Parameter-Efficient Fine-Tuning via Circular Convolution

Aochuan Chen[♂*] **Jiashun Cheng**[♂*] **Zijing Liu**[idea] **Ziqi Gao**[♂] **Fugee Tsung**[♂] **Yu Li**[idea] **Jia Li**[♂†]
[♂]The Hong Kong University of Science and Technology (Guangzhou)
The Hong Kong University of Science and Technology
[idea]International Digital Economy Academy
jialee@hkust-gz.edu.cn

Abstract

Low-Rank Adaptation (LoRA) has gained popularity for fine-tuning large foundation models, leveraging low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ to represent weight changes (*i.e.,* $\Delta\mathbf{W} = \mathbf{BA}$). This method reduces trainable parameters and mitigates heavy memory consumption associated with full delta matrices by sequentially multiplying $\mathbf{A}$ and $\mathbf{B}$ with the activation. Despite its success, the intrinsic low-rank characteristic may limit its performance. Although several variants have been proposed to address this issue, they often overlook the crucial computational and memory efficiency brought by LoRA. In this paper, we propose Circular Convolution Adaptation (C^3A), which not only achieves high-rank adaptation with enhanced performance but also excels in both computational power and memory utilization. Extensive experiments demonstrate that C^3A consistently outperforms LoRA and its variants across various fine-tuning tasks.

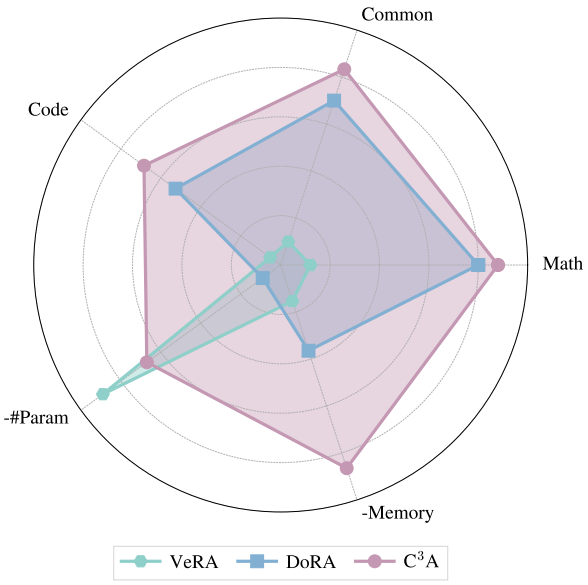

Figure 1: Performance comparison of C^3A and other methods relative to LoRA on LLaMA-8B. Higher values are better across all metrics. See Table 3 and Table 4 for more statistics.

1 Introduction

In recent years, Large Foundation Models (LFMs) have witnessed a pronounced ascendance in both scholarly and practical realms, attributable to their exceptional efficacy across diverse tasks in natural language processing (NLP) (Brown et al., 2020; Touvron et al., 2023), computer vision (CV) (Radford et al., 2021; Kirillov et al., 2023), and other domains (Li et al., 2024a,b, 2025). Distinguished by an extensive parameter count and significant computational requisites, these models have established unprecedented benchmarks in both accuracy and versatility. Nonetheless, their considerable size and intricate structure present formidable obstacles for efficient fine-tuning, especially within resource-constrained environments (Malladi et al., 2023; Zhang et al., 2024b). To mitigate these challenges, parameter-efficient fine-tuning (PEFT) techniques (Mangrulkar et al., 2022), exemplified by Low-Rank Adaptation (LoRA) (Hu et al., 2021), have emerged as highly effective solutions.

LoRA reduces the number of trainable parameters by leveraging low-rank matrices to approximate alterations in weights, thereby facilitating fine-tuning without degrading the model's efficacy. Specifically, LoRA can be articulated mathematically as follows:

$$\mathbf{Wx} = (\mathbf{W}_0 + \Delta\mathbf{W})\mathbf{x} = \mathbf{W}_0\mathbf{x} + \mathbf{B}(\mathbf{Ax}),$$

where $\mathbf{W}, \mathbf{W}_0, \Delta\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ are weight matrices, $\mathbf{B} \in \mathbb{R}^{d_1 \times r}$, $\mathbf{A} \in \mathbb{R}^{r \times d_2}$ are low-rank matrices formulated to construct $\Delta\mathbf{W}$, and $\mathbf{x} \in \mathbb{R}^{d_2}$ are the activations. The number of trainable parameters is $r(d_1 + d_2)$, thereby motivating the selection of $r \ll \min(d_1, d_2)$ (*e.g.,* $r = 8$ for $d_1 = d_2 = 1024$) to attain elevated parameter efficiency. Nonethe-

[*]Equal contribution.
[†]Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2025, pages 2004–2019

less, as elaborated by Zeng and Lee (2023), the potential of LoRA to encapsulate a target model is inherently constrained by r. In an effort to reconcile the dichotomy between performance and efficiency, Kopiczko et al. (2023) introduced Vector Random Matrix Adaptation (VeRA). VeRA attains comparable performance with a markedly reduced count of trainable parameters via fixed random-matrix projections. However, despite its minimal parameter count, VeRA demands considerable computational resources and memory capacity due to the extensive nature of the random matrices employed for projection. As depicted in Figure 1, other representative works share the same resource problem. This precipitates the following open research question within the scope of PEFT:

Beyond low parameter counts, how to achieve high-rank adaptation without incurring significant costs of time and memory?

To address this question, we introduce Circular Convolution Adaptation (C^3A), which incorporates the circular convolution operator (Bamieh, 2018). Circular convolution has garnered significant attention in both signal processing (Li et al., 2020) and cryptography (Dworkin et al., 2001) due to its exceptional efficiency and compactness. This operator can be equivalently expressed as multiplication by a circulant matrix, providing rank flexibility that is independent of the number of trainable parameters. Furthermore, by employing the Fast Fourier Transform (FFT), C^3A achieves superior time and memory efficiency compared to the direct multiplication (Bamieh, 2018), which makes it competitive with LoRA in terms of efficiency.

In addition, as explicated by Dosovitskiy et al. (2020), dense linear layers exhibit a deficiency of inductive biases, engendering a complex optimization landscape. Consequently, this hampers the effectiveness of transformers in comparison to Convolutional Neural Networks (CNNs) under conditions of limited data availability. Within the framework of a constrained training dataset for the downstream task, we postulate that a robust inductive bias could potentially augment adaption performance. The circular pattern in C^3A serves precisely as such an inductive bias.

In summary, circular convolution presents a promising solution for circumventing the rank limitations of LoRA at minimal costs. Our contributions can be summarized as follows:

❶ We introduce C^3A, a novel approach for PEFT. This method leverages the circular convolution operation and its equivalent circulant matrix to provide a flexible rank, which is free of linear constraint by the number of trainable parameters.

❷ Leveraging the elegant diagonalization of the circulant matrix, we implement both the forward pass and backpropagation using FFT. With the incorporation of FFT, the computation and memory efficiency of C^3A excels. C^3A strikes a unique balance between performance an efficiency.

❸ To offer greater flexibility in controlling the number of trainable parameters, we extend C^3A by incorporating block-circular convolution, which results in block-circulant matrices. This extension allows C^3A to achieve fully customizable parameter counts as well as adaptable rank configurations.

❹ We validate C^3A through comprehensive fine-tuning experiments across diverse tasks, which demonstrate C^3A's outstanding accuracy and memory merits compared to existing methods.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning

Research on PEFT has generally progressed along three main directions. The first direction involves partially updating the pre-trained neural network (*e.g.,* the layer norm (Basu et al., 2024) or the biases (Zaken et al., 2021)). Traditional methods relied on hand-crafted heuristics (Raghu et al., 2019) to identify which parameters are crucial and should be fine-tuned. More advanced approaches employ optimization techniques (Guo et al., 2020; Xu et al., 2021; Fu et al., 2023). For example, Guo et al. (2020) reformulated such a discrete optimization problem into a continuous one by employing Bernoulli masks and the Gumbel-softmax approximation (Jang et al., 2016).

The second direction emerged to maintain the integrity of the pre-trained model while enabling a high degree of parameter sharing through adapter-based methods (He et al., 2021; Rebuffi et al., 2017; Rücklé et al., 2020; Liu et al., 2022; Lian et al., 2022). These works focus on integrating additional modules, termed adapters, to fit the downstream task, effectively decoupling the pre-trained model parameters from those specific to the downstream task. Prompt Tuning (Brown et al., 2020; Gao et al., 2020; Chen et al., 2023; Zhang et al., 2024a) and Prefix Tuning (Li and Liang, 2021; Jia et al., 2022) also fall into this category, despite ignoring potential semantic meanings.

The final direction is characterized by delta-weight-based methods, such as Low-Rank Adaptation (LoRA) (Hu et al., 2021) and Orthogonal Fine-tuning (OFT) (Qiu et al., 2023). These methods bridge the gap between the pre-trained model and the downstream task by adaptive delta weights, which are stored separately while used in combination with the pre-trained weights. This unique design enables disentanglement of the pretrained and downstream-specific weights. Namely, it achieves parameter sharing and preserves the ability to integrate without additional inference cost. LoRA models the delta-weights by an additive matrix while OFT does it by a multiplicative one. To further improve either parameter efficiency or performance, many variants has been proposed for both of the methods (Kopiczko et al., 2023; Liu et al., 2024b, 2023; Yuan et al., 2024; Hayou et al., 2024b). However, these methods can hardly achieve high parameter efficiency and performance without incurring heavy computation and memory usage.

2.2 Circular Convolution

Circular convolution has been extensively studied in signal processing (Rabiner et al., 1978; McGillem and Cooper, 1984; Li et al., 2020) and cryptography (Dworkin et al., 2001; Gong et al., 2024). Owing to its computational advantages, circular convolution has also been explored in machine learning for generating long embeddings of high-dimensional data (Yu et al., 2014) and compressing heavily parameterized layers (Cheng et al., 2015; Ding et al., 2017). Remarkably, it achieves these efficiencies without significant performance degradation, which makes it a promising technique for fine-tuning applications.

Despite its success in small neural networks such as LeNet (Cheng et al., 2015), circular convolution has not demonstrated lossless performance in modern large foundational models (LFMs) or even in their base architecture, the transformer. This limitation may be attributed to the conflict between its high intrinsic bias (*i.e.,* the circulant pattern) and the vast amount of data required for training LFMs. Conversely, when fine-tuning LFMs, it is often impractical to collect as much data as needed for training from scratch. In such scenarios, the intrinsic bias of circular convolution could potentially serve as a regularization mechanism, thereby benefiting the optimization process of fine-tuning.

3 Method

In this section, we present C^{3}A (see an overview in Figure 2), a novel PEFT approach based on the circular convolution. C^{3}A follows LoRA's setting of learning an additive linear operation over the original dense linear transformation. However, instead of using low-rank decomposition and the matrix multiplication operator, C^{3}A resorts to circular convolution as this additive linear operation.

3.1 Notations

The adapted weight matrix, the original weight matrix, and the delta matrix are denoted by $\mathbf{W}$, $\mathbf{W}_0$, and $\Delta\mathbf{W}$, respectively ($\mathbf{W}, \mathbf{W}_0, \Delta\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$). The activation vector of the previous layer is denoted by $\mathbf{x} \in \mathbb{R}^{d_2}$. The post-transformation vector is $\mathbf{z}$, where $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{d_1}$, and the incremental part is denoted by $\Delta\mathbf{z}$, where $\Delta\mathbf{z} = \Delta\mathbf{W}\mathbf{x} \in \mathbb{R}^{d_1}$. The matrices $\mathbf{A}$ and $\mathbf{B}$ are low-rank matrices introduced by LoRA to represent $\Delta\mathbf{W}$, with r being their rank. r_v specifies the rank of the random projection matrix used in VeRA. The circular convolution kernel of C^{3}A is denoted by $\Delta\mathbf{w}$ and the circular convolution operator by $\star$. The loss function is represented by $\mathcal{L}$. The Fast Fourier Transform and its inverse are denoted by FFT and iFFT. The Hadamard product is denoted by $\circ$.

3.2 Circular Convolution

Firstly, for simplicity, we assume $d_1 = d_2 = d$ and $\Delta\mathbf{w} \in \mathbb{R}^d$. The circular convolution operator is defined as $\Delta\mathbf{z} = \Delta\mathbf{w} \star \mathbf{x} = \mathcal{C}(\Delta\mathbf{w})\mathbf{x}$, where $\mathcal{C}(\cdot)$ is a function which takes a vector and outputs the corresponding circulant matrix. Concretely, the first row of $\mathcal{C}(\Delta\mathbf{w})$ is $\Delta\mathbf{w}$ and the following rows are equal to the row above them periodically shifted to the right by one element. In math,

$$\mathcal{C}(\Delta\mathbf{w}) = \begin{bmatrix} \Delta w_1 & \Delta w_2 & \cdots & \Delta w_{d-1} & \Delta w_d \\ \Delta w_d & \Delta w_1 & \cdots & \Delta w_{d-2} & \Delta w_{d-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \Delta w_3 & \Delta w_4 & \cdots & \Delta w_1 & \Delta w_2 \\ \Delta w_2 & \Delta w_3 & \cdots & \Delta w_d & \Delta w_1 \end{bmatrix}.$$

Theoretically, the rank of $\mathcal{C}(\Delta\mathbf{w})$ is given by $d - \mathrm{Deg}(\gcd(f(x), x^d - 1))$ (Ingleton, 1956), where $\mathrm{Deg}(\cdot)$ denotes the degree of a polynomial, $f(x)$ is the polynomial associated with $\Delta\mathbf{w}$ (*i.e.,* $f(x) = \sum_{i=1}^{d} \Delta w_i x^{i-1}$), and $\gcd(\cdot)$ represents the greatest common divisor. Consequently, the theoretical upper bound on the rank of $\mathcal{C}(\Delta\mathbf{w})$ is d. By learning $\Delta\mathbf{w}$ in the $\mathbb{R}^n$ oracle, C^{3}A automatically achieves dynamic rank selection, which is

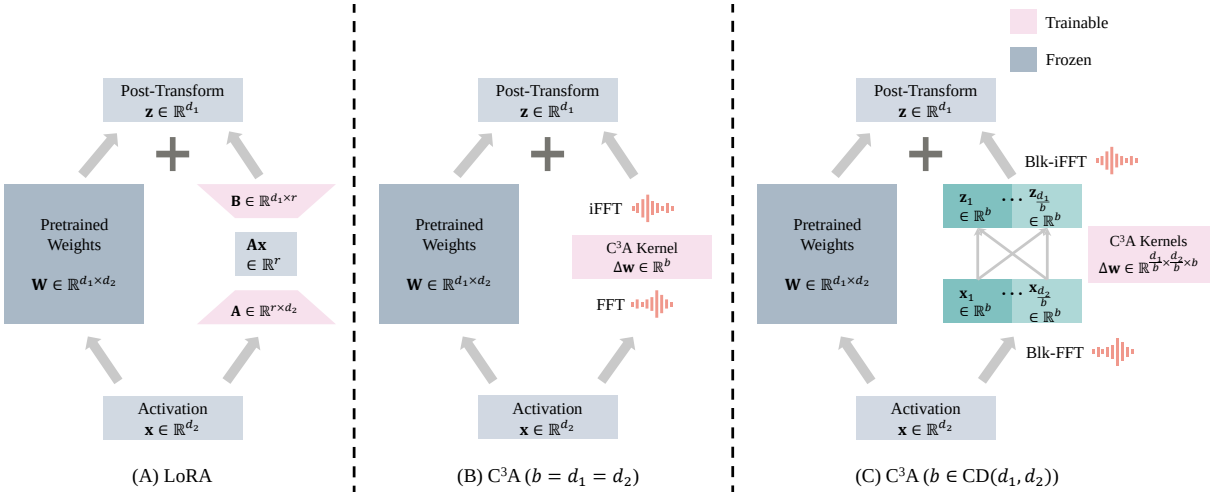

Figure 2: **Overview of LoRA (A) and our C^{3}A (B,C) method.** In LoRA, only low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ are trained and the delta weight is represented by their product (*i.e.,* $\Delta\mathbf{W} = \mathbf{BA}$). The total trainable parameter number is $r(d_1 + d_2)$, which is assosiated with the rank of the delta weight. In C^{3}A, circular convolution kernels $\Delta\mathbf{w}$ are tuned to adapt to the downstream task and the delta weight is represented by the (block-)circular matrix they construct (*i.e.,* $\Delta\mathbf{W} = \mathcal{C}_{(\mathrm{blk})}(\Delta\mathbf{w})$). The total trainable parameter count is $\frac{d_1 d_2}{b}$, which disentangles with the rank of the delta weight. Here, b is the block size and it should be a common divisor (CD) of d_1 and d_2.

not linearly constrained by the number of learnable parameters, unlike LoRA.

To achieve high efficiency, enlightened by Ding et al. (2017), we leverage the beautiful circulant structure of $\mathcal{C}(\Delta\mathbf{w})$, which makes it diagonalizable by the Fourier basis ($\mathbf{F}$) . In math, it can be described as $\mathcal{C}(\Delta\mathbf{w}) = \mathbf{F}\frac{\Lambda}{d}\mathbf{F}^{-1}$ (Golub and Van Loan, 1996), where Λ is its eigenvalues and can be calculated by a Fourier transform of the first row (*i.e.,* $\Lambda = \mathrm{diag}(\mathbf{F}\Delta\mathbf{w})$). Therefore, we can calculate $\Delta\mathbf{w} \star \mathbf{x}$ as

$$\begin{aligned}\Delta\mathbf{w} \star \mathbf{x} &= \mathbf{F}\mathrm{diag}(\frac{\mathbf{F}\Delta\mathbf{w}}{d})\mathbf{F}^{-1}\mathbf{x} \\ &= \mathrm{FFT}(\mathrm{FFT}(\Delta\mathbf{w}) \circ \mathrm{iFFT}(\mathbf{x})).\end{aligned} \tag{1}$$

3.3 Backpropagation

To effectuate backpropagation with optimal efficiency, it is imperative to obtain the analytical derivatives of the loss function $\mathcal{L}$ with respect to $\Delta\mathbf{w}$ and $\mathbf{x}$. Following the approach outlined in Ding et al. (2017), we aim to explain backpropagation using simpler language. By applying the chain rule, these derivatives are delineated as follows:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \frac{\partial\Delta\mathbf{z}}{\partial\mathbf{x}}\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}, \qquad \frac{\partial\mathcal{L}}{\partial\Delta\mathbf{w}} = \frac{\partial\Delta\mathbf{z}}{\partial\Delta\mathbf{w}}\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}. \tag{2}$$

Given that $\Delta\mathbf{z} = \mathcal{C}(\Delta\mathbf{w})\mathbf{x}$, it logically follows that $\frac{\partial\Delta\mathbf{z}}{\partial\mathbf{x}} = \mathcal{C}(\Delta\mathbf{w})$. Concerning $\frac{\partial\Delta\mathbf{z}}{\partial\Delta\mathbf{w}}$, we observe the commutative property of the circular convolution operation (*i.e.,* $\mathcal{C}(\Delta\mathbf{w})\mathbf{x} = \mathcal{C}(\mathbf{x})\Delta\mathbf{w}$), which implies $\frac{\partial\Delta\mathbf{z}}{\partial\Delta\mathbf{w}} = \mathcal{C}(\mathbf{x})$. Substituting these findings into Equation 2, we derive:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \mathcal{C}(\Delta\mathbf{w})\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}, \qquad \frac{\partial\mathcal{L}}{\partial\Delta\mathbf{w}} = \mathcal{C}(\mathbf{x})\frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}.$$

These expressions can also be interpreted as circular convolutions:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \Delta\mathbf{w} \star \frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}, \qquad \frac{\partial\mathcal{L}}{\partial\Delta\mathbf{w}} = \mathbf{x} \star \frac{\partial\mathcal{L}}{\partial\Delta\mathbf{z}}.$$

By meticulously executing this derivative computation in accordance with Equation 1, backpropagation can harness the computational efficacy facilitated by the FFT algorithm.

3.4 Block-Circular Convolution

Notwithstanding the elegance and efficiency of the circular convolution operator, it is subject to two fundamental limitations stemming from the constraint that the convolution kernel must match the dimensions of the activation vector: ① *It is inapplicable to non-square weight matrices.* ② *The count of learnable parameters remains fixed.* The first restriction hampers its applicability in scenarios such as fine-tuning a LLaMA3-8B model, where the weight matrix dimensions include 4096×1024. The second constraint diminishes the adaptability of C^{3}A, presenting challenges in addressing complex downstream tasks that necessitate a greater number of learnable parameters. To mitigate these limitations, we employ block-circular convolution (Ding et al., 2017). By partitioning the activation vector $\mathbf{x}$ and the post-transformation vector $\Delta\mathbf{z}$

into blocks of identical size, unique convolution kernels can be allocated to each pair of these blocks. Specifically,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{\frac{d_2}{b}} \end{bmatrix}$$
$$\Delta\mathbf{z} = \begin{bmatrix} \Delta\mathbf{z}_1 & \Delta\mathbf{z}_2 & \cdots & \Delta\mathbf{z}_{\frac{d_1}{b}} \end{bmatrix},$$

where b is the block size and b need to be a common divisor of d_1 and d_2. We will need $\frac{d_1 d_2}{b^2}$ convolution kernels to densely connect these blocks, which can be expressed in math as

$$\Delta\mathbf{z}_i = \sum_{j=1}^{\frac{d_2}{b}} \Delta\mathbf{w}_{ij} \star \mathbf{x}_j, i \in \{1, 2, \cdots, \frac{d_1}{b}\}.$$

This calculation can be represented by a block-circular matrix:

$$\Delta\mathbf{z} = \mathcal{C}_{\text{blk}}(\Delta\mathbf{w})\mathbf{x} \tag{3}$$
$$\mathcal{C}_{\text{blk}}(\Delta\mathbf{w}) = \begin{bmatrix} \mathcal{C}(\Delta\mathbf{w}_{11}) & \cdots & \mathcal{C}(\Delta\mathbf{w}_{1\,\frac{d_2}{b}}) \\ \mathcal{C}(\Delta\mathbf{w}_{21}) & \cdots & \mathcal{C}(\Delta\mathbf{w}_{2\,\frac{d_2}{b}}) \\ \cdots & \cdots & \cdots \\ \mathcal{C}(\Delta\mathbf{w}_{\frac{d_1}{b}1}) & \cdots & \mathcal{C}(\Delta\mathbf{w}_{\frac{d_1}{b}\,\frac{d_2}{b}}) \end{bmatrix}. \tag{4}$$

We refer our readers to Algorithm A1 in Appendix A for a Pytorch implementation. In this context, $\Delta\mathbf{w}_{ij} \in \mathbb{R}^b$, and it follows that $\frac{d_1 d_2}{b^2} b = \frac{d_1 d_2}{b}$ represents the number of learnable parameters. Notably, the parameter b serves as a hyperparameter modulating the quantity of learnable parameters, analogous to the role of r in LoRA. It is imperative to distinguish, however, that whereas r simultaneously governs the rank of the delta matrix and the number of learnable parameters, b exclusively influences the latter. This disentanglement of matrix rank and parameter count facilitates greater adaptability and potentially yields superior outcomes.

3.5 Complexity Analysis

We compare the time complexity and space complexity of LoRA, VeRA and C^{3}A in Table 1. Detailed analysis follows in this section.

3.5.1 Time Complexity

LoRA integrates low-rank matrices $\mathbf{A}$ and $\mathbf{B}$, which are successively multiplied with the activation vector, resulting in a computational complexity of $\mathcal{O}(r(d_1 + d_2))$. Generally, $r \ll \min(d_1, d_2)$. In contrast, VeRA, despite its high-rank structure and relatively few trainable parameters, suffers from a prohibitive computational complexity of $\mathcal{O}(r_v(d_1 + d_2))$, where r_v can exceed $\max(d_1, d_2)$. Consequently, striking an optimal balance between high rank and computational efficiency remains an elusive task.

On GPUs, the cuFFT backend automatically parallelizes FFT operations along the axes not being transformed, with the degree of parallelism p determined by the available resources. Thanks to the $\mathcal{O}(n \log n)$ complexity of the FFT algorithm used in Equation 1, C^{3}A achieves a time complexity of $\mathcal{O}(\frac{(d_1+d_2)}{p} \log b + \frac{d_1 d_2}{b})$. The first term is the time complexity for FFT and the second term is for aggregation. In practical scenarios, b is chosen as the greatest common divisor of d_1 and d_2 to achieve a high compression ratio. Given that, C^{3}A is comparable to LoRA in time complexity.

Method	Time	Space	
		# Param	# Other
LoRA	$\mathcal{O}(r(d_1 + d_2))$	$r(d_1 + d_2)$	0
VeRA	$\mathcal{O}(r_v(d_1 + d_2))$	$r_v + d_1$	$r_v(d_1 + d_2)$
C^{3}A	$\mathcal{O}(\frac{d_1+d_2}{p} \log b + \frac{d_1 d_2}{b})$	$\frac{d_1 d_2}{b}$	pb

Table 1: Time and space complexity comparison of LoRA, VeRA and C^{3}A. We split the space complexity into Parameter number and Other auxiliary tensors to help better understand the differences. We highlight that in practice, to achieve similar performance, $\frac{\max(d_1,d_2)}{b} \leq r \ll r_v$.

3.5.2 Space Complexity

We analyze the space complexity of LoRA, VeRA, and C^{3}A during training. The differences among these methods primarily arise from the trainable parameters and the auxiliary tensors required for the forward pass and backpropagation. LoRA does not rely on auxiliary tensors, while VeRA necessitates 2 random projection matrices, with a total size of $r_v(d_1 + d_2)$. Since r_v is by no means negligible, the memory usage of VeRA is significantly larger than that of LoRA.

In terms of C^{3}A, the only additional auxiliary tensor would be of size $pb \leq \min(d_1, d_2)$, which is reserved by the FFT algorithm. By selecting an appropriate b, which is often close to the greatest common divisor of d_1 and d_2, the space complexity of C^{3}A is optimized. Furthermore, because p scales with the available resources, the algorithm inherently manages dynamic memory consumption without additional effort.

4 Experiment

Baselines. We compare our C^{3}A with several representative PEFT methods, including BitFit (Zaken

	Methods	# Params	Mem	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
BASE	Full	124M	17.19G	$94.01_{\pm 0.39}$	$\mathbf{87.10}_{\pm 0.79}$	$62.00_{\pm 1.16}$	$92.40_{\pm 0.28}$	$\mathbf{77.33}_{\pm 2.68}$	$\mathbf{90.70}_{\pm 0.14}$	**83.92**
	BitFit	0.102M	12.60G	$93.30_{\pm 0.30}$	$85.80_{\pm 0.21}$	$59.21_{\pm 1.74}$	$91.96_{\pm 0.18}$	$73.07_{\pm 1.34}$	$90.18_{\pm 0.17}$	82.25
	$(IA)^3$	0.111M	19.86G	$92.98_{\pm 0.34}$	$85.86_{\pm 0.59}$	$60.49_{\pm 1.09}$	$91.56_{\pm 0.17}$	$69.10_{\pm 1.18}$	$90.06_{\pm 0.21}$	81.67
	$\text{LoRA}_{r=8}$	0.295M	13.75G	$\mathbf{94.50}_{\pm 0.41}$	$85.68_{\pm 0.74}$	$60.95_{\pm 1.57}$	$\mathbf{92.54}_{\pm 0.20}$	$76.68_{\pm 1.42}$	$89.76_{\pm 0.39}$	83.35
	$\text{VeRA}_{r=1024}$	0.043M	15.51G	$93.97_{\pm 0.17}$	$86.23_{\pm 0.41}$	$62.24_{\pm 1.91}$	$91.85_{\pm 0.17}$	$75.74_{\pm 1.56}$	$90.27_{\pm 0.25}$	83.38
	$\text{BOFT}^{m=2}_{b=8}$	0.166M	14.11G	$93.23_{\pm 0.50}$	$84.37_{\pm 0.54}$	$59.50_{\pm 1.25}$	$91.69_{\pm 0.12}$	$74.22_{\pm 0.84}$	$89.63_{\pm 0.37}$	82.11
	$\text{C}^3\text{A}_{b=768/1}$	0.018M	12.83G	$93.42_{\pm 0.26}$	$86.33_{\pm 0.32}$	$61.83_{\pm 0.96}$	$91.83_{\pm 0.04}$	$76.17_{\pm 1.39}$	$90.46_{\pm 0.29}$	83.34
	$\text{C}^3\text{A}_{b=768/6}$	0.111M	12.72G	$94.20_{\pm 0.16}$	$86.67_{\pm 0.54}$	$\mathbf{62.48}_{\pm 1.20}$	$92.32_{\pm 0.25}$	$77.18_{\pm 1.41}$	$90.16_{\pm 0.42}$	83.84
LARGE	Full	354M	43.40G	$95.75_{\pm 0.45}$	$\mathbf{88.35}_{\pm 0.64}$	$64.87_{\pm 1.25}$	$92.40_{\pm 0.28}$	$84.48_{\pm 1.14}$	$91.65_{\pm 0.14}$	86.25
	BitFit	0.271M	30.65G	$95.09_{\pm 0.27}$	$88.10_{\pm 0.76}$	$65.40_{\pm 0.76}$	$94.06_{\pm 0.14}$	$82.60_{\pm 1.15}$	$91.73_{\pm 0.20}$	86.16
	$(IA)^3$	0.295M	48.81G	$95.32_{\pm 0.20}$	$87.06_{\pm 0.57}$	$66.52_{\pm 1.10}$	$94.18_{\pm 0.15}$	$84.33_{\pm 2.38}$	$91.58_{\pm 0.39}$	86.50
	$\text{LoRA}_{r=8}$	0.786M	34.12G	$95.53_{\pm 0.35}$	$86.12_{\pm 0.86}$	$65.16_{\pm 0.76}$	$93.73_{\pm 0.30}$	$83.75_{\pm 0.51}$	$91.46_{\pm 0.21}$	85.96
	$\text{VeRA}_{r=256}$	0.061M	34.16G	$\mathbf{95.83}_{\pm 0.43}$	$87.72_{\pm 0.55}$	$63.66_{\pm 1.45}$	$94.11_{\pm 0.20}$	$83.03_{\pm 1.65}$	$91.12_{\pm 0.37}$	85.91
	$\text{BOFT}^{m=2}_{b=8}$	0.442M	34.98G	$95.76_{\pm 0.41}$	$88.28_{\pm 0.33}$	$64.72_{\pm 2.37}$	$93.89_{\pm 0.14}$	$82.82_{\pm 1.40}$	$91.03_{\pm 0.32}$	86.08
	$\text{C}^3\text{A}_{b=1024/1}$	0.049M	31.83G	$95.78_{\pm 0.05}$	$88.02_{\pm 0.62}$	$66.59_{\pm 1.20}$	$94.22_{\pm 0.25}$	$82.89_{\pm 0.67}$	$\mathbf{91.86}_{\pm 0.14}$	86.56
	$\text{C}^3\text{A}_{b=1024/8}$	0.393M	31.79G	$95.78_{\pm 0.15}$	$88.09_{\pm 0.47}$	$\mathbf{67.18}_{\pm 1.92}$	$\mathbf{94.26}_{\pm 0.19}$	$\mathbf{84.62}_{\pm 1.36}$	$91.81_{\pm 0.36}$	**86.96**

Table 2: Performance of different PEFT methods on the GLUE benchmark. We fine-tune pre-trained RoBERTa-Base and -Large models on 6 datasets. We report the Matthew's Correlation Coefficient (MCC) for CoLA, Pearson Correlation Coefficient (PCC) for STS-B, and accuracy (Acc.) for all the remaining tasks. For each metric, a higher score indicates better performance. "Avg." denotes the average score of each method across all datasets. The best results for each dataset are highlighted in **bold**. # Params does not include the classification head since each method uses a head of the same size. Memory cost (Mem) is measured on fixed length (*i.e.,* 256) data with a batchsize of 64.

et al., 2021), $(IA)^3$ (Liu et al., 2022), LoRA (Hu et al., 2021), VeRA (Kopiczko et al., 2023), BOFT (Liu et al., 2023) and DoRA (Liu et al., 2024b). BitFit fine-tunes biases. $(IA)^3$ is a SOTA method adding adapters. LoRA uses low-rank decomposition to compress additive delta matrices. VeRA reduces LoRA's trainable parameters while maintaining a high rank. BOFT compresses multiplicative delta matrices via orthogonal decomposition and butterfly factorization. DoRA separately learns magnitude and direction of delta matrices.

4.1 GLUE Benchmark

Settings. We assess our C^3A on the GLUE benchmark (Wang et al., 2018) covering various tasks like single-sentence classification, similarity, paraphrase, and inference. See Table A3 in Appendix C for more details. Datasets are split into train, validation, and test sets. Models are chosen based on validation performance and evaluated on the test set. We fine-tune RoBERTa-Base and RoBERTa-Large models (Liu et al., 2019). Unique hyperparameters are from original papers (*e.g.,* for VeRA's r and BOFT's b and m). Count of trainable parameters (# Params) exclude the classification head, which is uniform across methods. Shared hyperparameters (*i.e.,* learning rates for the classification head and other parameters) are determined by hyperparameter search. Regarding the memory test, input data is fixed to 256 tokens with a batch size of 64 for consistency.

Results. Results are presented in Table 2. Overall, $\text{C}^3\text{A}_{b=768/1}$ and $\text{C}^3\text{A}_{b=1024/1}$ achieve superior or comparable performance to baseline methods, despite using an exceptionally small number of trainable parameters. As the number of trainable parameters increases, models like $\text{C}^3\text{A}_{b=768/6}$ and $\text{C}^3\text{A}_{b=1024/8}$ significantly outperform the baselines. Moreover, compared to $(IA)^3$, LoRA, VeRA, and BOFT, C^3A distinguishes itself with remarkable memory efficiency. The only method with better memory efficiency is BitFit, which serves as an upper bound since it does not introduce new parameters. Furthermore, most of the delta matrices identified by C^3A are of full rank, indicating maximum capacity (Zeng and Lee, 2023) and providing a theoretical basis for outstanding performance.

4.2 Instruction Fine-tuning

Commonsense Reasoning. Our training utilizes Commonsense170K (Hu et al., 2023), aggregating multiple-choice questions from BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). Evaluation follows Hu et al. (2023) and Liu et al. (2024b), using greedy search and first-

	Methods	Params (%)	Mem	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
	ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA2-7B	LoRA$_{r=32}$	0.83	41.71G	71.0	81.4	79.6	87.4	83.2	82.6	67.5	81.5	79.3
	VeRA$_{r=16384}$	0.05	57.88G	68.9↓2.1	81.0↓0.4	77.4↓2.2	87.5↑0.1	81.3↓1.9	81.1↓1.5	62.8↓4.7	79.9↓1.6	77.5↓1.8
	BOFT$^{m=2}_{b=8}$			Out of Memory								
	DoRA$_{r=32}$	0.84	55.24G	72.3↑1.3	**84.2**↑2.8	78.7↓0.9	**88.5**↑1.1	84.0↑0.8	80.7↓1.9	**69.1**↑1.6	**83.7**↑2.2	80.2↑0.9
	C^{3}A$_{b=4096/32}$	0.35	44.89G	**73.4**↑2.4	83.4↑2.0	**82.1**↑2.5	88.3↑0.9	**84.9**↑1.7	**86.6**↑4.0	66.7↓0.8	82.5↑1.0	**81.0**↑1.7
LLaMA3-8B	LoRA$_{r=32}$	0.70	51.18G	73.8	88.2	80.4	94.0	85.5	87.5	78.1	84.0	83.9
	VeRA$_{r=16384}$	0.04	66.03G	72.2↓1.6	87.5↓0.7	80.0↓0.4	94.3↑0.3	83.9↓1.6	84.5↓3.0	75.6↓2.5	82.4↓1.6	82.5↓1.4
	BOFT$^{m=2}_{b=8}$			Out of Memory								
	DoRA$_{r=32}$	0.71	63.37G	75.3↑1.5	88.9↑0.7	**82.2**↑1.8	**96.7**↑2.7	86.0↑0.5	89.0↑1.5	**79.8**↑1.7	84.9↑0.9	85.3↑1.4
	C^{3}A$_{b=4096/32}$	0.26	56.08G	**76.9**↑3.1	**91.4**↑3.2	82.1↑1.7	94.9↑0.9	**86.9**↑1.4	**89.6**↑2.1	79.4↑1.3	**86.1**↑2.1	**85.9**↑2.0

Table 3: Comparison of various methods on the LLaMA2-7B and LLaMA3-8B models across eight commonsense reasoning datasets. For each language model, the best result on each dataset is shown in **bold**. The symbols ↑ and ↓ indicate relative improvement and decrease, respectively, compared to the LoRA method. Experiments are conducted on a single H800 GPU with 80GB HBM.

	Methods	Params (%)	Math		Avg.	Code				Avg.
			GSM8K	MATH		HumanEval	HumanEval+	MBPP	MBPP+	
LLaMA2-7B	Zero-Shot	-	7.3	1.1	4.2	11.0	9.8	30.2	24.1	18.8
	LoRA$_{r=32}$	0.83	60.5	11.7	36.1	31.7	28.0	35.4	30.4	31.4
	VeRA$_{r=16384}$	0.05	58.5↓2.0	9.4↓2.3	34.0↓2.1	28.9↓2.8	24.6↓3.4	35.1↓0.3	29.6↓0.8	29.5↓1.9
	DoRA$_{r=32}$	0.84	**62.3**↑1.8	12.4↑0.7	**37.3**↑1.2	33.2↑1.5	28.8↑0.8	**36.8**↑1.4	31.8↑1.4	32.7↑1.3
	C^{3}A$_{b=4096/32}$	0.35	61.5↑1.0	**13.0**↑1.3	37.2↑1.1	**33.8**↑2.1	**29.0**↑1.0	36.5↑1.1	**31.9**↑1.5	**32.8**↑1.4
LLaMA3-8B	Zero-Shot	-	33.1	5.3	19.2	33.5	29.3	61.4	51.6	44.0
	LoRA$_{r=32}$	0.70	77.2	28.2	52.7	57.9	52.4	64.8	55.3	57.6
	VeRA$_{r=16384}$	0.04	76.0↓1.2	26.7↓1.5	51.4↓1.3	56.3↓1.6	50.8↓1.6	64.0↓0.8	53.6↓1.7	56.2↓1.4
	DoRA$_{r=32}$	0.71	77.2↑0.0	**30.0**↑1.8	53.6↑0.9	59.1↑1.2	52.9↑0.5	**65.7**↑0.9	55.8↑0.5	58.4↑0.8
	C^{3}A$_{b=4096/32}$	0.26	**78.4**↑1.2	29.9↑1.7	**54.1**↑1.4	**59.5**↑1.6	**53.8**↑1.4	65.4↑0.6	**56.1**↑0.8	**58.7**↑1.1

Table 4: Comparison of various methods on the LLaMA2-7B and LLaMA3-8B models across math reasoning and code generation datasets. For each language model, the best result on each dataset is shown in **bold**. The symbols ↑ and ↓ indicate relative improvement and decrease, respectively, compared to the LoRA method. Experiments are conducted on a single H800 GPU with 80GB HBM.

keyword appearance for answer determination.

Mathematical Reasoning. We use MetaMathQA (Yu et al., 2023), containing 395K QA pairs from GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2020). Following Yu et al. (2023), we evaluate using greedy search on test sets, requiring chain-of-thought reasoning (Wei et al., 2022).

Code Generation. Training uses Magicoder-Evol-Instruct-110k (Wei et al., 2024), a decontaminated subset of WizardCoder (Luo et al., 2024). Evaluation on HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and their Plus variants follows EvalPlus (Liu et al., 2024a), reporting Pass@1 metrics.

Results. In Table 3 and Table 4, our principal experimental observations are summarized. The C^{3}A framework consistently surpasses LoRA and other baselines within the LLaMA series, with particular efficacy demonstrated in the most recent model, LLaMA3-8B. Noteworthy is the significant enhancement in the efficacy of LLaMA3-8B as a foundational model following the implementation of more sophisticated post-training techniques. This underscores the criticality of optimizing the fine-tuning protocols for this advanced model. It is also remarkable that C^{3}A achieves such results while employing less than half the parameter count of LoRA. Furthermore, C^{3}A achieves high rank adaptation and delivers optimal performance while maintaining remarkably low memory consumption. In contrast, VeRA and DoRA require substantially greater memory resources, and BoFT results in an OOM condition for a single H800, thereby underscoring the practicality and efficiency of C^{3}A. Taken together, the findings robustly underscore the superior efficacy of the C^{3}A methodology. We

refer readers to Appendix D for examples of model answers after different tuning methods.

4.3 Ablation Study

Initialization. We investigated initialization effects in C^3A compared to LoRA, which is known for initialization sensitivity due to its $\mathbf{A}$ and $\mathbf{B}$ matrices (Hayou et al., 2024a). We evaluated multiple initialization strategies across five distinct tasks. Each initialization method was tested with 5 independent runs per task, producing a total of 20 runs per task. As shown in Figure 3, testing zero, Gaussian, Kaiming uniform, and Xavier uniform initializations for C^3A's convolution kernels revealed performance variations mostly within standard deviations, demonstrating C^3A's robustness to initialization choices.

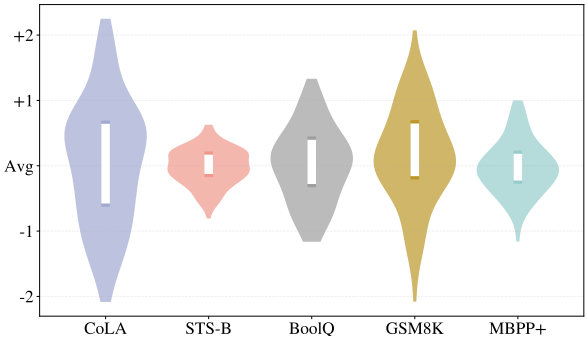

Figure 3: Violin plots for runs on different tasks. Within each violin plot, a white bar indicates the range of average performance for each initialization strategy. It is clear that the choice of initialization does not affect the final result more than the intrinsic stochasticity.

Expressiveness. We demonstrate C^3A's expressiveness on a synthetic dataset by placing 8 cluster centers on a 2D plane and sampling 30 points from their Gaussian distributions. A 3-layer MLP is used to classify these clusters. To compare the expressiveness of LoRA and C^3A, we replace the middle layer with either a low-rank layer or a circulant layer, ensuring that both layers have the same number of trainable parameters for a fair comparison.

The results are presented in Figure 4. We observe that $\text{LoRA}_{r=1}$ struggles with this simple classification task. In contrast, $\text{C}^3\text{A}_{b=128/2}$, despite using the same number of parameters, achieves a perfect classification, comparable to a standard linear layer. This demonstrates the high expressiveness of C^3A given the same parameter budget.

Scaling. To investigate the scaling effect of C^3A, we examine it from both data and model perspectives, following the setup in He et al. (2025). From a data perspective, we focus on the MATH dataset and vary the training data used. As shown in Figure 5, we observe that C^3A's performance improves faster than LoRA as more data is added, suggesting it is more effective at leveraging additional data. From the model perspective, we evaluate C^3A in small models (LLaMA3-8B, Mistral-7B) and large models (LLaMA3-70B, Mistral-8x7B). Results show that C^3A outperforms the LoRA baseline in both settings, indicating that its benefits generalize to different model scales. Experiments on various data quantities and model sizes showcase C^3A's scalability and robustness, making it a promising approach for adapting language models.

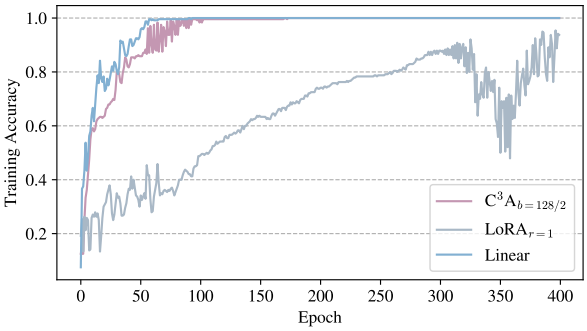

Figure 4: Training curves on the synthetic dataset. We refer our readers to Figure A1 in Appendix E for a visualization of the synthetic dataset.

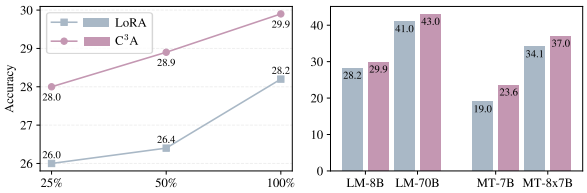

Figure 5: Data and model scaling of C^3A compared to LoRA. LM specifies LLaMA3 and MT specifies Mistral.

5 Conclusion

This manuscript introduces C^3A, a novel PEFT method. Unlike LoRA's low-rank decomposition, C^3A utilizes circular convolution and circulant matrices to construct the delta weight matrix. This approach allows independent control over the delta weight matrix's rank and the number of trainable parameters, enabling high-rank adaptation with limited parameter size. By employing FFT in forward and backward propagation, C^3A achieves significant computational and memory efficiency, presenting a compelling alternative to LoRA for model fine-tuning.

6 Limitations

Despite C^{3}A's efficiency and effectiveness within limited parameter budgets, it has some intrinsic drawbacks. First, due to the requirement of $b \in \gcd(d_1, d_2)$, C^{3}A may not be flexible enough to tackle all transformer architectures. Besides, the integration of FFT in C^{3}A necessitates specialized kernel support for optimal performance. While this support is well-developed and readily available for NVIDIA GPUs, it may not be as mature or optimized for edge devices with different hardware architectures. This can potentially hinder the deployment and efficiency of C^{3}A on resource-constrained edge platforms.

References

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

Bassam Bamieh. 2018. Discovering transforms: A tutorial on circulant matrices, circular convolution, and the discrete fourier transform. *arXiv preprint arXiv:1805.05533*.

Samyadeep Basu, Shell Hu, Daniela Massiceti, and Soheil Feizi. 2024. Strong baselines for parameter-efficient few-shot fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11024–11031.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Aochuan Chen, Yuguang Yao, Pin-Yu Chen, Yihua Zhang, and Sijia Liu. 2023. Understanding and improving visual prompting: A label-mapping perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19133–19143.

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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. 2017. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883.

Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE international conference on computer vision*, pages 2857–2865.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. 2014. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613.

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. *arXiv preprint arXiv:1905.10044*.

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 preprint arXiv:1803.05457*.

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

Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. 2017. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 395–408, New York, NY, USA. Association for Computing Machinery.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. 2001. Advanced encryption standard (aes).

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 12799–12807.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, USA.

Yanwei Gong, Xiaolin Chang, Jelena Mišić, Vojislav B Mišić, Jianhua Wang, and Haoran Zhu. 2024. Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods. *Cybersecurity*, 7(1):5.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024a. The impact of initialization on lora finetuning dynamics. *arXiv preprint arXiv:2406.08447*.

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

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Zhiwei He, Zhaopeng Tu, Xing Wang, Xingyu Chen, Zhijie Wang, Jiahao Xu, Tian Liang, Wenxiang Jiao, Zhuosheng Zhang, and Rui Wang. 2025. RaSA: Rank-sharing low-rank adaptation. In *The Thirteenth International Conference on Learning Representations*.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. 2019. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

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

A. W. Ingleton. 1956. The rank of circulant matrices. *Journal of the London Mathematical Society*, s1-31(4):445–460.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual prompt tuning. In *European Conference on Computer Vision*, pages 709–727. Springer.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026.

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

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561.

Changli Li, Hon Keung Kwan, and Xinxin Qin. 2020. Revisiting linear convolution, circular convolution and their related methods. *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1124–1131.

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

Yuhan Li, Peisong Wang, Zhixun Li, Jeffrey Xu Yu, and Jia Li. 2024a. Zerog: Investigating cross-dataset zero-shot transferability in graphs. *arXiv preprint arXiv:2402.11235*.

Yuhan Li, Peisong Wang, Xiao Zhu, Aochuan Chen, Haiyun Jiang, Deng Cai, Victor W Chan, and Jia Li. 2024b. Glbench: A comprehensive benchmark for graph with large language models. *Advances in Neural Information Processing Systems*, 37:42349–42368.

Yuhan Li, Xinni Zhang, Linhao Luo, Heng Chang, Yuxiang Ren, Irwin King, and Jia Li. 2025. G-refer: Graph retrieval-augmented large language model for explainable recommendation. In *Proceedings of the ACM on Web Conference 2025*, pages 240–251.

Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. 2022. Scaling & shifting your features: A new baseline for efficient model tuning. *Advances in Neural Information Processing Systems*, 35:109–123.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning

is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36.

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. *arXiv preprint arXiv:2402.09353*.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, et al. 2023. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. 2013. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075.

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`.

Clare D. McGillem and George R. Cooper. 1984. Continuous and discrete signal and system analysis.

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. *arXiv preprint arXiv:1809.02789*.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. 2012. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2023. Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362.

L. R. Rabiner, B. Gold, and C. K. Yuen. 1978. Theory and application of digital signal processing. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(2):146–146.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2019. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. 2021. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

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*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning*.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. *arXiv preprint arXiv:2109.05687*.

Felix Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. 2014. Circulant binary embedding. In *International conference on machine learning*, pages 946–954. PMLR.

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

Shen Yuan, Haotian Liu, and Hongteng Xu. 2024. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. *arXiv preprint arXiv:2405.17484*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Yuchen Zeng and Kangwook Lee. 2023. The expressive power of low-rank adaptation. *arXiv preprint arXiv:2310.17513*.

Yihua Zhang, Hongkang Li, Yuguang Yao, Aochuan Chen, Shuai Zhang, Pin-Yu Chen, Meng Wang, and Sijia Liu. 2024a. Visual prompting reimagined: The power of activation prompts.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. 2024b. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*.

Appendix

A Implementations

Algorithm A1 Block-Circular Convolution PyTorch Implementation

```
import torch
from torch.autograd import Function
from torch.fft import fft, ifft

class BlockCircularConvolution(Function):
    @staticmethod
    def forward(ctx, x, w):
        m, n, b = w.shape
        x = x.reshape(*x.shape[:-1], n, b)
        ctx.save_for_backward(x, w)
        x = torch.einsum(
            "...nb,mnb->...mb", ifft(x), fft(w)
        )
        x = fft(x).real
        x = x.reshape(*x.shape[:-2], -1)
        return x

    @staticmethod
    def backward(ctx, grad_output):
        x, w = ctx.saved_tensors
        m, n, b = w.shape
        grad_output = grad_output.reshape(
            *grad_output.shape[:-1], m, b
        )
        grad_output_fft = fft(grad_output)
        x_grad = fft(torch.einsum(
            "...mb,mnb->...nb",
            grad_output_fft, ifft(w)
        )).real
        x_grad = x_grad.reshape(
            *x_grad.shape[:-2], -1
        )
        w_grad = fft(torch.einsum(
            "...mb,...nb->mnb",
            grad_output_fft, ifft(x)
        )).real
        return x_grad, w_grad
```

We present the PyTorch implementation of Block-Circular Convolution in Algorithm A1. Furthermore, due to the inefficiency of directly assigning entries (as shown in Equation 4), we derive an alternative algorithm to compute the $\Delta\mathbf{W}$ more efficiently. Rather than direct assignment, we employ a forward process on the Identity matrix. Mathematically, this can be expressed as

$$\begin{aligned}\Delta\mathbf{W} &= \mathcal{C}_{\text{blk}}(\Delta\mathbf{w}) \\ &= \mathcal{C}_{\text{blk}}(\Delta\mathbf{w}) \cdot \mathbf{I}_{d_2} \\ &= \mathcal{C}_{\text{blk}}(\Delta\mathbf{w}) \cdot [\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_{d_2}] \\ &= [\mathcal{C}_{\text{blk}}(\Delta\mathbf{w})\mathbf{e}_1, \mathcal{C}_{\text{blk}}(\Delta\mathbf{w})\mathbf{e}_2, \cdots, \mathcal{C}_{\text{blk}}(\Delta\mathbf{w})\mathbf{e}_{d_2}] \\ &= [\Delta\mathbf{w} \star \mathbf{e}_1, \Delta\mathbf{w} \star \mathbf{e}_2, \cdots, \Delta\mathbf{w} \star \mathbf{e}_{d_2}]\,.\end{aligned}$$

Where $\mathbf{I}_{d_2} \in \mathbb{R}^{d_2 \times d_2}$ represents an Identity matrix and $\mathbf{e}_i$ is the ith column of it. In pytorch, we can efficiently compute the iFFT of $\{\mathbf{e}_i\}_{i=1,2,\cdots,d_2}$ by a column-wise iFFT of $\mathbf{I}_{d_2}$. We present the Pytorch implementation in Algorithm A2 as well.

Algorithm A2 Fast Algorithm of Getting $\Delta\mathbf{W}$

```
import torch
from torch.fft import fft, ifft

def get_circulant_fast(w):
    m, n, b = w.shape
    x = torch.eye(n*b)
    x = x.reshape(*x.shape[:-1], n, b)
    x = torch.einsum(
        "...nb,mnb->...mb", ifft(x), fft(w)
    )
    x = fft(x).real.flatten(start_dim=1).T
    return x
```

B Image Classification

Settings. In this study, we concentrate on the task of image classification leveraging Vision Transformer (ViT) models. Specifically, we employ both the Base and Large variants of this prominent foundational computer vision model, as delineated by (Dosovitskiy et al., 2020). These ViT models undergo pre-training on the expansive ImageNet-21K dataset (Ridnik et al., 2021). During the fine-tuning phase, we use an eclectic array of datasets encompassing Pets (Parkhi et al., 2012), Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), FGVC (Maji et al., 2013), and RESISC (Cheng et al., 2017). Statistics for these datasets are provided in Table A1.

Dataset	#Train	#Validation	#Test	#Class
Pets (Parkhi et al., 2012)	3,312	368	3,669	37
Cars (Krause et al., 2013)	7,329	815	8,041	196
DTD (Cimpoi et al., 2014)	4,060	452	1,128	47
EuroSAT (Helber et al., 2019)	16,200	5,400	5,400	10
FGVC (Maji et al., 2013)	3,000	334	3,333	100
RESISC (Cheng et al., 2017)	18,900	6,300	6,300	45

Table A1: Details about the vision datasets.

Results. Table A2 delineates a comprehensive summary of the outcomes derived from six distinct image classification datasets employing the ViT Base and Large models. The LoRA and C^{3}A techniques exhibit significant enhancements in performance relative to Head Tuning, thereby underscoring their efficacy within the realm of image classification. Remarkably, our methodology demonstrates a performance on par with LoRA while necessitating only half of the parameter count.

C GLUE Benchmark Details

The General Language Understanding Evaluation (GLUE) benchmark is a collection of nine natural language understanding tasks designed to test a model's performance on a diverse set of natural

	Method	# Params	Pets	Cars	DTD	EuroSAT	FGVC	RESISC	Avg.
BASE	Head	-	$90.28_{\pm 0.43}$	$25.76_{\pm 0.28}$	$69.77_{\pm 0.67}$	$88.72_{\pm 0.13}$	$17.44_{\pm 0.43}$	$74.22_{\pm 0.10}$	61.03
	Full	85.8M	$92.82_{\pm 0.54}$	$85.10_{\pm 0.21}$	$80.11_{\pm 0.56}$	$99.11_{\pm 0.07}$	$61.60_{\pm 1.00}$	$96.00_{\pm 0.23}$	85.79
	LoRA$_{r=16}$	0.59M	$93.76_{\pm 0.44}$	$78.04_{\pm 0.33}$	$78.56_{\pm 0.62}$	$98.84_{\pm 0.08}$	$\mathbf{56.64}_{\pm 0.55}$	$\mathbf{94.66}_{\pm 0.17}$	83.42
	C^{3}A$_{b=768/12}$	0.22M	$\mathbf{93.88}_{\pm 0.22}$	$\mathbf{79.05}_{\pm 0.35}$	$\mathbf{80.57}_{\pm 0.53}$	$\mathbf{98.88}_{\pm 0.07}$	$54.31_{\pm 0.79}$	$94.54_{\pm 0.23}$	**83.54**
LARGE	Head	-	$91.11_{\pm 0.30}$	$37.91_{\pm 0.27}$	$73.33_{\pm 0.26}$	$92.64_{\pm 0.08}$	$24.62_{\pm 0.24}$	$82.02_{\pm 0.11}$	66.94
	Full	303M	$94.30_{\pm 0.31}$	$88.15_{\pm 0.50}$	$80.18_{\pm 0.66}$	$99.06_{\pm 0.10}$	$67.38_{\pm 1.06}$	$96.08_{\pm 0.20}$	87.53
	LoRA$_{r=16}$	1.57M	$\mathbf{94.62}_{\pm 0.47}$	$\mathbf{86.11}_{\pm 0.42}$	$80.09_{\pm 0.42}$	$\mathbf{98.99}_{\pm 0.03}$	$63.64_{\pm 0.83}$	$95.52_{\pm 0.21}$	86.56
	C^{3}A$_{b=1024/16}$	0.79M	$94.48_{\pm 0.30}$	$84.94_{\pm 0.39}$	$\mathbf{82.62}_{\pm 0.52}$	$98.75_{\pm 0.19}$	$\mathbf{63.80}_{\pm 0.37}$	$\mathbf{95.94}_{\pm 0.16}$	**86.69**

Table A2: Fine-tuning results with ViT-Base and ViT-Large models on various image classification datasets. The models are fine-tuned for 10 epochs, and the best-performing model, based on validation set accuracy, is selected. The reported accuracy corresponds to the performance on the test set. The best results between LoRA and C^{3}A for each dataset are highlighted in **bold**. "Avg." denotes the average accuracy of each method across all datasets.

Corpus	Task	# Train	# Val	# Test	# Labels	Metrics	Domain
				Single-Sentence Tasks			
CoLA	Acceptability	8.55k	1.04k	1.06k	2	Matthews Corr.	misc.
SST-2	Sentiment	67.3k	872	1.82k	2	Accuracy	Movie reviews
				Similarity and Paraphrase Tasks			
MRPC	Paraphrase	3.67	408	1.73k	2	Accuracy/F1	News
STS-B	Sentence similarity	5.75k	1.5k	1.38k	1	Pearson/Spearman Corr.	misc.
QQP	Paraphrase	364k	40.4k	391k	2	Accuracy/F1	Social QA
				Inference Tasks			
MNLI	NLI	393k	19.65k	19.65k	3	Accuracy	misc.
QNLI	QA/NLI	105k	5.46k	5.46k	2	Accuracy	Wikipedia
RTE	NLI	2.49k	277	3k	2	Accuracy	News & Wikipedia

Table A3: Task descriptions and dataset statistics of the GLUE benchmark (Wang et al., 2018).

language understanding challenges. It was introduced by researchers from New York University, the University of Washington, and DeepMind in a 2018 paper titled "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding" (Wang et al., 2018).

The nine tasks included in the benchmark are:

1. **CoLA (Corpus of Linguistic Acceptability):** A binary classification task that tests a model's ability to determine whether a given sentence is grammatically acceptable.

2. **SST-2 (Stanford Sentiment Treebank):** A binary sentiment classification task using movie review excerpts.

3. **MRPC (Microsoft Research Paraphrase Corpus):** A binary classification task to determine whether two sentences are semantically equivalent.

4. **STS-B (Semantic Textual Similarity Benchmark):** A regression task that scores the semantic similarity of sentence pairs on a scale from 1 to 5.

5. **QQP (Quora Question Pairs):** A binary classification task to determine whether two questions are semantically equivalent.

6. **MNLI (Multi-Genre Natural Language Inference):** A ternary classification task that tests a model's ability to perform textual entailment across multiple genres.

7. **QNLI (Question Natural Language Inference):** A binary classification task that tests a model's ability to determine whether a sentence contains the answer to a given question.

8. **RTE (Recognizing Textual Entailment):** A binary classification task that tests a model's ability to determine whether a premise entails a hypothesis.

9. **WNLI (Winograd Natural Language Inference):** A binary classification task that tests a model's ability to resolve ambiguous pronouns in a sentence.

The GLUE benchmark has become a widely-used standard for evaluating the performance of pre-trained language models and fine-tuned models on a variety of natural language understanding tasks. The benchmark provides a single-number metric, the GLUE score, which is an average of the scores on each individual task. This allows for easy comparison of different models and architectures. We refer readers to Table A3 for more details.

D Instruction Fine-tuning Examples

Example D.1: MATH

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction: Find the number of square units bounded by the x-axis, the y-axis and the line $x+y=6$. Answer the above question. First think step by step and then answer the final number.

LoRA Answer: The line $x+y=6$ intersects the x-axis at $x=6$ and the y-axis at $y=6$. So, the line divides the square with side length 6 into two triangles. The area of each triangle is $\frac{1}{2}\times 6\times 6=18$. So, the area of the square is $2\times 18=\boxed{36}$. The final answer is: 36.

C^{3}A Answer: The line $x+y=6$ intersects the x-axis at $(6,0)$ and the y-axis at $(0,6)$. Thus, the line divides the first quadrant into two triangles, one with base 6 and height 6 and the other with base 6 and height 0. The area of the first triangle is $(6\times 6)/2=18$ square units, and the area of the second triangle is $(6\times 0)/2=0$ square units. Thus, the total area is $18+0=\boxed{18}$ square units. The final answer is: 18.

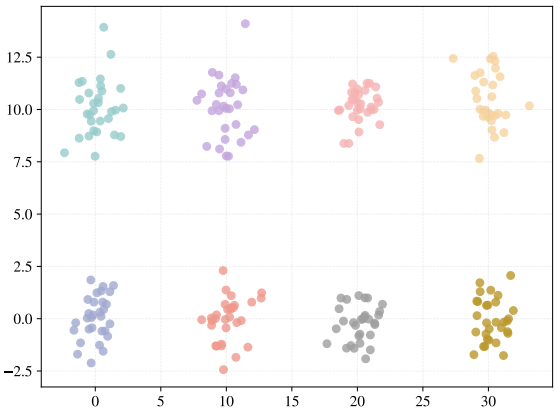

Figure A1: Synthetic dataset used for the expresiveness ablation study.

Example D.2: GSM8K

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction: Tim makes a special honey and jam mix. To make 1 jar of the mixture it takes 2 pounds of passion fruit and 1.5 pounds of jackfruit. It also takes 2 pounds of special honey. The passion fruit costs $6 per pound, the jackfruit is $8 per pound, and the honey is $10 per pound. He sells each jar for $50. How much profit does he make selling 10 jars? Answer the above question. First think step by step and then answer the final number.

LoRA Answer: The passion fruit costs $6\times 2=12$. The jackfruit costs $8\times 1.5=12$. The honey costs $10\times 2=20$. So the total cost is $12+12+20=44$. So he makes a profit of $50\times 10-44=\boxed{460}$. The final answer is: 460.

C^{3}A Answer: The passion fruit costs $6\times 2=12$. The jackfruit costs $8\times 1.5=12$. So the fruit costs $12+12=24$. The honey costs $10\times 2=20$. So the total cost per jar is $24+20=44$. So he makes a profit of $50-44=6$ per jar. So he makes a profit of $6\times 10=\boxed{60}$. The final answer is: 60.

E Synthetic Dataset

We visualize the synthetic dataset we use for the expressiveness ablation study in Figure A1.

F Hyperparameters

G Singular Value Comparison

Model	Hyperparameter	SST-2	MRPC	CoLA	QNLI	RTE	STS-B
Both	Optimizer	AdamW					
	LR Schedule	Linear					
	Warmup Ratio	0.06					
	C^3A Initialization	Xavier Uniform					
	Max Seq. Len	512					
Base	Epochs	40	80	80	40	80	80
	Batch Size	128	128	128	64	64	128
	Learning Rate ($C^3A_{b=768/6}$)	2E-1	3E-1	2E-1	7E-2	3E-1	2E-1
	Learning Rate (Head)	2E-4	4E-6	3E-2	8E-6	6E-3	4E-2
Large	Epochs	10	80	70	30	60	40
	Batch Size	128	128	128	32	64	128
	Learning Rate ($C^3A_{b=1024/8}$)	9E-2	3E-1	2E-1	7E-2	5E-2	2E-1
	Learning Rate (Head)	2E-4	5E-6	3E-3	8E-6	3E-3	5E-4

Table A4: Hyperparameter setup of C^3A for the GLUE benchmark.

Model	Hyperparameter	Pets	Cars	DTD	EuroSAT	FGVC	RESISC
Both	Optimizer	AdamW					
	LR Schedule	None					
	C^3A Initialization	Xavier Uniform					
	Epochs	10					
	Batch Size	64					
Base	Learning Rate ($C^3A_{b=768/12}$)	4E-1	4E+0	2E+0	2E+0	7E+0	2E+0
	Learning Rate (Head)	1E-2	1E-2	2E-2	8E-3	1E-2	2E-2
	Weight Decay	3E-4	5E-4	6E-5	2E-5	1E-5	2E-5
Large	Learning Rate ($C^3A_{b=1024/16}$)	7E-1	4E+0	2E+0	2E+0	4E+0	3E+0
	Learning Rate (Head)	3E-3	8E-3	7E-3	2E-2	1E-1	4E-3
	Weight Decay	4E-3	1E-5	2E-4	5E-4	2E-5	9E-5

Table A5: Hyperparameter setup of C^3A for image classification tasks.

Hyperparameter	Common	Math	Code
Optimizer	AdamW		
LR Scheduler	Cosine		
Batch Size	16	128	128
Warmup Ratio	0.05	0.03	0.03
Epoch	3	2	2
Dropout	0.05		
Learning Rate (LLaMA2-7B)	3E-1	4E-1	6E-1
Learning Rate (LLaMA3-8B)	3E-1	5E-1	6E-1

Table A6: Hyperparameter setup of C^3A for instruction tuning.

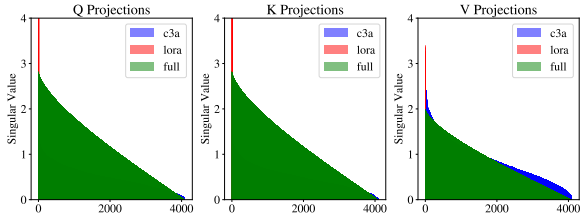

Figure A2: Singular values spectra of the weight difference between C^3A and LoRA, Full finetuning.