Parameter-Efficient Fine-Tuning of Large Language Models via Deconvolution in Subspace

Jia-Chen Zhang[1], Yu-Jie Xiong[1*], Chun-Ming Xia[1], Dong-Hai Zhu[1], Xi-He Qiu[1]
[1]School of Electronic and Electrical Engineering, Shanghai University of Engineering Science,
333 Longteng Road, Songjiang District, Shanghai, China
Correspondence: xiong@sues.edu.cn

Abstract

Large language model (LLM) is considered a milestone towards achieving Artificial General Intelligence (AGI). With its advanced emergent capabilities, it adapt to a wide range of specific applications. Fine-tuning LLMs for various downstream tasks has become a new paradigm. Low-Rank Adaptation (LoRA) is well-known for its parameter efficiency. It can reduce the number of parameters needed to fine-tune LLMs by several orders of magnitude. However, LoRA-based approaches encounter a significant limitation due to the bottleneck imposed by rank one decomposition. As the parameters count in LLMs increase, even rank one decomposition might surpass the number of parameters truly necessary for handling more downstream tasks. In this paper, we propose a new method for Parameter-Efficient Fine-Tuning (PEFT) via deconvolution in subspace, dubbed as DCFT. We innovatively use deconvolution to complete details and enhance knowledge in subspace incremental matrices, and dynamically control parameters by adjusting the kernel size, unconstrained by rank-one decomposition. Extensive experiments are conducted to validate the effectiveness of DCFT. Results show that compared to LoRA, DCFT achieve an 8× reduction in parameters, and still achieves highly impressive performance. Our code is available here: `https://github.com/Godz-z/DCFT`.

1 Introduction

LLMs are considered a potential spark for AGI (Xi et al., 2023). Due to their excellent situational adaptability and language comprehension abilities, LLMs have become the cornerstone of NLP tasks (Devlin et al., 2019; Liu et al., 2021; He et al., 2021; Radford et al., 2019). The success of GPT-3.5 has mainstreamed the development of LLMs towards larger parameter counts (Ouyang et al., 2024). Over the past few years, the parameter scale of pretrained language models has increased by thousands of times; for example, PaLM (Chowdhery et al., 2023) contains up to 540 billion parameters, while GPT-4 (OpenAI, 2023) contains up to 100 trillion parameters. Nevertheless, due to the knowledge boundaries of LLMs, their abilities in some downstream tasks are still limited. To expand these knowledge boundaries, it remains necessary to fine-tune LLMs on downstream tasks (Qiu et al., 2020; Liu et al., 2021).

However, the time and resource costs required to fine-tune all parameters are prohibitive. Various methods for parameter-efficient fine-tuning have been proposed to reduce these costs. The PEFT methods are divided into two categories based on whether the pretrained parameters are frozen (Lialin et al., 2023). Considering the potential issues of catastrophic forgetting and reduced generalization ability that may arise from modifying model parameters and architecture, LoRA (Hu et al., 2022) has become one of the most widely applied PEFT methods. LoRA introduces incremental updates to pretrained weights via the product of two low-rank matrices. It reduces training overhead by up to 70% and achieves comparable or superior performance to fine-tuning.

Nonetheless, due to the constraints of rank-one decomposition, LoRA-based approaches (Zhang et al., 2023b; Ding et al., 2023; yang Liu et al., 2024) have limitations in parameter efficiency. With the increasing parameter counts in LLMs, even rank one decomposition may exceed the necessary number of parameters for managing more complex models. Many researchers are dedicated to breaking this limitation. LoRA-FA (Zhang et al., 2023a) reduces trainable parameters and memory further by freezing the matrix $\mathcal{A}$ in LoRA. NOLA (Koohpayegani et al., 2024) decomposes LoRA's matrix $\mathcal{A}$ and $\mathcal{B}$ further into several small random matrices. Yet, these methods consistently follow

[*] Corresponding author.

Proceedings of the 31st International Conference on Computational Linguistics, pages 3924–3935
January 19–24, 2025. ©2025 Association for Computational Linguistics

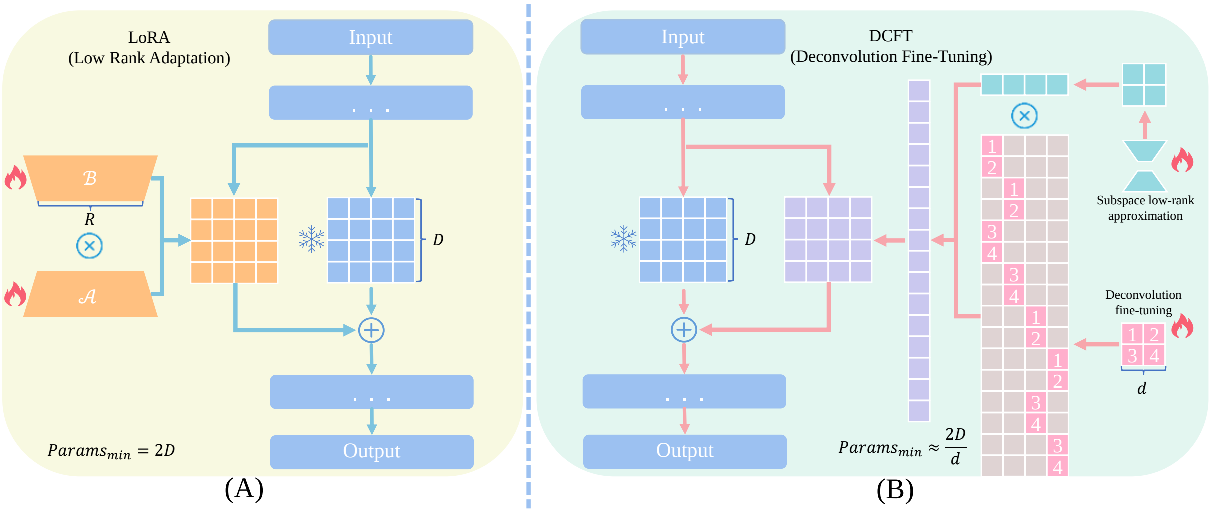

Figure 1: An illustration of the differences between LoRA and DCFT. The parameter calculation results represent the model's parameters when $r = 1$. D represents the dimension of the pretrained weights, and d represents the dimension of the convolution kernel.

the fundamental architecture of LoRA, encountering the persistent issue that parameter variations are contingent upon the size of R. Additionally, they have frozen some parameters, significantly constraining the model's capacity to learn from new data.

In this paper, we innovatively combine the feature reconstruction ability of deconvolution and the efficiency of subspace learning, dubbed as Deconvolution Fine-Tuning (DCFT). Deconvolution is a kind of CNN that performs upsampling by transposing the convolution kernel matrix (Zeiler et al., 2010). It is widely used in the fields of image processing and computer vision, commonly applied to tasks such as image reconstruction, semantic segmentation, and feature enhancement (Dumoulin and Visin, 2018; Zeiler and Fergus, 2014). For the first time, we apply deconvolution to the fine-tuning of large models, restoring and enhancing features learned from incremental matrices through deconvolution. We first learn a set of low-rank incremental matrices for LLMs within the subspace. Subsequently, by integrating orthogonal projection theory, we apply orthogonal constraints to maximize the learning space of these matrices. Then, we enhance and complete the matrices learned in the subspace through deconvolution to adapt them to the dimensions of the incremental matrices. By setting the stride equal to the convolution kernel size, we simplify the computation while better preserving the knowledge learned in the subspace and enhancing the stability of models.

Extensive experiments are conducted on various tasks and models to demonstrate the effectiveness of DCFT. Specifically, natural language understanding GLUE (Wang et al., 2018) is evaluated using DeBERTa and RoBERTa (He et al., 2021, 2023; Liu et al., 2021). The findings, including qualitative and quantitative results, indicate that DCFT outperforms existing methods. The contributions of this paper are as follows:

- We propose Deconvolution Fine-Tuning (DCFT), a novel method employing deconvolution to reconstruct subspace features in incremental matrices. Compared to LoRA, DCFT achieves improved performance while using only 14% of the parameters.
- We significantly reduce the computational complexity of DCFT and improve the fine-tuning efficiency through a series of methods including low-rank matrices, equal kernel stride, and larger convolution kernels.
- Extensive experiments are conducted to validate the effectiveness of our method. Notably, our model consistently surpasses parameter-efficient baselines, achieving superior performance with fewer parameters across a broad spectrum of downstream tasks.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning

PEFT is a technique for adapting LLMs by optimizing a small set of parameters while keeping most of the pre-trained parameters unchanged. This approach effectively reduces memory and computational costs, allowing large models to be applied in resource-constrained environments. Common PEFT techniques include Low-Rank Adaptation (LoRA) (Hu et al., 2022), Mixture of Experts (MoE), and Adaptable Embeddings (Adapters) (Houlsby et al., 2019; Li and Liang, 2021). These methods enhance the performance of the model on specific tasks by updating only a small part of the model while keeping the majority of the weights fixed, thus preserving the pre-trained knowledge.

2.2 LoRA-based approach

Due to the high cost of full-parameter fine-tuning of LLMs, numerous parameter-efficient methods have been introduced. Among these, LoRA (Hu et al., 2022) stands out by updating only a small subset of the model's parameters, thereby reducing memory overhead while maintaining performance on par with full-parameter fine-tuning. As Equation 3 shows, LoRA adds a low-rank adapter to the frozen pre-trained weights to learn incremental matrices.

$$W = W^{(0)} + \Delta = W^{(0)} + \mathcal{B}\mathcal{A}, \qquad (1)$$

where $\Delta \in \mathbb{R}^{din \times dout}$, $\mathcal{A} \in \mathbb{R}^{r \times dout}$, and $\mathcal{B} \in \mathbb{R}^{din \times r}$, with $r \in (din, dout)$. The dimensions of din and $dout$ are the same as those of the pre-trained matrix W. During fine-tuning, only A and B are updated. The rank r is chosen to be much smaller than the dimension of W. With less than 0.5% additional trainable parameters, the training overhead can be reduced up to 70%.

Building on this, AdaLoRA (Zhang et al., 2023b) introduces an adaptive mechanism for pruning, LoRA-FA (Zhang et al., 2023a) freezes the parameters of matrix $\mathcal{A}$, and NoLA (Koohpayegani et al., 2024) further decomposes matrix AB into the sum of multiple random matrices. DoRA (yang Liu et al., 2024) trains an additional set of parameters by decomposing the pre-trained weights. However, all these methods are based on the LoRA architecture, which leads to a significant issue: the number of trainable parameters is controlled by the rank r, which can only increase in multiples, and there exists a bottleneck in rank-one decomposition.

2.3 Convolution and Deconvolution

Convolution is a fundamental operation in many areas of signal processing and deep learning, especially in the context of Convolutional Neural Networks (CNNs) (Lecun et al., 1998). It involves sliding a filter (or kernel) over the input data to produce a feature map, which captures important spatial hierarchies in the data. Mathematically, the convolution operation for a 2D input can be expressed as:

$$Y(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot W, \qquad (2)$$

where $Y(i,j)$ represents the output feature map, X is the input feature map, W is the filter or kernel. The operation essentially involves multiplying the filter with corresponding input values and summing them to obtain the output at each spatial position. Convolution is widely used for tasks such as image recognition, object detection, and segmentation due to its ability to extract meaningful features from raw data.

Deconvolution (Zeiler et al., 2010), also known as transposed convolution or upsampling, is the reverse operation of convolution. While convolution reduces the spatial dimensions of the input, deconvolution aims to increase them, making it a crucial operation in tasks requiring output with higher spatial resolution, such as image generation and semantic segmentation. The deconvolution operation can be mathematically represented as:

$$X(i,j) = \sum_m \sum_n Y(i - s_m, j - s_n) \cdot W, \qquad (3)$$

where $X(i,j)$ is the reconstructed input, Y is the output feature map from the previous layer, W is the filter or kernel, and s_m and s_n denote the strides in the spatial dimensions. The transposed nature of this operation is reflected in the indices $i - s_m$ and $j - s_n$, where the input is effectively expanded.

Deconvolution is not merely the inversion of the convolution process but involves a more complex transformation where the spatial resolution of the feature map is increased, often leading to more refined and higher-quality outputs. This operation is critical in tasks where the generation or reconstruction of detailed spatial information is necessary.

3 Our Method

In this section, we provide a detailed description of the DCFT method's design, which enables flexible control over fine-tuning parameters and overcomes the constraints of rank one decomposition. Additionally, we introduce the optimization for efficiency employed in our method.

3.1 Deconvolution Fine-Tuning

The overall pipeline of DCFT is depicted in Figure 1. We first train a set of low-rank matrices $\mathcal{A}$ and $\mathcal{B}$ in the subspace. Orthogonal constraints are applied to matrix $\mathcal{A}$ and $\mathcal{B}$ to maximize the learning space:

$$R(\mathcal{A},\mathcal{B}) = \left\|\mathcal{A}^T\mathcal{A}-\mathcal{I}\right\|_F^2 + \left\|\mathcal{B}\mathcal{B}^T-\mathcal{I}\right\|_F^2, \qquad (4)$$

where equation 4 enhances the orthogonality of matrix $\mathcal{A}$ and $\mathcal{B}$ by approximating $\mathcal{A}^T\mathcal{A} = \mathcal{B}\mathcal{B}^T = \mathcal{I}$. Then, by applying the deconvolution operation, we upscale the knowledge learned in the subspaces, increasing the spatial dimensions of the incremental matrix. By leveraging the existing knowledge in the subspaces, we predict missing parts. This step allows us to obtain the incremental matrix with minimal parameters. Moreover, because deconvolution can learn the mapping from the subspace to the parent space, the incremental matrix obtained through deconvolution exhibits smoother and more precise details of implicit knowledge compared to that derived directly from the product of low-rank matrices.

As shown in Equation 5, the sub-feature matrix $\mathcal{F}$, obtained by multiplying the low-rank matrices, is subsequently multiplied by the transpose of the convolution kernel matrix C_s^T. This step completes and enhances the knowledge learned in the subspace. Ultimately, an incremental matrix of the same dimension as the frozen pretrained weights is obtained. The parameter increment matrix is added to the pretrained parameters to adapt to various downstream tasks, offering a plug-and-play capability. Our forward propagation process is as follows:

$$Conv^T(\mathcal{F}) = \mathcal{C}_s^T \cdot \mathcal{F} = \mathcal{C}_s^T \cdot (\mathcal{B} \cdot \mathcal{A}), \qquad (5)$$

$$W = W^{(0)} + \Delta = W^{(0)} + Conv^T(\mathcal{F}), \qquad (6)$$

where $\Delta \in \mathbb{R}^{(din \times dout)}$, $F \in \mathbb{R}^{(\frac{din}{d} \times \frac{dout}{d})}$, The dimensions of din and $dout$ are the same as those of the pre-trained matrix W. $Conv^T(*)$ denotes the deconvolution operation. s represents the stride of the transposed convolution. All of our trainable parameters are initialized randomly. We summarize the detailed algorithm in Algorithm 1.

3.2 Optimization for Efficiency

In this section, we introduce our approaches for optimizing computational efficiency. Compared to LoRA, which involves only two matrix multiplications, deconvolution operations significantly increase computational complexity. To address this challenge, we have implemented the following three principal measures to reduce the computational complexity of DCFT.

Algorithm 1 Algorithm of DCFT

```
Input: Dataset $\mathcal{D}$;total step $T$; Kernel size $d$; stride $s$.
Create matrix $\mathcal{A}, \mathcal{B}, \mathcal{C}_d$;
for $t = 1, \ldots, T$ do
    Sample a mini-batch from $\mathcal{D}$;
    //Subspace low-rank approximation
    Compute input matrix $\mathcal{F} = \mathcal{A} * \mathcal{B}$;
    //Deconvolution fine-tuning
    Set stride $s =$ kernel size $d$;
    Reshape matrix $\mathcal{C}_d \rightarrow$ matrix $\mathcal{C}_s^T$;
    Deconvolution reconstruction $Conv^T(\mathcal{F})$;
    Compute incremental matrix $\Delta$ as 6;
    Add to the pre-trained weights.
end for
```

3.2.1 Low-Rank Matrice

We employed the LoRA methodology to decompose the initial matrix F into the product of $\mathcal{A} * \mathcal{B}$. As shown in equation 5, where $\mathcal{A} \in \mathbb{R}^{(k \times \frac{dout}{d})}$ and $\mathcal{B} \in \mathbb{R}^{(\frac{din}{d} \times k)}$. In this paper, we set $k = 1$. By using low-rank matrices, we have reduced the parameters by an order of magnitude, effectively enhancing parameter efficiency and training speed. The change in the number of parameters after adopting the low-rank matrix is shown in Equations 7 and 8.

$$Params_{before} = (\frac{d_{out}}{d} \times \frac{d_{out}}{d}) \cdot layers, \qquad (7)$$

$$Params_{after} = (\frac{d_{out}}{d} + \frac{d_{out}}{d}) \cdot layers. \qquad (8)$$

3.2.2 Equal Kernel Stride

DCFT sets the stride equal to the dimension of the convolution kernel, ensuring that there is no overlap between the outputs of each convolution operation. This means that each input element is computed only once. This significantly reduces redundant calculations and prevents instability and unpredictability caused by overlapping computations. Under the condition that the output matrix dimensions are fixed, the Equation for the input matrix is as follows:

$$D_{in} = \frac{(D_{out} - K + 2P)}{S} + 1, \qquad (9)$$

where K represents the kernel size, P represents the padding size, and S represents the stride. When K equals S, the equation simplifies to:

$$D_{in} = \frac{D_{out}}{S}. \tag{10}$$

Therefore, setting the stride equal to the dimension of the convolution kernel can effectively reduce the requirements for the input matrix, maximize the capability of deconvolution feature completion, and further reduce the computation and parameters.

3.2.3 Convolution kernel size

DCFT can control the number of parameters by adjusting the size of the convolution kernel. When a larger convolution kernel ($d = 8$) is used, the network architecture is further simplified, leading to a more uniform distribution of computations, effectively eliminating checkerboard artifacts, and enhancing the quality of upsampling. Moreover, the adoption of a larger kernel significantly reduces the number of parameters in the deconvolution input matrix. This reduction in parameter gradient computations accelerates model convergence, thereby enhancing the fine-tuning efficiency of DCFT.

When a smaller convolution kernel ($d = 2$) is used, the network is better able to capture the local features of the input data, allowing the model to better understand and extract local information, which aids in the model's generalization ability on new data. Detailed experimental validation is provided in Section 4.4.

4 Experiments

In this section, we conduct a comprehensive evaluation of our method through a series of experiments. We implement DCFT for fine-tuning DeBERTaV3-base (He et al., 2023), DeBERTaV2-xxl and RoBERTa-large (Liu et al., 2021), we assess the effectiveness of the proposed algorithm on natural language understanding (NLU) and question answering (QA).

4.1 Experimental Settings

Implementation Details. We implement all algorithms using PyTorch (Paszke et al., 2019). Our implementation is based on the publicly available code-base of Huggingface Transformers3 (Wolf et al., 2020). Experiments involving the DeBERTaV3-base (He et al., 2021) are performed on an NVIDIA 2080-ti GPU and experiments about RoBERTa-large (Liu et al., 2021) and DeBERTaV2-xxl are conducted on NVIDIA A800 GPU. Due to the bottleneck of rank one in the LoRA-base model, we compare the results of rank one with DCFT.

Baselines. Our baselines include full-parameter fine-tuning and other widely recognized parameter-efficient methods, including Bitfit (Zhang et al., 2022) , Adapter tuning (Houlsby et al., 2019; Pfeiffer et al., 2021), LoRA (Hu et al., 2022), AdaLoRA (Zhang et al., 2023b), LoRA-FA (Zhang et al., 2023a), and SoRA (Ding et al., 2023).

4.2 Natural Language Understanding

Models and Datasets. We fine-tune DeBERTaV3-base, DeBERTaV2-xxl (He et al., 2023) and RoBERTa-large (Liu et al., 2021) and adopt the GLUE benchmark (Wang et al., 2018) for evaluation. It's a widely recognized benchmark for natural language understanding, including CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP (Wang et al., 2018), STS-B (Wang et al., 2018), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016) and RTE (Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2011). Dataset details are summarized in Appendix A.1.

Main results. We first evaluate the task of Natural Language Understanding based on the GLUE benchmark. Two results for DCFT are reported: the one with the highest training efficiency and the one with the strongest performance. The experimental performance of DCFT and other baselines is shown in Table 1. To ensure the fairness of the experiments, we used the same hyperparameters and ensured that the parameters were closely matched. The results show that on the base model, DCFT outperforms the baselines in most tasks, and the trainable parameters are significantly fewer than those of the baselines. For example, in the RTE task, the accuracy of DCFT reached 88.45%, which is 5.43% higher than that of SoRA. This demonstrates that for some simple tasks, even if the parameters are set to $r = 1$, the trainable parameters still far exceed the requirements. On the RoBERTa-large and DeBERTaV2-xxl models, due to the expansion of the dimensionality of the incremental parameter matrix, the issue of parameter redundancy caused by rank-one decomposition is further enhanced. This results in more significant performance improvements for DCFT in most tasks. For

Model	Method	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg.
DeB$^{V3}_{base}$	Fine-Tune	184M	69.21	95.64	89.22	91.10	91.59	89.98/89.95	93.78	82.49	87.82
DeB$^{V3}_{base}$	Bitfit	0.1M	68.70	94.38	87.16	86.68	89.71	87.45/87.45	91.90	76.12	85.18
	HAdapter	0.31M	67.65	95.41	89.25	90.18	91.31	**90.10**/90.02	93.52	83.39	87.60
	PAdapter	0.3M	69.06	94.72	89.71	90.01	91.38	89.89/90.06	93.87	84.48	87.90
	LoRA	0.17M	68.60	94.95	88.24	89.79	91.41	90.09/90.28	93.35	81.29	87.23
	SoRA	0.12M	70.24	95.14	89.22	90.13	91.41	90.08/**90.41**	93.43	83.02	87.85
	LoRA-FA	0.12M	70.24	95.14	89.22	90.13	91.41	90.08/**90.41**	93.43	83.02	87.85
	DCFT$^{\Delta}$(ours)	**0.024M**	**71.07**	95.53	90.93	89.20	91.56	89.40/89.63	93.59	**88.45**	88.73
	DCFT†(ours)	0.079M	70.43	**96.10**	**91.42**	89.60	**91.76**	89.97/90.12	**94.14**	**88.45**	**88.99**
RoB$_{large}$	Fine-Tune	335M	68.0	96.4	90.9	92.2	92.4	90.2	94.7	86.6	88.93
RoB$_{large}$	HAdapter	0.8M	66.3	96.3	87.7	91.5	91.5	90.3	94.7	72.9	86.40
	PAdapter	0.8M	67.8	**96.6**	89.7	**91.7**	91.9	**90.5**	94.8	83.8	88.35
	LoRA	0.8M	68.0	96.2	90.2	91.1	91.9	90.0	94.4	86.3	88.51
	LoRA-FA	3.7M	68.0	96.0	90.0	91.1	92.0	90.1	94.4	86.1	88.46
	DCFT$^{\Delta}$(ours)	**0.06M**	68.5	96.0	**90.9**	90.6	**92.2**	90.1	94.2	87.0	88.69
	DCFT†(ours)	0.21M	**69.3**	96.4	90.7	91.5	**92.2**	**90.5**	**95.1**	**89.9**	**89.31**

Table 1: Test results of DCFT and other baselines on the GLUE benchmark are presented. We report both matched and mismatched accuracies for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. DCFT$^{\Delta}$ represents the most efficient results, while DCFT† denotes the optimal results. Higher scores indicate better performance for all metrics. We employ consistent hyperparameters, detailed in Appendix A.1. Optimal values are used as the final results. The best result is highlighted in **bold**, and the second best is underlined.

instance, in the CoLA task, DCFT uses only 0.21M and 0.18M trainable parameters, leading LoRA by 1.3% and 0.6%, respectively, and surpassing other LoRA-based approaches.

Method	#Params	CoLA	SST-2	STS-B	QNLI
Fine-Tune	1.5B	72.0	97.2	92.9	96.0
LoRA	4.7M	72.4	**96.9**	93.0	96.0
DCFT$^{\Delta}$(ours)	**0.18M**	**73.0**	96.8	**93.1**	**96.2**

Table 2: Results of DCFT fine-tuned on four different datasets on DeBERTaV2-xxl. Only the results with the best training efficiency are reported. The best result is highlighted in **bold**.

4.3 Question Answering

Models and Datasets. We fine-tune DeBERTaV3-base (He et al., 2023) on two question answering (QA) datasets: SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018). The Stanford Question Answering Dataset (SQuAD) is a key NLP resource featuring over 100,000 questions on Wikipedia articles for training question-answering models. Dataset details are summarized in Appendix A.2.

Main results. To evaluate the effectiveness of DCFT fine-tuning for QA tasks, we fine-tuned the DeBERTaV3-base model and assessed its performance on the SQuAD dataset, with results detailed in Table 3. We employed identical hyperparameters across our training processes. The outcomes reveal that DCFT surpasses baseline models on two SQuAD datasets, while only necessitating half the parameter count of these baselines. Specifically, on the SQuAD v2.0 benchmark, DCFT achieved scores of 84.7% and 87.7% on two key evaluation metrics: exact match (EM) and F1 score, marking an enhancement of 1.1% and 1.0% over LoRA.

Model	Method	#Params	SQuADv1.1	SQuADv2.0
DeB$^{V3}_{base}$	Fine-Tune	184M	86.0 / 92.7	85.4 / 88.4
DeB$^{V3}_{base}$	HAdapter	0.08%	84.4 / 91.5	83.4 / 86.6
	PAdapter	0.08%	84.4 / 91.7	84.2 / 87.2
	LoRA	0.08%	86.6 / 92.9	83.6 / 86.7
	DCFT(ours)	**0.04%**	**87.3 / 93.4**	**84.7 / 87.7**

Table 3: Test results of DCFT and other baselines on the SQuAD benchmark are presented. We report EM and F1. Higher scores indicate better performance for all metrics. We employ consistent hyperparameters, detailed in Appendix A.2. The best result is highlighted in **bold**.

4.4 Kernel Sizes Analysis

For a fixed model, adapting to different downstream tasks is undoubtedly challenging. Some

Method&# Param	Metric	SST-2	CoLA	QNLI	RTE	MRPC	STS-B	Avg.
Full FT	Acc	95.63	69.19	94.03	83.75	89.46	91.60	87.28
DCFT($d = 2$)	Acc	**96.10**	70.43	**94.14**	**88.45**	**91.42**	91.76	**88.72**
0.079M	Time	9.57h	0.97h	13.67h	1.63h	1.25h	0.97h	4.68h
DCFT($d = 4$)	Acc	95.76	69.24	93.76	**88.45**	90.20	**91.80**	88.20
0.041M	Time	7.32h	0.68h	12.27h	1.47h	1.27h	0.79h	3.97h
DCFT($d = 6$)	Acc	95.07	70.69	93.65	86.28	89.95	91.73	87.90
0.029M	Time	6.97h	0.64h	12.16h	1.45h	1.24h	0.77h	3.87h
DCFT($d = 8$)	Acc	95.53	**71.07**	93.59	**88.45**	90.93	91.56	88.52
0.024M	Time	**5.01h**	**0.54h**	**10.19h**	**1.08h**	**0.98h**	**0.63h**	**3.16h**
DCFT($d = 12$)	Acc	95.76	67.87	93.57	84.12	89.46	91.73	87.09
0.023M	Time	5.43h	0.62h	11.45h	1.28h	1.05h	0.72h	3.43h

Table 4: The results of DCFT with different convolution kernel sizes were tested on six datasets from GLUE benchmark, and we reported the accuracy and time efficiency. d represents the dimension of the convolutional kernel. We use the same hyperparameters, with only the convolution kernel size varying.

parameters contribute minimally to the final outcomes, not only failing to enhance the model's capabilities but also impacting the convergence speed. Therefore, adaptively adjusting the parameter budget according to needs is crucial. In this section, we have presented the results and training times of DCFT with different convolutional kernel sizes on six different tasks, as shown in Table 4. The analysis reveals that the model trains fastest and achieves suboptimal overall results when using a kernel size of ($d = 8$). Moreover, when the parameter budget was increased, most tasks showed improved outcomes. When using a kernel size of ($d = 2$), the model achieves the best average results due to the small convolutional kernel's ability to perceive details. However, for the smaller dataset task of CoLA, increasing the parameter budget actually resulted in decreased performance. It is noteworthy that using larger convolutional kernels ($d = 12$) leads to a significant decline in model performance. The primary reason is likely that the enlarged receptive field enables the model to capture excessive irrelevant information, while overlooking critical details, ultimately resulting in deteriorated outcomes.

4.5 Step Analysis

In this section, we conduct an ablation experiment on DCFT with different step sizes. We fix the convolution kernel size at eight and keep other hyperparameters consistent. The accuracy (Acc) and training time (Time) for step sizes of $1, 2, 4$, and 8 across four tasks are recorded. We do not consider cases where the step size exceeds the convolution kernel size, as this would result in parameters not participating in the computation, which contradicts the principles of PEFT. The results are shown in Table 5. From the analysis, it can be concluded that when the step size equals the kernel size, the model achieves the optimal results with the shortest training time. As the step size decreases, the training time progressively increases. When the step size $= 1$, the model attains the second-best results but incurs nearly three times the training time compared to a step size of 8. We believe the main issue arises when the step size is 4 or 6, as the model is affected by checkerboard artifacts, leading to knowledge overlap and offset. However, with a step size of 1, although the model is still impacted by checkerboard artifacts, the increase in parameter count compensates for this due to enhanced learning capacity, at the expense of significantly prolonged training time. Therefore, the results indicate that setting the step size equal to the convolution kernel size is an effective approach.

Method&Params	Metric	CoLA	MRPC	STS-B	RTE
DCFT(step= 1)	Acc	70.96	90.69	91.48	**88.45**
165.5K	Time	1.99h	1.22h	1.71h	1.62h
DCFT(step= 2)	Acc	70.04	90.44	91.37	88.09
85K	Time	0.89h	0.99h	0.89h	1.14h
DCFT(step= 4)	Acc	65.97	88.97	90.73	87.36
44.9k	Time	1.03h	0.99h	0.86h	1.11h
DCFT(step= 8)	Acc	**71.07**	**90.93**	**91.50**	**88.45**
24.6k	Time	**0.54h**	**0.98h**	**0.63h**	**1.08h**

Table 5: DCFT uses results with different step sizes, while we adopt the same convolution kernel size ($d = 8$) and ensure that other hyperparameters remain consistent. The best results are highlighted in **bold**.

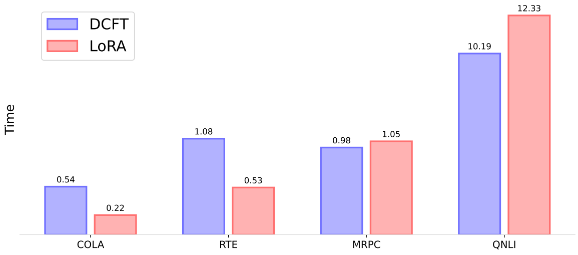

Figure 2: Illustration of the total training time for DCFT and LoRA on four datasets: COLA, RTE, MRPC, and QNLI.

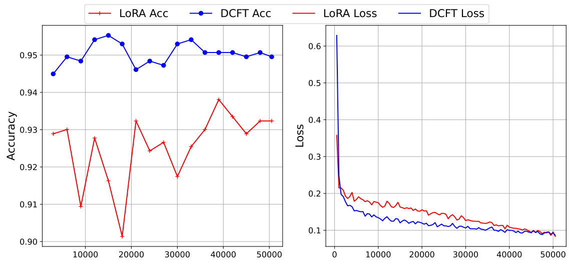

Figure 3: Accuracy and loss results of DCFT and LoRA on the SST-2 dataset.

4.6 Efficiency Analysis

In this section, we compare the training efficiency of DCFT and LoRA. Operating under the same computational infrastructure and with a batch size of 32, we calculate the total training time for four tasks, as shown in Figure 2. The results indicate that on smaller tasks like CoLA and RTE, DCFT's training efficiency is lower than LoRA's. However, as the task size increases, DCFT's training efficiency surpasses that of LoRA, particularly on the QNLI task, where it is 17.36% faster than LoRA.

Analysis shows that due to the high computational overhead of the deconvolution operation, DCFT performs slower than LoRA on small datasets. However, as the size of the task dataset increases, this complexity becomes negligible compared to the overall data processing load, and fewer parameters mean faster checkpoint storage and retrieval. Additionally, the advantage of fewer parameters in DCFT results in significantly reduced gradient computations at each step. Over long periods of computation, this difference accumulates, ultimately leading to a substantial reduction in training time. Furthermore, as shown in Table 2, DCFT converges faster than LoRA due to having fewer parameters. Therefore, with the trend towards larger models and fine-tuning datasets, the computational efficiency of DCFT will be further amplified.

Method	#Params	CoLA	MRPC	STS-B	RTE
DCFT$_{(Q,K)}$	6k	64.99	87.25	90.55	83.03
DCFT$_{(Q,V)}$	6k	64.50	89.95	91.43	83.03
DCFT$_{(Q,K,V)}$	9k	65.31	88.97	91.24	84.12
DCFT$_{(Q,K,V,O)}$	18.4k	68.37	89.95	91.50	85.92
DCFT$_{(ALL)}$	24k	**71.06**	**90.93**	**91.56**	**88.45**

Table 6: The results of applying DCFT across various layers are presented. The term ALL denotes the outputs from the query(Q), key(K), value(V), output matrix(O), and feed-forward layers.

4.7 Applying DCFT to Different layers

Notably, performance may fluctuate when applying parameter-efficient fine-tuning to different positions within the model. In this section, we apply DCFT to weight matrices at various positions to determine how to achieve optimal performance on downstream tasks. Using the DeBERTaV3-base model, we fine-tuned on the CoLA, MRPC, STS-B, and RTE datasets. As shown in Table 6, although DCFT is not primarily designed with parameter budget in mind, applying it to all weight matrices achieves the best results. Applying it specifically to the matrices Q, K, V, O follows closely in effectiveness, achieving the best results on the STS-B dataset. This suggests that, applying DCFT to all weight matrices is effective, and omitting the feed-forward layers result in a significant performance decline.

5 Conclusion

We propose a method named DCFT, which innovatively combines the reconstruction capabilities of deconvolution with the learning abilities of incremental matrices. This method enhances and completes the details of knowledge learned in subspaces and allocates parameter budgets based on the size of convolution kernels, free from the constraints of rank one decomposition. Additionally, we have implemented three principal measures to reduce the computational complexity of DCFT. This effectively lowers the model's computational complexity, allowing it to achieve better training efficiency on large datasets compared to LoRA. We conducted extensive experiments in natural language processing and question answering. The results show that DCFT achieves improved performance while reducing parameters by approximately 8×.

Acknowledgements

This work was supported in part by the Science and Technology Commission of Shanghai Municipality under Grant (21DZ2203100), in part by the National Natural Science Foundation of China under Grant (62006150), in part by Shanghai Local Capacity Enhancement project (21010501500) and in part by Science and Technology Innovation Action Plan of Shanghai Science and Technology Commission for social development project under Grant (21DZ1204900).

References

Luisa Bentivogli, Peter Clark, Ido Dagan, and et al. 2011. The seventh pascal recognizing textual entailment challenge. *Theory and Applications of Categories*.

David Bindel, James Demmel, William Kahan, and et al. 2002. On computing givens rotations reliably and efficiently. *Association for Computing Machinery Transactions on Mathematical Software*, pages 206–238.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, and et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, pages 1–113.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and et al. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

Ning Ding, Xingtai Lv, Qiaosen Wang, and et al. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4133–4145.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Vincent Dumoulin and Francesco Visin. 2018. A guide to convolution arithmetic for deep learning. *Preprint*, arXiv:1603.07285.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, and et al. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, pages 12799–12807.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and et al. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *Proceedings of the International Conference on Learning Representations*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and et al. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *Proceedings of the International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, and et al. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799.

Edward Hu, Yelong Shen, Phillip Wallis, and et al. 2022. Lora: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*.

Soroush Abbasi Koohpayegani, Navaneet K L, Parsa Nooralinejad, and et al. 2024. NOLA: Compressing lora using linear combination of random basis. In *The Twelfth International Conference on Learning Representations*.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 4582–4597.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *Preprint*, arXiv:2303.15647.

Zhuang Liu, Wayne Lin, Ya Shi, and et al. 2021. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the Chinese National Conference on Computational Linguistics*, pages 1218–1227.

OpenAI. 2023. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, and et al. 2024. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, page 15.

Adam Paszke, Sam Gross, Francisco Massa, and et al. 2019. *PyTorch: an imperative style, high-performance deep learning library*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

XiPeng Qiu, TianXiang Sun, YiGe Xu, and et al. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, page 1872–1897.

Alec Radford, Jeffrey Wu, Rewon Child, and et al. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Richard Socher, Alex Perelygin, Jean Wu, and et al. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Alex Wang, Amanpreet Singh, Julian Michael, and et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 353–355.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, pages 625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1112–1122.

Thomas Wolf, Lysandre Debut, Victor Sanh, and et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 38–45.

Zhiheng Xi, Wenxiang Chen, Xin Guo, and et al. 2023. The rise and potential of large language model based agents: A survey. *Preprint*, arXiv:2309.07864.

Shih yang Liu, Chien-Yi Wang, Hongxu Yin, and et al. 2024. DoRA: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*.

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833, Cham. Springer International Publishing.

Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. 2010. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.

Longteng Zhang, Lin Zhang, Shaohuai Shi, and et al. 2023a. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *Preprint*, arXiv:2308.03303.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, and et al. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the International Conference on Learning Representations*.

Qingru Zhang, Simiao Zuo, Chen Liang, and et al. 2022. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *Proceedings of the 39th International Conference on Machine Learning*, pages 26809–26823.

A Datasets

A.1 NLU Datasets

For evaluation, we adaopt the GLUE benchmark (Wang et al., 2018), including CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP (Wang et al., 2018), STS-B (Wang et al., 2018), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016) and RTE (Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2011). We present the dataset statistics of GLUE in the following table 7.

Dataset	Metric	#Train	#Valid	#Test	#Label
CoLA	Mcc	8.5k	1,043	1,063	2
SST-2	Acc	67k	872	1.8k	2
MRPC	Acc	3.7k	408	1.7k	2
QQP	Acc/F1	364k	40.4k	391k	2
STS-B	Corr	5.7k	1.5k	1.4k	1
MNLI	Acc(m/mm)	393k	20k	20k	3
QNLI	Acc	105k	5.5k	5.5k	2
RTE	Acc	2.5k	277	3k	2

Table 7: Dataset Sizes and Evaluation Metrics in the GLUE Benchmark. "Mcc," "Acc," "F1," and "Corr" denote the Matthews correlation coefficient, accuracy, F1 score, and Pearson correlation coefficient, respectively. "Acc(m/mm)" indicates accuracy results for matched and mismatched datasets within MNLI.

A.2 QA Datasets

For evaluation, we adaopt (QA) datasets: SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018). We present the dataset statistics of GLUE in the following table 8.

Dataset	Metric	#Train	#Valid
SQuAD v1.1	EM/F1	87,599	10,570
SQuAD v2.0	EM/F1	130,319	11,873

Table 8: Dataset Sizes and Evaluation Metrics in the SQuAD Benchmark. "EM," and "F1," denote the Exact Match and F1 Score.

B Sparse Regularization Theory

By using progressive projection matrices, we further increase the compression ratio p of the parameters. Additionally, this enhances the stability of the fine-tuning process. equations 11(Fu et al., 2023) theoretically demonstrate the role of sparsity in model stability. As the compression ratio p decreases, the upper bound also decreases. Therefore, a sparser model implies better stability.

$$E_{S,i\sim U(n)}[|\ell(A(S^i), z_i) - \ell(A(S), z_i)|] \\ \leq \frac{2\rho^2}{(\Lambda_{min} + 2(1-p))n}, \quad (11)$$

$E_{S,i\sim U(n)}[\cdot]$ is Pointwise Hypothesis Stability (PHS) (Bindel et al., 2002) which focuses on analyzing the change of model output after a training sample is removed. $\Lambda_{\text{min}} = \min\{\Lambda_1, \ldots, \Lambda_m\}$. $\ell(\cdot)$ represents the loss function. The variable ρ represents this measure of stability, reflecting the maximum impact of input variations on the output in the loss function.

C Orthogonal Projection Theory

It is a fundamental concept in linear algebra with applications across various fields including machine learning, statistics, and computer graphics. This theory revolves around the idea of projecting a vector onto a subspace in a way that minimizes the distance between the vector and the subspace, effectively finding the closest approximation within that subspace.

Mathematically, consider a vector u in R_n and a subspace $\mathbb{V}$ spanned by vectors $\{v_1, v_2, \ldots, v_k\}$. The orthogonal projection of u onto $\mathbb{V}$, denoted as $\mathbb{P}_{\mathbb{V}}(\mathbf{u})$, is given by:

$$\mathbb{P}_{\mathbb{V}}(\mathbf{u}) = \sum_{i=1}^{k} \frac{\mathbf{u}\cdot\mathbf{v}_i}{\mathbf{v}_i\cdot\mathbf{v}_i}\mathbf{v}_i. \quad (12)$$

This design allows each Low Rank block to capture information in different dimensions, thereby reducing information overlap and increasing the overall efficiency and effectiveness of the model. Additionally, the orthogonal training strategy helps prevent overfitting, making the model more robust when faced with new data.

D Checkerboard Artifacts

Checkerboard artifacts, commonly encountered in image generation or upsampling tasks using Convolutional Neural Networks (CNNs), are often associated with the use of strided or transposed convolutions. These operations can lead to uneven

Task	learning rate	batch size	r	Epochs	Dropout	γ	init_warmup	Δ_T
CoLA	5e-4	32	8	25	0.5	0.1	800	10
SST-2	8e-4	32	8	24	0.1	0	6,000	100
MRPC	1e-3	32	8	30	0.1	0	600	1
QQP	8e-4	32	8	20	0.1	0.15	8,000	100
STS-B	2.2e-3	32	8	25	0.1	0.2	800	10
MNLI	5e-4	32	8	7	0.1	0.15	8,000	100
QNLI	1.2e-3	32	8	6	0.1	0.1	2,000	100
RTE	1.2e-3	32	8	50	0.3	0.2	600	1

Table 9: Hyper-parameters setup of DCFT for GLUE benchmark.

Task	learning rate	batch size	r	Epochs	γ	init_warmup	Δ_T	t_f
AQuAD v1.1	1e-3	16	10	25	0.1	5,000	100	25000
AQuAD v2.0	1e-3	16	12	24	0.1	5,000	100	50000

Table 10: Hyper-parameters setup of DCFT for AQuAD benchmark.

coverage in the output images, resulting in visible grid-like patterns.

The formation of checkerboard artifacts can be described by the following equations. Suppose $f(x, y)$ represents the generated image and $k(x, y)$ is the convolutional kernel, applied with stride s during upsampling. Each pixel $p(x, y)$ in the generated image can be obtained through the convolution operation:

$$p(x, y) = \sum_{i,j} f(i, j) \cdot k(x - si, y - sj), \quad (13)$$

where i and j traverse all valid pixel coordinates. When the stride s is greater than 1, $k(x, y)$ may not cover some positions in x or y completely, causing discontinuities at these points in the output $p(x, y)$ and forming a checkerboard pattern.

To mitigate this issue, researchers have developed various strategies, such as replacing strided convolutions with interpolation methods (like bilinear interpolation) or optimizing the design of convolution kernels to avoid uneven overlaps. Another approach involves adjusting the relationship between stride and kernel size to ensure uniform influence on each output pixel. For instance, fractionally strided convolutions can be utilized, where the formula is modified to:

$$p(x, y) = \sum_{i,j} f(i, j) \cdot k\left(\frac{x}{s} - i, \frac{y}{s} - j\right). \quad (14)$$

Through these methods, checkerboard artifacts can be reduced or eliminated to enhance image quality.

E Hyperparameters

Regarding hyperparameters, We tune the learning rate and pick the best learning rate for every method. For each dataset, the batch size is set as identical for every method. The majority of hyperparameters used in evaluating DCFT on Natural Language Understanding and Question Answering are shown in Table 9 and 10.