HyperAdaLoRA: Accelerating LoRA Rank Allocation During Training via Hypernetworks without Sacrificing Performance

Anonymous ACL submission

Abstract

Parameter-Efficient Fine-Tuning (PEFT), especially Low-Rank Adaptation (LoRA), has emerged as a promising approach to fine-tuning large language models(LLMs) while reducing computational and memory overhead. However, LoRA assumes a uniform rank *r* for each incremental matrix, not accounting for the varying significance of weight matrices across different modules and layers. AdaLoRA leverages Singular Value Decomposition (SVD) to parameterize updates and employs pruning of singular values to introduce dynamic rank allocation, thereby enhancing adaptability. However, during the training process, it often encounters issues of slow convergence speed and high computational overhead. To address this issue, we propose HyperAdaLoRA, a novel framework that accelerates the convergence of AdaLoRA by leveraging a hypernetwork. Instead of directly optimizing the components of Singular Value Decomposition (P, Λ, Q), HyperAdaLoRA employs a hypernetwork based on attention mechanisms to dynamically generate these parameters. By pruning the outputs of the hypernetwork that generates the singular values, dynamic rank allocation is achieved. Comprehensive experiments on various datasets and models demonstrate that our method achieves faster convergence without sacrificing performance. Moreover, our method generalizes well to other LoRA-based approaches, highlighting its strong generalization capability.

1 Introduction

Parameter-efficient fine-tuning (PEFT) has emerged as a practical solution for adapting large language models (LLMs) to downstream tasks by updating a small subset of parameters, thereby reducing computational and memory overhead (Li and Liang, 2021; Lester et al., 2021; Zaken et al., 2022; Zhang et al., 2025a,b; Houlsby et al., 2019). A prominent PEFT method, Low-Rank Adaptation (LoRA) (Hu et al., 2022), is particularly notable for introducing trainable low-rank matrices into pre-trained weights during fine-tuning, reparameterizing weight updates as:

$$W = W^{(0)} + \Delta W = W^{(0)} + BA \qquad (1)$$

where $W^{(0)}, \Delta W \in \mathbb{R}^{d_1 \times d_2}$, $A \in \mathbb{R}^{r \times d_2}$ and $B \in \mathbb{R}^{d_1 \times r}$ with $r \ll \{d_1, d_2\}$. During fine-tuning, only matrices B and A are updated, substantially reducing the number of trainable parameters. However, LoRA assigns a uniform rank across all layers, neglecting the varying functional importance of different components, potentially limiting performance in deep or heterogeneous models (Hu et al., 2023; Zhang et al., 2023a).

Dynamic rank allocation methods have been introduced to tackle this challenge by adaptively assign different ranks r to various modules or layers. There are three main strategies: 1) Singular value decomposition (SVD) methods (Zhang et al., 2023b; Hu et al., 2023; Zhang et al., 2023a) decompose matrices into singular values and vectors, effectively capturing key components. However, the decomposition process is computationally intensive, with a time complexity of $O(n^3)$, and requires additional memory to store the singular values and vectors. 2) Single-rank decomposition (SRD) methods (Mao et al., 2024; Zhang et al., 2024; Liu et al., 2024b) decompose matrices into rank-1 components, enabling more granular rank allocation. Despite this flexibility, identifying and pruning rank-1 components necessitates multi-stage training, increasing algorithmic complexity. The iterative selection process can also introduce instability, particularly when essential components are mistakenly pruned. 3) Rank sampling methods (Valipour et al., 2022) dynamically allocates ranks during training by sampling from a range of ranks, offering post-training flexibility. However, the stochastic nature of this approach introduces gradient noise, potentially destabilizing convergence.

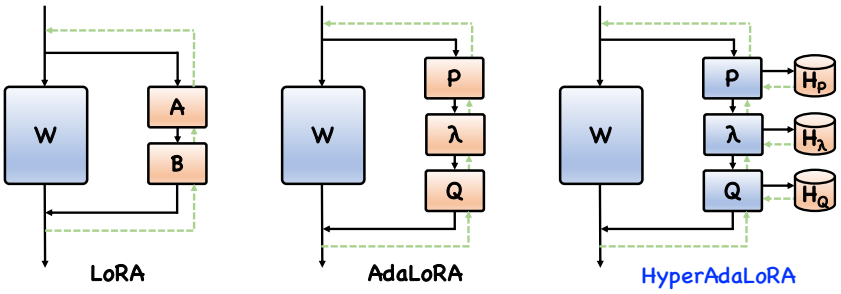


Figure 1: Comparison of LoRA, AdaLoRA, and HyperAdaLoRA frameworks, with black solid lines representing the forward process and green dashed lines indicating backpropagation (gradient flow). LoRA applies fixed-rank low-rank adaptations (A, B). AdaLoRA introduces dynamic rank allocation via Singular Value Decomposition (SVD) and singular value pruning (P, λ, Q). HyperAdaLoRA leverages a hypernetwork to dynamically generate SVD components (H_P, H_λ, H_Q), accelerating convergence and enhancing computational efficiency.

Hypernetworks (Ha et al., 2016) is a meta-model that generates parameters for a target network, decoupling parameter generation from model architecture. Some fine-tuning methods (Ortiz-Barajas et al., 2024; Liang et al., 2022; Li et al., 2025a) leverage hypernetworks to enhance performance on both single-task and multi-task settings. By producing task-specific parameters based on contextual inputs, hypernetworks enable dynamic adaptation without explicit iterative optimization, capturing intricate parameter adaptation patterns in real-time. Hypernetworks facilitate adaptive parameter optimization by modulating both the update direction and magnitude in response to the current model state (Lorraine and Duvenaud, 2018). This dynamic modulation fosters efficient navigation through high-dimensional parameter spaces, accelerating convergence while maintaining model expressiveness (Kirsch et al., 2018; Shi et al., 2022).

In this paper, we introduce HyperAdaLoRA, a novel framework that leverages hypernetworks (Ha et al., 2016) to achieve a significant improvement in convergence speed through parameter generation. Unlike traditional methods that directly train the incremental matrices P, Λ, and Q, HyperAdaLoRA employs task-specific hypernetworks to generate these matrices. Architecturally, our hypernetworks are based on a specific attention layer of BERT (Devlin et al., 2019), which enables them to capture the complex dependencies among parameters. Specifically, each hypernetwork takes the current state of P, Λ, or Q as input and outputs their updated versions. Dynamic rank allocation is realized by pruning the output of the hypernetwork that generates Λ. The training objective of HyperAdaLoRA is to minimize the discrepancy between the parameters generated by the hypernetworks and the ideal parameters. Extensive experiments demonstrate that our method achieves faster convergence while maintaining accuracy. Additionally, further extension experiments on other LoRA-based (SRD and rank-sampling) approaches validate the broad applicability of our method.

The main contributions of our paper can be summarized as follows:

- We introduce HyperAdaLoRA, a pioneering framework that utilizes hypernetworks to achieve substantial acceleration in convergence speed through advanced parameter generation.
- We employ attention based hypernetworks to capture the complex dependencies among parameters and accurately perform parameter updates during the training process.
- We conduct extensive experiments showing that HyperAdaLoRA achieves faster convergence without sacrificing accuracy, and it generalizes well to other LoRA-based approaches, highlighting its strong generalization capability.

2 Related Work

2.1 SVD-based Fine-tuning Method

SVD-Based methods parameterize LoRA's low-rank update matrix ΔW in a singular value decomposition form (e.g. splitting into $P\Lambda Q$) to dynamically adjust effective rank. The mathematical representation is as follows:

$$W = W^{(0)} + \Delta W = W^{(0)} + P\Lambda Q \qquad (2)$$

where $P \in \mathbb{R}^{d_1 \times r}$ and $Q \in \mathbb{R}^{r \times d_2}$ represent the left/right singular vectors, and the diagonal matrix $\Lambda \in \mathbb{R}^{r \times r}$ contains the singular values $\{\lambda_i\}_{1 \leq i \leq r}$ with $r \ll \min(d_1, d_2)$. For example, AdaLoRA (Zhang et al., 2023b) prunes less important singular values based on a sensitivity-derived importance score during training. SaLoRA (Hu et al., 2023) adaptively adjusts the rank by identifying and suppressing less informative singular components, optimizing parameter efficiency across layers. IncreLoRA (Zhang et al., 2023a) incrementally increases the rank during training, starting with a minimal rank and expanding as needed, balancing early training stability with later-stage expressiveness. These methods effectively capture principal components but incur $O(n^3)$ complexity and substantial memory overhead, impacting scalability and training stability, particularly with dynamic rank adjustment.

2.2 SRD-based Fine-tuning Method

SRD-based methods decompose the LoRA update $\Delta W = \sum_{i=1}^{r} u_i v_i^T$ into a series of rank-1 matrices, where each component $u_i v_i^T$ represents a distinct direction in the parameter space. This decomposition allows the model to assess and adjust each rank-1 update independently. AutoLoRA (Zhang et al., 2024) uses a meta-learning scheme to determine which rank-1 slices to retain or prune, while ALoRA (Liu et al., 2024b) trains a 'super-network' and reallocates ranks based on importance. SoRA (Ding et al., 2023) applies sparsity penalties to zero out less impactful components, and DoRA (Liu et al., 2024a) adjusts only the direction component, assigning a learnable scalar for each weight. However, SRD methods face limitations such as increased algorithmic complexity from multi-stage training and additional optimization for rank-1 component selection. Abrupt pruning can destabilize training, while iterative selection adds computational overhead and heightens sensitivity to hyperparameter tuning.

2.3 Rank Sampling-based Fine-tuning Method

Rank-sampling methods treat the LoRA rank as a random variable during training. DyLoRA (Valipour et al., 2022) implements this by sampling a truncation level $b \leq R$ in each iteration, zeroing out the bottom $R - b$ components and enabling the model to operate across multiple ranks without retraining. This approach eliminates the need for exhaustive rank search while also serving as a regularizer, concentrating key features in top components to potentially enhance generalization. However, training across multiple ranks can dilute performance at specific ranks compared to fixed-rank LoRA, and dynamic masking introduces slight computational overhead, potentially requiring additional training epochs to converge. Additionally, we note that some works (Li et al., 2025b,c) also introduce the concepts of non-zero initialization dynamics and token-wise input–output projections.

3 Method

3.1 Preliminary

Hypernetworks (Ha et al., 2016) are a type of neural network architecture used to generate the weights of another neural network (the target network). This can be expressed using the following formula:

$$\Theta = H(C; \Phi) \tag{3}$$

where Θ represents the weights of the target network, H denotes the hypernetwork, C is the context vector input to the hypernetwork and Φ corresponds to the weights of the hypernetwork itself.

The input to the hypernetwork (Ha et al., 2016) can be any information related to the target network, such as the input data of the target network, task requirements, or other contextual information. By conditioning parameter generation on these inputs, the hypernetwork can dynamically generate different parameters to adapt to different tasks or environments. This approach mitigates the need for learning a full set of parameters, thereby reducing model complexity and promoting generalization.

In neural architecture search (NAS), hypernetworks (Ha et al., 2016) can generate parameters for multiple sub-networks, thereby efficiently exploring different network architectures. By simultaneously training the hypernetwork and the sub-networks, the performance of a large number of candidate architectures can be evaluated in a relatively short time. This allows for the rapid screening of network structures with better convergence performance. This efficient architecture exploration method helps to find more optimal model architectures, thereby accelerating the overall training and convergence process. We present a more comprehensive theoretical analysis that elucidates how hypernetworks accelerate convergence (Appendix A).

3.2 Hypernetworks Accelerate Convergence

To accelerate the convergence of AdaLoRA, we propose a novel training strategy: employing a hypernetwork to dynamically generate the $P\Lambda Q$ parameters during training, rather than relying on traditional backpropagation for their updates. Specifically, at the beginning of training, we initialize the $P\Lambda Q$ parameters using a normal distribution. As training progresses, the parameters of the hypernetwork are continuously updated through backpropagation, thereby optimizing the generation process of the $P\Lambda Q$ parameters. In this process, the $P\Lambda Q$ parameters serve merely as intermediate results, with their ultimate goal being the optimized generation via the hypernetwork to achieve faster convergence. The design of the hypernetwork is crucial to this strategy. It takes the parameters before the update as input and, after a series of complex computations and optimization operations, outputs the updated parameters. To conserve computational resources and memory usage, we employ the same hypernetwork for the same parameters across different parameters. For example, we use a single hypernetwork to generate the updated values for the P parameters of all matrix weights. This design not only improves resource efficiency but also ensures consistency and stability in parameter updates. In the i-th iteration, the update process of $P\Lambda Q$ can be specifically represented as follows:

$$P_{i+1} = \mathcal{H}_P(P_i; \Phi_P) \tag{4}$$

$$\Lambda_{i+1} = \mathcal{H}_\Lambda(\Lambda_i; \Phi_\Lambda) \tag{5}$$

$$Q_{i+1} = \mathcal{H}_Q(Q_i; \Phi_Q) \tag{6}$$

where $\mathcal{H}_P$, $\mathcal{H}_\Lambda$, and $\mathcal{H}_Q$ represent the hypernetworks used to update the parameters P, Λ, and Q, respectively. Φ_P, Φ_Λ, and Φ_Q represent the parameters of these hypernetworks. P_i, Λ_i, and Q_i represent the parameters before the i-th iteration update, while P_{i+1}, Λ_{i+1}, and Q_{i+1} represent the parameters after the update.

The process of a hypernetwork generating updated parameters by taking model parameters as input is essentially a dynamic parameter generation process. From a mathematical perspective, this can be likened to a nonlinear transformation within the parameter space, aimed at more efficiently approximating the optimal parameters. From the standpoint of optimization theory, such personalized updates can more effectively explore the parameter space to identify better solutions. Traditional optimization methods, such as gradient descent, follow fixed rules for parameter updates. In contrast, a hypernetwork can dynamically adjust the direction and magnitude of updates based on the current state of the parameters. This flexibility enables it to better adapt to complex loss landscapes and accelerate convergence. As a result, the hypernetwork can more precisely adjust the model's state in each iteration, thereby reducing the number of iterations required to achieve convergence.

During the initial training phases, the hypernetwork designed for Λ generates a full-rank diagonal matrix. To facilitate adaptive budget allocation, we implement an iterative singular value pruning strategy based on magnitude thresholds. At each interval ΔT, the k smallest singular values within Λ are set to zero, effectively reducing the rank of the incremental update Δ. Subsequently, the hypernetworks adjust the patterns they generate to compensate for the pruned dimensions through a gradient-driven process of plasticity.

The loss function integrates task-specific objectives with orthogonality regularization, formulated as:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \gamma(\|P^\top P - I\|_F^2 + \|QQ^\top - I\|_F^2) \tag{7}$$

where $\gamma > 0$ denotes the regularization coefficient. This formulation ensures that the generated matrices P and Q approximate orthogonal transformations while maintaining compatibility with downstream tasks. We additionally provide details on the parameter matrix update process (Appendix C) and the pruning strategy (Appendix D).

3.3 Attention Driven Parameter Interaction

We adopt a BERT layer as the architecture of the hypernetwork and leverage the self-attention mechanism to capture the dependencies among parameters. Specifically, the interaction between the query and key in the attention mechanism mimics the associations between elements in the parameter matrix. This enables the hypernetwork to generate context-aware updates that preserve the structural patterns of the parameters. For any parameter p_i, its updated output takes into account all parameters in the matrix, as shown in the following equation:

$$p_{i+1} = \sum_{j=1}^{N} \text{Softmax}\left(\frac{Q_i K_j^T}{\sqrt{d}}\right) V_j \tag{8}$$

where p_{i+1} represents the updated value of parameter p_i, Q_i is the query vector associated with p_i, K_j

is the key vector associated with parameter p_j, and V_j is the value vector corresponding to parameter p_j. The total number of parameters in the matrix is denoted by N, and d represents the dimensionality of the query and key vectors. This formulation allows the hypernetwork to dynamically compute the updated value of p_i by aggregating information from all parameters in the matrix, capturing their interdependencies and preserving the structural patterns of the parameters.

4 Experiments

4.1 Experimental Setup

4.1.1 Benchmarks

We conduct a comprehensive evaluation of our method, covering a wide range of tasks in both natural language understanding (NLU) and natural language generation (NLG). In the realm of natural language understanding, our method is tested on challenging tasks from the GLUE benchmark (Wang, 2018): RTE (Wang, 2018) and WNLI (Wang, 2018). These tasks represent large scale entailment classification, small-scale binary entailment classification, and coreference resolution presented in the form of binary entailment classification, respectively. In the natural language generation domain, we assess our method using three widely recognized datasets: Stanford Alpaca (Taori et al., 2023), Magpie-Pro-300K-Filtered (Xu et al., 2024), and OpenPlatypus (Lee et al., 2023). In the following text, we sometimes abbreviate Magpie-Pro-300K-Filtered as Magpie.To further demonstrate the generalizability of our approach, we conduct additional evaluations on the GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) benchmarks.

4.1.2 Models

We use two prominent pretrained language models for NLU: RoBERTa-base (Liu, 2019), which is renowned for its strong performance across a wide range of NLU tasks, and DeBERTa-v3-base (He et al., 2021), an enhanced version that incorporates advanced pretraining techniques. For NLG, we employ two models: LLaMA3.1-8B (Grattafiori et al., 2024), a powerful 8 billion parameter model optimized for high-quality text generation, and Qwen2.5-7B (Yang et al., 2024), a model that demonstrates exceptional performance in various NLG tasks. We further evaluate our method on the larger Qwen2.5-14B (Yang et al., 2024).

4.1.3 Baselines

Our primary baseline for comparison is AdaLoRA (Zhang et al., 2023b). AdaLoRA uses $P\Lambda Q$ as trainable parameters that are dynamically updated during training. It allocates parameter budgets by parameterizing updates in an SVD form and prunes singular values based on importance scores during training. To further demonstrate the broad applicability of our method, we additionally conduct experiments by integrating it with LoRA (Hu et al., 2022), DoRA (SRD-based) (Liu et al., 2024a), and DyLoRA (rank-sampling based) (Valipour et al., 2022).

4.1.4 Implementation Details

Our experiments are conducted using the PyTorch framework (Paszke et al., 2019) and the Hugging Face Transformers library (Wolf, 2020), running on a cluster equipped with NVIDIA A100 40GB GPUs. In our experiments, we set the rank r to 3 and the orthogonality regularization coefficient γ to 0.1. We use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1×10^{-5} and a batch size of 64 for training. For the comparative experiments, we keep the hyperparameters for the base model fine-tuning consistent across different methods. We use ROUGE-1 and BLEU-4 as the NLG evaluation metrics. The hypernetwork is implemented as a single layer of TinyBERT, featuring a hidden dimension of 312 and 12 attention heads, with approximately 11.88 million parameters. For the CNN baseline, we adopt the ResNet-18 architecture, comprising 18 layers with hidden dimensions of 64, 128, and 256, totaling around 11.7 million parameters. Additionally, the MLP baseline consists of 15 layers with hidden dimensions of 256, 512, and 1024, amounting to approximately 11.4 million parameters. More training details can be found in Appendix E.

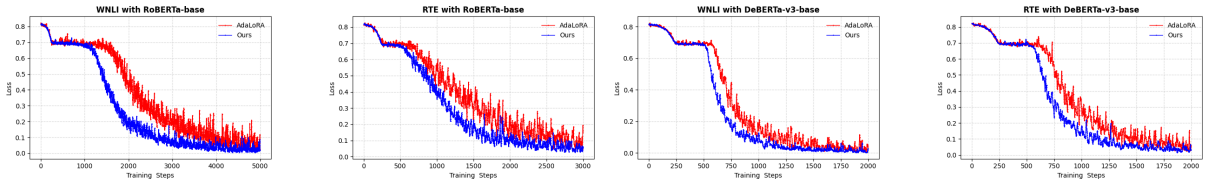
4.2 NLG Task Results

4.2.1 Performance Comparison

We first compare the final generation quality of HyperAdaLoRA and AdaLoRA after fine-tuning the LLaMA3.1-8B and Qwen2.5-7B models on the Stanford Alpaca and Magpie datasets. The results are summarized in Table 1 using BLEU-4 and ROUGE-1 scores. The results in Table 1 indicate that HyperAdaLoRA does not exhibit any performance degradation compared to AdaLoRA. In most configurations, the scores of the two models are very close, with HyperAdaLoRA occasionally

Model	Method	Stanford Alpaca		Magpie	
		BLEU-4	ROUGE-1	BLEU-4	ROUGE-1
LLaMA3.1-8B	AdaLoRA	55.06	58.51	70.69	56.76
	HyperAdaLoRA (ours)	**55.10**	**58.58**	**70.73**	**56.78**
Qwen2.5-7B	AdaLoRA	6.79	20.17	56.21	49.43
	HyperAdaLoRA (ours)	**6.79**	**20.19**	**56.22**	**49.43**

Table 1: Performance comparison between HyperAdaLoRA and AdaLoRA on NLG tasks using LLaMA3.1-8B and Qwen2.5-7B as backbones. The reported metrics include BLEU-4 and ROUGE-1.



(a) WNLI/RoBERTa-base (b) RTE/RoBERTa-base (c) WNLI/DeBERTa-v3-base (d) RTE/DeBERTa-v3-base

Figure 2: Comparison of training loss convergence between HyperAdaLoRA and AdaLoRA on natural language understanding tasks. The rows correspond to three natural language understanding tasks: RTE and WNLI. The columns represent two pretrained language models: RoBERTa-base and DeBERTa-v3-base.

showing a slight edge. This demonstrates that the significant improvements in convergence speed do not negatively affect the final quality of the generated outputs.

4.2.2 Training Efficiency

We further evaluate the effectiveness of HyperAdaLoRA in NLG tasks. We finetune the LLaMA3.1-8B and Qwen2.5-7B models on three instruction-following datasets: Stanford Alpaca, Magpie-Pro-300K-Filtered, and OpenPlatypus. Table 6 presents a comparison of the total training time. In all configurations, HyperAdaLoRA achieves shorter training times than AdaLoRA. This reduction is evident across datasets of varying sizes, from the large Magpie to the smaller Alpaca and OpenPlatypus datasets, consistent with the accelerated convergence. For instance, when fine-tuning LLaMA3.1-8B on the Stanford Alpaca dataset, HyperAdaLoRA takes 7250 seconds, compared to 8125 seconds for AdaLoRA. Similarly, when fine-tuning Qwen2.5-7B on the large Magpie-Pro dataset, HyperAdaLoRA has a training time of 14250 seconds, while AdaLoRA requires 15000 seconds. This consistent time advantage highlights the efficiency gains brought by hypernetwork based parameter generation. By more rapidly reaching effective parameter states, HyperAdaLoRA significantly reduces the total training duration needed for adaptation to these NLG tasks.

4.3 NLU Task Results

We compare the convergence speed of HyperAdaLoRA (which employs a BERT layered hypernetwork) with that of the baseline AdaLoRA. Figure 2 illustrates the training loss curves of these two methods on the RTE and WNLI datasets, using RoBERTa-base and DeBERTa-v3-base as backbone models. Across all experimental settings, HyperAdaLoRA consistently converges significantly faster than AdaLoRA. The loss curves of HyperAdaLoRA drop more steeply in the early stages of training and reach a lower loss plateau earlier in the training process. Our method achieves convergence with fewer training steps. As shown in Table 7, it also incurs lower per-step latency, further demonstrating its advantage in fine-tuning efficiency.

4.4 Analysis of Method Generalizability

4.4.1 Convergence Time on Other LoRA-Based Methods

To demonstrate this generality, we extend it beyond AdaLoRA to several representative methods, including LoRA, DoRA and DyLoRA. We evaluate these variants on LLaMA3.1-8B with a batch size of 8 using the Stanford Alpaca dataset. As shown in Table 2, our method consistently accelerates convergence across all settings. We provide the details of hypernetworks for LoRA, DoRA, and DyLoRA in Appendix B. Additional results are provided in Appendix G.

Method	LoRA		DoRA		DyLoRA	
	w/o Hyper	w/ Hyper	w/o Hyper	w/ Hyper	w/o Hyper	w/ Hyper
Training Time (s)	6893	**5958**	7525	**6267**	7129	**6132**

Table 2: Training time comparison on LLaMA3.1-8B with the Stanford Alpaca dataset.

Dataset	LoRA		AdaLoRA		DoRA		DyLoRA	
	w/o Hyper	w/ Hyper	w/o Hyper	w/ Hyper	w/o Hyper	w/ Hyper	w/o Hyper	w/ Hyper
GSM8K	71.19	71.21	72.38	72.54	72.45	72.46	72.52	72.53
HumanEval	42.26	42.27	43.54	43.56	43.51	43.53	43.43	43.45

Table 3: Performance comparison of different methods on LLaMA3.1-8B (GSM8K and HumanEval).

4.4.2 Performance on Reasoning Benchmarks

We conduct a comprehensive evaluation of various methods on the LLaMA3.1-8B model using two additional benchmark datasets, GSM8K and HumanEval. As demonstrated in the Table 3, our approach does not result in any performance degradation on these complex tasks.

4.4.3 Convergence Time on Larger Models

We further evaluate our method on the larger Qwen2.5-14B model with training load benchmarks on the Stanford Alpaca and OpenPlatypus datasets. As shown in the Table 4, our method achieves significantly faster convergence compared to the baselines.

Method	Stanford Alpaca	OpenPlatypus
AdaLoRA	15102	24857
Ours	**11553**	**18553**

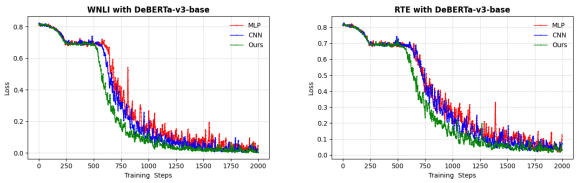
Table 4: Comparison of Training Time (s) on Qwen2.5-14B.

4.5 Hyperparameter Impact Analysis

We investigate the sensitivity of HyperAdaLoRA's NLG performance to the orthogonality regularization coefficient γ. We finetune LLaMA3.1-8B with γ values set to $\{0.1, 0.15, 0.2\}$. As shown in Table 5, performance remains relatively stable across the tested γ values. Although $\gamma = 0.2$ yields slightly better results in this specific setup, the differences are minimal. This indicates that HyperAdaLoRA is robust to variations in this hyperparameter, thereby simplifying its practical application.

γ	**Stanford Alpaca**		**Magpie**	
	BLEU-4	ROUGE-1	BLEU-4	ROUGE-1
0.10	55.10	58.58	70.73	56.78
0.15	55.06	58.42	70.70	56.68
0.20	55.12	58.30	70.72	56.78

Table 5: Performance of HyperAdaLoRA with different values of γ. We conduct experiments using the LLaMA3.1-8B model on the Stanford Alpaca and Magpie-Pro-300K-Filtered datasets.



(a) WNLI (b) RTE

Figure 3: Comparison of training loss convergence for different HyperAdaLoRA hypernetwork architectures (MLP, CNN, BERT layer) on the DeBERTa-v3-base model for NLU tasks. The analysis is conducted across three natural language understanding tasks.

4.6 Ablation Study

To demonstrate the contributions of our hypernetwork design, we compare the performance of HyperAdaLoRA with different hypernetwork architectures (MLP, CNN, and BERT layer) when finetuning DeBERTa-v3-base. Figure 3 shows the training loss curves of these three variants on the RTE and WNLI datasets. The results show that the choice of hypernetwork architecture affects the convergence speed. The BERT layer hypernetwork achieves the fastest convergence across all three datasets. This indicates that the attention mechanism is particularly effective at capturing the complex interdependencies among the elements of the P, Λ, and

Model	Method	Stanford Alpaca	Magpie-Pro-300K-Filtered	OpenPlatypus
LLaMA3.1-8B	AdaLoRA	8125	19600	11900
	HyperAdaLoRA (ours)	**6650**	**15720**	**9750**
Qwen2.5-7B	AdaLoRA	4240	15000	6750
	HyperAdaLoRA (ours)	**3500**	**11000**	**5500**

Table 6: Comparison of total training time (in seconds) for AdaLoRA and HyperAdaLoRA across natural language generation tasks. Experiments are conducted using the LLaMA3.1-8B and Qwen2.5-7B models on the Stanford Alpaca, Magpie-Pro-300K-Filtered, and OpenPlatypus datasets. HyperAdaLoRA consistently demonstrates lower training times across these settings.

Batch Size	AdaLoRA		HyperAdaLoRA (ours)	
	Memory (MB)	Latency (ms / step)	Memory (MB)	Latency (ms / step)
1	2758	123.16	2722	118.10
2	2798	132.60	2764	118.82
4	3232	149.38	3196	134.54
8	4115	189.39	4078	178.51
16	5906	284.33	5870	272.72
32	9600	486.16	9564	465.77
64	16566	882.55	16530	872.80

Table 7: Comparison of memory usage and training latency per step between AdaLoRA and HyperAdaLoRA across various batch sizes. The experimental settings are consistent with the implementation details described above.

Q matrices, thereby generating more efficient and targeted updates. In contrast, the MLP and CNN-based hypernetworks lag behind the BERT layer variant. The MLP, being the simplest architecture, shows the least acceleration, while the CNN provides intermediate results. This performance hierarchy is consistent with the representation capabilities of these architectures, further demonstrating the benefits of using complex mechanisms like attention to generate parameters in this context.

4.7 Efficiency Analysis

We analyze the computational load of our method compared to AdaLoRA. Table 7 presents the GPU memory usage and per step training latency for both methods under different batch sizes. As shown in Table 7, HyperAdaLoRA exhibits a slight reduction in memory usage compared to AdaLoRA. Additionally, HyperAdaLoRA consistently demonstrates lower training latency per step. Although the per step reduction may appear modest, the faster convergence rate demonstrated earlier leads to a significantly shorter total training time to reach the target performance level. Therefore, HyperAdaLoRA achieves a notable improvement in training efficiency. This efficiency gain mainly stems from two factors: first, our hypernetwork is lightweight and adds minimal computational overhead; second, by generating task-specific parameters conditioned on context, it enables dynamic adaptation without iterative optimization, accelerating convergence. By modulating update direction and magnitude, it efficiently explores high-dimensional parameter spaces while preserving model expressiveness. We provide an analysis of memory usage in Appendix I.

5 Conclusion

In this paper, we address the issue of slow convergence in AdaLoRA, an effective dynamic rank allocation method within the PEFT framework. To overcome this limitation, we propose HyperAdaLoRA, a novel and flexible framework that employs hypernetworks to dynamically generate the SVD-based low-rank parameters (P, Λ, Q) central to AdaLoRA. By adopting an attention-enhanced hypernetwork architecture, HyperAdaLoRA captures fine-grained parameter dependencies and produces more targeted updates during training. This design enables the model to explore the optimization space more efficiently, leading to notably faster convergence compared to standard AdaLoRA training methods. Extensive experiments across multiple benchmarks and model scales show that HyperAdaLoRA achieves faster convergence while maintaining comparable or even better final performance. Furthermore, extension experiments on other LoRA-based methods clearly demonstrate the versatility and broad applicability of our approach.

Limitations

In this work, we conduct extensive experiments to evaluate the effectiveness of our hypernetwork based training method in accelerating the training of various tasks. The results demonstrate that our approach can significantly speed up training without compromising performance. However, due to computational constraints, we have not yet been able to evaluate it on much larger models, such as those with 70 billion parameters. Exploring its scalability to models of this scale represents an important direction for future work.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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: 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. *arXiv preprint arXiv:2311.11696*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing (2021). *URL https://arxiv. org/abs/2111.09543*.

N. Houlsby, A. Giurgiu, S. Jastrzebski, and 1 others. 2019. Parameter-efficient transfer learning for nlp. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019:279–285.

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

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.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Louis Kirsch, Julius Kunze, and David Barber. 2018. Modular networks: Learning to decompose neural computation. *Advances in neural information processing systems*, 31.

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

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.

Mengtian Li, Jinshu Chen, Wanquan Feng, Bingchuan Li, Fei Dai, Songtao Zhao, and Qian He. 2025a. Hyperlora: Parameter-efficient adaptive generation for portrait synthesis. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 13114–13123.

Shiwei Li, Xiandi Luo, Xing Tang, Haozhao Wang, Hao Chen, Weihong Luo, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2025b. Beyond zero initialization: Investigating the impact of non-zero initialization on lora fine-tuning dynamics. *arXiv preprint arXiv:2505.23194*.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. 2025c. Beyond higher rank: Token-wise input-output projections for efficient low-rank adaptation. *arXiv preprint arXiv:2510.23123*.

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.

Chen Liang, Nikos Karampatziakis, Tuo Zhao, and Weizhu Chen. 2022. Hart: Efficient adaptation via regularized autoregressive parameter generation. *arXiv preprint arXiv:2207.01411*.

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

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. 2024b. Alora: Allocating low-rank adaptation for fine-tuning large language models. *arXiv preprint arXiv:2403.16187*.

Jonathan Lorraine and David Duvenaud. 2018. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*.

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*.

Jesus-German Ortiz-Barajas, Helena Gomez-Adorno, and Thamar Solorio. 2024. Hyperloader: Integrating hypernetwork-based lora and adapter layers into multi-task transformers for sequence labelling. *arXiv preprint arXiv:2407.01411*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and 1 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Haoxiang Shi, Rongsheng Zhang, Jiaan Wang, Cen Wang, Yinhe Zheng, and Tetsuya Sakai. 2022. Layer-connect: Hypernetwork-assisted inter-layer connector to enhance parameter efficiency. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3120–3126.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

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*.

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*, abs/2406.08464.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 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.

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*.

Hao Zhang, Mengsi Lyu, Yulong Ao, and Yonghua Lin. 2025a. Enhancing llm efficiency: Targeted pruning for prefill-decode disaggregation in inference. *Preprint*, arXiv:2509.04467.

Hao Zhang, Mengsi Lyu, Chenrui He, Yulong Ao, and Yonghua Lin. 2025b. Towards adaptive visual token pruning for large multimodal models. *arXiv preprint arXiv:2509.00320*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. 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*.

A Comprehensive Analysis of Hypernetworks for Accelerating Convergence

We present a more comprehensive theoretical analysis that elucidates how hypernetworks accelerate convergence.

Context-Aware Task-Specific Parameter Generation for Accelerated Convergence: Hypernetworks dynamically generate task-specific parameters based on contextual input information, enabling the direct output of optimal parameters or their low-rank representations for the current task. This approach circumvents the traditional, time-consuming explicit optimization process, thereby significantly reducing the number of training iterations. Such a mechanism is particularly well-suited for multi-task settings or scenarios with rapidly changing data distributions, allowing for rapid adaptation to new tasks without the need for parameter re-learning from scratch and effectively accelerating model convergence.

Hypernetworks as Learnable Inner Optimizers for Dynamic Update Strategies: Hypernetworks adjust the update direction and step size of parameters based on the current model state, effectively simulating a learnable inner optimizer. Under this mechanism, the model no longer relies on a fixed, uniform optimization strategy; instead, the hypernetwork determines how to update at each step, enabling a more flexible and efficient convergence trajectory. This approach inherits the principles of meta-learning while avoiding nested backpropagation, resulting in reduced computational cost and faster convergence.

Structured Parameter Optimization and Stability Enhancement in High-Dimensional Spaces via Hypernetworks: In high-dimensional parameter spaces, conventional optimization methods often suffer from getting trapped in local optima or oscillatory regions. Hypernetworks, by explicitly modeling the distribution of parameters, learn more structured update strategies that facilitate more efficient exploration of superior regions in the parameter space. In particular, by encoding contextual information as priors, hypernetworks provide more directed parameter generation schemes, thereby enhancing both the stability and efficiency of the training process.

B Details of Hypernetworks for LoRA, DoRA, and DyLoRA

In line with our formulation that extends AdaLoRA to HyperAdaLoRA in the paper, we now provide a parallel description of how hypernetworks are applied to LoRA, DoRA, and DyLoRA. For LoRA and DyLoRA, their weight update formula is the same as in standard LoRA: $\Delta W = BA$, where B and A are updated via backpropagation. We introduce two hypernetworks, H_B and H_A, to generate the updated parameters: $B_{i+1} = H_B(B_i; \Phi_B)$, $A_{i+1} = H_A(A_i; \Phi_A)$, where Φ_B and Φ_A are the trainable parameters of the hypernetworks. B_i and A_i denote the parameters before the i-th iteration, and B_{i+1} and A_{i+1} denote the parameters after the update. For DoRA, the original method factorizes the LoRA low-rank update into a direction component and a scalar component. Let the directional part be $\Delta W_{\text{dir}} = B_{\text{dir}} A_{\text{dir}}$, and let the scalar be s. The weight update can then be written as $\Delta W = s \cdot \Delta W_{\text{dir}} = s \cdot B_{\text{dir}} A_{\text{dir}}$. In standard DoRA, B_{dir} and A_{dir} are directly updated via backpropagation. Analogously, we introduce hypernetworks for the directional component in DoRA. In particular, we use two hypernetworks, $H_{B_{\text{dir}}}$ and $H_{A_{\text{dir}}}$, to generate the updated parameters: $B_{\text{dir},i+1} = H_{B_{\text{dir}}}(B_{\text{dir},i}; \Phi_{B_{\text{dir}}})$, $A_{\text{dir},i+1} = H_{A_{\text{dir}}}(A_{\text{dir},i}; \Phi_{A_{\text{dir}}})$, where $\Phi_{B_{\text{dir}}}$ and $\Phi_{A_{\text{dir}}}$ are the trainable parameters of the hypernetworks.

C Processing Details of Parameter Matrices (e.g., P)

We describe the processing pipeline of the parameter matrix P. Initially, P is sampled from a Gaussian distribution and treated as a temporary variable. At each training step, the hypernetwork H_P takes P as input and produces an updated version of P, which is then used in the forward pass. During backpropagation, gradients are propagated through P to update the parameters of H_P. This process is iteratively repeated throughout the training procedure.

D Implementation Details of the Pruning Strategy

During the fine-tuning process, every ΔT steps, we perform a pruning operation over the singular value sets Λ of the LoRA weight matrices. For each LoRA parameter matrix W_i, we compute its

current gradient ∇W_i and obtain its singular values $\{\sigma_{ij}\}_{j=1}^{r}$, where σ_{ij} denotes the j-th singular value of the i-th matrix. To quantify the relative importance of each singular direction, we define an importance score as

$$s_{ij} = |\sigma_{ij} \cdot \nabla \sigma_{ij}| \tag{9}$$

which reflects both the magnitude of the singular value and its sensitivity to the gradient, indicating its contribution to parameter adaptation. After computing all scores, we identify the k smallest s_{ij} across all matrices and prune their associated singular values. This targeted pruning reduces the rank allocation for less significant directions and enables dynamic redistribution of the global rank budget, prioritizing more impactful components and improving overall parameter efficiency during training. Our parameter k is fixed and predefined. It is worth noting that the pruning strategy is not the primary focus or core contribution of our work. It is directly inherited from the original design of AdaLoRA and is used solely to maintain consistency and fairness with prior work, rather than to introduce a new pruning technique.

E Other Training Details

During training, we employ a linear warm-up strategy to smoothly ramp up the learning rate, thereby enhancing training stability and convergence. Specifically, the learning rate is linearly increased from zero to the preset maximum value over the first 500 steps, after which the main learning rate scheduler (cosine annealing) takes over. Model parameters are initialized with a normal distribution. We set the average budget per weight to three, giving a total budget equal to three times the number of weights, and use a pruning interval of 100 steps. To ensure a fair comparison, all these training configurations are kept consistent.

F $P\Lambda Q$ Parameter Similarity Analysis

We conduct an average similarity analysis between the $(P\Lambda Q)$ parameters generated by our method and those produced by AdaLoRA. As shown in Table 8, the results indicate that the similarity between corresponding parameters consistently exceeds 0.98, demonstrating that the two methods are highly consistent at the parameter level and essentially equivalent in terms of performance. This finding further supports that our method can achieve comparable parameter representations while maintaining strong performance.

Parameter	P	Λ	Q
Cosine Similarity	0.9861	0.9878	0.9811

Table 8: Cosine similarity between the $(P\Lambda Q)$ parameters of our method and AdaLoRA.

G Additional Results on Training Time and Performance

We further present a comparative performance evaluation on the OpenPlatypus dataset, as well as a training time comparison of LLaMA3.1-8B on GSM8K and HumanEval. The corresponding results are summarized in Table 9 and 10. These results demonstrate that our method improves training efficiency without compromising performance.

Model	**Method**	**BLEU-4**	**ROUGE-1**
Qwen2.5-7B	AdaLoRA	19.87	44.24
	Ours	19.92	44.65
LLaMA3.1-8B	AdaLoRA	34.86	52.90
	Ours	35.95	53.00

Table 9: Performance comparison of HyperAdaLoRA and AdaLoRA on OpenPlatypus.

H Comparison with the γ = 0 Setting (without regularization)

We evaluate our method under the $\gamma = 0$ setting on the LLaMA3.1-8B model using the Stanford Alpaca and Magpie datasets. As shown in Table 11, setting $\gamma = 0$ leads to a slight, yet acceptable, drop in accuracy, while the training time remains largely unchanged. This indicates that our method can still achieve strong performance even without applying regularization.

I Memory Analysis

We provide here an analysis of how our method reduces memory consumption. The number of parameters introduced by our hypernetwork is much smaller than the total number of fine-tuning parameters introduced in the baselines (e.g., (P, Λ, Q) in AdaLoRA, or (A, B) in LoRA). In these baseline methods, the optimizer needs to maintain separate first- and second-moment states for all these

Dataset	LoRA w/o	LoRA w/	AdaLoRA w/o	AdaLoRA w/	DoRA w/o	DoRA w/	DyLoRA w/o	DyLoRA w/
GSM8K	9863	8110	10542	9011	10892	9054	12305	10527
HumanEval	8625	7210	10281	8620	9532	8029	10085	8547

Table 10: Comparison of training time (s) across methods with and without hypernetworks (w/o = without hypernetworks, w/ = with hypernetworks).

Method	Stanford Alpaca	Magpie	Stanford Alpaca (Time)	Magpie (Time)
Ours ($\gamma = 0$)	54.88	70.36	6970	14920
Ours	55.10	70.73	6650	15720

Table 11: Evaluation of our method under $\gamma = 0$ on the Stanford Alpaca and Magpie datasets.

fine-tuning parameters, which consumes a considerable amount of memory. In our method, (P, Λ, Q) / (A, B) are generated on demand by a shared lightweight hypernetwork and are no longer treated as independent trainable parameters, so the optimizer only needs to maintain states for the hypernetwork parameters. This substantially reduces the amount of optimizer states, leading to a slightly lower peak memory footprint.

J Training Stability of Our Method

Our proposed architecture is inherently stable to train for several reasons: (1) Shared optimization objective: the hypernetwork does not alter the main model's optimization objective but merely imposes a structured parametrization on the LoRA parameters, so the system still minimizes the same task loss instead of engaging in an adversarial game; (2) Low-dimensional update space: the hypernetwork only operates on low-rank LoRA factors, which imposes strong constraints on the parameter space and restricts updates to a structured low-dimensional subspace, leading to a smoother and more tractable optimization landscape; (3) Aligned gradient signals: the main model and the hypernetwork share the same task loss and gradient signal, so there are no competing objectives and their optimization directions are statistically aligned; (4) Lightweight hypernetwork design: the hypernetwork is relatively small and trained with conservative optimization hyperparameters, behaving more like a lightweight adaptation module rather than a second large backbone; (5) LoRA bounded update magnitude: the low-rank structure of LoRA naturally bounds the magnitude of updates and gradient norms, so even when local gradients are large, the effective update space is constrained by the rank and bottleneck dimensions, which in practice helps prevent gradient explosion and numerical divergence.

K Case Study

In Figure 4 and 5, we can see examples of two different natural language understanding tasks: WNLI and RTE. These examples demonstrate the models' capabilities in understanding and reasoning with textual information. These examples illustrate how models perform on different types of textual reasoning tasks when finetuned with our method. By fine-tuning RoBERTa-base and DeBERTa-v3-base models, we can enhance their performance on these tasks, thereby improving their ability to understand and reason with textual information.

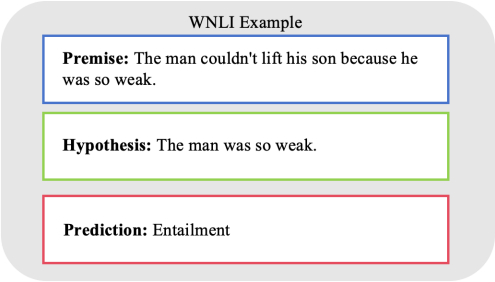


Figure 4: Examples generated by RoBERTa-base and DeBERTa-v3-base for the WNLI dataset.

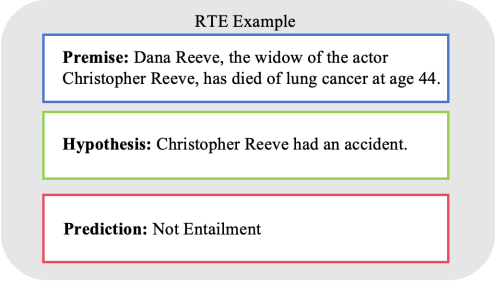


Figure 5: Examples generated by RoBERTa-base and DeBERTa-v3-base for the RTE dataset.