Sensitivity-LoRA : Low-Load Sensitivity-Based Fine-Tuning for Large Language Models

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have transformed both everyday life and scientific research. However, adapting LLMs from general-purpose models to specialized tasks remains challenging, particularly in resource-constrained environments. Low-Rank Adaptation (LoRA), a prominent method within Parameter-Efficient Fine-Tuning (PEFT), has emerged as a promising approach to LLMs by approximating model weight updates using low-rank decomposition. However, LoRA is limited by its uniform rank r allocation to each incremental matrix, and existing rank allocation techniques aimed at addressing this issue remain computationally inefficient, complex, and unstable, hindering practical applications. To address these limitations, we propose **Sensitivity-LoRA**, an efficient fine-tuning method that dynamically allocates ranks to weight matrices based on both their global and local sensitivities. It leverages the second-order derivatives (Hessian Matrix) of the loss function to effectively capture weight sensitivity, enabling optimal rank allocation with minimal computational overhead. Our experimental results have demonstrated robust effectiveness, efficiency and stability of Sensitivity-LoRA across diverse tasks and benchmarks.

1 Introduction

Large language models (LLMs) have become transformative tools across a wide spectrum of tasks and applications (Ding et al., 2022; Qin et al., 2023; Zhu et al., 2023b,a; Li et al., 2023; Zhang et al., 2023a; Huang et al., 2023; Wang et al., 2023). Despite these advancements, fine-tuning remains a critical technique for effectively adapting LLMs from general-purpose models to specialized applications, especially in resource-constrained environments. However, full-parameter fine-tuning can be prohibitively resource-intensive, requiring significant computational power and GPU capacity. To address this limitation, the research community introduced parameter-efficient fine-tuning (PEFT) (Houlsby et al., 2019a; Lester et al., 2021; Li and Liang, 2021; Zaken et al., 2022; Hu et al., 2022a), which aims to balance accuracy and efficiency by selectively updating a subset of model parameters.

LoRA (Hu et al., 2022b), a prominent PEFT method, approximates model weight updates using low-rank decomposition, leveraging the low intrinsic dimension of over-parameterized models (Li et al., 2018; Aghajanyan et al., 2020). During training, the update of the weight matrix (ΔW) can be approximated as the product of two smaller matrices B and A, expressed as:

$$\Delta W \approx B \cdot A \tag{1}$$

where $\Delta W \in R^{d_1 \times d_2}$, $A \in R^{r \times d_2}$ and $B \in R^{d_1 \times r}$ with $r \ll \{d_1, d_2\}$. Thus, it approximates the update of the weight matrix with fewer parameters. However, the full potential of LoRA remains constrained by its inherent design limitations. Specifically, it assumes a uniform rank r for each incremental matrix, not accounting for the varying significance of weight matrices across different modules and layers (Hu et al., 2023; Zhang et al., 2023b).

To address this limitation, dynamic rank allocation has emerged as a key solution by allocating the rank r to each different module or layer according to its specific requirements. Existing methods achieve this through three main approaches: singular value decomposition (SVD), single-rank decomposition (SRD), and rank sampling. SVD-based methods (Zhang et al., 2023c; Hu et al., 2023; Zhang et al., 2023b) decompose matrix BA into an SVD form and selectively truncate the singular values in order to allocate the matrix rank. However, this process is computationally expensive and requires additional memory to store singular values and vectors. SRD-based methods (Mao et al.,

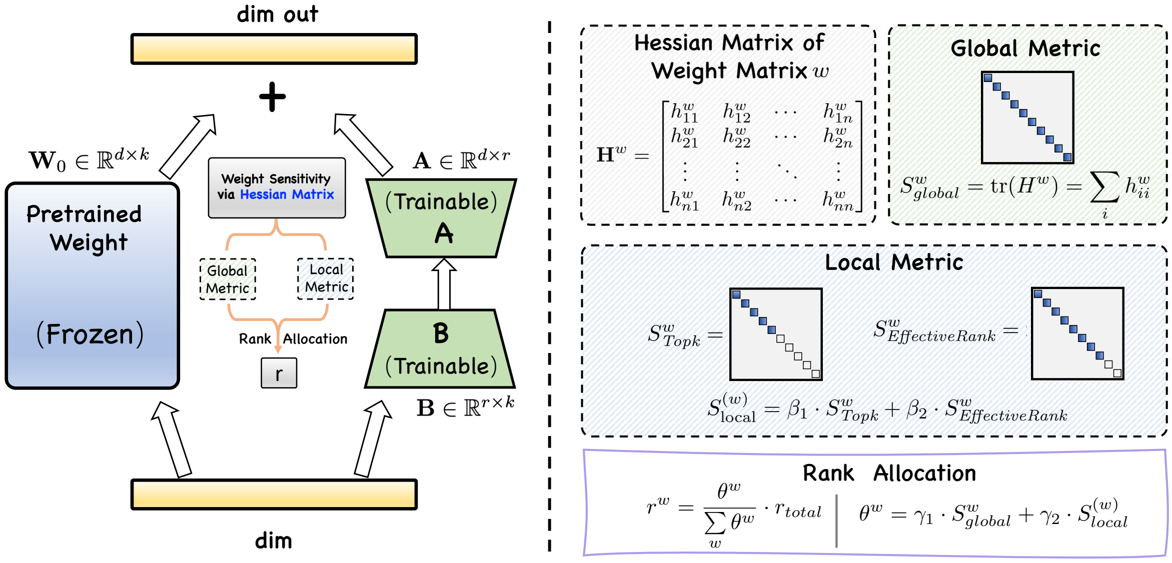

Figure 1: Pipeline of the Sensitivity-LoRA Method: **Step 1 - Sensitivity detection** via Hessian-based metrics, including global and local sensitivity measures. ($h_{ij}^w = \frac{\partial^2 E^w}{\partial w_i \partial w_j}, h_{ii} = \frac{\partial^2 E^w}{\partial w_i^2}$, where E^w denotes the change in loss function regarding weight matrix w; $\mathrm{tr}(H^w)$ denotes the trace of H^w.) **Step 2 - Dynamic rank allocation** based on global and local sensitivity. (r^w denotes the allocated rank of weight matrix w, r_{total} denotes the total rank of all weight matrices in the model.)

2024; Zhang et al., 2024; Liu et al., 2024) decompose matrix BA into single-rank components and allocate the ranks by selecting the proper components. However, optimizing single-rank components and the pruning process can increase computational complexity, potentially offsetting efficiency gains. Rank sampling-based methods (Valipour et al., 2022) allocate ranks directly by random sampling. However, the randomness introduced by sampling could increase training instability.

In order to design a dynamic rank allocation method that introduces extremely low overhead and ensures stability, we propose Sensitivity-LoRA, which can rapidly allocate rank to the weight matrix based on the sensitivity of the parameters, without incurring a significant computational load. Specifically, we utilize the second derivatives of the loss function with respect to the parameters (Hessian matrix) to ascertain the sensitivity of each parameter within the weight matrix. To comprehensively evaluate the sensitivity of the parameter matrix, we employ metrics such as the trace of the Hessian matrix, *Topk* and *Effective Rank* to measure its global and local sensitivities respectively. By integrating various metrics, we determine the rank allocation weights corresponding to the weight matrices to achieve rank allocation. The efficiency, stability, and generality of our approach have been validated through extensive experiments on various tasks, such as sentiment analysis, natural language inference, question answering, and text generation.

In summary, the main contributions of our paper are listed as follows:

- We design a dynamic rank allocation method that introduces minimal overhead and ensures stability.
- We introduce the second derivatives of the loss function with respect to the weight matrix to measure their sensitivity.
- We achieve rank allocation by taking into account both the global and local sensitivity of the weight matrix.
- Extensive experiments demonstrate the effectiveness, stability, and efficiency of our method.

2 Related Work

Existing PEFT approaches can be classified into four main types in terms of memory efficiency, storage efficiency, and inference overhead, as follows:

2.1 Additive PEFT

Additive PEFT introduces lightweight modules into the model architecture, such as adapters and soft prompts, while keeping the pre-trained backbone frozen. Adapters add small networks with down-projection and up-projection matrices, enabling task-specific learning with minimal parameter updates (Houlsby et al., 2019a; Lester et al., 2021). Soft prompts prepend learnable embeddings to the input sequence, allowing fine-tuning by modifying input activations only (Li and Liang, 2021; Zaken et al., 2022). These methods typically require updating less than 1% of the total parameters, significantly reducing computation and memory costs, making them ideal for resource-constrained environments (Hu et al., 2022a).

2.2 Selective PEFT

Selective PEFT fine-tunes a subset of the existing parameters in a pre-trained model, rather than adding new modules. It employs binary masks to identify and update only the most important parameters while keeping the majority frozen. Techniques like Diff pruning and FishMask leverage Fisher information or parameter sensitivity analysis to select critical parameters for fine-tuning (Zaken et al., 2022; Li and Liang, 2021). This approach avoids increasing model complexity and is particularly suited for scenarios where only a small fraction of the model contributes significantly to performance.

2.3 Reparameterized PEFT

Reparameterized PEFT utilizes low-rank parameterization techniques to represent model weights in a reduced form during training. LoRA (Low-Rank Adaptation) is a prominent example, introducing low-rank matrices to fine-tune specific weights while maintaining high inference efficiency (Hu et al., 2022a). Other methods, such as Compacter, use the Kronecker product for parameter reparameterization, further reducing memory requirements and computational costs (Houlsby et al., 2019a). Reparameterized PEFT is highly effective for large-scale models where resource constraints are critical.

2.4 Hybrid PEFT

Hybrid PEFT combines the strengths of Additive, Selective, and Reparameterized PEFT methods into a unified framework. For example, UniPELT integrates LoRA, adapters, and soft prompts, allowing dynamic selection of the most suitable module for specific tasks through gating mechanisms (Zaken et al., 2022). This hybrid approach enhances adaptability and task performance by leveraging the complementary advantages of different PEFT strategies (Li and Liang, 2021; Hu et al., 2022a).

3 Methodology

In this section, we firstly introduce the concept of weight sensitivity with a formal definition of global and local sensitivity metrics of weight matrices. Next, we propose effective allocation strategies to optimize the dynamic rank allocation process based on these sensitivity metrics. The pipeline of our method is presented in Figure 1.

3.1 Weight Sensitivity

Consider a neural network whose dynamics is driven by a collection of parameters w and a loss function E, which guides its learning dynamics. When a small perturbation δw is introduced to the parameters, the resulting change in the loss function can be expressed using a Taylor series expansion up to the second-order term, with higher-order terms captured by $O(\|\delta w\|^3)$ as follows:

$$E(w+\delta w) = E(w) + g^T\delta w + \frac{1}{2}\delta w^T H \delta w + O(\|\delta w\|^3) \quad (2)$$

where g denotes the gradient vector of the loss function E with respect to the parameters w, indicating the rate of change of the loss function in the direction of each parameter. H represents the Hessian matrix of the loss function E, which is a matrix of second-order partial derivatives and contains information about the curvature of the loss function at the current parameter point.

The change in the loss function ΔE can be represented by the following expression:

$$\Delta E = g^\top \delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^\top H \delta\mathbf{w} + O(\|\delta_w\|^3) \quad (3)$$

By expanding the components of ΔE, we have:

$$\Delta E = \sum_i g_i \delta w_i + \frac{1}{2}\sum_{i,j} h_{ij}\delta w_i \delta w_j + O(\|\delta_w\|^3) \quad (4)$$

where g_i and h_{ij} are the gradient and Hessian elements, respectively.

For a well-trained neural network, when the parameter w is located at a local minimum of the loss function, the gradient g becomes zero. Then, the above equation can be simplified to

$$\Delta E = \frac{1}{2}\sum_{i,j} h_{ij}\delta w_i \delta w_j + O(\|\delta_w\|^3) \quad (5)$$

Additionally, several studies have demonstrated that the Hessian matrix H tends to be diagonally dominant, suggesting that the interactions between different parameters can be largely disregarded (LeCun et al., 1989; Dong et al., 2020; Frantar et al., 2022). Then, the above equation can be simplified to

$$\Delta E \approx \frac{1}{2}\sum_i h_{ii}\delta_{w_i}^2 + O(\|\delta_w\|^3) \quad (6)$$

Given that the perturbation in the weights ($\delta\mathbf{w}$) is sufficiently small, the higher-order term becomes negligible compared to the quadratic term, and therefore can be disregarded. Consequently, the above formula can be further simplified to

$$\Delta E \approx \frac{1}{2}\sum_i h_{ii}\delta_{w_i}^2 \quad (7)$$

Consequently, the diagonal elements of the Hessian matrix serve as a reliable indicator of weight sensitivity.

3.2 Rank Allocation Metric

3.2.1 Global Metric

The global sensitivity measurement aims to evaluate the overall impact of an entire parameter (or weight) matrix on model output. It quantifies how variations in this weight matrix affect the loss function. To capture this dynamics, the Hessian matrix, which consists of the second-order partial derivatives of the loss function with respect to the weight matrix, is used. Given that the Hessian matrix tends to be diagonal-dominant at the minimum, its trace can serve as an effective global sensitivity indicator. Formally, the global sensitivity S_{global}^w of weight matrix w can be defined as:

$$S_{global}^w = \mathrm{tr}(H^w) = \sum_i h_{ii}^w \quad (8)$$

where h_{ii}^w is the i-th diagonal element of the Hessian matrix H^w, and $\mathrm{tr}(H^w)$ denotes the trace of H^w. Since the diagonal elements reflect the impact of individual parameter changes on the loss function, a larger trace value indicates that the model is more sensitive to its changes. This suggests more parameters make significant contributions to the changes in the loss function, emphasizing their role in model performance.

3.2.2 Local Metric

While certain weight matrices might have low overall sensitivities, specific weight elements within these matrices can still have high sensitivity, significantly impacting model performance. As such, it is essential to account for local sensitivity to capture finer-grained variations in parameter influence on the loss function. To address this, we introduce two metrics: *Topk* and *Efficient Rank*.

The *Topk* metric approximates local sensitivity of a weight matrix by averaging its largest k diagonal elements of Hessian matrix, based on the assumption that most of the matrix's energy or sensitivity is concentrated in these large values. By focusing on *Top k* diagonal elements, the *Topk* metric can guide us to prioritize these critical weights during weight pruning or optimization processes. It reduces computational complexity while preserving the most impactful weights for model performance. The computation formula for the *Topk* metric of weight matrix w is as follows:

$$S_{Topk}^w = \frac{1}{k}\sum_{i=1}^{k}\lambda_i^w \quad (9)$$

where λ_i^w represents the diagonal elements of Hessian matrix H^w sorted in descending order, and k denotes the number of diagonal elements selected.

The *Effective Rank* metric determines the minimum rank that captures most of the energy of a weight matrix based on the cumulative contribution of the diagonal elements of its Hessian matrix. By establishing a threshold for the cumulative contribution rate (such as 0.9 or 0.95), the *Effective Rank* metric identifies the minimum number of eigenvalues needed to achieve this threshold, thereby appropriately ranking the weight matrix. The key benefit of this metric is ensuring the stability of the rank allocation process. The formula for *Effective Rank* of weight matrix w is as follows:

$$S_{EffectiveRank}^w = \min\left\{k \mid \frac{\sum_{j=1}^{k}\lambda_j^w}{\sum_{j=1}^{m}\lambda_j^w} \geq \alpha\right\} \quad (10)$$

where λ_j^w is the j-th diagonal element of H^w in non-increasing order, m is the total number of diagonal elements, and k is the minimum number of diagonal elements required for the cumulative contribution rate to reach the threshold α.

To ensure the effectiveness and stability, we integrate *Topk* and *Efficitive Rank* metrics together to define the local sensitivity metric S_{local}^w of weight matrix w as follows:

$$\beta_1 = \frac{\sigma^{S_T}}{(\mu^{S_T})^2} \quad \beta_2 = \frac{\sigma^{S_E}}{(\mu^{S_E})^2} \quad (11)$$

$$S^w_{local} = \beta_1 \cdot S^w_{Topk} + \beta_2 \cdot S^w_{EffectiveRank} \tag{12}$$

where σ^{S_T} and σ^{S_E} represent the standard deviations of the *Topk* and *Efficitive Rank* metrics for all weights, while μ^{S_T} and μ^{S_E} denote the corresponding mean values. We utilize the standard deviation of metrics to design allocation weights. The larger the standard deviation of a metric, the more widely its values are distributed, which imply a greater amount of information contained within that metric. The squared average values represent the normalization of the metric scale and the standard deviation. The effectiveness of this design is demonstrated through experiments.

3.3 Rank Allocation Strategy

Taking into account both global and local metrics, we define a refined rank allocation strategy to determine the rank allocation weights θ^w of weight matrix w by integrating global and local sensitivities:

$$\gamma_1 = \frac{\sigma^{S_g}}{(\mu^{S_g})^2} \quad \gamma_2 = \frac{\sigma^{S_l}}{(\mu^{S_l})^2} \tag{13}$$

$$\theta^w = \gamma_1 \cdot S^w_{global} + \gamma_2 \cdot S^w_{local} \tag{14}$$

where σ^{S_g} and σ^{S_l} represent the standard deviations of the global and local metrics for all weights, while μ^{S_g} and μ^{S_l} denote the corresponding mean values. The reason for this design is mentioned in the preceding text. Hence, we can derive the formula for rank allocation as follows:

$$r^w = \frac{\theta^w}{\sum\limits_{w} \theta^w} \cdot r_{total} \tag{15}$$

where r^w denotes the rank allocated to weight matrix w, and r_{total} represents the total rank of all weight matrices in the model.

4 Experiments

4.1 Experimental Setup

Models and Benchmarks. We evaluate the performance of our method across diverse NLG (Natural Language Generation) and NLU (Natural Language Understanding) tasks. For the NLU tasks, we select RoBERTa-base (Liu, 2019) as the base model and evaluate its performance on various subtasks of the GLUE (General Language Understanding Evaluation) benchmark (Wang, 2018): MNLI (Williams et al., 2017), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), CoLA (Warstadt, 2019), QNLI (Rajpurkar et al., 2018), QQP [1], RTE (Wang, 2018) and STS-B (Cer et al., 2017). For the NLG tasks, we conduct experiments using two large language models, Qwen2.5-7B (Yang et al., 2024) and LLaMA3.1-8B (Grattafiori et al., 2024), and evaluate their performance on two representative NLG datasets: Magpie-Pro (Xu et al., 2024) and OpenPlatypus (Lee et al., 2023). We also visualize the global and local rank allocation results for each layer of GPT-2 Large (Radford et al., 2019) and RoBERTa-base (Liu, 2019).

Evaluation Metrics. We report a comprehensive set of standard evaluation metrics. For NLG tasks, we utilize BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004) to assess the quality of generated text. For NLU tasks, we employ the Matthew's correlation coefficient for the CoLA task, the Combined Score for STS-B, and accuracy for the remaining NLU tasks.

Baselines. We adopt several representative methods, including HAdapter (Houlsby et al., 2019b), PAdapter (Pfeiffer et al., 2020), LoRA (Hu et al., 2022b) with uniform rank allocation, AdaLoRA (Zhang et al., 2023c) and DyLoRA (Valipour et al., 2022), as our baselines. More details can be found in the Appendix A.1.

In addition, more implementation details can be found in the Appendix A.2.

4.2 Main Results

We evaluate the effectiveness of Sensitivity-LoRA on NLU tasks by finetuning the RoBERTa-base model across the tasks in the GLUE benchmark. As shown in Table 1, Sensitivity-LoRA demonstrates outstanding performance in a variety of natural language understanding tasks. Specifically, our method achieves the highest average score of 85.94, outperforming all baselines. Sensitivity-LoRA leverages the second order derivatives of the loss function to extract weight wise importance metrics, incorporating both local and global sensitivity. Based on these metrics, it dynamically determines the optimal rank allocation, thereby achieving exceptional performance.

To further assess the effectiveness of our method on NLG tasks, we compare Sensitivity-LoRA against baselines on two diverse datasets: Magpie-Pro and OpenPlatypus, utilizing Qwen2.5-7B and LLaMA3.1-8B. As shown in Table 2, our method consistently outperforms all baselines on evaluation metrics, including BLEU-4, ROUGE-1, and

[1] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

Method	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
HAdapter	86.76	94.03	87.01	57.84	93.19	90.42	78.75	90.91	84.86
PAdapter	86.95	94.11	86.54	57.95	93.37	90.55	79.42	90.97	84.98
LoRA	87.26	93.46	87.08	58.83	92.95	90.50	79.39	91.03	85.06
AdaLoRA	87.32	93.57	87.28	59.00	93.08	90.62	79.56	91.21	85.20
DyLoRA	87.24	93.65	87.28	58.98	93.00	90.57	79.59	91.17	85.19
Sensitivity-LoRA (ours)	**87.58**	**94.59**	**87.73**	**60.20**	**93.62**	**90.74**	**81.81**	**91.27**	**85.94**

Table 1: Performance comparison between baseline methods and the proposed approach on the GLUE benchmark using the RoBERTa-base model. Higher values indicate better performance across all tasks. Bolded values denote the best performance in each task.

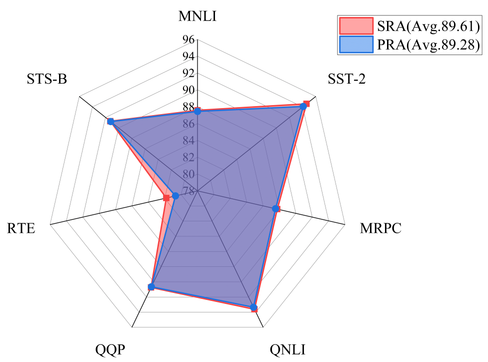

Figure 2: Comparison of the evaluation results for RoBERTa-base finetuned on several datasets from the GLUE benchmark, using both PRA and SRA rank allocation methods.

ROUGE-L. On Qwen2.5-7B, our method achieves the highest average score of 37.98, significantly outperforming others. On LLaMA3.1-8B, it further demonstrates its advantage by attaining an average score of 49.57, surpassing AdaLoRA (48.80), LoRA (48.37), and other adapter based methods. Notably, our method achieves substantial gains in BLEU-4 (71.25) and ROUGE-1 (57.35) on the Magpie-Pro dataset and leads across all three metrics on OpenPlatypus. These results highlight the superior generalization and effectiveness of our sensitivity aware finetuning strategy across various models and generation scenarios.

4.3 Ablation Study

In this section, we present a detailed set of ablation studies to thoroughly evaluate the effectiveness of each component of our method. The evaluation results are summarized in Table 3, where we finetune the RoBERTa-base model on the GLUE benchmark using three different strategies: the proposed global metric S_g, the local metric S_l, and their combination. Our findings indicate that both S_g-LoRA and S_l-LoRA significantly outperform the vanilla LoRA baseline, which employs a uniform rank allocation strategy. This clearly demonstrates that incorporating either global or local sensitivity information can lead to more informed and effective rank assignments. Furthermore, our full method, which integrates both global and local metrics, achieves the highest average score of 85.94. This result underscores the complementary nature of the two types of sensitivity and highlights the benefits of combining both perspectives to guide finetuning. Overall, these results validate the effectiveness of our sensitivity aware rank allocation mechanism and provide strong evidence for the advantages of leveraging both global and local sensitivity information in optimizing model performance.

4.4 Comparison of Rank Allocation Methods

In this section, we compare two rank allocation methods for model weights, based on global and local sensitivity metrics. The Progressive Rank Allocation (PRA) method first sorts the metrics in descending order, subsequently allocating ranks progressively within a specified range. Weights with higher sensitivity are allocated higher ranks. For example, assume there are 6 weights sorted by sensitivity. The average number of r allocated to each matrix is 5, and there are 3 categories in total. The allocation of r for the weights is 6, 6, 5, 5, 4, and 4, respectively. The Scaled Rank Allocation (SRA) method (mentioned in Section 3.3) allocates ranks according to the proportion of each weight's metric relative to the model's total metrics. To visually compare the effectiveness of these two allocation methods, we apply both strategies to the sensitivity metrics and subsequently combine them using the corresponding rank allocation strategy. We then finetune RoBERTa-base on some datasets from the GLUE benchmark. As illustrated in Figure 2, SRA consistently outperforms PRA on various datasets, demonstrating its robustness and superior adaptability. This consistent improve-

Model	Method	Magpie-Pro			OpenPlatypus			Avg.
		BLEU-4	ROUGE-1	ROUGE-L	BLEU-4	ROUGE-1	ROUGE-L	
Qwen2.5-7B	HAdapter	54.71	49.11	32.42	19.39	43.95	22.51	37.01
	PAdapter	54.83	49.15	32.24	19.42	44.03	22.54	37.04
	LoRA	55.03	48.82	32.42	19.72	43.83	22.53	37.06
	AdaLoRA	55.66	49.13	32.75	19.87	44.24	22.67	37.39
	DyLoRA	55.59	49.21	32.82	19.86	44.18	22.59	37.37
	Sensitivity-LoRA (ours)	**56.31**	**50.04**	**33.57**	**20.13**	**44.77**	**23.07**	**37.98**
LLaMA3.1-8B	HAdapter	69.28	56.23	41.08	34.73	52.31	35.61	48.20
	PAdapter	69.30	55.05	41.97	34.66	51.47	36.01	48.07
	LoRA	69.67	55.89	41.78	34.64	52.35	35.92	48.37
	AdaLoRA	70.40	56.31	42.17	34.86	52.90	36.15	48.80
	DyLoRA	70.36	56.26	42.16	34.89	52.80	36.20	48.78
	Sensitivity-LoRA (ours)	**71.25**	**57.35**	**43.02**	**35.30**	**53.79**	**36.69**	**49.57**

Table 2: Evaluation results on NLG tasks using Qwen2.5-7B and LLaMA3.1-8B as backbone models. We compare Sensitivity-LoRA with other PEFT baselines on two representative datasets, Magpie-Pro and OpenPlatypus. Metrics reported include BLEU-4, ROUGE-1, and ROUGE-L.

Method	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
LoRA	87.26	93.46	87.08	58.83	92.95	90.50	79.39	91.03	85.06
S_g-LoRA	87.45	94.08	87.51	59.60	93.35	90.68	80.19	91.24	85.51
S_l-LoRA	87.41	94.05	87.49	59.60	93.33	90.64	80.19	91.27	85.50
Ours	**87.58**	**94.59**	**87.73**	**60.20**	**93.62**	**90.74**	**81.81**	**91.27**	**85.94**

Table 3: Ablation study on the GLUE benchmark using the RoBERTa-base model. We compare the performance of different rank allocation strategies: the uniform baseline (LoRA), global sensitivity based allocation (S_g-LoRA), local sensitivity based allocation (S_l-LoRA), and the proposed combined method (Ours).

ment across datasets suggests that the SRA enables more effective allocation decisions, leading to better overall results. Consequently, we adapt the SRA method for rank allocation in this paper.

4.5 Rank Allocation Under Different Metrics

In Figure 3, we present the global and local rank allocation results for GPT-2 Large and RoBERTa-base, utilizing the SRA rank assignment method detailed in Section 3.3. As illustrated, the global sensitivity metric, Hessian Trace, allocates a larger rank budget to the intermediate and deeper layers of the models, with relatively less emphasis on the initial layers. In contrast, the local sensitivity metric, *Topk*, primarily focuses on the middle layers, assigning more ranks to these regions. The *Efficient Rank* approach, however, assigns higher ranks to the initial layers and exhibits a decreasing trend in rank allocation for subsequent layers. Each of these three sensitivity metrics highlights different aspects of the models, demonstrating that relying on a single source of information for decision-making is insufficient. This highlights the necessity of Sensitivity-LoRA, which integrates these diverse information sources to achieve dynamic rank allocation.

4.6 Overhead Analysis

Cost of Obtaining the Hessian Matrix. Computing the Hessian matrix is a complex process, especially for large models. Some methods propose processing the weight matrix by rows, allowing the Hessian matrix to be approximated through operations on activation values (Frantar et al., 2022; Li et al., 2025). Additionally, Cholesky decomposition is employed to enhance computational stability. In essence, we only need to perform forward inference on the model using a calibration set, and the intermediate results can be used to approximate the Hessian matrix. This significantly reduces the computational cost of the Hessian matrix. For example, when using the PIQA (Bisk et al., 2020) dataset as the calibration set for LLaMA3.1-8B, the computation, including both metric calculation and rank allocation, can be completed in just 25.78 seconds. When using only a portion of the dataset, the computation can be finished in under 10 seconds without introducing significant errors. In contrast, other methods, such as AdaLoRA, which determine rank allocation during training, can significantly increase training time, ranging from minutes to hours. Compared to these methods, our approach introduces negligible additional computa-

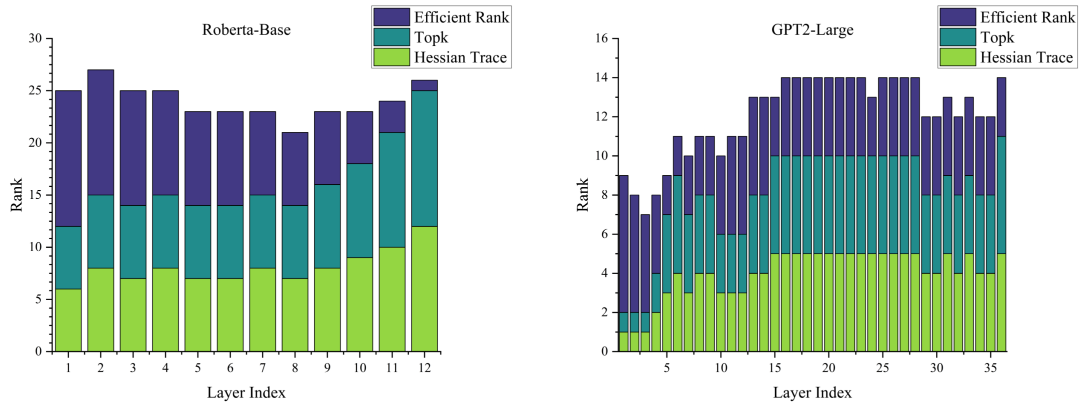

Figure 3: The rank allocation for each layer of GPT-2 Large and RoBERTa-base under different rank allocation metrics. Different colors represent different allocation metrics, and the height of each bar in the histogram corresponds to the rank allocated to that layer by the respective metric.

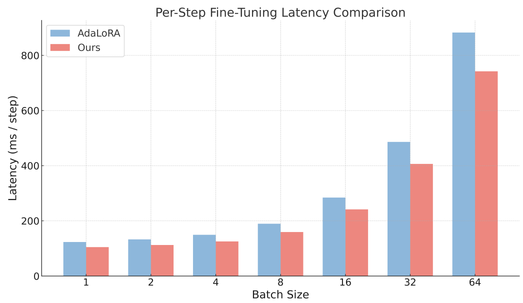

Figure 4: Comparison of per-step fine-tuning latency (ms) between AdaLoRA and our HyperAdaLoRA across varying batch sizes

tional overhead. Additionally, when we calibrate using different calibration sets, such as PIQA (Bisk et al., 2020) and WikiText2 (Merity et al., 2016), we obtain nearly identical results for the Hessian matrix, which further validates the stability of our method.

Memory Analysis. Our method allocates the rank before training, whereas other approaches, such as AdaLoRA, require continuous rank reallocation during training. Specifically, our method has almost exactly the same memory footprint as the conventional LoRA method during training, without introducing any additional overhead. This design not only avoids the extra burden associated with dynamic rank reallocation but also ensures the efficiency and stability of the training process.

Latency Analysis. To assess the training efficiency of our method, we measure its per-step latency and compare it with that of AdaLoRA across various batch sizes, as shown in Figure 4. Our approach consistently exhibits lower latency under all configurations, with the performance gap widening as the batch size increases. This trend indicates superior scalability of our method. The improvement is attributed to our sensitivity-aware rank assignment strategy, which eliminates the runtime overhead associated with AdaLoRA's dynamic scheduling. These results confirm that our method enables more efficient adaptation while significantly reducing computational costs.

4.7 Additional Results

We conduct experiments to validate the effectiveness of the design of the allocation parameters $(\beta_1, \beta_2, \gamma_1, \gamma_2)$. The results demonstrate that combining standard deviation with scale normalization can achieve more effective rank allocation (Appendix B.1). Additionally, we investigate the impact of hyperparameters k and α on the performance of our method. The experiments show that our approach maintains robust performance across various hyperparameter configurations (Appendix B.2). Furthermore, we test our method on specific text examples and obtain favorable results (Appendix C).

5 Conclusion

In this work, we introduce Sensitivity-LoRA, a method that efficiently allocates ranks to weight matrices based on their sensitivity, without a significant computational burden. Sensitivity-LoRA first performs sensitivity utilization by analyzing both global and local sensitivities. It utilizes the second-order derivatives (Hessian matrix) of the loss function to accurately capture parameter sensitivity. Next, it optimizes rank allocation by aggregating global and local sensitivities, ensuring a comprehensive and fair evaluation metric. Extensive experiments consistently demonstrate the efficiency, effectiveness and stability of our method.

6 Limitations

In this paper, we conduct extensive experiments on large language models to validate the effectiveness of our proposed finetuning method. While our findings demonstrate the potential of the method in enhancing model performance, there are still areas that warrant further exploration. Specifically, we do not yet extend our evaluation to large vision models and multimodal large language models, which could provide additional insights into the generalizability and scalability of our approach. Addressing these domains will be a key focus in future work. Additionally, although our method shows promising results in the tested datasets, its robustness under low-resource and domain-specific datasets, such as those involving medical or scientific data, remains to be thoroughly assessed. Exploring these datasets could reveal further nuances in the method's adaptability and potential for specialized applications.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

G. Ding et al. 2022. Efficient fine-tuning for resource-constrained systems. *Proceedings of the Machine Learning Conference*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

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

N. Houlsby, A. Giurgiu, S. Jastrzebski, et al. 2019a. Parameter-efficient transfer learning for nlp. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019:279–285.

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

E. J. Hu, Y. Shen, P. Wallis, et al. 2022a. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations (ICLR)*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022b. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yahao Hu, Yifei Xie, Tianfeng Wang, Man Chen, and Zhisong Pan. 2023. Structure-aware low-rank adaptation for parameter-efficient fine-tuning. *Mathematics*, 11(20):4317.

E. Huang et al. 2023. Evaluating large language models in complex scenarios. *Journal of Computational Linguistics*.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Ariel N Lee, Cole J Hunter, and Nataniel Ruiz. 2023. Platypus: Quick, cheap, and powerful refinement of llms. *arXiv preprint arXiv:2308.07317*.

B. Lester, R. Al-Rfou, and N. Constant. 2021. The power of scale for parameter-efficient prompt tuning. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021:3045–3061.

C. Li et al. 2023. Fine-tuning techniques for efficient model adaptation. *AI Research Journal*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*.

X. Li and P. Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation tasks. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021:4582–4597.

Yuhang Li, Ruokai Yin, Donghyun Lee, Shiting Xiao, and Priyadarshini Panda. 2025. Gptqv2: Efficient finetuning-free quantization for asymmetric calibration. *arXiv preprint arXiv:2504.02692*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. 2024. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. 2024. Dora: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. *arXiv preprint arXiv:2405.17357*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

A. Qin et al. 2023. Advances in state-of-the-art natural language processing. *Journal of NLP Research*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

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

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

F. Wang et al. 2023. Practical applications of llms in specialized domains. *Specialized AI Applications*.

A Warstadt. 2019. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Thomas Wolf. 2020. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

E. Zaken, Y. Goldberg, and S. Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformers. *Transactions of the Association for Computational Linguistics (TACL)*, 10:1–16.

D. Zhang et al. 2023a. Parameter-efficient fine-tuning methods for llms. *Journal of Machine Learning Research*.

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

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

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

B. Zhu et al. 2023a. Expanding frontiers in large language models. *AI Frontier Research*.

B. Zhu et al. 2023b. Large language models: Progress and applications. *Advances in NLP*.

A Experimental Setup

A.1 Baselines

We adopt several representative methods, including HAdapter (Houlsby et al., 2019b), PAdapter (Pfeiffer et al., 2020), LoRA (Hu et al., 2022b) with uniform rank allocation, AdaLoRA (Zhang et al., 2023c) and DyLoRA (Valipour et al., 2022), as our baselines. HAdapter (Houlsby et al., 2019b) and PAdapter (Pfeiffer et al., 2020) are parameter-efficient fine-tuning methods based on adapters. They achieve rapid adaptation to specific tasks by inserting lightweight adapter modules into pre-trained models, eliminating the need to fine-tune the entire model. LoRA (Hu et al., 2022b) approximates parameter updates by adding low-rank decomposition matrices to the weight matrices of pre-trained models, thereby reducing the number of parameters required for fine-tuning. AdaLoRA (Zhang et al., 2023c) dynamically adjusts the rank of low-rank matrices in different model layers to match their varying contributions to model performance. DyLoRA (Valipour et al., 2022) is a dynamic low-rank adaptation technique that sorts the representations learned by adapter modules at different ranks during training and trains LoRA blocks to cover a range of ranks rather than a single rank.

A.2 Implementation Details

Our code is implemented using the PyTorch (Paszke et al., 2019) framework and Transformers (Wolf, 2020) libraries, with all experiments conducted on four NVIDIA A100 GPUs. When we calibrate using different calibration sets, such as PIQA (Bisk et al., 2020) and WikiText2 (Merity et al., 2016), we obtain nearly identical results for the Hessian matrix, which further validates the stability of our method. The details of the approximate Hessian matrix computation can be found in Section 4.6. We designate the local metric S_{Topk} with k set to half of the total number of diagonal elements, and set the parameter α in the *Efficient Rank* metric to 0.85. The values of some parameters $(\beta_1, \beta_2, \gamma_1, \gamma_2)$ follow the settings described in Section 3, and the effectiveness of this design is demonstrated in Section 4. We set the average rank of each matrix to 4 for NLU and 8 for NLG. The comparison methods are required to use a similar number of finetuning parameters. The training is performed using the Adam optimizer with a learning rate of 5×10^{-4}, a batch size of 32 for 10 epochs.

B Parameter Analysis

B.1 Effectiveness of Allocation Parameter

We conduct validation experiments on the allocation parameters $(\beta_1, \beta_2, \gamma_1, \gamma_2)$ to assess the effectiveness of the proposed weighted formula (which is consistently applied to the local sensitivity weight β_1, β_2 and the global local fusion weight γ_1, γ_2, as mentioned in Section 3). Specifically, we compare our design with a method that does not consider the standard deviation of the metrics and instead assigns equal weights to each metric (using $\beta_1 = \frac{0.5}{\mu^{S_T}}, \beta_2 = \frac{0.5}{\mu^{S_E}}, \gamma_1 = \frac{0.5}{\mu^{S_g}}, \gamma_2 = \frac{0.5}{\mu^{S_l}}$). As shown in Table 4, our parameter strategy achieves superior performance across all tasks, with an average score of 85.94 compared to the other of 85.65. The performance improvements are particularly significant in the RTE, CoLA and SST-2 tasks, indicating that our approach has better adaptability to different tasks. These results demonstrate that combining the standard deviation with scale normalization can achieve more expressive and stable sensitivity modeling, thereby enabling more effective rank allocation and overall finetuning performance.

B.2 Hyperparameter Analysis

In this study, we delve into the influence of the hyperparameters k and α on the performance of our method. These hyperparameters are crucial for the computation of the *Topk* and *Effective Rank* metrics, respectively. As illustrated in Table 5, we examine two representative configurations: $k = \frac{N}{3}, \alpha = 0.80$ and $k = \frac{N}{2}, \alpha = 0.85$, where N denotes the total number of diagonal elements in the Hessian matrix. Under both settings, our method demonstrates remarkable consistency, achieving average scores of 85.93 and 85.94, respectively. These strong results highlight the robustness of our sensitivity based rank allocation framework to reasonable variations in the metric configuration. Moreover, they underscore the effectiveness of integrating the *Topk* and *Effective Rank* metrics, which successfully capture salient parameter sensitivities across different thresholds.

C Case Study

Figure 5 presents the performance of the GPT-2 Large and RoBERTa-base models fine-tuned using

Method	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
$0.5/\mu$	87.49	94.32	87.61	59.90	93.47	90.69	80.50	91.18	85.65
Ours	**87.58**	**94.59**	**87.73**	**60.20**	**93.62**	**90.74**	**81.81**	**91.27**	**85.94**

Table 4: Performance comparison of our allocation weight parameter strategy and equal allocation weight parameter. We compare our proposed formulation $\beta_1, \beta_2, \gamma_1, \gamma_2$ (σ/μ^2) with a baseline $\beta_1, \beta_2, \gamma_1, \gamma_2$ $(0.5/\mu)$. Results are reported on the GLUE benchmark using RoBERTa-base.

k	α	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
$N/3$	0.80	87.55	94.52	87.72	**60.21**	93.62	90.70	**81.82**	**91.34**	85.93
$N/2$	0.85	**87.58**	**94.59**	**87.73**	60.20	**93.62**	**90.74**	81.81	91.27	**85.94**

Table 5: Performance comparison of the RoBERTa-base under different hyperparameters (k and α). The k denotes the number of top Hessian diagonal elements used in S^w_{Topk}, and α is the cumulative contribution threshold used in $S^w_{EffectiveRank}$.

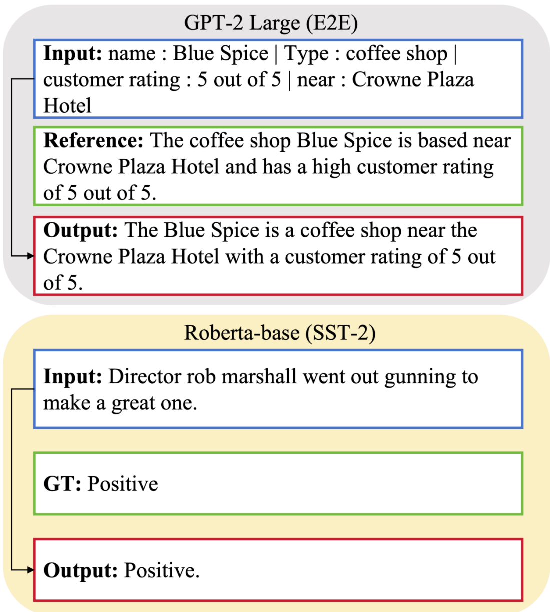

Figure 5: The Case Study of the GPT-2 Large and RoBERTa-base models. The blue boxes represent the input test data, the green boxes indicate the reference text or ground truth output, and the red boxes highlight the model's actual output.

the dynamic rank allocation method (Sensitivity-LoRA) on the E2E and SST-2 datasets. For the E2E dataset, the GPT-2 Large model generates fluent and grammatically correct natural language text that closely aligns with the reference while retaining the input information. This indicates that the model effectively processes structured inputs and excels at generating accurate and coherent natural language descriptions. For the SST-2 dataset, the RoBERTa-base model achieves strong performance in sentiment classification tasks, accurately classifying input text as "Positive." These results demonstrate the effectiveness of the Sensitivity-LoRA method in enhancing model performance on both text generation and classification tasks.