AFA-LoRA: Enabling Non-Linear Adaptations in LoRA with Activation Function Annealing

Anonymous ACL submission

Abstract

Low-Rank Adaptation (LoRA) is a widely adopted parameter-efficient fine-tuning (PEFT) method. However, its linear adaptation process limits its expressive power. This means there is a gap between the expressive power of linear training and non-linear training. To bridge this gap, we propose AFA-LoRA, a novel training strategy that brings non-linear expressivity to LoRA while maintaining its seamless mergeability. Our key innovation is an annealed activation function that transitions from a non-linear to a linear transformation during training, allowing the adapter to initially adopt stronger representational capabilities before converging to a mergeable linear form. We implement our method on supervised fine-tuning, reinforcement learning, and speculative decoding. The results show that AFA-LoRA reduces the performance gap between LoRA and full-parameter training. This work enables a more powerful and practical paradigm of parameter-efficient adaptation.

1 Introduction

The growth of Large Language Models (LLMs)(Vaswani et al., 2017; Team et al., 2025; Achiam et al., 2023; Liu et al., 2024a; Bai et al., 2023) has revolutionized natural language processing. However, due to the extremely large number of parameters in these massive models, fully fine-tuning them for downstream tasks is usually infeasible. This challenge led to the development of the Parameter Efficient Fine-Tuning (PEFT) method, which aims to achieve competitive performance by training only a small fraction of the model's parameters. Among these, Low-Rank Adaptation (LoRA) (Hu et al., 2022) has become a major method. LoRA freezes the weights W of the pre-trained model and inserts a trainable rank decomposition matrix (adapter) into each layer. The update is parameterized as $\Delta W = \frac{\alpha}{r} BA$, where $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are the low-rank matrices with rank r, and α is a scaling hyperparameter. This design allows the adapter to be seamlessly merged back into the main model after training. ($W_{\text{new}} = W + \frac{\alpha}{r} BA$).

Despite its wide application, LoRA's expressive power is limited. From a design perspective, LoRA's forward propagation process is linear and lacks the nonlinear transformation capability inherent in the feedforward layer of the basic model, which is fully utilized during full parameter fine-tuning. By introducing non-linear functions into LoRA's training, we aim to reduce the difference in performance between LoRA and full fine-tuning. A seemingly straightforward solution would be to introduce non-linear activation functions (e.g., ReLU) between the LoRA matrices. However, this approach creates a new problem: the resulting non-linear adapter can no longer be merged into the main model through simple matrix addition.

To resolve this conflict, we propose AFA-LoRA (Activation Function Annealing LoRA), a novel training strategy that combines the advantages of nonlinear training and linear integration. Our main point is that the need for nonlinearity is especially critical in the initial training phase of the model, while the fusion capability only needs to be guaranteed to be linear at the end of training. AFA-LoRA introduces an annealed activation function, $y = \beta \cdot \sigma(x) + (1 - \beta) \cdot x$, placed between the A and B matrices of LoRA. The weight β is annealed from 1 to 0 over the training process. Initially, the adapter behaves as a powerful non-linear projector ($\beta = 1$), maximizing learning capacity. As training progresses, it smoothly and differentiably transitions into a linear function ($\beta = 0$), guaranteeing mergeability upon convergence.

We evaluated AFA-LoRA on a variety of tasks. In the Supervised Fine-Tuning (SFT) benchmark, it narrowed the performance gap between standard LoRA and fully parametric fine-tuning. Secondly,

we integrated AFA-LoRA into the GRPO framework(Shao et al., 2024) for reinforcement learning, demonstrating that it also effectively reduces the gap between GRPO-LoRA and full-parameter GRPO, showing that it works well for more than just SFT. We also integrated AFA-LoRA into the draft model in Eagle, a popular speculative decoding framework. We added LoRA adapters to the draft model in Eagle and trained them together with the main weights. This enabled the draft model to accept longer token sequences, showing that AFA-LoRA can be well adapted to different tasks.

In summary, our contributions are:

- **Problem Formulation:** We point out the trade-off in LoRA between learning ability and integratability. Adding non-linearity helps models learn better, but it usually makes merging harder—a challenge for any mergeable PEFT method.
- **Method Innovation:** We introduce Activation Function Annealing (AFA), a training method that implements non-linear functions initially, then smoothly switches to linear ones for inference. This way, AFA-LoRA maintains full mergeability while boosting performance.
- **Experimental Validation:** We show that AFA-LoRA works well on supervised fine-tuning, reinforcement learning (GRPO), and speculative decoding (Eagle). In all cases, it narrows the gap with full fine-tuning and still allows for easy merging after training.

2 Theory

This section presents the theoretical background for Activation Function Annealing (AFA). We define AFA as a method that enables the exploration of both linear and non-linear function spaces during training. In addition, we analyze its advantages from an optimization perspective, showing how AFA can improve model adaptation and convergence.

2.1 Formalization

The fine-tuning process can be viewed as learning a parameterized function $F_\theta(x)$ that adapts a pre-trained model. Within this framework, an adapter module (e.g., a LoRA branch) constitutes a specific functional component. The standard LoRA adapter applies a purely linear transformation: $F_{\text{LoRA}}(x) = W_2 W_1 x$. This means there is no non-linear activation between the two matrices; it is equivalent to applying the identity function.

The core of our method is the introduction of a time-dependent activation function σ_{AFA}, defined as

$$\sigma_{\text{AFA}}(x; t) = \beta(t) \cdot \sigma(x) + (1 - \beta(t)) \cdot x, \quad (1)$$

where $t \in [0, T]$ denotes the training step, and $\beta(t)$ is an annealing coefficient that decreases monotonically from $\beta(0) = 1$ to $\beta(T) = 0$. The function σ is a standard non-linear activation function, such as ReLU. The resulting adapter using AFA is given by $F_{\text{AFA}}(x; t) = W_2(t) \cdot \sigma_{\text{AFA}}(W_1(t)x; t)$.

2.2 Key Properties

The AFA method has several features that help explain why it works well in practice. First, it satisfies clear boundary conditions: at the start of training ($t = 0$), $\sigma_{\text{AFA}}(x; 0) = \sigma(x)$, endowing the adapter with full non-linear capacity; at convergence ($t = T$), $\sigma_{\text{AFA}}(x; T) = x$, reducing the adapter to a linear function that can be seamlessly merged into the main model. Second, provided $\sigma(x)$ and $\beta(t)$ are continuous, $\sigma_{\text{AFA}}(x; t)$ is continuous in its arguments, ensuring a smooth and stable optimization trajectory.

Most importantly, AFA enables a dynamic expansion of the searching space. Let $\mathcal{F}_{\text{Linear}}$ represent the space of linear adapters and $\mathcal{F}_{\text{Nonlinear}}$ the space of non-linear adapters with a fixed σ. The AFA strategy defines a continuous family of intermediate spaces $\mathcal{F}_{\text{AFA}}(t)$. This family originates from the non-linear space, $\mathcal{F}_{\text{AFA}}(0) = \mathcal{F}_{\text{Nonlinear}}$, and terminates in the linear space, $\mathcal{F}_{\text{AFA}}(T) = \mathcal{F}_{\text{Linear}}$. Crucially, for any $t < T$, the linear space is a proper subset, $\mathcal{F}_{\text{Linear}} \subset \mathcal{F}_{\text{AFA}}(t)$. This guaranties that AFA searches a strictly richer space than standard LoRA throughout most of the training process, while finally converging to a mergeable solution.

2.3 Optimization Landscape Perspective

The advantage of AFA can be further understood through the lens of optimization. Full fine-tuning operates on a complex, high-dimensional loss landscape $\mathcal{L}_{\text{Full}}(\Theta)$. In contrast, LoRA constrains the optimization to a lower-dimensional subspace $\mathcal{L}_{\text{LoRA}}(\theta)$, which may lack access to high quality minima present in the full landscape.

The AFA strategy can be viewed as a guided search. Initially, with $\beta \approx 1$, the optimization occurs in an expanded space $\mathcal{L}_{\text{AFA}}$, leading to the discovery of more complex and deeper feature adaptations. As β anneals to zero, the search space keeps getting smaller, guiding the optimization trajectory from the promising region found in the expanded space back into the constrained linear subspace $\mathcal{L}_{\text{LoRA}}$. This process effectively guides the model parameters to a superior solution that can still be merged into the main model, thereby reducing the performance gap between LoRA and full fine-tuning.

2.4 Generality of the Framework

The AFA formulation presented in Eq. (1) is not limited to the LoRA framework. It can be used in many cases where we want a neural network component to learn with more flexibility during training but need it to fit a certain structure at deployment. The concept can be generalized to any scenario where a target component $C_{\text{target}}(x)$ is augmented with a more expressive component $C_{\text{boost}}(x)$ during training via annealing. Using AFA in LoRA, where the final goal is a linear adapter, is a strong example of this general method.

3 Related Work

We build on developments in PEFT and explore new adaptations of activation functions for neural networks. Here, we summarize representative related works that set the stage for our contribution.

3.1 Parameter-Efficient Fine-Tuning

The high cost of full fine-tuning has led to the creation of PEFT-based approaches. Early approaches include adapter-based methods, which insert small, trainable modules between layers of a pre-trained model (Houlsby et al., 2019), and prompt-based techniques like prompt-tuning and prefix-tuning, which optimize continuous input vectors (Lester et al., 2021; Li and Liang, 2021). Among these methods, Low-Rank Adaptation (LoRA) (Hu et al., 2022) is widely used because it is efficient, and its adapters can be easily merged into the main model after training; thus, there is no extra cost during inference. In our method, we use LoRA as a starting point and focus on improving its ability to learn complex patterns while retaining the integrable features.

3.2 Advances in the LoRA Framework

The success of LoRA has inspired extensive research aimed at improving its efficiency and performance. One approach is adaptive parameter allocation (e.g. AdaLoRA) (Zhang et al., 2023). AdaLoRA changes the rank of LoRA matrices during training to better utilize computational resources. Other innovative methods include VeRA (Kopiczko et al., 2023), which focuses on parameter reduction, and DoRA (Liu et al., 2024b), which decouples the magnitude and direction of weight updates. QLoRA (Dettmers et al., 2023) allows for fine-tuning large models using quantized weights, which saves memory and speeds up training.

Most of these methods explore how to set up or train the LoRA, but the core idea of linear adaptation remains unchanged. To address this issue, some recent methods attempt to change the internal structure of adapters. Among the aforementioned works, our method stands out because it directly tackles the trade-off between learning ability and easy merging. As far as we know, AFA-LoRA is the first to add a time-dependent non-linearity during training that gradually fades away, allowing the adapter to learn more at first and then merge smoothly into the main model.

3.3 Activation Functions in Model Adaptation

Activation functions can effectively introduce non-linearity into neural networks. They have been widely used in pre-training and full fine-tuning, but the exploration of their applications within PEFT adapters is still limited. Most PEFT adapters only retain the activations within the main model and do not add new non-linear components to their adapters. In our method, we propose introducing a temporary non-linear activation to the adapter during training, which helps to improve performance. This idea is similar to activation annealing used elsewhere, but here we apply it in PEFT while making sure that merging after training remains easy.

Previous work such as PReLU (He et al., 2015) has shown that allowing the activation function to change during training is feasible. Our method follows this idea by gradually changing the non-linearity in adapters over time.

3.4 Summary and Positioning

Our proposed AFA-LoRA stands out among PEFT methods. Instead of just changing LoRA's parameters, it improves the performance of adapters by introducing a temporary non-linear component during training. This non-linearity will converge to linear space complexity over the iterations, so there is no extra cost when using the model for inference. Our method directly addresses the trade-off between learning ability and integratability, providing a flexible solution that can help build better integrable adapters.

4 Method

In this part, we will present details about our proposed AFA-LoRA method. To better understand the motivation behind AFA-LoRA, we will recap the standard LoRA as a preliminary first.

4.1 Preliminaries: Low-Rank Adaptation (LoRA)

LoRA approximates the weight update of a pre-trained matrix $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ with a low-rank decomposition. The forward pass can be formulated as follows:

$$h = W_0 x + \Delta W x = W_0 x + BAx, \qquad (2)$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable matrices with rank $r \ll \min(d_{\text{in}}, d_{\text{out}})$. After training with LoRA, the forward pass is merged as $W' = W_0 + BA$, resulting in zero inference overhead. However, the linearity of ΔW does not fully explore the expressive power of the pre-trained main model.

4.2 Activation Function Annealing (AFA)

To enhance expressivity while preserving integrability, we insert an annealed activation function between A and B. The adapter starts as a non-linear function and converges to a linear function.

4.2.1 Annealed Activation Function

We set up the annealed activation function ϕ by smoothly mixing a non-linear function σ (e.g., ReLU) with the identity function:

$$\phi(x; \beta) = \beta \cdot \sigma(x) + (1 - \beta) \cdot x, \qquad (3)$$

Where, β is a value that decreases from 1 to 0 as training continues. This means that when $\beta = 1$, we will adapted non-linear function $\sigma(x)$ for updating, and when $\beta = 0$, we simply use x.

4.2.2 AFA-LoRA Forward Computation

The AFA-LoRA forward pass is given by:

$$\begin{aligned} h &= W_0 x + B\phi(Ax; \beta(t)) \\ &= W_0 x + B\left[\beta(t)\sigma(Ax) + (1 - \beta(t))Ax\right] \end{aligned} \qquad (4)$$

Where, t is the training step. The architectural change is shown in Figure 1. At the end of training ($\beta(T) = 0$), this calculation will converge to $h = W_0 x + BAx$, which means all the extra weights can be easily merged into the main model.

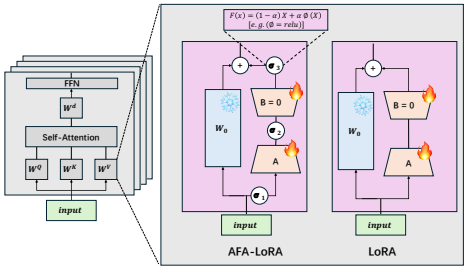

Figure 1: Architectural comparison of (a) Standard LoRA and (b) AFA-LoRA. The annealed activation function ϕ is placed beside the A and B matrices.

4.3 Annealing Schedule

The annealing schedule for $\beta(t)$ is defined over a range of training steps. A linear schedule is formulated as:

$$\beta(t) = \max\left(0,\ 1 - \frac{\max(0, t - T_{\text{start}})}{T_{\text{end}} - T_{\text{start}}}\right). \qquad (5)$$

In the experiment, unless otherwise specified, setting $T_{\text{start}} = 0$ and $T_{\text{end}} = 0.3T$ anneals β over the first 30% of training. The schedule profile is illustrated in Figure 2.

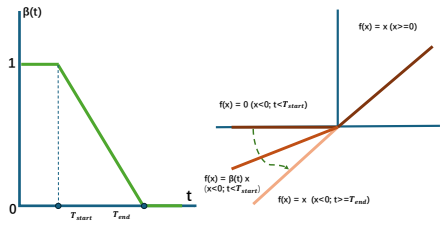

Figure 2: Illustration of ReLU-to-linear activation annealing and the decay schedule for $\beta(t)$ during training.

4.4 Training Algorithm

Algorithm 1 summarizes the training steps for AFA-LoRA. The proposed AFA-LoRA is an effective extension of standard LoRA by only adapting a time scheduler $\beta(t)$ and non-linear function ϕ during each forward pass.

Algorithm 1 AFA-LoRA Training

Input: Model W_0, Dataset D, Total steps T, Annealing range $[T_{\text{start}}, T_{\text{end}}]$
Initialize LoRA parameters $\theta = (A, B)$.
for $t = 1$ to T **do**
  $\beta(t) = 1 - (t - T_{\text{start}})/(T_{\text{end}} - T_{\text{start}})$
  Sample batch $(x, y) \sim D$
  Forward pass: $h = W_0 x + B[\beta(t)\sigma(Ax) + (1 - \beta(t))Ax]$
  Compute loss $\mathcal{L}$
  Update θ via gradient descent
end for
Merge adapter: $W' = W_0 + BA$

5 Experiments

We applied our AFA-LoRA across three scenarios to comprehensively evaluate its performance. In the supervised fine-tuning domain, we evaluate the method's capability to enhance commonsense reasoning using the Llama-3-8B model on the Commonsense-170K dataset, with performance evaluated across eight diverse benchmarks, including ARC-Challenge, BoolQ, and HellaSwag, through accuracy metrics. In the experiment of reinforcement learning, we integrate AFA-LoRA into the GRPO framework to optimize mathematical problem-solving policies on the GSM8K dataset, employing Qwen2.5 models ranging from 3B to 32B parameters and quantifying improvements through reward gains and reductions in performance gaps relative to full fine-tuning. Finally, in speculative decoding, we jointly train the draft model and AFA-LoRA adapters within the Eagle framework on the ShareGPT dataset using Llama3.1-8B and evaluate token acceptance rates. Each experimental paradigm employs distinct model architectures, adaptation strategies, and evaluation methodologies to validate AFA-LoRA's capabilities across diverse scenarios.

5.1 Supervised Fine-Tuning Experiments

We include both LoRA and DoRA (Weight-Decomposed Low-Rank Adaptation) (Liu et al., 2024b) as baselines in our experiments. DoRA is an advanced parameter-efficient fine-tuning method that splits weight updates into direction and magnitude, enabling it to capture more complex changes than standard LoRA.

We implemented activation function annealing with LoRA and DoRA on supervised fine-tuning with the Llama-3-8B (AI@Meta, 2024) model and the Commonsense-170K (Hu et al., 2023) dataset, which is designed for commonsense reasoning. We evaluate performance across eight benchmarks: ARC-Challenge (Clark et al., 2018), ARC-Easy, BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), Social IQA (Sap et al., 2019), and WinoGrande (Sakaguchi et al., 2021). Our experiments compare AFA-enhanced methods with three baselines: full parameter fine-tuning (Full-SFT), standard LoRA, and DoRA. For both LoRA and DoRA, we try seven different ways of placing the annealed activation function in the adapter structure, using rank $r = 32$ and scaling factor $\alpha = 64$. The value of β decreases linearly over the first 30% of training steps, so in Equation (5), we set $T_{start} = 0$ and $T_{end} = \frac{T}{3}$.

Table 1 shows that both AFA-LoRA and AFA-DoRA provide clear improvements over their baselines. The best AFA-LoRA setup achieves an average accuracy of 86.16%, which is 0.59% higher than the standard LoRA (85.57%). For DoRA, the top AFA-DoRA variant reaches 86.34%, a gain of 0.89% over the DoRA baseline (85.45%). Importantly, the best AFA-DoRA version reduces the gap to full fine-tuning (87.07%) by about 54.94%, while the best AFA-LoRA reduces this gap by around 39.33%.

In conclusion, our experimental results demonstrate that using activation function annealing with LoRA or DoRA improve accuracy on commonsense reasoning tasks. This approach contributes to narrow the difference between lightweight tuning methods and full fine-tuning technique.

5.2 Reinforcement Learning Experiments

We conduct reinforcement learning experiments using the GRPO (Group Relative Policy Optimization) framework, implemented with the Verl training system. This method uses preference optimization within groups of responses to keep policy updates stable. We test Qwen2.5-Instruct models of different sizes (3B, 7B, 14B, and 32B)(Team, 2024; Yang et al., 2024) on GSM8K(Cobbe et al., 2021), a dataset for math reasoning tasks. For training, we use the AdamW optimizer with model-specific learning rates (2×10^{-5} for 3B; 1.5×10^{-5} for 7B/14B; 1.3×10^{-5} for 32B). The global batch size is set to 1024 and is split into PPO mini-batches of 256 samples for each global batch. Each sequence has up to 512 prompt tokens and up to

Table 1: Commonsense reasoning evaluation results (Accuracy %) on Llama-3-8B. Results show the 30% decay configuration for each AFA placement variant. Avg is the macro-average across all 8 tasks. The best result for each method is **bold**.

Method	Placement	ARC-C	ARC-E	BoolQ	HellaSwag	OpenBookQA	PIQA	Social IQA	WinoGrande	Avg
FULL-SFT	–	82.94	92.68	74.98	96.71	89.40	90.26	83.21	86.11	87.07
LoRA Baseline	–	80.03	90.99	74.56	96.03	87.00	88.63	81.17	86.11	85.57
AFA-LoRA	σ-A-B	81.23	91.41	75.57	95.72	86.00	88.08	81.93	85.95	85.74
	A-σ-B	79.78	91.25	75.60	96.03	86.80	88.52	81.37	86.66	85.75
	A-B-σ	81.66	91.46	74.80	95.66	87.20	89.06	81.99	86.42	86.03
	σ-A-σ-B	80.38	90.91	74.95	96.11	87.40	88.52	80.04	87.45	85.72
	A-σ-B-σ	80.72	91.41	75.08	95.87	86.40	89.17	81.06	87.37	85.89
	σ-A-B-σ	81.14	91.33	75.44	95.71	88.40	88.68	81.53	87.06	**86.16**
	σ-A-σ-B-σ	80.29	90.82	75.96	95.74	87.20	89.06	80.60	85.87	85.69
DORA Baseline	–	80.46	90.45	75.66	95.77	85.00	87.92	81.06	87.29	85.45
AFA-DORA	σ-A-B	80.37	91.07	75.19	95.23	85.60	89.11	81.11	87.21	85.61
	A-σ-B	79.69	91.25	75.93	95.80	86.40	88.74	80.45	86.74	85.63
	A-B-σ	80.97	91.62	75.56	95.90	88.40	89.11	82.59	86.58	**86.34**
	σ-A-σ-B	80.20	91.07	75.99	96.05	87.60	89.33	81.16	87.21	86.08
	A-σ-B-σ	79.52	90.74	75.35	95.75	85.40	89.88	81.21	87.05	85.61
	σ-A-B-σ	81.48	91.58	76.17	95.66	86.80	89.00	81.42	86.97	86.14
	σ-A-σ-B-σ	81.65	91.07	76.60	95.70	87.00	88.57	81.26	87.21	86.13

Table 2: Comprehensive GRPO Evaluation on GSM8K: Training and Validation Performance

Model	Metric	Full	LoRA	σA-B	AσB	ABσ	σAσB	σABσ	AσBσ	σAσBσ
3B	Train Reward	97.50	95.27	95.41	95.63	95.61	95.72	95.86	**96.04**	95.68
	Val Reward	88.32	87.19	87.26	**88.70**	87.34	88.02	87.87	88.40	88.17
	Val Gain	-	0.0%	6.2%	**133.6%**	13.3%	73.5%	60.2%	107.1%	86.7%
7B	Train Reward	98.77	97.13	97.17	**97.60**	97.50	97.44	97.36	97.15	97.25
	Val Reward	92.80	92.19	93.03	92.34	**93.10**	92.65	92.87	92.65	92.65
	Val Gain	-	0.0%	137.7%	24.6%	**149.2%**	75.4%	111.5%	75.4%	75.4%
14B	Train Reward	98.83	97.29	97.46	**97.93**	97.56	97.46	97.54	97.64	97.54
	Val Reward	95.45	94.47	95.22	**95.83**	95.15	95.07	95.75	95.30	94.84
	Val Gain	-	0.0%	76.5%	**138.8%**	69.4%	61.2%	130.6%	84.7%	37.8%
32B	Train Reward	99.10	97.03	97.32	97.48	97.38	97.44	**97.50**	97.15	97.27
	Val Reward	96.66	96.21	96.44	96.44	96.51	96.44	96.21	**96.82**	96.66
	Val Gain	-	0.0%	51.1%	51.1%	66.7%	51.1%	0.0%	**135.6%**	100.0%

Note:
- **Gain** represents the percentage improvement over standard LoRA gap reduction: Gain $= \frac{\text{Method Reward} - \text{LoRA Reward}}{\text{Full Reward} - \text{LoRA Reward}} \times 100\%$
- σ indicates the position of the ReLU activation relative to LoRA matrices A/B
- **Bold values** denote the best performance per metric and model size
- Gain values exceeding 100% indicate the method outperformed the Full-Train baseline

1024 response tokens. The loss includes KL divergence regularization ($\beta = 0.001$), but no entropy term is used. For each prompt, five responses are generated using vLLM (Kwon et al., 2023). We adapt gradient checkpointing and FSDP to save memory and speed up training (Zhao et al., 2023). All experiments are carried out with a total of fifteen epochs. All variants of LoRA are set with rank $r = 64$ and scaling factor $\alpha = 32$. For AFA-LoRA, β decays linearly over the first 30% steps in all tested placements.

Table 2 shows that AFA-LoRA is highly versatile across different evaluation scenarios. Importantly, this method not only consistently narrows the performance gap with full parameter fine-tuning but also often surpasses it on validation metrics. This is a significant achievement for a parameter-efficient approach. For example, the ABσ configuration achieves a 149.2% improvement at the 7B scale, which means it generalizes even better than full fine-tuning in some cases. These consistent gains across model sizes show substantial task-agnostic benefits, especially for mid-sized models where annealed non-linearity provides significant benefits. All these improvements are achieved while maintaining full intergratability after training, so there is no extra inference cost, and deployment remains practical compared to traditional fine-tuning methods.

Overall, these results demonstrate that AFA-LoRA delivers clear and consistent improvements to reinforcement learning tasks across a wide range

of model sizes. The method is especially effective for mid-sized models, where it often achieves even better generalization than full fine-tuning. These findings highlight the value of activation function annealing as a simple yet powerful technique for improving adaptation quality in large language models under RL optimization.

5.3 Speculative Sampling Experiments

We apply AFA-LoRA adapter to every linear layer in Eagle's draft model. Each adapter uses an activation function that gradually decays from non-linear and linear transformation during training.

For each input x, the output is:

$$y = W_{\text{main}}\, x + B[\, \beta\sigma(Ax) + (1-\beta)Ax] \quad (6)$$

where A, B are trainable low-rank matrices, σ is an activation function (ReLU/GeLU/SiLU), and β decays from 1 to 0 during training.

Joint training with AFA-LoRA adapters enhances the draft model's capacity, particularly beneficial for smaller models like Eagle. Post-training, adapters merge into base weights without inference overhead.

We evaluated our method on ShareGPT dataset with Llama-3.1-8B as the draft model, using AdamW optimizer with a learning rate of 6×10^{-5} in BF16 precision.

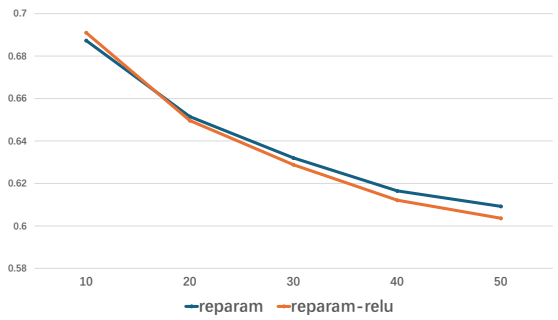

Figure 3: Llama3.1-8b training loss-epoch curves for Eagle-1 on the ShareGPT dataset

Table 3 shows that applying AFA-LoRA adapters and training the draft model in Eagle with the adapters contributes to better results than using the standard Eagle alone. In both chain and tree decoding setups, Eagle_LoRA achieves a higher average of accepted tokens per prompt compared to naive Eagle, demonstrating that this joint training approach improves the draft model's capacity for speculative decoding.

SiLU (Elfwing et al., 2018) annealing yields the highest acceptance rates (Table 3), with +1.0%/+0.8% gains on HumanEval for chain/tree decoding versus LoRA baseline.

Table 3: Average accepted tokens per prompt for speculative decoding on ShareGPT (Llama-3.1-8B), comparing standard Eagle to versions jointly trained with LoRA adapters and various activation annealing methods, for both chain ($d = 4$) and tree ($d = 5$) models.

Chain Decoding				
Activation	**MT-Bench**	**GSM8K**	**HumEval**	**Alpaca**
Eagle	1.6974	1.8861	2.2523	1.5810
Eagle_LoRA	1.6931	1.9208	2.2758	1.6032
Eagle_LoRA_ReLU	1.7297	1.9230	2.2711	1.6077
Eagle_LoRA_SiLU	**1.7306**	1.9308	**2.2981**	**1.6349**
Eagle_LoRA_GeLU	1.7161	**1.9330**	2.2841	1.5925
Tree Decoding				
Activation	**MT-Bench**	**GSM8K**	**HumEval**	**Alpaca**
Eagle	3.1406	3.4137	3.8272	3.0851
Eagle_LoRA	3.1803	3.4765	3.8609	**3.1677**
Eagle_LoRA_ReLU	**3.2124**	3.4479	3.8806	3.1514
Eagle_LoRA_SiLU	3.2067	3.4849	**3.8933**	3.1654
Eagle_LoRA_GeLU	3.2020	**3.4891**	3.8693	3.1559

Note:
- Metrics represent average accepted tokens per prompt. AFA-LoRA with ReLU, SiLU and GeLU use step decay (5 epoch decay and 45 epoch stay linear).

Figure 3 shows the progressive advantage of AFA-LoRA during training, showing the loss curves of the Llama-3.1-8B Eagle-1 models on the ShareGPT dataset. While both the pure LoRA baseline and the AFA-LoRA based on ReLU start from similar initial loss values, their trajectories rapidly diverge as training progresses. The key observation is the increasing divergence between the trajectories as the training continues. The gap of performance between naive LoRA and AFA-LoRA demonstrates that the annealed non-linearity provides compounding benefits throughout training, with its relative advantage growing more pronounced during the later optimization stages when fine-grained feature refinement becomes critical.

Overall, adding AFA-LoRA adapters contributes to performance gain of Eagle in accepting tokens during speculative decoding. This shows that our method is a strong choice for improving draft models in real-world generation tasks.

6 Ablation Studies

In this section, we explore the impact of different annealing schedules and activation functions on AFA-LoRA performance. While our main experiments (Section 5) focused on ReLU activation with a 30% decay duration, here we evaluate SiLU

Table 4: Commonsense reasoning evaluation results (Accuracy %) with different decay steps and activation functions. Results are grouped by method (DORA/LoRA) and activation function (GeLU/SiLU). The best average for each method is **bold**.

Method	Activation	Decay	ARC-C	ARC-E	BoolQ	HellaSwag	OpenBookQA	PIQA	Social IQA	WinoGrande	Avg
DORA	GeLU	30%	82.00	91.46	75.81	95.79	87.60	88.63	81.01	85.48	**85.97**
		60%	81.57	90.66	75.65	95.73	86.20	89.34	81.47	86.19	85.85
		100%	80.72	90.91	74.68	95.72	86.20	88.74	80.91	85.87	85.47
	SiLU	30%	80.20	91.46	75.26	95.95	87.40	88.79	81.06	85.64	**85.72**
		60%	81.31	91.08	75.54	95.54	87.60	88.63	80.45	85.64	**85.72**
		100%	78.75	90.99	75.35	95.74	88.20	88.08	81.27	86.27	85.58
LoRA	GeLU	30%	80.80	91.12	76.42	95.96	86.60	89.39	80.45	86.90	**85.96**
		60%	81.48	91.33	75.87	95.89	86.80	88.90	80.45	86.50	85.90
		100%	80.03	90.95	74.71	95.68	85.40	89.17	81.27	86.11	85.41
	SiLU	30%	80.20	90.95	74.83	95.89	87.40	89.23	80.55	86.66	**85.71**
		60%	79.52	90.70	75.08	95.99	87.00	89.12	81.47	86.35	85.65
		100%	78.67	90.53	75.69	95.56	87.00	88.74	80.91	86.27	85.42

and GeLU activations across three decay schedules. We performed ablation experiments using the same Llama3-8B LoRA SFT task described in Section 5.1, keeping the same hyperparameters (rank $r = 32$, scaling factor $\alpha = 64$) and evaluation benchmarks. The key difference lies in the annealing configurations: instead of changing the position of activation function as in the table 1, we fix the position with the manner of A-σ-B and change the activation function type ($\sigma \in \{\text{SiLU}, \text{GeLU}\}$) and the decay schedules ($T_{\text{end}} \in \{0.3T, 0.6T, T\}$).

Table 4 presents comprehensive results across all configurations. Across both LoRA and DoRA variants, the 30% decay schedule consistently achieves the best average performance. For LoRA with GeLU, the 30% decay yields 85.96% average accuracy, outperforming 60% decay (85.90%) and 100% decay (85.41%). Similarly, DoRA with GeLU peaks at 85.97% with 30% decay. This pattern holds across activation functions, demonstrating that early annealing followed by extended linear training is the optimal strategy.

Figure 3 shows why early annealing works. During the decay phase, models with activation annealing show slightly higher training loss than the baseline. However, as the training steps going further, the loss of our method outperforms the baseline with a more optimal convergence. This indicates that the nonlinear patterns captured during early training create a stronger foundation for later optimization. The training process reveals that the nonlinear patterns captured during the early training contribute to the performance gain and improve the optimality of convergence in the later training steps.

These ablation studies confirm our key insight: AFA-LoRA's effectiveness arises from utilizing the the nonlinear capabilities of activation functions during early training. The 30% decay schedule achieves the best performance by providing sufficient nonlinear capacity for initial feature adaptation alongside adequate linear training for convergence integration. Therefore, we use this 30% decay setting by default in nearly all subsequent experiments (SFT in Section 5.1 and GRPO in Section 5.2).

7 Conclusion

We propose Activation Function Annealing (AFA) as a simple yet effective way to enhance the expressive power of mergeable adapters like LoRA. By gradually changing non-linear activations to linear functions during training, AFA allows parameter-efficient fine-tuning to perform better without losing representational information for merging. Our experiments in supervised fine-tuning, reinforcement learning, and speculative decoding show that AFA-based adapters can reduce the performance gap with full-model adaptation. These gains come from small architectural changes and do not add any extra cost when deploying models for inference.

This work suggests new possibilities for designing efficient model adapters. In future research, it will be meaningful to explore adaptive annealing schedules for different tasks, apply this strategy to other other components in neural networks with diverse architectures, and automatically choose activation functions for specific domains. We believe that activation function annealing is a strong foundation for building high-performance yet practical adaptation techniques.

8 Limitations

While AFA-LoRA performed well across multiple tasks, this study still has some limitations that warrant further investigation in future work.

AFA-LoRA introduces additional hyperparameters, including annealing time (T_{start}, T_{end}), activation function type (ReLU, SiLU, GeLU), and their placement in the adapter. While experiments show that 30% annealing time and SiLU activation perform well on most tasks, these optimal configurations may vary across different tasks, datasets, or model sizes, requiring task-specific tuning. Additionally, the continuous gradient variations during annealing may affect training stability, particularly in large-scale distributed training or resource-limited environments. Future work could explore adaptive annealing strategies, more stable optimization methods, and systematic evaluations under various hardware setups to enhance the method's robustness and practicality.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

AI@Meta. 2024. Llama 3 model card.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

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

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

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

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

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.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

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

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

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

A.G. Sutton, R.S.and Barto. 1998. *Reinforcement learning: An introduction*. MIT Press Cambridge.

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.

Meituan LongCat Team, Bei Li, Bingye Lei, Bo Wang, Bolin Rong, Chao Wang, Chao Zhang, Chen Gao, Chen Zhang, Cheng Sun, and 1 others. 2025. Longcat-flash technical report. *arXiv preprint arXiv:2509.01322*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

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

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, and 1 others. 2023. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.