Analysis of Adaptive Low-Rank Adaptation and Quantization Strategies (Optimization for AI AI51101 Final Report)

Jeongmin Ahn
UNIST AIGS, 20255558
jeongmin0209@unist.ac.kr

Gyun Lee
UNIST AIGS, 20255573
gyunini@unist.ac.kr

Abstract

This report investigates optimization strategies for Parameter-Efficient Fine-Tuning (PEFT), specifically focusing on AdaLoRA and its integration with quantization. First, we validate that AdaLoRA outperforms static LoRA by adaptively allocating higher ranks to Feed-Forward Networks (FFN) and top layers. Second, we propose a "Dynamic Rank Scheduling" method to address the limitation of starting with a high budget. However, we identify that this approach suffers from the "Cold Start Problem" and local minima entrapment, yielding suboptimal performance (CoLA MCC 0.6757 vs. 0.69). Finally, we propose and evaluate **QLoRA + AdaLoRA**, combining 4-bit Normal Float (NF4) quantization with adaptive pruning. This method achieves a $\sim 5\times$ reduction in memory usage (351MB $\rightarrow$ 67MB) while maintaining or exceeding the accuracy of full-precision AdaLoRA on GLUE benchmarks. Our project is available here.

1 Introduction (Problem Setup)

Large Language Models (LLMs) require immense computational resources for fine-tuning. While Low-Rank Adaptation (LoRA) (Hu et al., 2021) mitigates this by using fixed-rank decomposition, it treats all modules equally, ignoring the fact that FFNs and upper layers are often more critical for performance. AdaLoRA (Zhang et al., 2023) addresses this via adaptive rank allocation. However, it relies on a pruning-based mechanism that necessitates a high initial rank to explore the parameter space, which incurs extra computational and memory overhead during the warm-up phase."

In this project, we conduct a three-stage analysis:

1. **Replication:** We verify AdaLoRA's claim that adaptive allocation outperforms fixed allocation using DeBERTaV3 on GLUE and NLG tasks.
2. **Method 1 (Dynamic Rank Scheduling):** We propose an "Expand on Plateau" strategy to start with minimal ranks ($r = 2$) and grow capacity dynamically, utilizing Momentum Reset and Variance Grafting to stabilize optimization.
3. **Method 2 (QLoRA + AdaLoRA):** To maximize the memory-performance trade-off, we integrate AdaLoRA's pruning mechanism with QLoRA's 4-bit quantization, Double Quantization, and Paged Optimizers.

2 Methods

2.1 Preliminaries: AdaLoRA

2.1.1 SVD-based Adaptation and Pruning

Standard LoRA limits flexibility by assigning uniform ranks across all layers. AdaLoRA overcomes this by parameterizing the incremental update in the form of Singular Value Decomposition (SVD):

$$W = W^{(0)} + P\Lambda Q \tag{1}$$

where Λ is a diagonal matrix of singular values. To bypass the computational cost of exact SVD, AdaLoRA enforces orthogonality on P and Q via a regularization term $R(P, Q) = \|P^\top P - I\|_F^2 + \|QQ^\top - I\|_F^2$. This allows direct manipulation of Λ to control the effective rank.

2.1.2 Importance-Aware Rank Allocation

The model dynamically prunes less important singular values based on a sensitivity metric. The importance score $S_{k,i}$ aggregates the contributions of the singular value and its vectors:

$$S_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_j s(P_{k,ji}) + \frac{1}{d_2}\sum_j s(Q_{k,ij}) \tag{2}$$

where sensitivity is approximated as $s(w) \approx |w \cdot \nabla_w \mathcal{L}|$. To mitigate gradient noise, AdaLoRA stabi-

lizes this metric using an Exponential Moving Average (EMA) and an uncertainty term $\bar{U}^{(t)}$, defining the final score as $s^{(t)}(w_{ij}) = \bar{I}^{(t)}(w_{ij}) \cdot \bar{U}^{(t)}(w_{ij})$.

2.1.3 Global Budget Scheduling

AdaLoRA employs a global budget scheduler to balance exploration and exploitation. The training follows a specific trajectory: (1) **Warm-up:** starting with a high initial budget ($1.5\times$ target) to explore the search space, (2) **Cubic Decay:** gradually pruning ranks based on importance scores, and (3) **Final Stage:** fixing the remaining budget for stable fine-tuning. This ensures resources are concentrated on critical modules like FFNs and top layers.

2.2 Proposed Method 1: Dynamic Rank Scheduling

To overcome the computational inefficiency of pruning-based methods that require a high initial budget, we proposed "Dynamic Rank Scheduling via Loss Plateau." The core idea is to reverse the pruning approach: start with a minimal rank ($r = 2$) and expand the capacity only when the model struggles to converge.

Algorithm: Expand on Plateau We implemented a scheduler that monitors the validation loss L_{val}. Let r_{eff} be the current effective rank and ϵ be a minimum improvement threshold. If the improvement is insignificant ($L_{best} - L_{val}^{(t)} < \epsilon$) for a patience of P consecutive epochs, we trigger a *Rank Expansion*:

$$r_{eff} \leftarrow \min(r_{eff} + \Delta r, r_{max}) \tag{3}$$

This strategy allows the model to escape local minima by expanding the search space into new dimensions.

Stabilization Techniques Expanding the rank dynamically introduces uninitialized parameters into an ongoing optimization process, which disrupts the AdamW optimizer state. To mitigate gradient explosions and loss spikes, we implemented the following mechanisms based on our code analysis:

- **Zero-Initialization Strategy:** When adding new ranks, we initialize the new rows of matrix A using Kaiming Uniform initialization and the new columns of matrix B to zero.
- **Momentum Reset:** The first moment estimate (momentum, m) for the newly added parameters is reset to 0, as they have no prior gradient history.
- **Variance Grafting:** The second moment estimate (variance, v) determines the adaptive learning rate. Initializing v_{new} to 0 causes excessively large updates. Instead, we graft the optimization history by initializing v_{new} with the *mean variance* of the existing trained parameters:

$$v_{new} \leftarrow \frac{1}{N_{old}} \sum_{i \in \mathcal{P}_{old}} v_i \tag{4}$$

  This synchronizes the learning pace of new parameters with the existing ones, stabilizing the training dynamics.

2.3 Proposed Method 2: QLoRA + AdaLoRA

We combined QLoRA's (Dettmers et al., 2023) memory efficiency with AdaLoRA's adaptive performance. The architecture consists of a 4-bit quantized base model and an AdaLoRA adapter (SVD-based). Key components include:

- **NF4 Quantization:** Utilizing 4-bit Normal Float (NF4), which is information-theoretically optimal for zero-centered normal distributions of weights.
- **Double Quantization:** Quantizing the quantization constants themselves to save additional memory (approx. 0.373 bits per parameter).

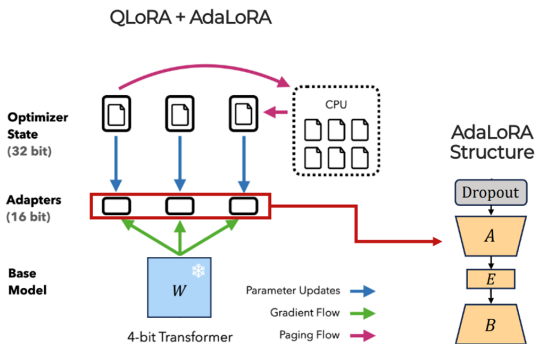


Figure 1: Layer-wise rank distribution of AdaLoRA (Large Budget) with GLUE/MNLI task. The model allocates higher ranks to FFN modules and top layers.

Architecture Overview. Figure 1 illustrates the architecture of our experiment that combines QLoRA and AdaLoRA. Following the QLoRA design, we quantize the base PLM to 4-bit (NF4) while keeping trainable components in higher precision. Specifically, the base transformer weights are stored in 4-bit, whereas the adapter parameters are maintained in 16-bit to preserve training stability. In addition, optimizer states remain in 32-bit precision, and we employ paged optimizers, mitigating out-of-memory (OOM) issues.

Applying AdaLoRA on Adapters. On top of the 4-bit quantized base model, we apply AdaLoRA specifically to the adapter modules. In our setting, we approximate the SVD of adapter updates and conduct *singular value pruning* to remove less important directions. This enables the model to reallocate a limited rank budget toward more influential components, effectively concentrating capacity on the most important ranks while keeping the overall trainable footprint small.

3 Experiments

3.1 Experimental Setup

Datasets and Models We utilized **DeBERTaV3-base** as the backbone model. For evaluation, we selected four representative tasks from the **GLUE benchmark**: MNLI, SST-2, CoLA, and QNLI. Additionally, we used the **CNN/DailyMail** dataset to evaluate generation capabilities (NLG).

Baselines We compared our proposed methods against three strong baselines:

- **LoRA:** Standard low-rank adaptation. To ensure a fair comparison, we evaluated LoRA under two budget settings: *Small* ($r = 2$) and *Large* ($r = 8$).
- **AdaLoRA:** Adaptive rank allocation with SVD-based pruning, also evaluated under Small (Target $r = 2$) and Large (Target $r = 8$) budget constraints.
- **QLoRA:** 4-bit quantized adaptation using NF4 data type.

Implementation Details All experiments were conducted using the **Hugging Face PEFT library** (Mangrulkar et al., 2022) library. Since optimal settings vary by dataset size and complexity, we applied **task-specific hyperparameters** as follows:

- **SST-2:** Learning rate 6×10^{-5}, Batch size 8, Epochs 16.
- **MNLI:** Learning rate 8×10^{-4} (AdaLoRA) / 1×10^{-4} (LoRA), Batch size 32, Epochs 6.
- **CoLA:** Learning rate 5×10^{-4}, Batch size 32, Epochs 25.
- **QNLI:** Learning rate 1.2×10^{-3}, Batch size 32, Epochs 5.
- **CNN/DailyMail:** Learning rate 5×10^{-4}, Batch size 32, Epochs 15.

Common settings included the AdamW optimizer and a linear learning rate scheduler with a 10% warmup. For AdaLoRA, we set the initial rank $r_{init} = 12$, orthogonality regularization weight $\gamma = 0.1$, and update interval $\Delta T = 100$ steps.

Rank Visualization Methodology To analyze how AdaLoRA allocates the parameter budget, we implemented a visualization tool (`visualize_rank.py`). This tool loads the saved model checkpoints and extracts the `rank_pattern` dictionary, which records the final singular value count for each module. We then mapped these values to their corresponding layers and module types (e.g., W_q, W_v, W_{f1}) to generate **layer-wise rank distribution heatmaps**. This allows us to empirically verify whether the model focuses on critical components like FFNs or top layers.

Metrics We report accuracy for MNLI, SST-2, and QNLI; Matthew's Correlation Coefficient (MCC) for CoLA; and ROUGE-1/2/L scores for the CNN/DailyMail text summarization task.

4 Results and Analysis

4.1 Comparative Performance Analysis

We first evaluate the performance of AdaLoRA against the fixed-rank LoRA baseline across varying budget constraints. Table 1 summarizes the results on GLUE (NLU) and CNN/DailyMail (NLG) benchmarks.

NLU Tasks (GLUE) As shown in Table 1, AdaLoRA consistently outperforms LoRA in most settings. A striking improvement is observed in the **QNLI** task under the Small Budget setting, where AdaLoRA achieves an accuracy of **94.56%**, significantly surpassing LoRA's 88.56%. This suggests

Task		Small Budget		Large Budget	
Dataset	Metric	**LoRA** ($r = 2$)	**AdaLoRA** (Tgt $r = 2$)	**LoRA** ($r = 8$)	**AdaLoRA** (Tgt $r = 8$)
MNLI	Accuracy	0.9010	**0.9049**	0.9045	**0.9062**
SST-2	Accuracy	0.9541	**0.9599**	0.9576	**0.9610**
CoLA	MCC	68.71	**69.15**	69.53	**69.92**
QNLI	Accuracy	0.8856	**0.9456**	0.9387	**0.9460**
CNN/DailyMail	ROUGE-1	44.2432	**44.3125**	44.3312	**44.5662**
	ROUGE-2	21.2376	**21.3367**	21.3691	**21.3989**
	ROUGE-L	30.7821	**31.0128**	31.1024	31.0752

Table 1: Comparative results of LoRA and AdaLoRA on GLUE and CNN/DailyMail benchmarks. AdaLoRA outperforms LoRA in 7 out of 8 settings, showing particular robustness in low-budget regimes.

that adaptive rank allocation is crucial when the parameter budget is extremely limited ($r \approx 2$), allowing the model to effectively distribute capacity where it is most needed.

NLG Task (CNN/DailyMail) For the text summarization task, AdaLoRA demonstrates superior generation quality. It achieves consistently higher scores across ROUGE-1, ROUGE-2, and ROUGE-L metrics compared to the baseline.

4.2 Analysis of Rank Allocation Dynamics

To understand how AdaLoRA achieves efficiency, we tracked the dynamic evolution of singular value allocation by saving model checkpoints at every epoch.

Concentration on Critical Modules Our heatmap visualizations (Figure2)

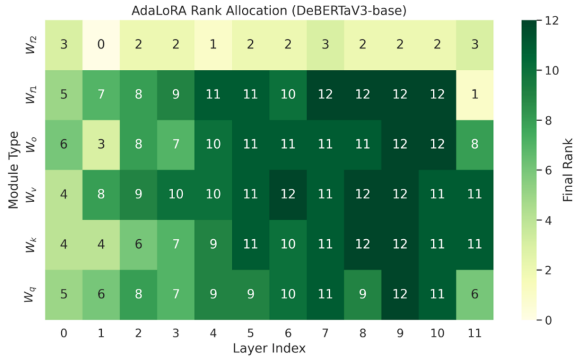


Figure 2: Layer-wise rank distribution of AdaLoRA (Large Budget) with GLUE/MNLI task. The model allocates higher ranks to FFN modules and top layers.

reveal that AdaLoRA does not distribute ranks uniformly. Instead, the surviving parameters are strictly concentrated in Feed-Forward Network (FFN) modules and Top layers of the Transformer. This empirically verifies the hypothesis that upper layers and FFNs are more critical for task performance than self-attention modules or lower layers.

Pruning Behavior In the *Large Budget* setting, we observed a gradual pruning process. In contrast, under the *Small Budget* setting (target $r = 2$), the model aggressively pruned most layers to zero rank, preserving capacity only for the most essential components. This selective preservation explains why AdaLoRA maintains high performance even when the average rank is extremely low. **Detailed visualization of rank distributions for each task and budget setting can be found in Appendix A.**

4.3 Failure Analysis: Dynamic Rank Scheduling

Despite the theoretical motivation, our proposed "Dynamic Rank Scheduling" method yielded suboptimal results. On the CoLA task, it achieved an MCC of **0.6757**, which is lower than the static baseline range (0.69–0.70). We identified four primary reasons for this performance gap:

1. **Cold Start Problem:** When rank expansion is triggered, existing parameters are already highly optimized, whereas the newly added ranks start from initialization.

2. **Limitations of Variance Grafting:** Although we applied Variance Grafting to transfer optimizer statistics, it could not achieve perfect synchronization between the mature and new parameters.

3. **Suboptimal Convergence:** Before the expansion trigger, the model likely converged to a suboptimal solution within the restricted subspace ($r = 2$).

4. **Local Minima Trap:** Once the model is trapped in a deep local minimum within the low-rank space, simply adding new dimensions (Δr) proved insufficient to help the model escape and reach the global optimum.

TASK LIST			LoRA	AdaLoRA	QLoRA	QLoRA + AdaLoRA
NLU Task – GLUE	MNLI	base / adapter	351.76 MB / 0.64 MB	351.76 MB / 0.62 MB	67.55 MB / 0.64 MB	67.55 MB / 0.68 MB
		trainable adapter param	334,083	325,083	333,314	357,336
	SST-2	base / adapter	351.76 MB / 0.64 MB	351.76 MB / 0.76 MB	67.55 MB / 0.64 MB	67.55 MB / 0.72 MB
		trainable adapter param	333,314	400,355	333,314	375,768
	CoLA	base / adapter	351.76 MB / 0.64 MB	351.76 MB / 0.77 MB	67.55 MB / 0.64 MB	67.55 MB / 0.78 MB
		trainable adapter param	333,314	404,954	333,314	410,328
	QNLI	base / adapter	351.76 MB / 0.64 MB	351.76 MB / 0.65 MB	67.55 MB / 0.64 MB	67.55 MB / 0.66 MB
		trainable adapter param	333,314	340,442	333,314	345,816

Table 2: Memory usage (base/adapter) and trainable adapter parameters for GLUE tasks.

TASK LIST		Eval Metric	LoRA	AdaLoRA	QLoRA	QLoRA + AdaLoRA
NLU Task – GLUE	MNLI	acc	0.9010	**0.9049**	0.8977	0.9013
	SST-2	acc	0.9529	0.9599	0.9530	**0.9644**
	CoLA	mcc	68.71	0.6915	0.6886	**0.7027**
	QNLI	acc	0.8856	0.9455	0.9422	**0.9464**

Table 3: Accuracy results on GLUE tasks for LoRA, AdaLoRA, QLoRA, and QLoRA+AdaLoRA (Ours).

4.4 Efficacy of QLoRA + AdaLoRA

Table 4 presents the results of integrating quantization with adaptive pruning. The combined method (Ours) achieved comparable or superior performance to full-precision AdaLoRA on 3 out of 4 tasks.

Task	LoRA	AdaLoRA	QLoRA	Ours
MNLI	0.9010	**0.9049**	0.8977	0.9013
SST-2	0.9529	0.9599	0.9530	**0.9644**
CoLA	68.71	0.6915	68.86	**70.27**
QNLI	0.8856	0.9455	0.9422	**0.9464**

Table 4: Performance comparison on GLUE tasks. Ours (QLoRA + AdaLoRA) demonstrates superior performance despite quantization.

Memory Efficiency: The base model size for DeBERTaV3 reduced from **351.76 MB** (FP32/BF16) to **67.55 MB** (4-bit), achieving a $\sim 5\times$ reduction. This confirms that combining quantization with adaptive rank pruning is a viable strategy to recover performance lost during quantization while maintaining extreme memory efficiency.

Experimental Setup. We evaluate four GLUE NLU tasks (MNLI, SST-2, CoLA, QNLI) under a unified training protocol. For each task, we train (i) LoRA, (ii) AdaLoRA, (iii) QLoRA, and (iv) our combined method (QLoRA + AdaLoRA, small).

Heatmap Analysis of Rank Allocation. We first analyze whether quantizing the base PLM changes AdaLoRA's rank allocation behavior.

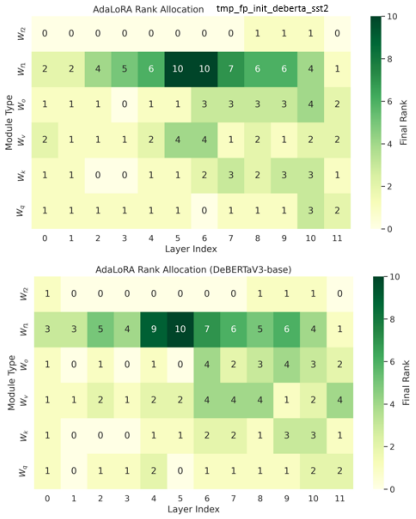


Figure 3: QLoRA rank heatmap.

Figure 3 visualizes the layer-wise rank allocation on SST-2, where the upper heatmap corresponds to our method and the lower heatmap corresponds to full-precision AdaLoRA. The two heatmaps exhibit an almost identical allocation pattern, indicating that applying QLoRA-style 4-bit quantization to the base model does not significantly distort AdaLoRA's adaptive rank redistribution on the adapter side. We observed the same qualitative trend across all four datasets, suggesting that the adaptive pruning mechanism remains stable under base-model quantization.

Memory Cost Measurement. We next quantify the memory reduction achieved by quantization. Compared to the original DeBERTaV3-base footprint (351.76 MB for the base model), QLoRA reduces the base model memory to 67.55 MB, achieving more than a $5\times$ reduction (Table 2). This setting allows us to isolate and directly observe the effect of 4-bit quantization, while keeping the adapter memory overhead small. Notably, our method introduces only a minor increase in adapter memory compared to QLoRA (e.g., 0.66–0.78 MB range depending on the task), while benefiting from AdaLoRA-style rank adaptation.

Accuracy Analysis and Overall Trend. Finally, we report downstream task performance in Table 3. Across all four datasets, AdaLoRA provides a consistent but modest improvement over LoRA, whereas QLoRA occasionally shows a slight drop due to quantization. Importantly, the last column (Ours) demonstrates that combining QLoRA with AdaLoRA can recover or even improve performance: on three out of four tasks, our method matches or exceeds AdaLoRA, despite operating with a 4-bit quantized base model. This supports our hypothesis that adaptive rank allocation and pruning can compensate for the performance loss induced by quantization.

5 Limitation and Takeaways

5.1 Limitations of Dynamic Scheduling

While our proposed "Dynamic Rank Scheduling" aimed to solve the high initial cost of pruning methods, it revealed a fundamental challenge in optimization: the **Cold Start Problem**.

- **Optimization Mismatch:** We observed that newly added parameters (initialized from zero or random) lag significantly behind the highly optimized existing parameters. Even with *Variance Grafting*, the optimizer struggles to synchronize their learning trajectories.
- **Future Direction:** Instead of complex distillation, a more practical solution would be a **"Freeze-and-Thaw"** strategy. Upon rank expansion, we could temporarily freeze the mature parameters and exclusively train the new ranks for a few epochs. Alternatively, applying **Discriminative Learning Rates**—assigning a higher learning rate to new parameters—could accelerate their convergence to match the existing model's state.

5.2 Synergy of Quantization and Adaptation

Our integration of QLoRA and AdaLoRA demonstrated a highly effective synergy.

- **Robustness of Pruning:** We found that AdaLoRA's sensitivity-based pruning remains robust even when the base model is quantized to 4-bit. This suggests that the "importance" of singular values is preserved regardless of weight precision.
- **Practical Impact:** Achieving a $\sim 5\times$ memory reduction (351MB $\rightarrow$ 67MB) while maintaining performance comparable to full-precision AdaLoRA offers a practical "sweet spot" for deploying LLMs on resource-constrained edge devices.

5.3 Conclusion

In this project, we validated that adaptive budget allocation (AdaLoRA) is consistently superior to fixed-rank allocation (LoRA), particularly in low-budget regimes ($r \approx 2$). Although dynamic expansion proved difficult due to optimization instability, our alternative approach of combining **4-bit Quantization with Adaptive Pruning** proved to be a viable and efficient strategy. We conclude that future PEFT research should focus not only on parameter efficiency but also on the **dynamic allocatability** of those parameters to maximize model capacity.

References

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Edward J Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

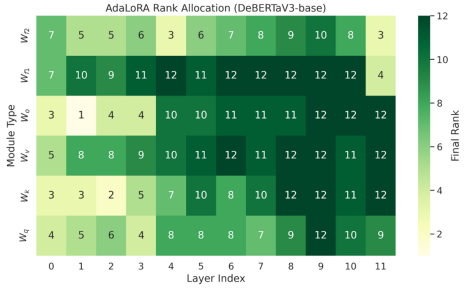
Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Stas Bekman, and Clement Delangue. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

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

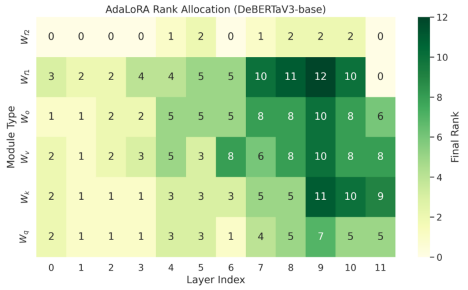
Appendix

A Rank Allocation Visualization

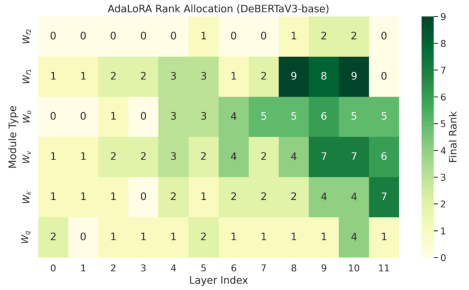
In this section, we present the layer-wise rank distribution heatmaps. These visualizations demonstrate how AdaLoRA adaptively allocates parameter budgets across different layers and modules during the training process. Brighter colors indicate higher rank allocation.



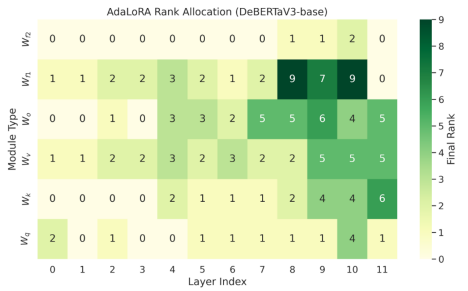
(a) MNLI (Small Budget - Step 1)



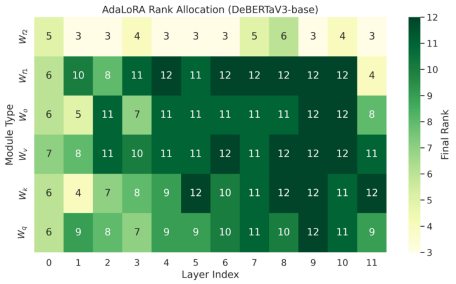
(b) MNLI (Small Budget - Step 2)



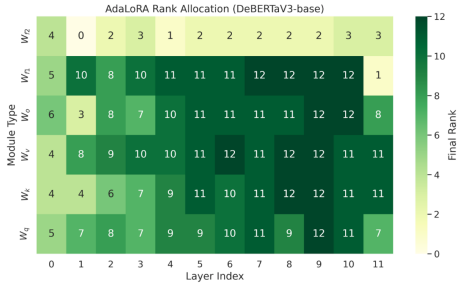
(c) MNLI (Small Budget - Step 3)



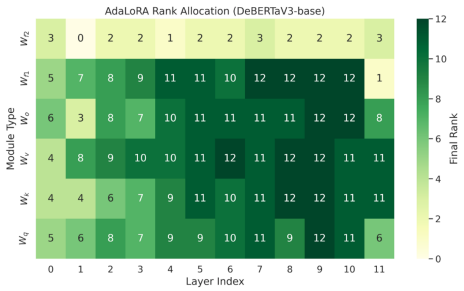
(d) MNLI (Small Budget - Final)



(e) MNLI (Large Budget - Step 1)



(f) MNLI (Large Budget - Step 2)



(g) MNLI (Large Budget - Final)

Figure 4: Rank evolution visualization for MNLI task. (a)-(d) show the progressive pruning in the Small Budget setting, while (e)-(g) show the Large Budget setting. AdaLoRA consistently assigns higher ranks to FFN modules (W_{f1}, W_{f2}) and top layers.