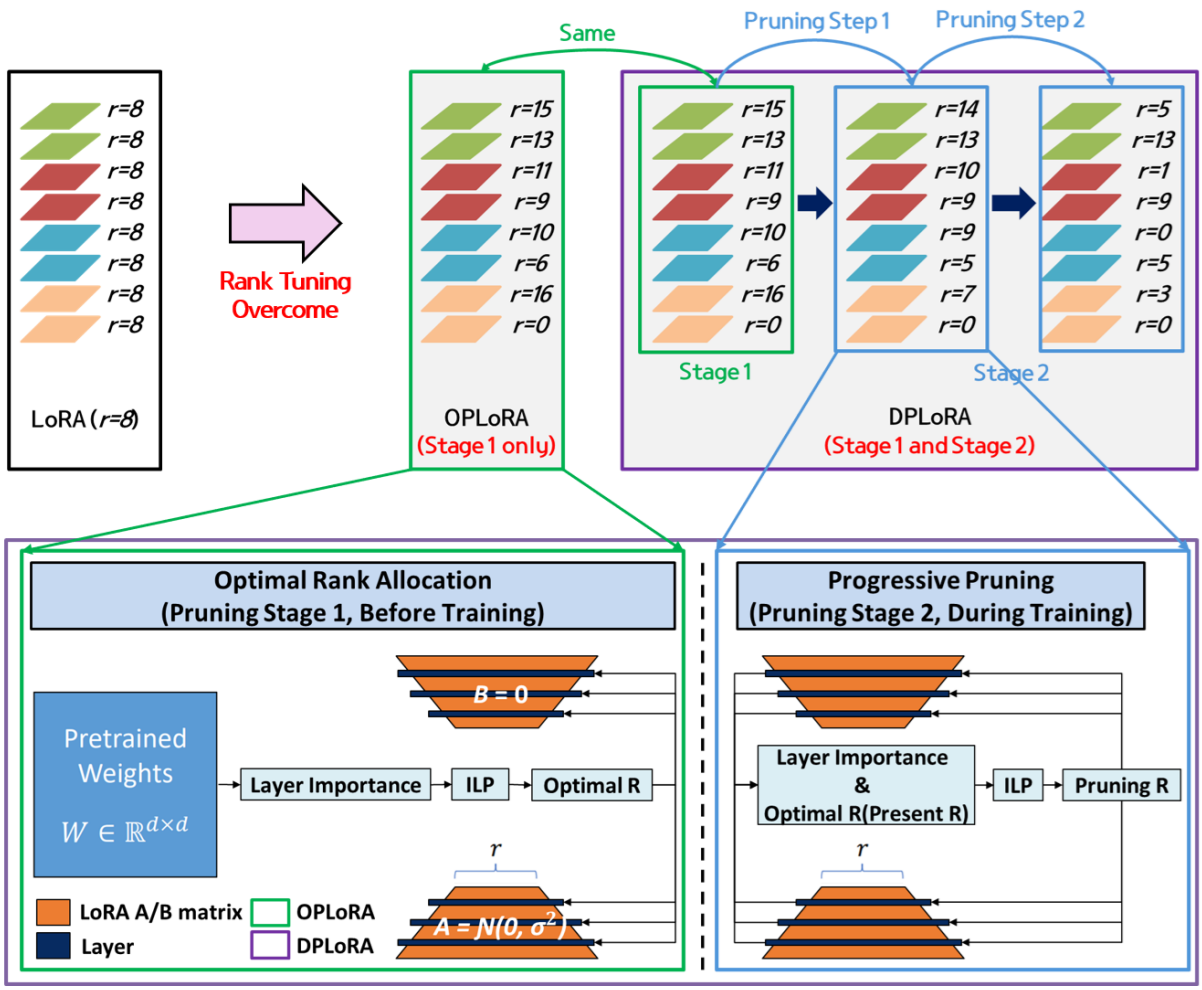

Figure 1: Unified framework of DPLoRA, illustrating the transition from initial rank allocation (Pruning Stage 1) to progressive pruning (Pruning Stage 2).

40%, and at efficiency-focused settings ($p = 0.8$), it cuts training time by nearly half. These findings confirm that coordinating rank allocation via optimization yields superior trade-offs compared to fixed or greedy baselines.

Our contributions are summarized as follows:

- We formulate rank allocation as an Integer Linear Programming (ILP) problem[3]. This formulation provides a principled method to solve the combinatorial challenge of rank selection, ensuring a coordinated parameter distribution under budget constraints unlike manual or iterative importance-based approaches.

- We design a synergistic two-stage framework that combines the stability of gradient-based initialization with the adaptability of progressive pruning, addressing the limitations of both rigid static allocations and potentially unstable dynamic adjustments.

- We demonstrate that both stages of our framework—OPLoRA (Stage 1) and the full DPLoRA (Stages 1 and 2)—substantially reduce computational costs while achieving state-of-the-art or highly competitive performance across diverse architectures, validating the practical utility of our optimization-based strategy.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning (PEFT)

PEFT methods adapt large language models (LLMs) with minimal overhead. Early approaches like Adapter Tuning (Houlsby et al., 2019), which insert bottleneck layers, and Prefix-Tuning (Li and Liang, 2021) or Prompt Tuning (Lester et al., 2021), which tune continuous prompts, all freeze pre-

[3]ILP usage inspired by LQ-LoRA (Guo et al., 2024), which applies it to quantization, while we apply it to rank allocation.

trained weights while training only a small number of parameters.

2.2 Refinements to the LoRA Update Mechanism

Low-Rank Adaptation (LoRA) (Hu et al., 2022) injects trainable low-rank matrices ($\Delta W = AB$). Building on this foundation, one line of research improves LoRA by modifying the structure or training of the update matrices A and B. DoRA (Liu et al., 2024) disentangles LoRA update magnitude and direction components for stability. VeRA (Kopiczko et al., 2024) shares a single pair of frozen random matrices across layers and learns small scaling vectors, drastically reducing parameters. LoRA-FA (Zhang et al., 2023b) freezes the randomly initialized matrix A and trains only B to reduce memory. While effective, they operate under a fixed-rank assumption, ignoring optimal allocation.

2.3 Importance-Based Automated Rank Allocation

A second line of research automates rank selection via importance-based layer-wise adjustment. These methods start from an initially large rank and progressively remove or zero out less important components during training. For example, AdaLoRA (Zhang et al., 2023c) adopts an SVD-like decomposition and prunes singular values based on sensitivity-based importance scores. SoRA (Ding et al., 2023) inserts a gate vector optimized via proximal gradient with soft-thresholding to drive redundant rank components to zero.

In contrast, another line of work explores dynamic or constructive strategies for rank adaptation. DyLoRA (Valipour et al., 2023) enables dynamic inference across ranks via truncated updates and frozen-update rank sampling to progressively order information. IncreLoRA (Zhang et al., 2023a) incrementally increases ranks during training to mitigate premature pruning of important components. ElaLoRA (Chang et al., 2025) dynamically prunes and expands ranks based on gradient-derived importance scores, enabling adaptive layer-wise rank allocation. Finally, AutoLoRA (Zhang et al., 2024) introduces a meta-learning framework that optimizes selection variables for rank-1 components to automatically derive layer-wise optimal ranks.

Although these methods automate rank selection, their reliance on iterative adjustments, proxy metrics, or search processes may lead to suboptimal or locally driven solutions. In contrast, our framework, which includes both OPLoRA (Stage 1) and the full DPLoRA (Stages 1 and 2), formulates rank allocation as a principled optimization problem using Integer Linear Programming (ILP). To the best of our knowledge, we are the first to frame LoRA rank allocation as an exact integer optimization problem with a global budget constraint, distinguishing our approach from prior importance-based heuristic methods.

3 Methodology

3.1 OPLoRA: Optimal Rank Allocation (Pruning Stage 1)

Layer Importance Estimation: Layer importance is measured by the mean of squared gradients, approximating the diagonal of the empirical Fisher Information matrix (Guo et al., 2024) and capturing each layer's sensitivity to the loss. Mathematically, importance I_l^{init} of layer l is computed via Equation 1 using gradients of the pre-trained weights θ_l with respect to the loss function $\mathcal{L}$. To ensure stability, we estimate this expectation by averaging squared gradients over five mini-batches, using up to 500 samples from dataset $\mathcal{D}$.

$$I_l^{\text{init}} = \mathbb{E}_{x\sim\mathcal{D}}\left[\text{mean}\left(\left(\nabla_{\theta_l}\mathcal{L}(x)\right)^2\right)\right] \quad (1)$$

Performance Gain Estimation: Performance gain $G_{l,r}$ when assigning rank r to layer l is modeled via Equation 2, where $M_l = \min(d_l^{\text{in}}, d_l^{\text{out}})$ represents the maximum possible rank, with d_l^{in} and d_l^{out} denoting the input and output dimensions. The exponential term captures diminishing returns, ensuring that marginal benefits saturate as rank r increases.

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

Objective Function: We formulate an Integer Linear Programming (ILP) problem for optimal rank allocation. Decision variables are binary, defined as $x_{l,r} \in \{0, 1\}$, where $x_{l,r} = 1$ indicates rank r is allocated to layer l. We define the objective function $\mathcal{J}_{\text{Stage1}}$ as the total performance gain, which we seek to maximize.

$$\mathcal{J}_{\text{Stage1}} = \sum_l \sum_r G_{l,r} \cdot x_{l,r} \quad (3)$$

Parameter Cost Modeling: Parameter cost $C_{l,r}$ of allocating rank r to layer l represents the number of trainable parameters introduced by LoRA matrices

$A_l \in \mathbb{R}^{d_l^{\text{in}} \times r}$ and $B_l \in \mathbb{R}^{r \times d_l^{\text{out}}}$, and is used as a constraint in the ILP formulation.

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

Constraints of Optimal Rank Allocation (Pruning Stage 1): The ILP enforces three constraints: (1) each layer l receives exactly one rank; (2) the total parameter cost must not exceed the global budget $\mathcal{B}_{\text{initial}}$; and (3) the aggregate rank within each layer type set $\mathcal{M}$ (e.g., FFN, attention) is at least one to prevent component collapse. Formally, our optimization problem can be expressed as:

$$\begin{aligned} \max \quad & \mathcal{J}_{\text{Stage1}} \\ \text{s.t.} \quad & \sum_r x_{l,r} = 1, \quad \forall l \\ & \sum_l \sum_r C_{l,r} \cdot x_{l,r} \leq \mathcal{B}_{\text{initial}}, \\ & \sum_{l \in \Omega_m} \sum_r r \cdot x_{l,r} \geq 1, \quad \forall m \in \mathcal{M} \end{aligned} \tag{5}$$

where Ω_m denotes the set of layers belonging to type m. These constraints apply to both Stage 1 and 2.

3.2 Progressive Pruning (Pruning Stage 2)

Enhanced Layer Importance Estimation: Unlike Stage 1, which estimates layer importance from full pre-trained weights (θ_l), Stage 2 derives importance solely from trainable LoRA matrices (A_l, B_l) as the original weights are frozen. We apply an Exponential Moving Average (EMA) with decay factor β to track dynamic importance $I_l^{(t)}$ during training, where $I_l^{(t)}$ denotes the importance of layer l at time t as shown in Equation 6. I_l^{update} is the geometric mean of the two means of squared gradients from both adapter matrices, weighted by $\sqrt{r_l^{\text{current}}}$ to reflect the greater influence of high-rank layers while preventing disproportionate inflation via sub-linear scaling, as detailed in Equation 7.

$$I_l^{(t)} = \beta \cdot I_l^{(t-1)} + (1 - \beta) \cdot I_l^{\text{update}} \tag{6}$$

$$I_l^{\text{update}} = \mathbb{E}_{x \sim \mathcal{D}} \left[\sqrt{\text{mean}\left((\nabla_{A_l} \mathcal{L}(x))^2 \right) \cdot \text{mean}\left((\nabla_{B_l} \mathcal{L}(x))^2 \right)} \right] \times \sqrt{r_l^{\text{current}}} \tag{7}$$

Performance Loss Estimation: Performance loss incurred when reducing the rank of layer l from r_l^{current} to r^{new} is defined in Equation 8.

$$L_{l,r^{\text{new}}} = I_l^{(t)} \cdot \frac{r_l^{\text{current}} - r^{\text{new}}}{r_l^{\text{current}}}, \quad (r_l^{\text{current}} \geq r^{\text{new}}) \tag{8}$$

Objective Function: Binary variables $x_{l,r^{\text{new}}} \in \{0, 1\}$ indicate the allocation of new rank r^{new} to layer l. We define the objective function $\mathcal{J}_{\text{Stage2}}$ as the total loss plus a Rank Stability Regularizer $P_{l,r^{\text{new}}}$ to encourage smooth and stable rank updates. We seek to minimize this function.

$$\mathcal{J}_{\text{Stage2}} = \sum_l \sum_{r^{\text{new}}} (L_{l,r^{\text{new}}} + P_{l,r^{\text{new}}}) \cdot x_{l,r^{\text{new}}} \tag{9}$$

The regularizer $P_{l,r^{\text{new}}}$ (Equation 10) enforces smoothness via two importance-scaled terms: a rank change penalty (γ term) and a stability incentive (δ term) using the indicator function.

$$\begin{aligned} P_{l,r^{\text{new}}} = & \gamma \cdot I_l^{(t)} \cdot |r^{\text{new}} - r_l^{\text{current}}| \\ & - \delta \cdot I_l^{(t)} \cdot \mathbf{1}_{r^{\text{new}} = r_l^{\text{current}}} \end{aligned} \tag{10}$$

Constraints of Progressive Pruning (Pruning Stage 2): While maintaining the fundamental single-rank constraint from Stage 1, we impose two additional constraints specific to the progressive phase: (1) a strictly non-increasing rank trajectory ($r^{\text{new}} \leq r_l^{\text{current}}$); and (2) a dynamic parameter budget $\mathcal{B}_{\text{step}}(t)$ that decreases at each pruning step t, governed by a Bézier-based scheduler. Formally, the optimization problem is formulated as:

$$\begin{aligned} \min \quad & \mathcal{J}_{\text{Stage2}} \\ \text{s.t.} \quad & \sum_{r^{\text{new}}} x_{l,r^{\text{new}}} = 1, \quad \forall l \\ & \sum_{r^{\text{new}}} r^{\text{new}} \cdot x_{l,r^{\text{new}}} \leq r_l^{\text{current}}, \quad \forall l \\ & \sum_l \sum_{r^{\text{new}}} C_{l,r^{\text{new}}} \cdot x_{l,r^{\text{new}}} \leq \mathcal{B}_{\text{step}}(t) \end{aligned} \tag{11}$$

Bézier-based Pruning Scheduler: To ensure smooth and controlled parameter reduction, we employ a cubic Bézier curve to schedule the parameter budget $\mathcal{B}_{\text{step}}(\tau)$ from the initial parameter budget $\mathcal{B}_{\text{initial}}$ over N pruning steps (Equation 12). Here, τ denotes the pruning step index ($1 \leq \tau \leq N$). The reduction ratio $R(\tau)$ is defined by control points $P = [0.0, 0.2, 0.8, 1.0]$, where P_i denotes the i-th element of P, ensuring smooth convergence toward the target ratio R_{target} (Equation 13). The pruning trigger $T_{\text{trigger}}(\tau)$ (Equation 14) spaces pruning steps between the warm-up period T_{delay} and total training steps T_{total}, dividing by $N + 1$ to ensure recovery after the final pruning step.

$$\mathcal{B}_{\text{step}}(\tau) = \mathcal{B}_{\text{initial}} \cdot (1 - R(\tau)) \tag{12}$$

$$R(\tau) = R_{\text{target}} \cdot \sum_{i=0}^{3} \binom{3}{i} P_i \cdot \left(\frac{\tau}{N}\right)^i \cdot \left(1 - \frac{\tau}{N}\right)^{3-i} \tag{13}$$

$$T_{\text{trigger}}(\tau) = T_{\text{delay}} + \tau \cdot \frac{T_{\text{total}} - T_{\text{delay}}}{N+1} \tag{14}$$

3.3 Adaptive Rank Adjustment Strategy

After solving the Stage 2 ILP, we resize the LoRA adapters under the non-increasing rank constraint $r^{\text{new}} \leq r_l^{\text{current}}$ by reducing the dimensions of matrices $A_l \in \mathbb{R}^{d_l^{\text{in}} \times r_l^{\text{current}}}$ and $B_l \in \mathbb{R}^{r_l^{\text{current}} \times d_l^{\text{out}}}$.

Dimension Pruning Criterion: Unlike gradient-based ILP, we use weight magnitude to preserve accumulated knowledge. For each rank dimension i, importance score $S_{l,i}$ is the product of L2 norms of the corresponding column $(A_l)_{:,i}$ and row $(B_l)_{i,:}$.

$$S_{l,i} = \|(A_l)_{:,i}\|_2 \cdot \|(B_l)_{i,:}\|_2 \tag{15}$$

This score captures the contribution of each rank dimension to the learned transformation.

Top-k Pruning Execution: We select the indices $\mathcal{I}_l^{\text{top}}$ of the top-k dimensions ranked by $S_{l,i}$ scores, where $k = r^{\text{new}}$. Pruned matrices are constructed by slicing the original matrices to retain these indices (Equation 16), preserving core learned features by retaining dimensions with the largest weight magnitudes.

$$A_l^{\text{new}} = (A_l)_{:,\mathcal{I}_l^{\text{top}}}, \quad B_l^{\text{new}} = (B_l)_{\mathcal{I}_l^{\text{top}},:} \tag{16}$$

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 using the standard train/validation splits of the GLUE benchmark (Wang et al., 2018), which includes tasks such as sentiment analysis, paraphrase detection, and natural language inference, with performance measured by their respective standard metrics (e.g., Accuracy, F1, Matthews Correlation), as detailed in the caption of Table 1. Additionally, we assess instruction-following capabilities using the original Stanford Alpaca dataset (52K instructions) (Taori et al., 2023), where performance is quantitatively evaluated using MT-BENCH (Zheng et al., 2023). 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 family for the instruction-following task on Alpaca. Our approach is compared against strong baselines specific to each task: for GLUE, these include BitFit, Adapter, standard LoRA, SoRA, and AdaLoRA; for Alpaca, we use standard 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 baseline runs that were re-evaluated using the same hyperparameter settings (e.g., learning rate, training epochs) as our OPLoRA and DPLoRA setups for a fair comparison. To solve the Integer Linear Programming (ILP) problems for optimal rank allocation, we utilized the PuLP library in Python, employing the COIN-OR Branch and Cut (CBC) solver.

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 a high baseline score of 85.4 using 0.9M parameters, and BitFit* delivering a competitive 85.2 with just 0.1M parameters. Notably, even when we double the initial rank budget of AdaLoRA (AdaLoRA+, 0.8M), its average score only improves marginally from 85.2 to 85.5.

Our proposed OPLoRA framework demonstrates superior efficacy on the GLUE benchmark by outperforming all baseline methods with an average score of 85.9. This improvement is achieved

with only 1.2M trainable parameters, yielding higher performance compared to LoRA ($r = 8$, 85.3 avg) while requiring fewer parameters (1.2M vs. 1.3M, a 7.7% reduction). This result strongly validates our importance-based rank allocation strategy.

Even more strikingly, DPLoRA ($p = 0.4$) sets a new state-of-the-art among compared PEFT baselines with the highest average score of 86.1, surpassing even OPLoRA while using significantly fewer parameters (0.7M vs. 1.2M, a 41.7% reduction). Crucially, it also outperforms the expanded AdaLoRA+ baseline (86.1 vs. 85.5) despite utilizing fewer trainable parameters (0.7M vs. 0.8M). This demonstrates that our progressive pruning strategy effectively eliminates parameter redundancy while preserving the model's essential representational capability. Furthermore, the DPLoRA variants showcase an exceptional performance-efficiency trade-off. Specifically, DPLoRA ($p = 0.6$) outperforms standard LoRA (85.3) with an average score of 85.4, while utilizing only 0.5M parameters, i.e., 61.5% fewer than the LoRA baseline (0.5M vs. 1.3M). Even at a higher pruning rate, DPLoRA ($p = 0.8$) retains a respectable average of 84.7 using a mere 0.2M parameters, demonstrating DPLoRA's versatility across diverse resource constraints.

Computation Efficiency. Table 2 demonstrates the computational advantages of our proposed methods. Compared to the LoRA ($r = 8$) baseline (set as 100% relative time), OPLoRA achieves a training time reduction of 13% (87% relative time). Furthermore, DPLoRA ($p = 0.8$) achieves a remarkable 46% training time reduction (54% relative time). These gains stem from two factors. First, OPLoRA's static optimization (Stage 1) allocates fewer total parameters than the baseline, resulting in a 13% time reduction from the outset. Second, DPLoRA introduces progressive pruning (Stage 2), which actively removes redundant ranks during training. This dynamic reduction allows DPLoRA ($p = 0.8$) to cut the training time by nearly half compared to the baseline, making it highly suitable for resource-constrained environments.

4.3 Results on Instruction Following

Performance on MT-BENCH across Model Scales: To evaluate the instruction-following capabilities and scalability of our framework, we conducted experiments on MT-BENCH using models of varying sizes: LLaMA 3.2 (1B, 3B) and LLaMA 3 (8B). We compared our methods against standard LoRA and DoRA (Weight-Decomposed Low-Rank Adaptation). As shown in Table 3, our approach demonstrates consistent effectiveness across different scales.

Across all model sizes, OPLoRA consistently achieves the highest performance gains over both baselines. Specifically, on LLaMA 3 8B, OPLoRA achieves a score of 5.20, surpassing both LoRA ($r = 8$, 5.13) and DoRA (4.97), while maintaining efficient parameter usage (20.5M; a 2.3% and 8.1% reduction, respectively). Similarly, on the 3B model, OPLoRA improves the score to 4.48, exceeding both LoRA (4.46) and DoRA (4.33) with 11.7M parameters (a 4.1% and 9.3% reduction, respectively). These results indicate that our optimal rank allocation is superior to fixed-rank or weight-decomposed strategies across various scales.

Furthermore, DPLoRA ($p = 0.6$) demonstrates exceptional parameter efficiency and generalization across all model scales. Notably, on LLaMA 3 8B, DPLoRA achieves a performance identical to DoRA (4.97) while requiring 63.2% fewer trainable parameters (8.2M vs. 22.3M). A similar trend is observed in the LLaMA 3.2 3B model, where DPLoRA maintains a high score of 4.10 while utilizing 61.4% fewer parameters than the LoRA baseline (4.7M vs. 12.2M). While a performance-efficiency trade-off is naturally observed in the highly constrained 1B model, where DPLoRA achieves a score of 2.51 while requiring 60.7% fewer parameters than the LoRA baseline (2.2M vs. 5.6M), the overall results confirm that our progressive pruning strategy effectively identifies and prunes redundant ranks without fundamentally compromising the model's generative quality.

4.4 Ablation Study

We conducted a step-by-step ablation study across varying pruning rates ($p = 0.6, 0.7, 0.8$) to evaluate model robustness under extreme compression regimes. For the initial analyses (Step 1 and Step 2), we fixed the pruning scheduler to a standard Linear scheduler. This controlled setting isolates the impact of importance metrics and stabilization mechanisms before optimizing the pruning trajectory in Step 3.

Step 1: Determining Core Metrics. We first evaluate the importance metrics for ILP layer selection. The Squared Gradient metric (g^2) consistently demonstrates superior performance, significantly outperforming Weight or Activation-based metrics

Table 1: Performance and the number of trainable parameters (in millions) on the GLUE benchmark. We report accuracy for most tasks, Matthews correlation for CoLA, and Pearson correlation for STS-B. For MNLI, we report the average accuracy of the matched and mismatched sets. Higher is better. AdaLoRA+ denotes AdaLoRA run with its official default settings while only increasing the target parameter budget. (* denotes results from original papers).

Method	#Trainable Params(M)	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Avg.
Full FT	125.0	60.7	87.5	88.6	92.7	**91.7**	75.8	93.8	90.3	85.1
LoRA ($r = 8$)	1.3	63.4	87.2	87.8	92.5	90.5	76.3	94.4	90.7	85.3
SoRA	0.9	58.4	**87.7**	87.3	92.9	91.6	76.8	93.8	90.3	84.8
AdaLoRA+	0.8	61.7	87.5	87.4	92.7	90.0	78.9	94.4	91.0	85.5
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	**63.7**	87.5	89.8	**93.0**	90.9	77.6	**94.8**	90.0	85.9
DPLoRA ($p = 0.4$)	0.7	63.6	87.5	89.2	**93.0**	90.9	80.0	94.6	90.2	**86.1**
DPLoRA ($p = 0.5$)	0.6	61.4	87.2	89.5	92.8	90.7	79.3	94.0	90.0	85.6
DPLoRA ($p = 0.6$)	0.5	61.2	86.8	89.5	92.8	90.5	78.3	94.7	89.8	85.4
DPLoRA ($p = 0.7$)	0.4	59.5	86.4	89.6	92.8	90.1	77.1	93.5	89.2	84.8
DPLoRA ($p = 0.8$)	0.2	60.5	85.5	89.1	92.3	89.2	77.6	94.0	89.2	84.7

Table 2: Training runtime efficiency across GLUE tasks († denotes the same hyperparameter matching).

Method	#Trainable Params(M)	Training Time (min)	Relative Time (%)
LoRA ($r = 8$)†	1.3	2,889	100
Adapter †	0.9	1,595	55
BitFit †	0.1	1,085	38
OPLoRA (ours)	1.2	2,502	87
DPLoRA (ours, $p = 0.8$)	0.2	**1,570**	**54**

Table 3: Instruction-following performance on MT-BENCH across different LLaMA model sizes, evaluated by GPT-4. Higher scores indicate better performance. († denotes the same hyperparameter matching). Values are reported to two decimal places to distinguish performance ties.

Model	Method	#Trainable Params(M)	MT-BENCH
LLaMA 3.2 1B	LoRA ($r = 8$)†	5.6	3.07
	DoRA	6.0	2.83
	OPLoRA (ours)	5.5	**3.12**
	DPLoRA (ours, $p = 0.6$)	**2.2**	2.51
LLaMA 3.2 3B	LoRA ($r = 8$)†	12.2	4.46
	DoRA	12.9	4.33
	OPLoRA (ours)	11.7	**4.48**
	DPLoRA (ours, $p = 0.6$)	**4.7**	4.10
LLaMA 3 8B	LoRA ($r = 8$)†	21.0	5.13
	DoRA	22.3	4.97
	OPLoRA (ours)	20.5	**5.20**
	DPLoRA (ours, $p = 0.6$)	**8.2**	4.97

across all tasks.

With the ILP metric established as Squared Gradient, we next analyze the physical dimension pruning criteria presented in Table 4. Here, the Weight L2 Norm demonstrates superior efficacy compared

Table 4: Core Metric Analysis: Comparison of Layer Importance Metrics and Dimension Pruning Criteria with Linear Scheduler. Reported values represent the average score across pruning rates $p = 0.6, 0.7, 0.8$. Values are reported to two decimal places to distinguish performance ties.

Task	Layer Importance (ILP)	Dimension Pruning Criterion	
		Weight L2 Norm	Squared Gradient
MRPC	Squared Gradient (g^2)	**88.81**	88.75
	Absolute Gradient ($\|g\|$)	87.83	88.62
	Squared Weight (W^2)	85.10	84.94
	Squared Activation (A^2)	83.31	81.43
RTE	Squared Gradient (g^2)	**76.85**	76.45
	Absolute Gradient ($\|g\|$)	75.21	75.61
	Squared Weight (W^2)	71.12	72.00
	Squared Activation (A^2)	70.20	70.24

to the Squared Gradient. Specifically, in MRPC, Weight L2 achieves 88.81 (surpassing the gradient-based score of 88.75), and in RTE, it leads with 76.85 compared to 76.45. This confirms that preserving accumulated weight magnitude is more effective for retaining model capability than relying on instantaneous sensitivity. Consequently, we adopt the combination of Squared Gradient (for ILP) and Weight L2 Norm (for Dimension Pruning) as the optimal configuration.

Step 2: Impact of Rank Stability Mechanisms. Building upon the metrics selected in Step 1, we examined the effectiveness of the stabilization mechanisms within the Pruning Stage 2 ILP formulation: the Exponential Moving Average (EMA) for importance smoothing and the Rank Stability Reg-

Table 5: Stability Analysis: Impact of Rank Stability Mechanisms with Linear Scheduler. We report the accuracy (%) and the relative rank to demonstrate robustness across tasks.

Task	Configuration	Accuracy (%)				Rank
		$p = 0.6$	$p = 0.7$	$p = 0.8$	Avg	
MRPC	**EMA + Regularizer (Ours)**	89.5	88.9	88.0	88.8	2
	Regularizer Only	89.4	89.1	87.2	88.5	4
	EMA Only	89.4	88.6	**90.1**	**89.4**	**1**
	Neither	**89.6**	**89.4**	87.4	88.8	2
RTE	**EMA + Regularizer (Ours)**	**78.6**	76.5	**75.5**	**76.9**	**1**
	Regularizer Only	76.8	**77.1**	**75.5**	76.5	2
	EMA Only	77.5	76.9	73.8	76.1	4
	Neither	76.7	77.0	75.3	76.3	3

Table 6: Scheduler Analysis: Comparison of Pruning Schedules. We report the accuracy (%) across varying pruning rates to demonstrate robustness under extreme compression levels.

Task	Scheduler	Accuracy (%)			
		$p = 0.6$	$p = 0.7$	$p = 0.8$	Avg
MRPC	**Bézier (Ours)**	**89.5**	**89.6**	**89.1**	**89.4**
	Linear	**89.5**	88.9	88.0	88.8
	Early Pruning	89.0	89.0	88.4	88.8
RTE	**Bézier (Ours)**	78.3	77.1	**77.6**	**77.7**
	Linear	**78.6**	76.5	75.5	76.9
	Early Pruning	78.2	**77.5**	76.1	77.3

ularizer for preventing structural oscillation. As shown in Table 5, relying on a single component can lead to inconsistent performance across tasks. For instance, while 'EMA Only' achieves the top rank in MRPC, it drops to the lowest rank (Rank 4) in RTE. In contrast, our proposed combination 'EMA + Regularizer' demonstrates superior robustness, achieving Rank 1 in the challenging RTE task (76.9%) and maintaining a competitive Rank 2 in MRPC. This confirms that the synergy between smoothing importance scores and the stability regularizer is essential for ensuring robust generalization across diverse downstream tasks.

Step 3: Impact of Pruning Scheduler. Finally, we evaluated the impact of the pruning scheduler, which dictates the rate of rank reduction over time. We compared our Bézier curve-based scheduler against standard Linear and Early Pruning strategies. As shown in Table 6, the Bézier scheduler achieves the highest or comparable average accuracy across tasks (MRPC: 89.4, RTE: 77.7).

Crucially, the advantage of the Bézier scheduler becomes most apparent in the extreme compression regime ($p = 0.8$). While the Linear scheduler suffers a significant performance drop at $p = 0.8$ (e.g., MRPC: 88.0, RTE: 75.5), the Bézier scheduler maintains superior performance (MRPC: 89.1, RTE: 77.6). This confirms that the smooth reduction trajectory allows the model to adapt more effectively to capacity constraints compared to aggressive linear reduction.

4.5 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 and modules, highlighting the suboptimality of fixed-rank approaches. Subsequently, Figure 3 demonstrates the effectiveness of progressive pruning: even as parameters are reduced by 60% (blue line), performance rapidly reaches near-optimal levels and remains stable (red line). This illustrates that our two-stage approach achieves significant parameter efficiency without degrading final task performance.

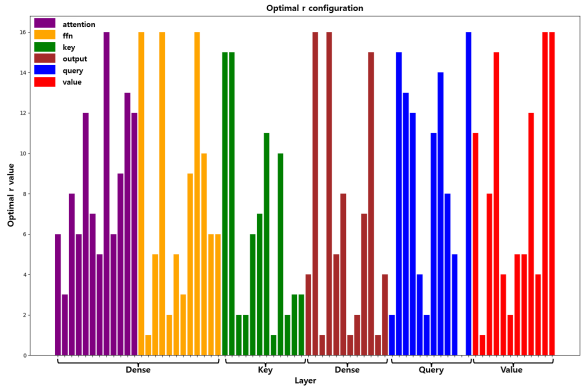

Figure 2: Initial heterogeneous rank allocation for RoBERTa-base on CoLA.

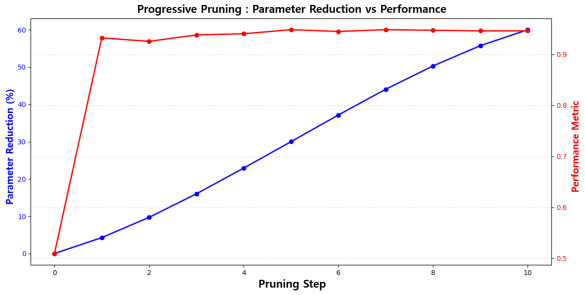

Figure 3: Progressive pruning on SST-2 (DPLoRA, $p = 0.6$). Performance remains stable while parameter reduction reaches 60%.

ILP Computation Overhead: We address potential concerns regarding the computational overhead of Integer Linear Programming (ILP). Our analysis in Table 7 confirms that the cost of our framework is negligible across various model sizes.

Table 7: ILP solver execution time (in seconds) for the SST-2 and Alpaca tasks using DPLoRA ($p = 0.6$). We report the total accumulated time and the average time computed over all 11 optimization steps (1 initial allocation plus 10 progressive pruning steps).

Model	Task	Total Time (s)	Avg. Time (s)
RoBERTa-base	SST-2	0.45	0.04
LLaMA 3.2 1B	Alpaca	0.77	0.07
LLaMA 3.2 3B	Alpaca	2.47	0.22
LLaMA 3 8B	Alpaca	1.89	0.17

For a model like RoBERTa-base, the total ILP computation time for the entire training process is less than half a second (0.45s). Even for the much larger LLaMA 3 family, the cumulative overhead remains remarkably low, totaling just 2.47 seconds in the most demanding case. Given that fine-tuning large-scale models typically spans several hours, an overhead of less than 3 seconds represents a negligible 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.

5 Conclusion

This study introduces a dual-pruning compression framework that combines optimal LoRA rank allocation with progressive pruning. Extensive experiments on the GLUE and Alpaca (MT-BENCH) benchmarks demonstrate that our proposed framework—OPLoRA (Stage 1) and the full DPLoRA (Stages 1 and 2)—not only surpasses the performance of standard LoRA but also achieves superior results compared to various state-of-the-art PEFT methods. Crucially, the consistent success on both RoBERTa-base (encoder-only) and LLaMA 3 family (decoder-only) models highlights the framework's broad applicability across different architectural paradigms. Furthermore, our framework provides significant efficiency gains, reducing training time by up to 46% compared to standard LoRA. Additionally, the substantial reduction in adapter size enhances storage efficiency, offering distinct advantages for multi-tenant deployment scenarios. Our contribution is a two-stage pruning approach that automates optimal rank selection, thereby eliminating the need for manual rank tuning. This approach significantly reduces the number of trainable LoRA parameters while simultaneously boosting performance, enabling efficient deployment in resource-constrained environments and supporting green AI initiatives.

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, incorporating hardware-aware constraints (e.g., multiples of 8 or 16 for Tensor Core utilization), and creating more efficient, self-adaptive optimization mechanisms.

6 Limitations

The proposed approach has limitations. First, the task-specific rank configurations derived from our optimization process limit direct application to multi-task learning scenarios, where a shared representation is typically required. Second, since LoRA adapters are typically merged with backbone weights during inference, our method reduces training memory and storage but not inference latency compared to full fine-tuning. Third, while DPLoRA significantly reduces the number of trainable parameters, the actual impact on total training time is hardware-dependent. On dedicated GPUs, kernel launch overheads can offset the theoretical computational gains, occasionally resulting in a marginal increase in overall training time.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korean government (MSIT) (No. IITP-2026-RS-2024-00346737), and by grants from the Institute for Information & Communications Technology Planning & Evaluation (IITP) funded by the Ministry of Science and ICT (MSIT), Korea, through the Global Scholars Invitation Program (No. IITP-2026-RS-2024-00459638), the Graduate School of Metaverse Convergence at Sungkyunkwan University (No. IITP-2026-RS-2023-00254129), and the ICT Challenge and Advanced Network of HRD (ICAN) support program (No. IITP-2026-RS-2023-00259497). The authors also acknowledge the use of the ELSA HPC cluster at The College of New Jersey for conducting the research reported in this paper; this cluster is funded in part by the National Science Foundation (NSF) under grant numbers OAC-1826915 and OAC-2320244.

References

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

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

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

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)*.

Han Guo, Philip Greengard, Eric Xing, and Yoon Kim. 2024. LQ-loRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *The Twelfth International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International conference on machine learning (ICML)*, 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 *Proceedings of International Conference on Learning Representations (ICLR)*.

Dawid J Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. *Proceedings of International Conference on Learning Representations (ICLR)*.

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)*.

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

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

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models.

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. *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

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

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

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

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

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. AutoLoRA: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies (Volume 1: Long Papers)*, pages 5048–5060.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, and Eric Xing. 2023. Judging LLM-as-a-judge with MT-BENCH and chatbot arena. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:46595–46623.

A GLUE and Alpaca Results with Statistical Variability

To substantiate the robustness and reproducibility of our proposed methods, we report performance metrics across both the GLUE and Alpaca benchmarks, derived from the average of three independent trials initialized with distinct random seeds (42, 777, and 2025). All outcomes are reported as the mean score $\pm$ the standard deviation, thereby quantifying the variability inherent in stochastic training dynamics and weight initialization. This rigorous evaluation protocol ensures that the reported findings are statistically grounded and not artifacts of specific random seeds. For experimental consistency, we employed task-appropriate hyperparameter configurations; specific details for GLUE are provided in Appendix B, while specifications for Alpaca are outlined in Appendix C.

B Hyperparameter Settings on GLUE

This appendix details the hyperparameter configurations used for the GLUE benchmark experiments. While certain settings remained constant across all tasks—specifically, a fixed LoRA parameter budget of 1,263,000 and a batch size of 32—we independently optimized other key hyperparameters to ensure peak performance for each method. These task-specific parameters include the learning rate, the number of training epochs, and the maximum sequence length.

For published baselines, we adopted configurations from their respective original papers where feasible; otherwise, we applied the same hyperparameter settings as our method for a fair comparison. Additional details on reproducibility measures are provided in Appendix I.

C Hyperparameter Settings on Alpaca

This appendix outlines the hyperparameter configurations for the Alpaca instruction-following task, covering experiments on the LLaMA 3.2 1B, LLaMA 3.2 3B, and LLaMA 3 8B models.

To accommodate the substantial memory constraints of the LLaMA 3 8B model, we utilized a physical batch size of 2 combined with 8 gradient accumulation steps to achieve an effective batch size of 16, thereby ensuring stable training. For the LLaMA 3.2 1B and 3B models, configurations were similarly optimized to balance resource management with performance. The complete lists of hyperparameters used for these tasks are provided in Table 13 through Table 15.

D Hyperparameter Sensitivity and Robustness

A potential concern when introducing a new framework is the overhead of hyperparameter tuning for unseen downstream tasks. While DPLoRA involves several configuration variables, the majority of these (e.g., target parameter budget and candidate rank values) are deterministic environmental constraints rather than tunable hyperparameters. In practice, only three hyperparameters require actual tuning: δ (referred to as Stable Layer Bonus in our configuration tables), β (Imp. EMA Decay), and γ (Mom. Penalty Weight).

To demonstrate the robustness of DPLoRA, we conduct a systematic sensitivity analysis on these three key hyperparameters. Our hyperparameter selection follows a stage-wise procedure: we first identify an effective range on a log scale (e.g., $\{0.01, 0.1, 1.0\}$) and then perform fine-grained tuning within that range. Table 16 presents the performance variations on the QNLI and SST-2 datasets around our default configuration ($\delta = 0.05$, $\beta = 0.6$, $\gamma = 0.25$). As shown, our hyperparameters exhibit remarkably low sensitivity. Specifically, the performance varies by only 0.57/0.34 points for $\delta \in [0.05, 0.2]$, 0.06/0.16 points for $\beta \in [0.4, 0.8]$, and 0.16/0.50 points for $\gamma \in [0.1, 0.5]$ on QNLI and SST-2, respectively.

Furthermore, to confirm that a hyperparameter search is not strictly mandatory for new tasks, we evaluated DPLoRA ($p = 0.4$) using only the fixed default values ($\beta = 0.6$, $\gamma = 0.25$, $\delta = 0.05$) across the entire GLUE benchmark. As reported in Table 17, DPLoRA achieves an average score of 85.7 without any task-specific tuning. This result remains highly competitive with the heavily tuned setting (86.1, as shown in Table 1) and outpaces many baseline methods, confirming that our coarse default values are highly robust and significantly mitigate the difficulty of parameter tuning for practical deployments.

E Stability Analysis of Layer Importance Estimation

To investigate the stability of the initial layer importance estimation and the impact of the evaluation sample size, we compared the importance scores computed using our default setting of 5 evaluation

Table 8: GLUE benchmark results (mean ± standard deviation). All results are averaged over three random seeds.

Method	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
Full 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 ($r = 8$)	63.42±0.63	87.24±0.14	87.75±0.42	92.53±0.16	90.48±0.03	76.29±1.46	94.38±0.61	90.66±0.32
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	63.74±1.85	87.50±0.05	89.79±0.75	92.98±0.06	90.90±0.03	77.62±1.88	94.80±0.35	89.97±0.35
DPLoRA ($p = 0.4$)	63.59±1.13	87.48±0.13	89.22±0.85	93.01±0.21	90.85±0.05	80.02±0.91	94.61±0.57	90.17±0.20
DPLoRA ($p = 0.5$)	61.35±1.98	87.18±0.15	89.46±0.85	92.78±0.37	90.69±0.06	79.30±0.83	93.96±0.65	90.01±0.24
DPLoRA ($p = 0.6$)	61.19±0.76	86.81±0.02	89.46±0.74	92.82±0.13	90.47±0.17	78.34±0.36	94.65±0.24	89.77±0.29
DPLoRA ($p = 0.7$)	59.48±1.39	86.42±0.03	89.62±0.99	92.77±0.36	90.12±0.05	77.14±1.27	93.54±0.24	89.22±0.61
DPLoRA ($p = 0.8$)	60.51±0.83	85.45±0.19	89.05±2.00	92.25±0.50	89.19±0.31	77.62±0.36	94.00±0.43	89.23±0.80

Table 9: Instruction-following performance on MT-BENCH (mean ± std. dev.), evaluated by GPT-4. Higher scores are better. († denotes the same hyper-parameter matching).

Model	Method	#Params (M)	MT-BENCH Score
LLaMA 3.2 1B	LoRA ($r = 8$)†	5.6	3.07 ± 0.07
	DoRA	6.0	2.83 ± 0.09
	OPLoRA (ours)	5.5	3.12 ± 0.04
	DPLoRA ($p = 0.6$, ours)	2.2	2.51 ± 0.01
LLaMA 3.2 3B	LoRA ($r = 8$)†	12.2	4.46 ± 0.06
	DoRA	12.9	4.33 ± 0.18
	OPLoRA (ours)	11.7	4.48 ± 0.10
	DPLoRA ($p = 0.6$, ours)	4.7	4.10 ± 0.04
LLaMA 3 8B	LoRA ($r = 8$)†	21.0	5.13 ± 0.10
	DoRA	22.3	4.97 ± 0.10
	OPLoRA (ours)	20.5	5.20 ± 0.07
	DPLoRA ($p = 0.6$, ours)	8.2	4.97 ± 0.12

mini-batches (up to 500 data samples) against a significantly larger set of 50 evaluation mini-batches.

Table 18 presents the comparison results on the MNLI and QQP datasets across three random seeds. We report the Pearson correlation coefficient between the scores, the rank retention rate (i.e., the proportion of layers that remain active, $r > 0$), and the exact match rate of the assigned ranks.

The results demonstrate that varying the evaluation sample size leads to only marginal differences, confirming that our estimation process is highly stable. The Pearson correlation is extremely high, averaging 0.996 for MNLI and 0.998 for QQP. Furthermore, the average rank retention rates are 75.2% and 83.9%, respectively. It is also worth noting that different datasets yield distinct importance distributions (e.g., varying exact match and retention rates). This reflects an intended adaptation to task-specific characteristics rather than estimation instability.

F Robustness of OPLoRA (Pruning Stage 1) to Out-of-Distribution Tasks

When downstream tasks exhibit substantial distributional divergence from the pre-training corpus, there is a concern that Stage 1 might excessively prune critical layers. Our framework structurally prevents this irrecoverable pruning through two mechanisms.

First, layer importance is evaluated using squared gradients on the target data (Equation 1). Consequently, layers requiring greater domain adaptation naturally yield larger squared gradient values and are proportionally allocated higher initial ranks by the ILP.

Second, the ILP enforces a non-zero aggregate rank per layer type (Equation 5, constraint 3), serving as a strict safeguard against the structural collapse of essential components.

While a dedicated evaluation under out-of-distribution scenarios (e.g., cross-lingual or cross-modal fine-tuning) falls outside the scope of this paper, it represents a valuable direction for subsequent research.

G Expandability to Diverse Model Families

Our proposed DPLoRA framework fundamentally optimizes the rank allocation of standard LoRA modules. Therefore, it is theoretically applicable to any neural network architecture where LoRA can be integrated.

To verify this architectural generalizability beyond the LLaMA 3 family presented in the main text, we conducted a preliminary experiment using the Qwen2.5-7B model. Specifically, we applied DPLoRA ($p = 0.6$) to fine-tune Qwen2.5-7B on the Alpaca dataset. Under this setting, the framework operated successfully, yielding an MT-BENCH score of 5.04 with only 7.6M trainable parameters.

This preliminary observation confirms that DPLoRA can be seamlessly adapted to diverse large language model families without encounter-

Table 10: DPLoRA hyperparameter configurations across GLUE tasks. The same settings are used for all pruning rates ($p = 0.4, 0.5, 0.6, 0.7, 0.8$).

Setting	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**
LoRA Budget	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M
Batch Size	32	32	32	32	32	32	32	32
Pruning Steps	10	10	10	10	10	10	10	10
Learning Rate	3.7e-04	2.4e-04	4.6e-04	4.8e-04	4.4e-04	2.1e-04	2.8e-04	3.5e-05
Epochs	67	27	57	49	10	63	38	67
Weight Decay	0.196	0.435	0.307	0.283	0.173	0.198	0.482	0.093
Max Grad Norm	1.177	1.573	1.498	0.450	0.652	0.870	0.932	0.670
Max Seq Length	320	128	128	128	128	256	256	128
Warmup Ratio	0.071	0.054	0.038	0.022	0.073	0.111	0.029	0.092
Imp. EMA Decay	0.041	0.202	0.530	0.976	0.396	0.744	0.030	0.979
Mom. Penalty Weight	0.089	0.054	0.028	0.915	0.094	0.083	0.392	0.066
Stable Layer Bonus	0.026	0.056	0.025	0.05	0.012	0.015	0.052	0.033
Recovery Steps	100	500	100	100	100	100	100	100
Ext. Recovery Steps	200	1000	200	200	200	200	200	200

Table 11: OPLoRA hyperparameter configurations across GLUE tasks.

Setting	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**
LoRA Budget	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M	1.26M
Batch Size	32	32	32	32	32	32	32	32
Pruning Steps	10	10	10	10	10	10	10	10
Learning Rate	3.7e-04	2.4e-04	4.6e-04	4.8e-04	4.4e-04	2.1e-04	2.8e-04	3.5e-05
Epochs	67	27	57	49	10	63	38	67
Weight Decay	0.196	0.435	0.307	0.283	0.173	0.198	0.482	0.093
Max Grad Norm	1.177	1.573	1.498	0.450	0.652	0.870	0.932	0.670
Max Seq Length	320	128	128	128	128	256	256	128
Warmup Ratio	0.071	0.054	0.038	0.022	0.073	0.111	0.029	0.092

Table 12: Full Fine-Tuning hyperparameter configurations across GLUE tasks.

Setting	**CoLA**	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**
Batch Size	32	32	32	32	32	32	32	32
Learning Rate	1e-5	2e-5	1e-5	2e-5	2e-5	1e-5	2e-5	1e-5
Epochs	20	5	20	10	5	20	10	20
Weight Decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Seq Length	128	128	128	128	128	128	128	128
Warmup Steps	100	3000	100	1000	2800	100	500	100

Table 13: Hyperparameter configurations on the Alpaca task on LLaMA 3.2 1B.

Setting	**OPLoRA**	**DPLoRA** ($p = 0.6$)	**LoRA** ($r = 8$)
LoRA Budget	5.55M	5.55M	-
Pruning Steps	-	10	-
Imp. EMA Decay	-	0.274	-
Mom. Penalty Weight	-	0.084	-
Stable Layer Bonus	-	0.151	-
Recovery Steps	-	100	-
Ext. Recovery Steps	-	200	-
Shared Hyperparameters			
Batch Size		2	
Learning Rate		2.7e-04	
Epochs		5	
Weight Decay		0.141	
Max Grad Norm		1.374	
Max Seq Length		512	
Warmup Ratio		0.059	
Grad Accumulation		8	
LoRA Dropout		0.207	
Imp. Chunk Size		-	

Table 14: Hyperparameter configurations on the Alpaca task on LLaMA 3.2 3B.

Setting	**OPLoRA**	**DPLoRA** ($p = 0.6$)	**LoRA** ($r = 8$)
LoRA Budget	11.85M	11.85M	-
Pruning Steps	-	10	-
Imp. EMA Decay	-	0.056	-
Mom. Penalty Weight	-	0.605	-
Stable Layer Bonus	-	0.025	-
Recovery Steps	-	100	-
Ext. Recovery Steps	-	200	-
Shared Hyperparameters			
Batch Size		4	
Learning Rate		3e-05	
Epochs		18	
Weight Decay		0.037	
Max Grad Norm		1.184	
Max Seq Length		512	
Warmup Ratio		0.071	
Grad Accumulation		16	
LoRA Dropout		0.252	
Imp. Chunk Size		2	

Table 15: Hyperparameter configurations on the Alpaca task on LLaMA 3 8B.

Setting	**OPLoRA**	**DPLoRA** ($p = 0.6$)	**LoRA** ($r = 8$)
LoRA Budget	20.50M	20.50M	-
Pruning Steps	-	10	-
Imp. EMA Decay	-	0.147	-
Mom. Penalty Weight	-	0.296	-
Stable Layer Bonus	-	0.07	-
Recovery Steps	-	100	-
Ext. Recovery Steps	-	200	-
Shared Hyperparameters			
Batch Size		1	
Learning Rate		3.4e-05	
Epochs		7	
Weight Decay		0.03	
Max Grad Norm		1.287	
Max Seq Length		512	
Warmup Ratio		0.073	
Grad Accumulation		16	
LoRA Dropout		0.179	
Imp. Chunk Size		1	

Table 16: Hyperparameter sensitivity analysis on QNLI and SST-2. The default values are $\delta = 0.05$, $\beta = 0.6$, and $\gamma = 0.25$.

Parameter	**Value**	**QNLI**	**SST-2**
δ	0.05	93.01	94.76
	0.10	92.44	94.42
	0.20	92.54	94.53
β	0.4	92.45	94.27
	0.6	92.39	94.11
	0.8	92.42	94.23
γ	0.1	92.48	94.42
	0.25	92.39	94.11
	0.5	92.32	94.61

ing architectural bottlenecks. We leave a comprehensive, multi-seed evaluation across varying base models—including Mixture-of-Experts (MoE) architectures—for future work.

H Computing Infrastructure

The experimental evaluation was conducted across two distinct computing environments, each tailored to the specific computational demands of the GLUE and Alpaca benchmarks.

For the GLUE benchmark, all experiments were executed on a unified environment provided by the Backend.AI platform. This system featured a 20-core CPU, 20GB of RAM, and a fractional GPU resource (0.3 FGPU) allocated with 24GB of VRAM. For the instruction-following tasks on Alpaca, we utilized a separate environment equipped with a 4-core CPU, 64GB of RAM, and an NVIDIA L40S GPU (48GB VRAM).

To ensure reproducibility, a standardized software stack was employed across all experiments, based on the PyTorch NGC container (version 24.09). The core components included PyTorch 2.6.0, Python 3.10.12, CUDA 12.6, and the PuLP 3.0.2 optimization library.

I Reproducibility Settings

To ensure the reproducibility of our results, we implemented strict deterministic protocols across all experiments. Specifically, we fixed the Python hash seed to a constant value and initialized consistent random seeds for both PyTorch and NumPy. Additionally, we enabled deterministic algorithms within the CUDA backend. For the memory-intensive Alpaca experiments, we explicitly con-

Table 17: GLUE benchmark performance of DPLoRA ($p = 0.4$) using default hyperparameters ($\beta = 0.6, \gamma = 0.25, \delta = 0.05$) without any task-specific tuning. The results are averaged over three random seeds.

Setting	#Trainable Params(M)	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Avg.
DPLoRA ($p = 0.4$, Default)	0.7	62.3	87.4	88.6	92.4	90.9	79.7	94.2	90.0	85.7

Table 18: Stability of layer importance estimation across different evaluation sample sizes (5 evaluation mini-batches vs. 50 evaluation mini-batches). We report the Pearson correlation, rank retention rate ($r > 0$), and exact rank match rate across three random seeds on the MNLI and QQP datasets.

Task	Seed	Pearson	Retention ($r > 0$)	Exact Match
MNLI	42	0.998	78.4%	72.2%
	777	0.994	69.4%	66.7%
	2025	0.997	77.8%	72.2%
	AVG	**0.996**	**75.2%**	**70.4%**
QQP	42	1.000	91.7%	86.1%
	777	0.993	60.0%	58.3%
	2025	1.000	100.0%	95.8%
	AVG	**0.998**	**83.9%**	**80.1%**

figured the PyTorch CUDA memory allocator to utilize `expandable_segments`, a setting designed to mitigate memory fragmentation and prevent Out-Of-Memory (OOM) errors.

Regarding the ILP optimization, we enforced deterministic behavior in the CBC solver by fixing the random seed, imposing a strict time limit, restricting execution to a single thread, and defining a tight optimality gap. Furthermore, all PyTorch DataLoaders were instantiated with fixed random generators, and we applied custom worker initialization functions to maintain consistency in multi-processing environments.

As detailed in the experimental setup, all results presented in this paper, including the main results and ablation studies, were derived from experiments repeated with three distinct random seeds (42, 777, and 2025). This rigorous setup ensures that our findings are statistically robust and not artifacts of a specific initialization.

J Practical Solvability of ILP-Based Optimization

The Integer Linear Programming (ILP) problem formulated for optimal rank allocation is known to be NP-hard, theoretically exhibiting exponential time complexity in the worst-case scenario. However, in practice, the specific structure of our problem facilitates efficient and reliable solutions due to several key factors.

First, the decision space is inherently restricted, as the set of candidate rank values is finite and limited. Second, the constraint matrix is predominantly sparse, with the global parameter budget serving as the primary coupling constraint. Third, we leverage the CBC solver (via PuLP), which utilizes highly optimized branch-and-cut algorithms tailored for such structured combinatorial problems.

Although we imposed conservative time limits (e.g., 600 seconds) as a safeguard against worst-case behavior, the solver consistently converged well within these bounds during our experiments. As evidenced by the empirical results in Table 7, the actual computational overhead remains negligible, demonstrating the practical efficiency of this approach.

K Robustness Analysis across Deployment Scenarios

To evaluate the practical viability of DPLoRA, we analyze its behavior under three distinct scenarios representing real-world deployment constraints: (1) extreme memory constraints (Edge AI), (2) data scarcity (Specialized Domains), and (3) architectural diversity (Service Scalability).

Scenario A: Extreme Compression for Edge Deployment. We simulate a deployment scenario on memory-constrained edge devices by pushing the pruning rate to an extreme limit ($p = 0.8$). Figure 4 illustrates the trade-off in this regime. In the High Performance Zone ($p \leq 0.5$), DPLoRA ($p = 0.4$) achieves peak performance, significantly surpassing the LoRA baseline (dashed line) and Adapter. It also outperforms the AdaLoRA+ baseline (86.1 vs. 85.5; see Table 1 for details) while utilizing fewer trainable parameters (0.7M vs. 0.8M). Crucially, in the Extreme Efficiency Zone ($p \geq 0.7$), the model operates with fewer parameters (≈ 0.2M) than the standard AdaLoRA (≈ 0.3M) while exhibiting graceful degradation (84.7). This demonstrates DPLoRA's robust Pareto efficiency across various parameter budgets and its suitability for

on-device applications where storage efficiency is prioritized over absolute peak performance.

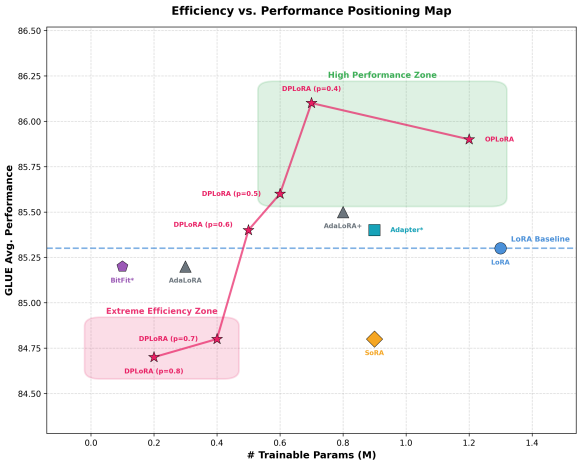

Figure 4: Efficiency vs. Performance Positioning Map. The plot identifies a High Performance Zone (green) for superior scores and an Extreme Efficiency Zone (pink) for extreme compression. DPLoRA demonstrates a continuous trajectory connecting both regions, offering a versatile trade-off compared to static baselines.

Scenario B: Data Scarcity Impact. We investigate the model's robustness in low-resource environments (e.g., specialized domains), where overfitting is a prevalent challenge. We categorize GLUE tasks into High-Resource (e.g., MNLI) and Low-Resource (e.g., RTE, MRPC) groups. Figure 5 reports the performance retention rate relative to standard LoRA ($r = 8$, represented as 100%). While performance is comparable across all methods in the High-Resource regime, a significant divergence emerges in the Low-Resource regime. Baselines such as SoRA suffer noticeable degradation ($\approx 98\%$), whereas DPLoRA ($p = 0.4$) not only remains stable but achieves a substantial performance gain, establishing a Robustness Gap of +1.8%. This indicates that DPLoRA effectively generalizes even with limited training data, making it a reliable solution for specialized domains where large-scale datasets are unavailable.

Scenario C: Architecture Scalability for Service Expansion. Finally, we simulate a service expansion scenario where a single compression framework must adapt to vastly different model scales, ranging from lightweight classifiers to massive generative agents. We compare the performance retention rates on an encoder-only model (RoBERTa-base) and a family of decoder-only LLMs (LLaMA 3.2 1B/3B and LLaMA 3 8B). As shown in Figure 6, OPLoRA consistently outperforms the standard LoRA baseline (red dashed line) across all architectures, peaking at 101.6% on LLaMA 3.2 1B. DPLoRA ($p = 0.6$) also demonstrates robust adaptability by maintaining 100.1% retention on the encoder-only RoBERTa-base. While DPLoRA experiences a performance drop on the smaller decoder model (81.8% on LLaMA 3.2 1B), its retention rate improves significantly as the model scales up, recovering to 91.9% on LLaMA 3.2 3B and reaching a highly competitive 96.9% on LLaMA 3 8B. This trend suggests that larger LLMs possess higher parameter redundancy, making them more resilient to our aggressive progressive pruning strategy. These results confirm that our framework is highly architecture-agnostic and provides a scalable compression solution for diverse deployment environments.

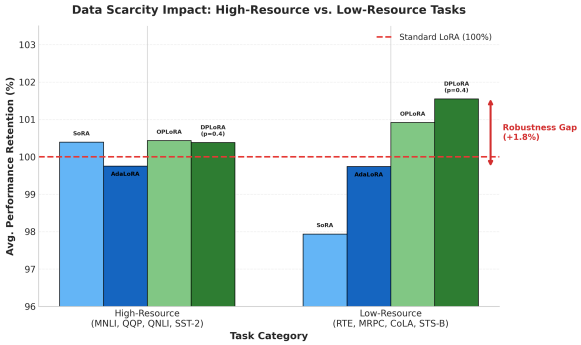

Figure 5: Data Scarcity Impact Analysis. We compare performance retention rates across task categories. While performance is similar in high-resource tasks, DPLoRA demonstrates superior robustness in low-resource settings, significantly outperforming baselines by preventing overfitting.

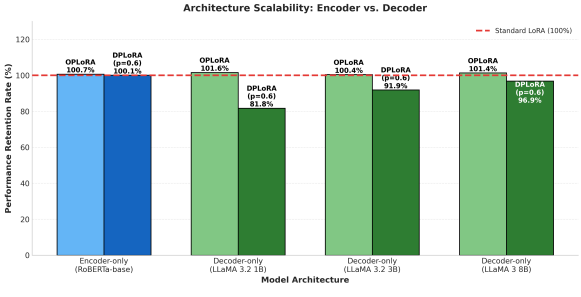

Figure 6: Architecture Scalability Analysis. Comparison of performance retention rates across Encoder-only (RoBERTa) and Decoder-only (LLaMA 3 family) models. The results demonstrate that our framework achieves performance comparable to or surpassing the baseline (100%), proving its effectiveness for scalable service expansion across diverse architectures.

L Use of AI Assistants

We used AI assistants (ChatGPT, Claude) for grammar polishing and code debugging. All technical contributions, experimental design, and analyses are the authors' own work.