Dual-Module Collaborative LoRA for Effective Large Language Model Fine-Tuning

Lecheng Cao LECHENGCAO@SMAIL.NJU.EDU.CN
Junhao Zhang JUNHAO.ZHANG@SMAIL.NJU.EDU.CN
Xiaohan Zhang XHZHANG@SMAIL.NJU.EDU.CN
Huaxiong Li[*†] HUAXIONGLI@NJU.EDU.CN
Nanjing University

Editors: Hung-yi Lee and Tongliang Liu

Abstract

To enable parameter-efficient fine-tuning of large language models (LLMs), Low-Rank Adaptation (LoRA) reduces parameters by freezing pretrained weights W_0 and approximating updates via low-rank matrices $\Delta W = BA$. However, standard LoRA neglects the differential impact of low-rank matrix components on model performance and suffers from slow convergence due to random initialization. To address this, we propose a dual-module architecture: The shared module inherits pretrained weights' core semantic representations through principal component initialization, retaining residuals in the original model.The expert module incorporates a selection mechanism guided by importance screening, with orthogonality constraints imposed through loss regularization to ensure independence in parameter update directions. The shared module accelerates convergence by updating world knowledge, while the expert module dynamically screens domain knowledge to achieve efficient allocation of updated budgets. Extensive experiments under identical configurations show our method achieves 76.8% average accuracy on Commonsense 170k (Llama 2-7B), surpassing LoRA by 2.1%. On GSM8K and HumanEval, it outperforms LoRA by 2.3% and 9.7%, respectively.

Keywords: LoRA; fine-tuning; singular value decomposition; pre-trained models

1. Introduction

Pre-trained language models have advanced significantly, scaling from millions to hundreds of billions of parameters while demonstrating remarkable pattern recognition and knowledge emergence capabilities (Achiam et al. (2023)). Although these models exhibit strong cross-domain generalization, substantial distributional shifts between pre-training corpora and domain-specific data challenge specialized applications (Parthasarathy et al. (2024)). This has established parameter-efficient fine-tuning (PEFT) as a vital research frontier (Han et al. (2024)). PEFT methods substantially reduce the computational, memory, and storage costs of full-parameter tuning while maintaining comparable downstream task performance.

As a parameter-efficient fine-tuning method, Low-Rank Adaptation (LoRA) freezes the original pre-trained weights W_0 and introduces trainable low-rank matrices to approximate

* Corresponding Author
† This work was partially supported by National Natural Science Foundation of China under Grants Nos. 62576161, 62176116, and 62276136.

parameter updates (Hu et al. (2021)). It parameterizes the incremental matrix ΔW using the product of two low-rank matrices A and B:

$$W = W_0 + \Delta W = W_0 + BA. \tag{1}$$

where $\Delta W \in \mathbb{R}^{d_1 \times d_2}$, $A \in \mathbb{R}^{r \times d_2}$, $B \in \mathbb{R}^{d_1 \times r}$ and $r \ll \{d_1, d_2\}$. Remarkably, LoRA achieves performance comparable to full fine-tuning while utilizing merely 2% of the pre-trained model's trainable parameters.

Building on LoRA's high parameter efficiency and flexible adaptability, recent innovations include adaptive dimension adjustment via singular value thresholding (Zhang et al. (2023)) and optimized low-rank matrix initialization strategies that accelerate knowledge updates (Meng et al. (2024)). However, critical challenges remain. First, during domain adaptation tasks—characterized by narrow data distributions, strong task specificity and limited annotated samples—LoRA outputs exhibit high concentration, causing dimensional redundancy in latent representations (Zhou et al. (2024)). But hasty pruning of redundant components may hinder the low-rank matrix's capacity to update world knowledge, while abrupt compression of the parameter space exacerbates gradient competition phenomena. Second, Gaussian and zero initialization of matrices A and B induces slow convergence during complex fine-tuning due to local optima trapping (Meng et al. (2024)). Although biased initialization strategies enhance stability and convergence speed, they may disrupt parameter isotropy, causing premature convergence in pretrained feature subspaces.

To address these limitations, we draw inspiration from prior LoRA studies and Mixture-of-Experts (MoE) architectures. MoE employs dynamic sparse activation for efficient scaling: decomposing neural networks into specialized expert subnets and routing tasks via gating networks, enabling computation with partial parameter activation (Shazeer et al. (2017)). Notably, the recently proposed MoE architecture in DeepSeek-V3, incorporating a shared-and-routed expert design with auxiliary-loss-free load balancing, significantly enhances model performance and generalization capability (Liu et al. (2024)).

We propose a novel dual-module collaborative architecture for pretrained model fine-tuning. Our key contributions are:

- We design a dual-module framework. The shared module, initialized via SVD, processes all inputs to capture universal patterns, providing stable foundational representations that accelerate convergence.The expert module filters specialized matrices through importance evaluation, masking low-importance components for efficient domain-specific adaptation.

- We validated the effectiveness of our fine-tuning approach across multiple datasets. Through ablation studies, matrix freezing experiments and training overhead analysis, we demonstrate the functional roles and computational costs of individual modules during fine-tuning.

2. Related Works

Parameter-Efficient Fine-Tuning (PEFT) aims to reduce the number of parameters requiring updates during adaptation, thereby lowering computational and costs (Zhang et al. (2025)).

Mainstream PEFT approaches can be broadly categorized into four classes (Han et al. (2024)): additive PEFT, selective PEFT, reparameterized PEFT and hybrid PEFT.

Low-Rank Adaptation (LoRA), as a reparameterized PEFT method, leverages the low-rank property of matrices to reduce the number of parameters updated during adaptation (Hu et al. (2021)). By introducing two low-rank matrices A, B, it substitutes weight updates with the low-rank product $\Delta W = BA$. Compared to full fine-tuning, LoRA demonstrates advantages including parameter efficiency, zero inference overhead, modular adaptation and knowledge preservation. Building on LoRA's framework, researchers have advanced studies in parameter efficiency enhancement (Zhang et al. (2023)), parameter quantization (Hubara et al. (2018)), and rank adaptation strategies (Ren et al. (2024)). Our method represents an improvement upon LoRA, retaining most structural advantages of the original framework while enhancing its performance through enhanced parameter efficiency.

Building upon the foundational LoRA architecture, researchers have sought to enhance LoRA to more effectively capture task-specific features. A significant branch of this effort involves integrating LoRA with Mixture-of-Experts (MoE) architectures. MoE architecture comprises expert networks and a gating network within each MoE layer (Shazeer et al. (2017)). By routing inputs to the most suitable expert via the gating mechanism, MoE enables efficient processing of domain-specific tasks by activating only a subset of parameters, thereby achieving parameter efficiency.

Integrating LoRA with MoE allows models to learn multiple expert low-rank matrix pairs, with a gating network selecting experts based on input content. This enhances knowledge retention during fine-tuning and enables effective multi-task handling (Luo et al. (2024)). While our method differs philosophically from MoE, its dual-module architecture and expert module design draw inspiration from MoE's structure to enhance fine-tuning performance.

3. Proposed Method

We innovatively constructs a dual-module collaborative architecture that decomposes the incremental matrix ΔW of LoRA into a shared module incremental matrix ΔW_{share} and an expert module incremental matrix ΔW_{expert}, while adjusting the pretrained matrix. The schematic diagram of the proposed method is illustrated in Figure 1. The forward pass is formulated as:

$$y = W_{\text{res}}x + \Delta W_{\text{share}}x + \Delta W_{\text{expert}}x = W_{\text{res}}x + B_{\text{share}}A_{\text{share}}x + B_{\text{expert}}EA_{\text{expert}}x, \quad (2)$$

where $A_{\text{share}} \in \mathbb{R}^{d_{\text{in}} \times r_{\text{share}}}$, $B_{\text{share}} \in \mathbb{R}^{r_{\text{share}} \times d_{\text{out}}}$, $A_{\text{expert}} \in \mathbb{R}^{d_{\text{in}} \times r_{\text{expert}}}$, $B_{\text{expert}} \in \mathbb{R}^{r_{\text{expert}} \times d_{\text{out}}}$ and diagonal matrix $E \in \mathbb{R}^{r_{\text{expert}} \times r_{\text{expert}}}$. The configuration satisfies $d \gg r_{\text{expert}} \geq r_{\text{share}}$. During fine-tuning, the adjusted pretrained weight matrix W_{res} remains frozen, while normal gradient updates are applied to low-rank matrices in both modules. For the diagonal matrix E, to leverage its expert screening function, additional specialized updates are applied at periodic intervals.

3.1. Shared Module Based On SVD

The shared module, serving as the fundamental feature processor that receives all inputs, should possess the capability for universal feature extraction and maintain a feature space

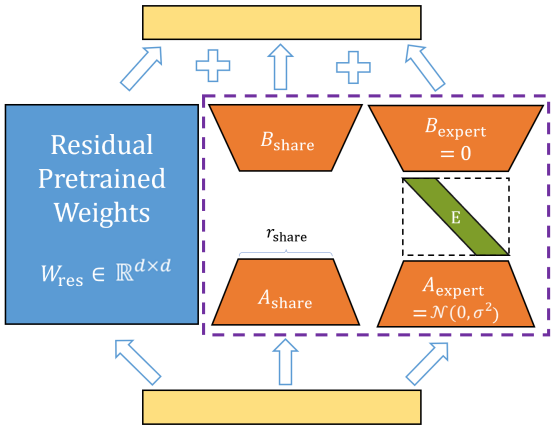

Figure 1: Overall Methodology

characterized by the prior knowledge representation of the pre-trained model. Therefore, we have made certain adjustments to both its initialization and training processes.

3.1.1. Initialization Method

The pre-trained weight matrix $W_0 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ undergoes economy-sized singular value decomposition:

$$W_0 = USV^\top, \tag{3}$$

where $U \in \mathbb{R}^{d_{\text{in}} \times \min(d_{\text{in}}, d_{\text{out}})}$ and $V \in \mathbb{R}^{\min(d_{\text{in}}, d_{\text{out}}) \times d_{\text{out}}}$ are the matrices of singular vectors, $V^\top$ is the transpose of matrix V; $S \in \mathbb{R}^{\min(d_{\text{in}}, d_{\text{out}}) \times \min(d_{\text{in}}, d_{\text{out}})}$ is a diagonal matrix, with its diagonal elements being the singular values $\{\sigma_i\}_{1 \ll i \ll \min(d_{\text{in}}, d_{\text{out}})}$ arranged in descending order.

Based on the magnitude of the singular values, the singular triplet $\delta = \{U, S, V\}$ is partitioned. The first r_{share} primary singular values and their corresponding singular vectors constitute the primary triplet $\delta_{\text{main}} = \{U_{[:,:r_{\text{share}}]}, S_{[:r_{\text{share}},:r_{\text{share}}]}, V_{[:,:r_{\text{share}}]}\}$. This triplet retains the principal information of the pre-trained weights, satisfying $\sum_{i=1}^{r_{\text{share}}} \sigma_i^2 \approx \sum_{i=1}^{\min(d_{\text{in}}, d_{\text{out}})} \sigma_i^2$. The remaining triplet, containing the residual singular values and vectors, is denoted as the residual triplet $\delta_{\text{res}} = \{U_{[:,r_{\text{share}}:]}, S_{[r_{\text{share}}:,r_{\text{share}}:]}, V_{[:,r_{\text{share}}:]}\}$. The objective of the shared module is to assign the primary triplet δ_{main} to two low-rank matrices such that $A_{\text{share}} B_{\text{share}} = U_{[:,:r_{\text{share}}]} S_{[:r_{\text{share}},:r_{\text{share}}]} V^\top_{[:,:r_{\text{share}}]}$, thereby efficiently approximating the principal components of the original matrix via low-rank adapters.

We evenly decompose $S_{[:r_{\text{share}},:r_{\text{share}}]}$ and combine its components with $U_{[:,:r_{\text{share}}]}$ and $V^\top_{[:,r_{\text{share}}:]}$ respectively, assigning the results to the adapter matrices of the shared module.

$$A_{\text{share}} = U_{[:,:r_{\text{share}}]} S^{\frac{1}{2}}_{[:r_{\text{share}},:r_{\text{share}}]} \in \mathbb{R}^{d_{\text{in}} \times r_{\text{share}}}, \tag{4}$$

$$B_{\text{share}} = S^{\frac{1}{2}}_{[:r_{\text{share}},:r_{\text{share}}]} V^\top_{[:,:r_{\text{share}}]} \in \mathbb{R}^{r_{\text{share}} \times d_{\text{out}}}. \tag{5}$$

The remaining residual triplet δ_{res} is combined and assigned to the pre-trained weight to correct the approximation error, thereby preventing the original world knowledge from

adversely affecting the updated world knowledge fine-tuned for the target domain:

$$W_{\text{res}} = U_{[:,r_{\text{share}}:]} S_{[r_{\text{share}}:,r_{\text{share}}:]} V^{\top}_{[:,r_{\text{share}}:]} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}. \tag{6}$$

The shared module, initialized with principal components of pre-trained weights, captures the model's fundamental knowledge extraction capability despite minimal parameters. As singular values quantify dimensional information content, and dominant singular values retain most model information (Golub and Van Loan (2013)), initializing with the primary triplet δ_{main} minimizes Frobenius norm approximation error. This enables direct fine-tuning of the most critical weights, accelerating convergence and enhancing stability. The initialization incurs negligible overhead through the Fast SVD technique (Halko et al. (2011)), with detailed cost analysis in Subsection 4.4.

3.1.2. Training Adjustments

The shared module encodes the representational capacity of the pre-trained model's prior knowledge upon parameter initialization. Functioning as the core component during fine-tuning, this module performs critical functions, including extracting foundational semantic features from inputs, establishing semantic associations and updating world knowledge systems.Given these essential characteristics, we implement a gradient protection scheme: The dropout rate for the shared module is permanently fixed at zero.

This design is motivated by two principles. First, large language models pre-trained on massive corpora converge in knowledge-dense parameter spaces, where weights already possess optimized generalization capabilities. Introducing random dropout to the shared module—initialized via principal components of these pre-trained weights—could disrupt well-established semantic feature representations, thereby degrading world knowledge modeling. Second, while the shared module disables dropout, the proposed method retains randomized dropout in expert modules. This dual-channel design effectively mitigates task-specific overfitting during fine-tuning. Through this adjustment, the method preserves the pre-trained model's fundamental representational capacity within the shared module, providing stable initialization for downstream task optimization.

3.2. Expert Module Based On Importance Screening

During LoRA for domain-specific tasks, the output distributions of LoRA increments across layers are relatively concentrated, with each low-rank matrix containing a significant portion of components with minimal impact (Zhou et al. (2024)). Therefore, the purpose of the expert module is to identify and retain high-contribution components within each low-rank matrix, thereby achieving efficient parameter budget allocation for specific fine-tuning task.

We construct a diagonal matrix E to perform expert selection, adjusting the influence of the triplet $\zeta = \{B_{\text{expert}}, E, A_{\text{expert}}\}$ on the model's output by controlling the values within E. For the incremental matrix ΔW_{expert} of the expert module, its structure follows from matrix properties:

$$\Delta W_{\text{expert}} = B_{\text{expert}} E A_{\text{expert}} = \sum_{i=1}^{r_{\text{expert}}} B_{\text{expert}_i} E_i A_{\text{expert}_i}, \tag{7}$$

where $A_{\text{expert}_i} \in \mathbb{R}^{d_{\text{in}} \times 1}$ denotes the i-th column of matrix A_{expert}, $B_{\text{expert}_i} \in \mathbb{R}^{1 \times d_{\text{out}}}$ denotes the i-th row of matrix B_{expert}, and E_i is the i-th element of the diagonal matrix E. Each $\Delta W_{\text{expert}_i} = B_{\text{expert}_i} E_i A_{\text{expert}_i} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, for $i \in [1, r_{\text{expert}}]$, can be regarded as an individual expert matrix. Thus, the incremental matrix ΔW_{expert} of the expert module can be viewed as being composed by concatenating a series of these smaller expert matrices, and the number of experts approximately equals the low-rank dimension r_{expert} of the expert module. The expert module evaluates the contribution of each expert matrix to the incremental output and performs screening accordingly.

To this end, the expert module must address two critical issues: How to evaluate the importance scores I for each of the r_{expert} experts within every low-rank module across each layer, enabling their importance ranking; How to ensure that controlling the diagonal matrix E effectively facilitates the screening of expert knowledge.

3.2.1. Importance Assessment

For the first issue, we adapts sensitivity-based importance screening (Molchanov et al. (2019); Sanh et al. (2020); Liang et al. (2021); Zhang et al. (2022, 2023)). Sensitivity-based importance screening posits that during model training, the importance score I of element w_{ij} at row i and column j in weight matrix W is determined by its sensitivity score S and uncertainty score U. The sensitivity score S is defined as the absolute value of the product between the parameter weight and its gradient:

$$S(w_{ij}) = |w_{ij} \cdot \nabla_{w_{ij}} \mathcal{L}|. \tag{8}$$

However, since sensitivity scores S are estimated on minibatches, stochastic sampling variability introduces substantial uncertainty when applying Formula 8. Therefore, during training step t, the proposed method employs linear moving averaging on sensitivity scores S^t to mitigate evaluation errors from individual batch samples, yielding the updated sensitivity score $\bar{S}^t$. Here $0 < \beta_1 < 1$ represents the smoothing sensitivity factor for the exponential moving average:

$$\bar{S}^t(w_{ij}) = \beta_1 \bar{S}^{t-1}(w_{ij}) + (1 + \beta_1) S^t(w_{ij}). \tag{9}$$

The uncertainty score U is defined as the absolute difference between the original sensitivity score S and its linearly averaged counterpart $\bar{S}$:

$$U(w_{ij}) = |S(w_{ij}) - \bar{S}(w_{ij})|. \tag{10}$$

Similarly, during training step t, linear moving averaging is applied to uncertainty scores U^t, producing the updated uncertainty score $\bar{U}^t$. The smoothing sensitivity factor $0 < \beta_2 < 1$ governs the exponential moving average for uncertainty:

$$\bar{U}^t(w_{ij}) = \beta_2 \bar{U}^{t-1}(w_{ij}) + (1 + \beta_2) U^t(w_{ij}). \tag{11}$$

By multiplying the sensitivity score $\bar{S}^t$ and the uncertainty score $\bar{U}^t$, the importance score $I^t(w_{ij})$ for each parameter is obtained:

$$I^t(w_{ij}) = \bar{S}^t(w_{ij}) \bar{U}^t(w_{ij}). \tag{12}$$

For the expert module approach, to achieve expert screening, it is essential to determine the importance of each expert matrix triplet $\zeta_i^t = \{B_{\text{expert}_i}^t, E_i^t, A_{\text{expert}_i}^t\}_{1 \le i \le r_{\text{expert}}}$ during the t-th training iteration. $A_{\text{expert}_i}^t = \{A_{\text{expert}_{ij}}^t\}_{1 \le j \le d_{\text{in}}} \in \mathbb{R}^{d_{\text{in}} \times 1}$ denotes the i-th column of A_{expert}^t, $B_{\text{expert}_i}^t = \{B_{\text{expert}_{ji}}^t\}_{1 \le j \le d_{\text{out}}} \in \mathbb{R}^{1 \times d_{\text{out}}}$ denotes the i-th row of B_{expert}^t, and E_i^t represents the i-th element of the diagonal matrix E^t. Following Formula 12, the importance score for each expert triplet in the t-th update is derived as:

$$I^t(\zeta_i^t) = I^t(E_i^t) + \frac{1}{d_{\text{out}}} \sum_{j=1}^{d_{\text{out}}} I^t(B_{\text{expert}_{ji}}^t) + \frac{1}{d_{\text{in}}} \sum_{j=1}^{d_{\text{out}}} I^t(A_{\text{expert}_{ij}}^t). \tag{13}$$

3.2.2. Expert Screening

Regarding the second issue, by ensuring that the B_{expert} and A_{expert} matrices in the triplet are approximately orthogonal, each expert can independently influence the model's output while guaranteeing information uniqueness along each parameter direction. This enables effective screening of experts across different knowledge directions through a single diagonal matrix E. To maintain orthogonality of the B_{expert} and A_{expert} matrices during training, we introduce a loss constraint term L_{orthog}:

$$L_{\text{orthog}} = \mathrm{R}(B_{\text{expert}}, A_{\text{expert}}) = \| B_{\text{expert}}^\top B_{\text{expert}} - I \|_{\mathrm{F}} + \| A_{\text{expert}} A_{\text{expert}}^\top - I \|_{\mathrm{F}} . \tag{14}$$

The constraint term L_{orthog}, multiplied by an orthogonality penalty coefficient γ, is added to the original forward propagation loss L_{forward} to form the total loss for backpropagation:

$$L_{\text{total}} = L_{\text{forward}} + \gamma L_{\text{orthog}}. \tag{15}$$

Through the method in 3.2.1, each expert matrix's importance score $I^t(\zeta_i^t)$ is computed post-backpropagation per training iteration. During expert selection, matrices are ranked by these scores, with the top-k^t experts selected (k^t denoting the selection scope). Given t_{total} total training iterations with transition points t_1 and t_2, the selection scope k^t undergoes three distinct phases: it maintains an initial value k_{init} until t_1, square decay between t_1 and t_2, and finally stabilizes at k_{final} until the end. This dynamic selection strategy is formally expressed in Formula 16. Given the concentrated output distribution of LoRA experts, we set $k_{\text{init}} = r_{\text{expert}}$ and k_{final} slightly smaller than r_{expert}.

$$k^t = \begin{cases} k_{\text{init}} & 0 \le t < t_1 \\ k_{\text{final}} + (k_{\text{init}} - k_{\text{final}}) \left(1 - \frac{t - t_1}{t_{\text{total}} - t_1 - t_2}\right)^2 & t_1 \le t < t_{\text{total}} - t_2 \\ k_{\text{final}} & \text{o.w.} \end{cases} . \tag{16}$$

This phased strategy enables comprehensive expert importance exploration during initial training, prevents premature rank collapse through gradual mid-phase reduction, and leverages stabilized expert importance distributions during late-stage training to optimize expert system performance.

Consequently, during training timesteps $t \in [0, t_1)$, since the selection scope $k_{\text{init}} = r_{\text{expert}}$, the method does not require expert selection and proceeds with normal gradient updates. During training timesteps $t \in [0, t_2)$, at ΔT step intervals, the method computes

the current importance score $I^t(\zeta_i^t)$ for each expert matrix and implements expert selection by retaining diagonal elements in E corresponding to the top-k^t experts, while zeroing out others. The specific formulation is as follows:

$$\begin{aligned} E_i^{t+1} &= \Gamma\left(\widetilde{E}_i^t, I^t(\zeta_i^t)\right), \text{when } t = n\Delta T, \\ \Gamma\left(\widetilde{E}_i^t, I^t(\zeta_i^t)\right) &= \begin{cases} \widetilde{E}_i^t, & \text{if } I^t(\zeta_i^t) \text{ is in top-}k^t \text{ of } \{I^t(\zeta_j^t)\}_{1\le j\le r_{\text{expert}}} \\ 0, & \text{o.w.} \end{cases}. \end{aligned} \tag{17}$$

Where $n \in \mathbb{N}^+$. $\widetilde{E}i^{(t)}$ denotes the i-th element of the diagonal matrix E updated via backpropagation of the loss during the t-th training round:

$$\widetilde{E}^t = E^t - \eta \nabla_E \mathcal{L}\left(B_{\text{expert}}^t, E^t, A_{\text{expert}}^t\right). \tag{18}$$

During $t \in [t_2, t_{\text{total}})$, the method discontinues importance evaluation and instead fixes the selection scores to $I^{t_2}(\zeta_i^{t_2})$, maintaining the selection formula in Formula 17. This design leverages stabilized expert importance distributions while preventing training instability that could arise from stochastic gradient-induced erroneous masking of critical matrices, particularly given the constant selection scope k^t for $t \in [t_2, t_{\text{total}})$ as defined in Formula 16.

The proposed method implements masking in diagonal matrix E rather than direct matrix pruning, primarily to mitigate irrecoverable loss from erroneously pruned critical experts. When element E_i is incorrectly zeroed, its impact on the importance score $I^t(\zeta^t i)$ in Formula 13 remains negligible, preserving reactivation potential. Conversely, direct pruning of low-importance components necessitates discarding corresponding triplet elements $\zeta^t i$, which permanently precludes reactivation and causes irreversible information loss. Furthermore, since the B_{expert} and A_{expert} matrices are not strictly orthogonal, direct pruning could also adversely affect the utility of other experts.

4. Experiments

All experiments in this chapter utilized a single NVIDIA A100-PCIE (80GB) GPU. Training employed cosine-annealed AdamW optimization with LoRA adapters applied across all base model linear layers. For our method, LoRA dropout rate was set to 0.05 for expert module and 0 for shared module. Complete hyperparameters are tabulated in Table 1. BFloat16 computation was utilized for both base models and adapters to conserve resources. Unless specified, reported losses exclusively reflect forward propagation loss (L_{forward}), excluding orthogonal regularization (L_{orthog}) for cross-method comparability.

4.1. Natural Language Generation Experiments

All experiments employed the Llama 2-7B model (Touvron et al. (2023)), sequentially fine-tuned and evaluated across three specialized domains: commonsense reasoning was assessed by fine-tuning on the Commonsense 170K dataset (Hu et al. (2023)) with an 8:2 train-test split and evaluating across eight sub-datasets; mathematical capability was validated through MetaMathQA (Yu et al. (2023)) fine-tuning and GSM8K (Cobbe et al. (2021)) evaluation; coding proficiency was tested via CodeFeedback (Zheng et al. (2024)) fine-tuning followed by evaluation on HumanEval (Chen et al. (2021)) and MBPP (Austin et al.

Table 1: Training hyperparameters for natural language generation tasks

Hyperparameter	Commonsense 170k	MetaMathQA	CodeFeedback
Learning Rate	2e-5	1e-5	1e-5
Batchsize	64	64	64
Epochs	1	1	1
LoRA Rank	8	128	128
LoRA Alpha	32	128	128
r_{share}	4	64	32
r_{expert}	4	64	96
t_1 Ratio	0.2	0.15	0.15
t_2 Ratio	0.5	0.5	0.5
β_1	0.85	0.85	0.85
β_2	0.85	0.85	0.85
ΔT	10	10	10
γ	0.3	0.1	0.1
k_{final}	3	48	80

(2021)). Hyperparameters for each dataset correspond to columns in Table 1, respectively. Only 100k-sample subsets of both training datasets were used to reduce computational costs.

Table 2 comprehensively compare our method with original LoRA across commonsense reasoning and specialized domains in natural language generation. In commonsense tasks, our approach outperforms LoRA on most datasets such as social QA (BoolQ +0.6%, SIQA +0.7%, HellaSwag +0.9%) and scientific and physical QA (PIQA +0.8%, ARC-easy +7.1%, ARC-challenge +5.9%, OpenBookQA +2.5%), with the sole exception being Winogrande where it marginally underperforms. For specialized domains, our method achieves +2.3% accuracy on GSM8K mathematical problems and +9.7% on HumanEval coding tasks, though slightly underperforming on simpler MBPP coding tasks. These results evidence superior commonsense reasoning capabilities alongside marked gains in mathematical and coding proficiency versus original LoRA.

Across commonsense reasoning , mathematical reasoning , and code generation tasks (Figure 2), our method(blue curve) consistently demonstrates three convergent advantages than LoRA(orange curve): (1) accelerated initial convergence despite higher starting loss (evidenced by 15% faster descent in early commonsense training, rapid reduction below baseline within 50 code epochs); (2) superior steady-state performance with final loss reductions of 10.9% (Commonsense), 2.3% (MetaMathQA) and 4.3% (CodeFeedback) versus original LoRA; (3) enhanced gradient efficiency where lower gradient norms achieve better performance—attributed to expert selection mechanisms. These behavior stems from dual mechanisms: the expert module initially allocates gradient budget to orthogonalization constraints, elevating early loss but subsequently enabling superior resource allocation via constraint-driven expert selection, maintaining performance with reduced gradient norms; the shared module's direct optimization on principal components of pretrained weights facilitates rapid loss reduction.

Table 2: The accuracy of different methods in different dataset. The best results are denoted in bold.

Training Dataset	Trainable Parameter	Validation dataset	LoRA	Test
Commonsense	20m(0.29%)	BoolQ	68.9	**69.5**
		PIQA	80.7	**81.5**
		SIQA	77.4	**78.1**
		HellaSwag	78.1	**79.0**
		WinoGrande	**78.8**	77.1
		ARC-e	77.8	**84.9**
		ARC-c	61.3	**67.2**
		OBQA	74.8	**77.3**
		Average ACC	74.7	**76.8**
MetaMathQA	320m(4.53%)	GSM8K	36.5	**38.8**
CodeFeedback	320m(4.53%)	HumanEval	10.4	**20.1**
		MBPP	**34.4**	33.3

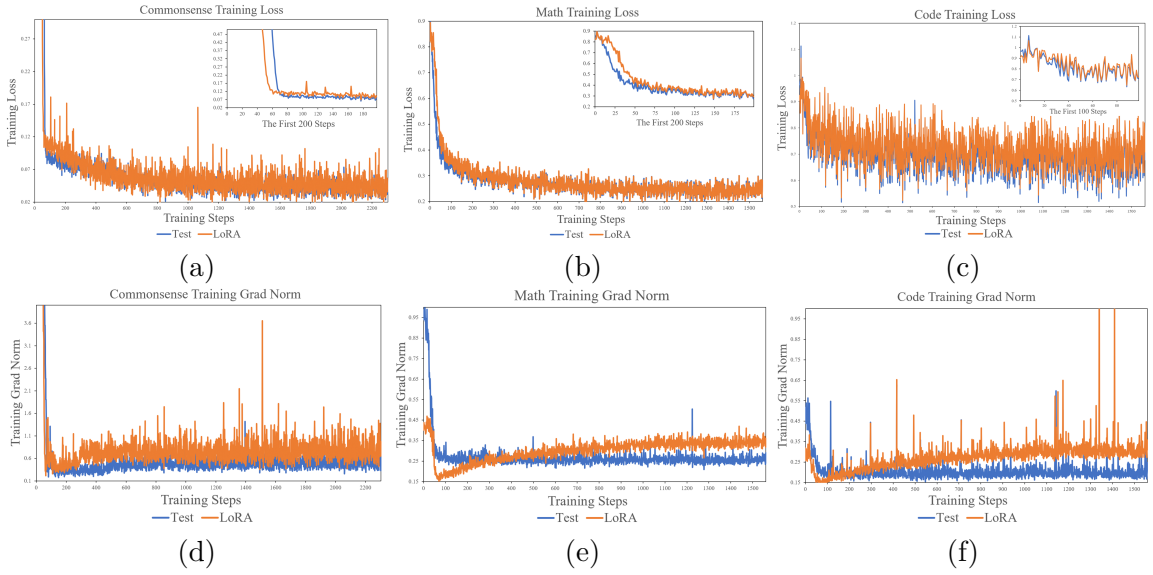

Figure 2: (a), (b) and (c) present the training loss curves for different methods on the Commonsense, MetaMathQA and CodeFeedback dataset, respectively. (d), (e) and (f) present the radient norm curves on corresponding dataset.

4.2. Low-rank Matrix Freezing Experiment

Prior studies suggest that constraining the learning rate for matrix A in low-rank adaptation may enhance fine-tuning performance (Hayou et al. (2024)). We therefore investigate the impact of weight freezing strategies on our method through four configurations: (1) origin_test: freezing only original weight matrices; (2) freeze_As_test: freezing original

weights and initialized A_{share}; (3) freeze_Ae_test: freezing original weights and A_{expert}; (4) freeze_A_test: freezing original weights and all A matrices.

Initial experiments were conducted on a simplified dataset. Using a 15k random subset from Commonsense 170k, we yield the training and test loss curves in Figure 3. To further validate low-rank matrix freezing effects at scale, we fine-tuned Llama 2-7B on 100k subsets of MetaMathQA and CodeFeedback using baseline (origin_test, blue curve) and optimal configuration in simplified dataset (freeze_As_test, orange curve). Hyperparameters correspond to columns in Table 1, respectively. Subsequent evaluation covered GSM8K, HumanEval, and MBPP.

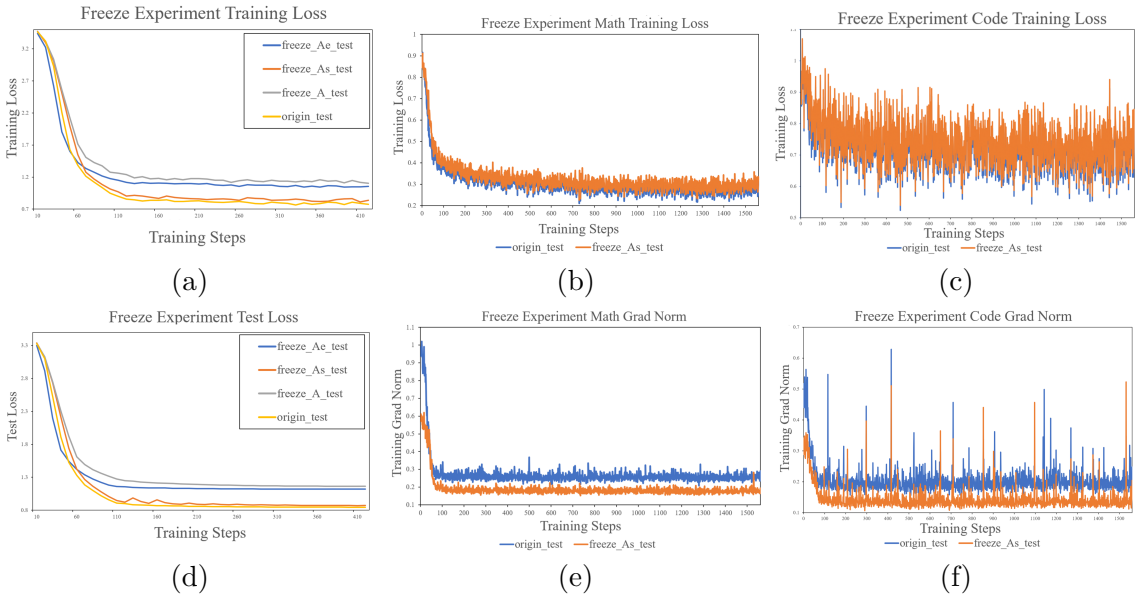

Figure 3: (a) and (d) illustrate the training loss curves and test loss curves under different weight freezing methods in the Commonsense 15k dataset. (b) and (e) present the training loss curves and gradient norm curves for different methods on the MetaMathQA dataset. (c) and (f) present the corresponding curves on the CodeFeedback dataset.

Experimental analysis reveals critical impacts of parameter matrix freezing strategies on model performance. Freezing the shared matrix A_{share} in simple scenarios causes only marginal validation loss increase ($< 3\%$), confirming its efficacy as the core carrier of pre-trained knowledge—its parameter space inherently contains principal feature directions, and freezing essentially constrains optimization within fixed low-rank subspaces. However, expert modules prove indispensable for dynamic adaptation: freezing A_{expert} triggers significant validation loss surge (32.5%), with degradation comparable to full low-rank matrix A freezing (37.6%). Table 3 further quantifies task performance differences across freezing strategies. The "freeze_As_test" row shows two entries in the "Trainable Parameter" column because distinct hyperparameter configurations were used for MetaMathQA (247M parameters) and CodeFeedback (283M parameters). Notably, complex tasks (e.g., GSM8K,

HumanEval) exhibit substantial performance degradation when freezing A_{share} (accuracy drops of 6.1% and 9.7%), while simpler tasks (e.g., MBPP) show minimal or slightly positive effects. Update trajectory visualizations (Figure 3) mechanistically explain this phenomenon: freezing A_{share} reduces gradient norms while elevating training loss, indicating constrained parameter update capacity that impedes effective knowledge adaptation.

Table 3: Experimental results of low-rank matrix freezing experiments on large datasets. The best results are denoted in bold

Method	Trainable Parameters	GSM8K	HumanEval	MBPP
freeze_ As_ test	247m(3.50%) / 283m(4.01%)	32.7	10.4	**34.1**
origin_ test	320m (4.53%)	**38.8**	**20.1**	33.3

4.3. Module Ablation Experiments

To investigate module contributions during fine-tuning, ablation experiments were conducted on 100k subsets of MetaMathQA and CodeFeedback datasets (hyperparameters is shown Table 1), evaluating via GSM8K, HumanEval, and MBPP. Four method variants were tested: (1) Test-Base(blue curve): core structure only (retaining diagonal matrix E vs. original LoRA), disabling shared initialization and dropout modification, expert assessment and selection; (2) Test-Share(orange curve): enabling shared initialization and dropout modification, disabling expert mechanisms; (3) Test-Expert(gray curve): enabling expert assessment and selection, disabling shared mechanisms; (4) Test-All(yellow curve): full module activation. Results are quantified in Table 4 and visualized in Figure 4.

Table 4: The ablation experiments' results. The best results are denoted in bold.

Method	Trainable Parameters	GSM8K	HumanEval	MBPP
Test-Base	320m (4.53%)	36.4	10.3	33.6
Test-Share	320m (4.53%)	35.6	16.5	32.3
Test-Expert	320m (4.53%)	37.1	10.4	**34.6**
Test-All	320m (4.53%)	**38.8**	**20.1**	33.3

Ablation studies reveal distinct module roles: The shared module rapidly adapts pretrained world knowledge to target domains, generating larger gradient norms and boosting accuracy on complex tasks (e.g., HumanEval +6.2%), at the cost of altering original knowledge and biasing updates toward pretrained principal components (GSM8K -0.8%, MBPP -1.3%). Expert selection enhances fundamental problem-solving (GSM8K +0.7%, optimal MBPP 34.6%) but offers marginal gains for challenging tasks (HumanEval +0.1%), allocating partial gradient budget to orthogonalization constraints for lower norms yet slower descent. Critically, joint activation (Test-All) enables synergy: the shared module updates core world knowledge while experts supplement domain-specific knowledge, achieving accelerated convergence with superior update directions (HumanEval 20.1%, GSM8K 38.8%), validating complementary world/domain knowledge modeling mechanisms.

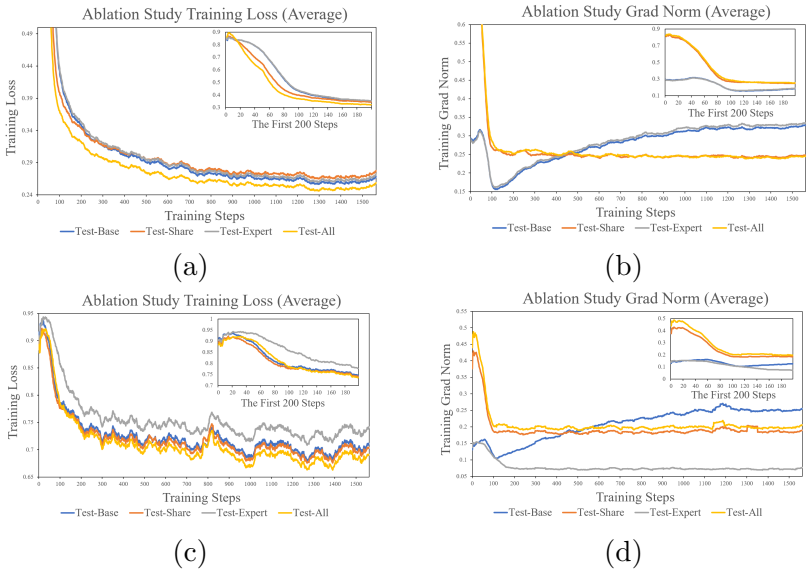

Figure 4: (a) and (b) present the training loss and gradient norm curves of different methods in the ablation experiments on MetaMathQA dataset. (c) and (d) present the corresponding curves of respective methods on CodeFeedback dataset. Experimental results average training loss and gradient norms over 50-step cycles.

4.4. Comparison of Training Costs

As described in the methodology (Section 3) and experiments (Subsection 4.1), our dual-module architecture enhances original low-rank adaptation, achieving superior fine-tuning performance across datasets, while introducing computational overhead from shared module initialization and expert screening. To quantify trade-offs, we measure training time and memory consumption versus baseline under identical hyperparameters on Llama 2-7B, isolating costs via four configurations: Test-Base (no modules), Test-Share (shared module only), Test-Expert (expert module only), Test-All (full system). Experiments used Commonsense 170K, plus 100k subsets of MetaMathQA and CodeFeedback to mitigate computational demands. Hperparameters for each dataset detailed in Table 1's respective dataset column.

As shown in Table 5, which compares training time and GPU memory usage across methods. Regarding training time, the proposed method introduces negligible overhead in shared module initialization compared to Test-Base, thanks to the Fast SVD technique (Halko et al. (2011)). However, the expert filtering mechanism in Test-Expert incurs an 8% increase due to importance evaluation and screening, resulting in a 9% total overhead for the full implementation (Test-All). For GPU memory usage, the shared module's principal component initialization and dropout adjustment add negligible additional usage overhead, while the expert module's importance score computation increases memory consumption by 5%. Consequently, the complete framework (Test-All) requires 5% more maximum GPU memory than the baseline. These results indicate that our method does not impose significant training time or memory overhead. Primarily because: (1) additional computa-

tions occur exclusively during shared module initialization; (2) expert incremental matrices ΔW_{expert} importance screening executes only during specific training periods; (3) low-rank parameters remain orders-of-magnitude smaller than pretrained weights. Updating these matrices' importance scores entails negligible computational cost compared to full-model forward-backward propagation.

Table 5: Comparison of training time and memory cost

	Dataset	Test-Base	Test-Share	Test-Expert	Test-All
Time cost(min)	Commonsense	222	226	264	267
	MetaMathQA	247	250	260	261
	CodeFeedback	269	271	277	279
Memory cost(Gib)	Commonsense	32.4	32.4	32.6	32.6
	MetaMathQA	53.4	53.4	56.3	56.2
	CodeFeedback	49.6	49.6	52.8	52.8

5. Conclusions and Perspectives

This paper enhances Low-Rank Adaptation (LoRA) through a dual-module architecture. The shared module initializes with principal components of pretrained weights, allocating residuals to original matrices. Zero-dropout training in this module prioritizes world knowledge updates for accelerated convergence. The expert module enforces low-rank matrix orthogonality via loss constraints, computes sensitivity-based importance scores for ΔW_{expert}, and selectively updates important dimensions to optimize budget allocation for specialized task accuracy. Cross-domain validation (commonsense reasoning, mathematics, programming) with module freezing and training costs experiments confirms its superiority.

However, several unresolved issues persist: (1) Current application is limited to language model linear layers. Systematic investigation is still required to determine whether the method can be adapted to convolutional layers to enhance the performance of visual models. (2) The expert module's importance ranking necessitates extensive manual hyperparameter tuning, whose sensitivity significantly impacts model outputs. Answers to these questions are currently being actively explored.

References

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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

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

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

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

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304. 01933*, 2023.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of machine learning research*, 18(187):1–30, 2018.

Chen Liang, Simiao Zuo, Minshuo Chen, Haoming Jiang, Xiaodong Liu, Pengcheng He, Tuo Zhao, and Weizhu Chen. Super tickets in pre-trained language models: From model compression to improving generalization. *arXiv preprint arXiv:2105.12002*, 2021.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. MoELoRA: Contrastive learning guided mixture of experts on parameter-efficient fine-tuning for large language models. *arXiv preprint arXiv:2402.12851*, 2024.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning LLMs from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*, 2024.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Jiahuan Pei. MELoRA: Mini-ensemble low-rank adapters for parameter-efficient fine-tuning. *arXiv preprint arXiv:2402.17263*, 2024.

Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv: 1701.06538*, 2017.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*, 2025.

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. PLATON: Pruning large transformer models with upper confidence bound of weight importance. In *International conference on machine learning*, pages 26809–26823. PMLR, 2022.

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

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. OpenCodeInterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

Hongyun Zhou, Xiangyu Lu, Wang Xu, Conghui Zhu, Tiejun Zhao, and Muyun Yang. LoRA-drop: Efficient LoRA parameter pruning based on output evaluation. *arXiv preprint arXiv:2402.07721*, 2024.