Memory-Efficient Training with In-Place FFT Implementation

Xinyu Ding[1], **Bangtian Liu, Siyu Liao**[1]*, **Zhongfeng Wang**[1]
[1]School of Integrated Circuits, Sun Yat-sen University
{dingxy33, liaosy36, wangzf83}@mail.sysu.edu.cn, liubangtian@gmail.com,

Abstract

Fast Fourier Transforms (FFT) are widely used to reduce memory and computational costs in deep learning. However, existing implementations, including standard FFT and real FFT (rFFT), cannot achieve true in-place computation. In particular, rFFT maps an input of size n to a complex output of size $n/2 + 1$, causing dimensional mismatch and requiring additional memory allocation. We propose the first real-domain, fully in-place FFT framework (rdFFT) that preserves input-output memory space consistency. By leveraging butterfly operation symmetry and conjugate properties in the frequency domain, we design an implicit complex encoding scheme that eliminates intermediate cache usage entirely. Experiments on multiple natural language understanding tasks demonstrate the method effectiveness in reducing training memory cost, offering a promising direction for frequency-domain lightweight adaptation.

1 Introduction

Large-scale neural models have achieved remarkable success across a wide range of applications, such as natural language processing [1][2] and computer vision [3]. As model sizes increase, memory consumption has emerged as a significant challenge. Notably, model training memory cost is larger than the deployment cost, primarily due to the backpropagation process [4]. Thus, reducing memory usage has become a critical research study, especially for the training stage.

Numerous methods have been proposed to reduce memory usage during model training and deployment. Commonly used approaches such as model distillation [5], quantization [6], and pruning [7] mainly aim to reduce memory consumption by decreasing the number or precision of model parameters. In contrast, this work takes a novel approach at the arithmetic operator level, implementing in-place Fast Fourier Transform (FFT) operations for model training through the use of circulant-structured parameter matrices [8–10]. The FFT operator has been widely employed in various neural network architectures due to its efficiency in capturing global patterns and enabling structured transformations. For instance, Fourier-based fine-tuning methods such as FourierFT [11] and Block Circulant Adapter (BCA) [10] rely heavily on FFT to perform efficient parameter transformations in the frequency domain.

It should be noted that FFT-based operators typically involve both FFT and IFFT computations, which generate intermediate tensors that consume memory and use different data type since the result of FFT is complex number. Numerous studies have explored memory optimization strategies for FFT operators, particularly in high-performance programming libraries such as FFTW [12] and cuFFT [13]. These libraries support in-place computation, where input and output share memory space. However, they suffer from the inability to maintain the original (input) memory space and lack of support for bfloat16 data type that is widely used for modern neural models.

*Siyu Liao is the corresponding author.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

To overcome the limitations of existing FFT libraries, we introduce rdFFT—a real-domain, fully in-place Fourier transform that produces the same output as rFFT, but operates entirely within the original n real-valued input memory space. We notice that the first and middle point of FFT results have zeros in their imaginary part. Thus, they can be squeezed together so the final output only requires n real-valued input memory space. Besides, our method exploits the symmetry of butterfly operations and the conjugate structure of real-valued spectra to implicitly encode complex information within real-valued tensors. This design enables in-place computation without the need for auxiliary buffers or dimension mismatches, facilitating seamless integration into modern deep learning workflows. Crucially, our in-place design supports consistent forward and backward passes entirely within the real domain. In summary, our main contributions are as follows:

- We propose rdFFT, a novel real-valued, fully in-place Fourier transform operator that eliminates memory space mismatches, offering improved practicality and usability for neural network applications compared to existing libraries.
- We introduce a new memory layout design, develop a novel butterfly execution scheme for IFFT computation, and provide support for the bfloat16 data type, which is widely used in modern neural networks.
- We integrate our rdFFT into neural network models via circulant-structured parameter matrix and validate on real-world models, achieving zero-memory allocation for intermediate tensor computations.

2 Related Works

Current automatic differentiation frameworks often discourage in-place operations due to their challenges in gradient computation during training. The key benefit of in-place computation lies in its ability to save memory by eliminating the need for storing intermediate tensors. For instance, combining batch normalization and activation into a single in-place operation has been shown to reduce memory usage by up to half [14]. In neural network based hardware placement tasks [15], in-place operations are found beneficial due to their inherent memory efficiency. Similarly, in-place operations are integrated into convolutional neural networks for anomaly detection [16], achieving notable memory savings. The in-place operation can also be extended to broader contexts. For example, in in-place distillation [17], a student model is distilled directly from the teacher model without additional memory allocation.

The FFT operator can be found across various neural network models, particularly in tasks involving the Fourier domain, which is common in computer vision. Some convolutional neural networks run entirely in the Fourier domain [18]. In beampattern synthesis [19], an IFFT operator is applied to the hidden representations generated by the neural network. Additionally, 2D FFT has proven beneficial for fine-tuning large language models [11]. Circulant-structured matrix-vector products can also be efficiently computed using 1D FFT and IFFT, facilitating neural network training [10]. Beyond these applications, FFT also plays a key role in spectral convolution [20] and in approximating global attention mechanisms [21].

From the perspective of automatic differentiation, each operator can be seen as a neural network layer, with or without trainable parameters. For example, the low-rank fine-tuning method for large language models [22] represents a full weight matrix using two low-rank matrices, which is equivalent to introducing two linear layers, thereby requiring storage for intermediate activations. Similarly, circulant-structured weight matrices [10] leverage FFT and IFFT to transform inputs and parameters. Although FFT operators themselves does not contain trainable parameters, they still require the preservation of intermediate results to support automatic differentiation and to handle complex-valued data types that is different from the real-valued inputs. Moreover, libraries such as FFTW [12] and cuFFT [13] demand pre-allocated memory buffers of size $N + 2$ real numbers. This memory pre-allocation should be handled during the neural model loading phase, which complicates integration and limits the practicality of using these libraries in real-world neural network applications. Besides, they also does not support bfloat16 data type which is common in modern neural networks.

3 Preliminary

3.1 Standard FFT and rFFT

Fast Fourier Transform (FFT) is a computationally efficient algorithm for computing the Discrete Fourier Transform and its inverse. Given a sequence of N real or complex values $x(n), n = 0, 1, \ldots, N-1$, the FFT and inverse FFT are defined as follows:

$$y_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn}, \quad x_n = \frac{1}{N}\sum_{k=0}^{N-1} y_k \cdot e^{i\frac{2\pi}{N}kn}. \tag{1}$$

This formulation, referred to as the standard FFT, transforms N input elements into N complex outputs. However, when the input is real, the FFT exhibits Hermitian symmetry, i.e., $y_{N-k} = \overline{y_k}$. Real-valued input based FFT (rFFT) exploits this property by computing only the first $N/2 + 1$ complex values.

The rFFT significantly reduces the memory requirement compared to the complex-valued FFT, decreasing the output size from $2N$ real numbers (i.e., N complex numbers) to $N + 2$ real numbers. This improvement is based on the following fundamental property of the FFT:

Theorem 1 (Conjugate Symmetry of Real FFT [23])**.** *Let $x \in \mathbb{R}^N$ be a real-valued sequence, and let y_k denote its Fast Fourier Transform (FFT). Then the FFT output satisfies the conjugate symmetry property:*

$$y_{N-k} = \overline{y_k}, \quad \textit{for all } k = 1, 2, \ldots, \left\lfloor \frac{N}{2} \right\rfloor. \tag{2}$$

This property implies that the FFT of a real-valued signal is *redundant*—the full frequency-domain spectrum can be uniquely reconstructed from only the first $\lfloor \frac{N}{2} \rfloor + 1$ complex coefficients: $y_0, y_1, \ldots, y_{\lfloor \frac{N}{2} \rfloor}$. Leveraging this redundancy, rFFT implementations store only the non-redundant half of the spectrum, thereby reducing both computation and memory footprint.

However, this comes at the cost of mismatched memory sizes between the input and output: in the FFT, N real-valued inputs are transformed into outputs that take the memory space of $N + 2$ real values; in the IFFT, the complex inputs taking the space of $N + 2$ real values are mapped back to N real outputs. In both directions, the input and output cost different amount of memory.

This memory misalignment is difficult for neural network training, where tensors are generally allocated with fixed shapes. The requirement to expand an N-element real tensor to $N + 2$ elements necessitates either pre-allocation or runtime reallocation, which causes integration difficulty, prevents true in-place computation and potentially incur significant overhead in memory-constrained environments.

3.2 In-place Transform via Butterfly Operation

The FFT can be efficiently computed using the Cooley–Tukey algorithm [24], which recursively decomposes given FFT into smaller FFTs. As illustrated in the Butterfly Operation Diagram section of Fig.1, a 16-point FFT is first recursively decomposed into smaller FFTs—two 8-point FFTs, then four 4-point FFTs, and finally eight 2-point FFTs. Following this hierarchical decomposition, the FFT is computed through a series of butterfly operations applied at each level of the recursion. At the core of the algorithm lies the butterfly operation, a fundamental computation that transforms a pair of complex values into two outputs using addition, subtraction, and multiplication by a twiddle factor.

As illustrated in the blue-shaded region of Fig.1, the "Complex-to-Complex FFT" section provides two butterfly operations, both originating from FFT16, which are highlighted in red. The butterfly operation for an r-point FFT is defined as:

$$\begin{aligned} y_k &= x_k + x_{k+\frac{r}{2}} \cdot W_N^m, \\ y_{k+\frac{r}{2}} &= x_k - x_{k+\frac{r}{2}} \cdot W_N^m, \end{aligned} \tag{3}$$

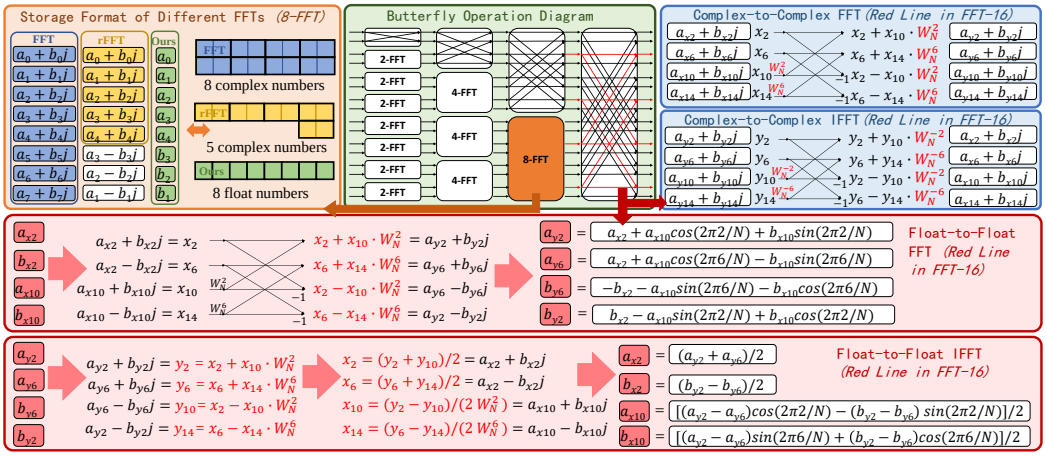


Figure 1: Overview of our method and its differences from standard FFT and rFFT implementations. The green section depicts the Butterfly Operation Diagram using a 16-point FFT (16-FFT) as an example. The orange section illustrates the storage formats of different FFT implementations, shown on an 8-point FFT (8-FFT). Two representative butterfly computation paths in the 16-FFT are highlighted in red, and expanded into: (i) the blue section showing Complex-to-Complex FFT and IFFT operations, and (ii) the red section showing Float-to-Float FFT and IFFT operations—both derived from the red paths in the 16-FFT diagram. This figure summarizes the key computational flows and memory layouts addressed by our in-place real-domain FFT design.

where W_N^m is the twiddle factor, N is the total size of the FFT, and r is the size of the current sub-FFT. The exponent m is given by $m = 1 + \log_2(N/r)$. For each r-point FFT, there are $\frac{r}{2}$ such butterfly pairs, with $k = 0, 1, \ldots, r/2 - 1$. The same butterfly operations are used in the "Complex-to-Complex IFFT" section, differing only in the choice of twiddle factors. The overall structure of the Cooley–Tukey FFT and its inverse (IFFT) is almost the same, except that the IFFT applies a final normalization factor of $1/N$.

It can be noticed that butterfly operations are inherently in-place, allowing outputs to overwrite inputs without the need for extra memory. This makes FFT especially suitable for memory-efficient implementations. However, since most neural network computations involve real-valued tensors, converting these to complex-valued representations usually requires additional memory allocation. With real-valued input and in-place constraint, given the aforementioned memory mismatch, the butterfly process is not directly applicable.

3.3 FFT based Model Training via Circulant Matrix

The circulant matrix based neural network training can be converted to FFT and IFFT operations for acceleration [8]. Given a real-valued circulant weight matrix $\mathbf{C} \in \mathbb{R}^{N \times N}$ defined by its first column $\mathbf{c} \in \mathbb{R}^N$, the linear transformation $\mathbf{y} = \mathbf{C}\mathbf{x}$ can be equivalently computed in the frequency domain as follows:

$$\mathbf{y} = \texttt{IFFT}\left(\texttt{FFT}(\mathbf{c}) \odot \texttt{FFT}(\mathbf{x})\right), \tag{4}$$

where $\odot$ denotes elementwise multiplication.

This structure not only accelerates computation but also simplifies the gradient computation during training. By leveraging the linearity and conjugate of the FFT results [10], gradients with respect to both input $\mathbf{x}$ and parameter $\mathbf{c}$ can be computed by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \texttt{IFFT}\left(\overline{\texttt{FFT}(\mathbf{c})} \odot \texttt{FFT}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}}\right)\right), \quad \frac{\partial \mathcal{L}}{\partial \mathbf{c}} = \texttt{IFFT}\left(\overline{\texttt{FFT}(\mathbf{x})} \odot \texttt{FFT}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}}\right)\right), \tag{5}$$

where $\overline{\texttt{FFT}(\cdot)}$ means taking the conjugate of the FFT results. There is also block circulant matrix based training that aims to fit non-square matrix [9], where the matrix is divided into blocks by partition size p.

4 Method

4.1 In-place FFT with Real-valued Input

While the Cooley–Tukey algorithm enables efficient in-place computation of the Fast Fourier Transform (FFT), it operates inherently in the complex domain. However, in modern neural networks, the vast majority of model parameters are real-valued. Transitioning between real and complex representations introduces unnecessary memory overhead, especially in memory-constrained training settings. To address this issue, we propose new strategies to reduce the memory footprint of Fourier-based transformations at the layer level.

Conjugate Symmetry in Cooley–Tukey Sub-FFTs. The Cooley–Tukey algorithm recursively decomposes a length-N FFT into smaller FFTs. If the original input sequence $x \in \mathbb{R}^N$ is real-valued, then each recursively computed sub-FFT also receives a real-valued input (or a linear combination of real values). Since the FFT is a linear operation and conjugate symmetry is preserved under linear combinations, each sub-FFT applied to real-valued data also satisfies the conjugate symmetry property:

$$y_{r-k} = \overline{y_k}, \quad \text{for all } k = 1, \ldots, \left\lfloor \frac{r}{2} \right\rfloor, \tag{6}$$

where y denotes the output of a size-r FFT block. Thus, conjugate symmetry is preserved at every level of the Cooley–Tukey decomposition. This property enables memory-efficient computation in real-input FFTs throughout the recursive stages.

Squeeze $N + 2$ into N. Although the rFFT reduces memory usage of complex-valued FFT by exploiting conjugate symmetry, it stores $N/2 + 1$ complex numbers in the memory space of $N + 2$ real numbers, which is different from the the original real-valued input of length N. This result does not allow for true in-place computation within the original real-valued buffer. We further analyze the structure of the sub-FFTs in the Cooley–Tukey decomposition. For any r-point FFT of real-valued input, the output satisfies:

$$y_0,\ y_{r/2} \in \mathbb{R}, \quad \text{and} \quad y_k = \overline{y_{r-k}},\ \text{for } k = 1, \ldots, r/2 - 1.$$

This implies that only the real parts of y_0 and $y_{r/2}$ need to be stored. For remaining $r - 2$ complex values, we only store the real and imaginary parts of $y_1, \ldots, y_{r/2-1}$, and the rest can be reconstructed via conjugation. As a result, we successfully reduce the memory space of $N + 2$ real values to the size of N real values.

Memory Layout Design. Based on aforementioned observation, we propose a data layout design where each complex coefficient y_k ($1 \leq k < r/2$) stores its real part at index k and stores its imaginary part at the conjugate-symmetric index $r - k$. In this way, the entire frequency-domain representation fits into a real-valued buffer of size r, without requiring any complex-valued memory. The special cases y_0 and $y_{r/2}$, which are always real, each occupy a single real-valued slot. The "Storage Format of Different FFTs" in Fig.1 illustrates our layout design in comparison to standard FFT and rFFT formats.

Proposition 1. *In the Cooley–Tukey FFT algorithm on real-valued input, at each stage of the recursive decomposition, every conjugate-symmetric pair and its butterfly counterparts form a symmetric four-element group. This structural symmetry ensures that the butterfly operations preserve conjugate symmetry and can be performed entirely in-place.*

Proof. Consider two conjugate-symmetric outputs $x(a_1)$ and $x(b_1)$ from an m-point FFT stage, where their indices are given by

$$a_1 = 2km + \tfrac{m}{2} - i, \quad b_1 = 2km + \tfrac{m}{2} + i$$

for some integers k and i. In the next $2m$-point FFT stage, these values participate in butterfly operations with their counterparts at

$$a_2 = (2k + 1)m + \tfrac{m}{2} - i, \quad b_2 = (2k + 1)m + \tfrac{m}{2} + i.$$

Define the center of the $2m$-point FFT block as $c = (2k+1)m$. Then, the offsets from the center are:

$$c - a_1 = \tfrac{m}{2} + i, \qquad c - b_1 = \tfrac{m}{2} - i,$$
$$c - a_2 = -\tfrac{m}{2} + i, \qquad c - b_2 = -\tfrac{m}{2} - i.$$

This confirms that the set $\{a_1, b_1, a_2, b_2\}$ is symmetric with respect to the center index c, and forms two conjugate-symmetric pairs placed at symmetric offsets.Since the FFT butterfly operations preserve conjugate symmetry when applied to conjugate inputs, the output values at this stage also maintain the same symmetry.

Therefore, all computations involving these four values can be performed in-place, and the symmetric layout remains valid in the next FFT stage. □

This symmetry ensures that at every level of the decomposition, each butterfly involving a conjugate pair produces another conjugate pair. As a result, conjugate symmetry is recursively preserved across all stages of the Cooley–Tukey FFT algorithm.

As a concrete example, consider the case shown in Fig. 1 (Float-to-Float FFT), where an 8-point FFT stage has a conjugate pair located at indices $a = 2$ and $b = 6$. In the following 16-point FFT stage, these values participate in butterfly operations with elements at indices 10 and 14. The resulting index pairs $(2, 14)$ and $(6, 10)$ are symmetric with respect to the center of the 16-point block and thus also form conjugate pairs.

Consequently, the interleaved memory layout—where the real part of y_k is stored at index k and the imaginary part at index $r - k$—remains consistent throughout the entire FFT computation. This enables the algorithm to be executed entirely in-place within a real-valued buffer, eliminating the need to explicitly store redundant conjugate components.

Based on the recursive structure and symmetry, we summarize our in-place real-domain FFT algorithm:

1. Store real and imaginary parts of conjugate pairs in an interleaved layout within the input real-valued memory space;
2. Perform all butterfly operations entirely in-place without auxiliary buffers;
3. Reconstructs the complex frequency spectrum from a real input of length N.

Overall, our design enables fully in-place FFT computation for real-valued input, with zero memory overhead and compatible with the Cooley–Tukey algorithm.

4.2 In-place IFFT with Symmetric Complex-valued Input

While the FFT benefits from conjugate symmetry in real-valued inputs, the inverse transform receives conjugate-symmetric complex values as input. However, unlike FFT computation, the outputs of sub-IFFTs in the Cooley–Tukey recursion are not guaranteed to be real-valued or conjugate-structured. This makes it difficult to directly apply the aforementioned in-place memory sharing mechanism.

To overcome the input difference in IFFT, we exploit the linearity of the FFT. Since each butterfly operation is a linear combination of its inputs, we can reverse the forward FFT computation structure to recover the original real-valued signal. Specifically, we compute the inverse using the same butterfly graph but reverse the direction of data flow and scale the results appropriately.

As illustrated in Fig.1 (Float-to-Float IFFT section), we compute intermediate outputs from the complex input Y as follows:

$$\begin{aligned} x_2 &= \frac{y_2 + y_{10}}{2}, \quad x_6 = \frac{y_6 + y_{14}}{2}, \\ x_{10} &= \frac{y_2 - y_{10}}{2W_N^2}, \quad x_{14} = \frac{y_6 - y_{14}}{2W_N^6}, \end{aligned} \tag{7}$$

where $W_N^k = \exp(-2\pi ik/N)$ denotes the k-th twiddle factor.

In this formulation, we split each conjugate pair into symmetric and anti-symmetric components, enabling recovery of the original signal through only real-domain operations. By carefully reusing buffer locations during this process, we implement the in-place IFFT computation without auxiliary memory for intermediate complex arrays.

Table 1: Peak GPU memory usage (in MB) measured during single layer training (up to the end of the backward pass) for different methods under varying input shapes. For inputs of shape $D = 4096$, LoRA uses rank 64; for $D = 1024$, the rank is 32. Entries marked as "N/A" indicate that the specified block size (e.g., 4096) is incompatible with the given input shape (e.g., 1024). Values in parentheses denote how many times memory is reduced compared to full fine-tuning.

GPU Mem. (MB)	D = 4096			D = 1024		
	B=1	B=16	B=256	B=1	B=16	B=256
full-finetune	144.33	145.50	164.25	24.27	24.56	29.25
lora	20.31(×7.11)	21.25(×6.85)	39.38(×4.17)	16.77(×1.45)	17.00(×1.44)	21.69(×1.35)
$\text{fft}_{p=128}$	3.65(×39.55)	35.88(×4.06)	551.50(×0.30)	0.25(×95.22)	2.66(×9.22)	41.22(×0.71)
$\text{rfft}_{p=128}$	3.14(×45.93)	35.14(×4.14)	547.13(×0.30)	0.22(×111.20)	2.53(×9.72)	40.30(×0.73)
$\text{ours}_{p=128}$	**1.06**(×135.78)	**2.00**(×72.73)	**20.50**(×8.01)	**0.08**(×308.73)	**0.34**(×71.35)	**5.03**(×5.81)
$\text{fft}_{p=256}$	1.89(×76.24)	19.03(×7.65)	293.25(×0.56)	0.15(×166.80)	1.61(×15.24)	25.08(×1.17)
$\text{rfft}_{p=256}$	1.62(×89.17)	18.35(×7.93)	286.06(×0.57)	0.12(×194.92)	1.48(×16.63)	24.02(×1.22)
$\text{ours}_{p=256}$	**0.56**(×256.36)	**1.50**(×96.97)	**20.25**(×8.11)	**0.05**(×512.42)	**0.33**(×74.74)	**5.02** (×5.83)
$\text{fft}_{p=512}$	1.02(×141.97)	10.63(×13.68)	164.50(×1.00)	0.09(×267.23)	1.09(×22.60)	17.03(×1.72)
$\text{rfft}_{p=512}$	0.86(×167.28)	10.03(×14.51)	156.66(×1.05)	0.08(×312.61)	0.96(×25.68)	15.05(×1.94)
$\text{ours}_{p=512}$	**0.31**(×461.14)	**1.38**(×105.78)	**20.13** (×8.16)	**0.03** (×764.69)	**0.32** (×76.56)	**5.01**(×5.84)
$\text{fft}_{p=1024}$	0.58(×249.23)	6.44(×22.59)	100.22(×1.64)	0.06(×382.35)	0.83(×29.76)	13.01(×2.25)
$\text{rfft}_{p=1024}$	0.49(×295.88)	5.88(×24.73)	92.24(×1.78)	0.05(×447.79)	0.70(×35.15)	11.02(×2.65)
$\text{ours}_{p=1024}$	**0.19**(×767.76)	**1.31**(×110.82)	**20.06** (×8.19)	**0.02** (×1,014.39)	**0.32** (×77.50)	**5.00**(×5.84)
$\text{fft}_{p=4096}$	0.25(×575.07)	3.30(×44.12)	52.05(×3.16)	N/A	N/A	N/A
$\text{rfft}_{p=4096}$	0.21(×698.79)	2.78(×52.25)	44.04(×3.73)	N/A	N/A	N/A
$\text{ours}_{p=4096}$	**0.09**(×1,531.54)	**1.27**(×114.92)	**20.02**(×8.21)	N/A	N/A	N/A

Symmetry in Circulant Matrix based Training. According to Eq. 4, even though the FFT results are naturally symmetric for real-valued inputs, there is elementwise multiplication between two FFT results. Given that $\overline{A \cdot B} = \overline{A} \cdot \overline{B}$, it follows that the elementwise multiplication result maintains the symmetry property as in FFT results of input and circulant weight vector. Therefore, the IFFT operation input in Eq. 4 and Eq. 5 are with symmetric complex-valued input.

5 Experiments

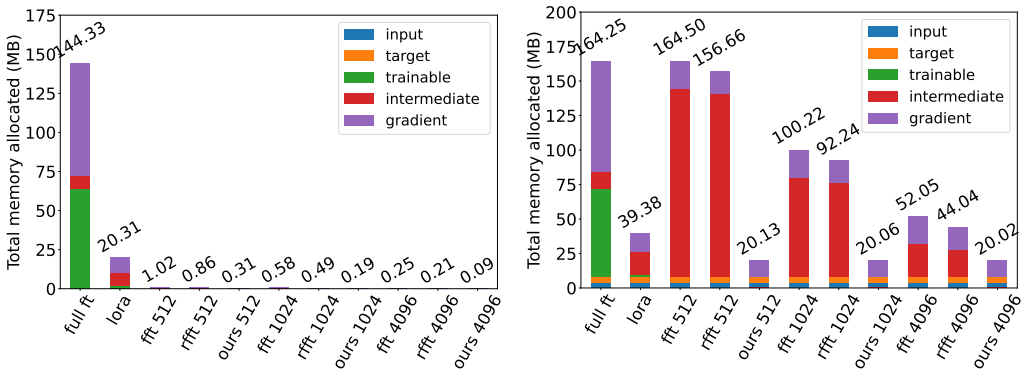
All our experiments compare three different FFT implementations: **(1) fft:** standard complex-valued FFT using `torch.fft.fft/ifft` from PyTorch [25]; **(2) rfft:** real-input FFT using `torch.fft.rfft/irfft`, exploiting Hermitian symmetry; **(3) ours:** custom CUDA-based real-domain FFT with in-place forward/backward implementation, reusing the input real-valued memory for intermediate result storage.

We also include two common baselines: **(1) FF**: updating all trainable parameters; **(2) lora**[22]: low-rank adaptation with parameter-efficient updates. For fair comparison, all runs use the same training configuration (batch size, optimizer, precision).

5.1 Memory Efficiency

To evaluate the memory efficiency of our proposed in-place training, we conduct experiments in two settings: **(1) single-layer analysis:** we perform training on a singular linear layer with different training methods, where circulant matrix based training [10] are accomplished with different FFT implementations; **(2) full-model training:** we apply the circulant fine-tuning approach [10] to RoBERTa-large and LLaMA2-7B and monitor memory usage throughout training. All circulant variants share the same number of trainable parameters, differing only in FFT backend.

Peak memory is recorded using PyTorch memory profiler. To better understand the memory distribution, we also visualize the breakdown of memory usage (model weights, trainable params, gradients, others) in both the single-layer and full-model training.



(a) Memory allocation ($B = 1$)

(b) Memory allocation ($B = 256$)

Figure 2: Memory breakdown during single-layer fine-tuning with hidden dimension $D = 4096$, under two batch sizes: (a) $B = 1$ and (b) $B = 256$. *Intermediate tensors* are allocated during the forward pass, while *gradients* appear in the backward pass. This illustrates how batch size impacts memory allocation for activations and gradients.

5.1.1 Single Layer Training

Setups. To isolate the memory overhead introduced by different FFT implementations, we conduct controlled experiments on a single fine-tuned layer using an NVIDIA A100 GPU. We vary the input dimension $\mathbf{x} \in \mathbb{R}^{B \times D}$, with $D \in \{1024, 4096\}$ and batch size $B \in \{1, 16, 256\}$. For circulant-based methods, we further vary the block size to evaluate its influence on memory usage. Peak memory is recorded during both forward and backward passes, and the results are reported in Tab.1. Fig.2 shows the memory breakdown for the large setup ($D = 4096$) under batch sizes $B = 1$ and $B = 256$, highlighting the memory footprint of intermediate tensors created during forward computation and gradients allocated during backpropagation. This breakdown reveals the impact of in-place FFT on reducing transient memory usage.

Results. Tab.1 and Fig.2 clearly demonstrate the effectiveness of our in-place method. When the batch size is small, all circulant variants show memory advantages over full fine-tuning and LoRA. However, as the block size decreases and the batch size increases, standard FFT-based circulant layers incur increasing overhead from intermediate tensors, especially during the forward pass. In contrast, our method performs the entire forward computation in-place, introducing no intermediate tensors. As shown in Fig.2, this leads to significant memory savings in the single-layer setup. Moreover, by overwriting the `grad_output` in-place at the final stage of the backward pass, our method also reduces memory usage during gradient computation.

5.1.2 Full Model Training

Setups. We conduct full-model experiments on both LLaMA2-7B and RoBERTa-large using an NVIDIA A100 GPU. For LLaMA2-7B, we use the GSM8K dataset with `per_device_train_batch_size` set to 2 and `gradient_accumulation_steps` set to 4. For RoBERTa-large, we use the MRPC dataset with a batch size of 32. These configurations follow the standard precision training setups for each task [10], and we retain them to reflect real-world GPU memory cost. We use stochastic gradient descent (SGD) as the optimizer in all experiments.

Results. From Tab.2, we observe that all methods have the same *base model* memory, and the memory for *trainable_params* is negligible compared to the total model size. It can be noticed that *others* take up the second largest memory consumption, which is managed by PyTorch framework for storing activations, dynamic memory allocation and release, etc. In LLaMA2-7B, *gradient* memory is approximately twice that of *trainable_params* because the forward pass uses `bf16` to reduce memory usage, but gradients must be stored in `float32` as backward computations do not support `bf16`. In contrast, RoBERTa-large uses full-precision training, so the gradient memory matches the parameter memory size.

Table 2: Peak GPU memory usage across different training stages during one epoch on LLaMA2-7B and RoBERTa-large. *model* indicates memory used to load the base model; *trainable* refers to memory allocated for trainable parameters; *gradient* denotes the analytically estimated memory for gradients of trainable parameters; *others* represents the remaining memory, computed as the difference between the peak usage and the sum of the above three, accounting for buffers, activations, and miscellaneous overhead.

Method	LLaMA2-7b					Method	RoBERTa-Large				
	model (GB)	trainable (MB)	gradient (MB)	others (GB)	total (GB)		model (GB)	trainable (MB)	gradient (MB)	others (GB)	total (GB)
FF	12.61	0.00	6144.00	8.28	26.90	FF	1.33	0.00	192.00	4.63	6.15
$\text{lora}_{r=32}$	12.61	48.00	96.00	6.20	18.96	$\text{lora}_{r=8}$	1.33	3.00	3.00	4.90	6.24
$\text{lora}_{r=64}$	12.61	96.00	192.00	6.26	19.15	$\text{lora}_{r=16}$	1.33	6.00	6.00	4.92	6.26
$\text{fft}_{p=512}$	12.61	6.00	12.00	8.17	20.81	$\text{fft}_{p=256}$	1.33	0.75	0.75	5.39	6.72
$\text{rfft}_{p=512}$	12.61	6.00	12.00	6.65	19.28	$\text{rfft}_{p=256}$	1.33	0.75	0.75	4.80	6.13
$\text{ours}_{p=512}$	**12.61**	**6.00**	**12.00**	**5.30**	**17.93**	$\text{ours}_{p=256}$	**1.33**	**0.75**	**0.75**	**4.44**	**5.77**
$\text{fft}_{p=1024}$	12.61	3.00	6.00	6.58	19.20	$\text{fft}_{p=512}$	1.33	0.38	0.38	5.41	6.74
$\text{rfft}_{p=1024}$	12.61	3.00	6.00	6.55	19.17	$\text{rfft}_{p=512}$	1.33	0.38	0.38	4.76	6.08
$\text{ours}_{p=1024}$	**12.61**	**3.00**	**6.00**	**5.30**	**17.92**	$\text{ours}_{p=512}$	**1.33**	**0.38**	**0.38**	**4.44**	**5.77**
$\text{fft}_{p=4096}$	12.61	0.75	1.50	8.09	20.71	$\text{fft}_{p=1024}$	1.33	0.19	0.19	5.39	6.72
$\text{rfft}_{p=4096}$	12.61	0.75	1.50	6.55	19.16	$\text{rfft}_{p=1024}$	1.33	0.19	0.19	4.76	6.09
$\text{ours}_{p=4096}$	**12.61**	**0.75**	**1.50**	**5.29**	**17.91**	$\text{ours}_{p=1024}$	**1.33**	**0.19**	**0.19**	**4.44**	**5.77**

It is also worth noting that fft and rfft implementations do not support `bf16` arithmetic, limiting their memory optimization potential during the forward pass. While full fine-tuning does not incur extra memory for *trainable_params* (as the base model is updated directly), it still requires large *gradient* storage due to the number of trained parameters.

Our method, in contrast, outperforms all FFT and inverse FFT operations in-place with native support for real-valued `bf16` input. Besides, our method also outperforms LoRA adapters which has been widely adopted due to its small parameter amount and memory consumption. As a result, it consistently achieves the lowest peak memory usage across training steps, showing superior memory efficiency in practical fine-tuning setups.

5.2 Runtime Efficiency and Numerical Accuracy

To comprehensively evaluate the proposed **rdfft** operator, we analyze both its low-level computational efficiency and numerical accuracy (operator-level), as well as its end-to-end performance when integrated into large-scale fine-tuning tasks (model-level). This dual perspective allows us to capture both micro-level operator behavior and macro-level training characteristics.

5.2.1 Operator-Level Evaluation

At the operator level, Tab.3 presents both runtime and numerical accuracy. At small to medium sizes (512 and 1024), ours achieves competitive runtime with rfft, suggesting that in-place execution can be efficient when synchronization cost is limited. At larger sizes (4096), however, the runtime overhead increases due to CUDA thread-block limitations, where synchronization is required both within and across blocks. Moreover, the inverse transform (ours) is faster than the forward one, since it reuses the butterfly structure in reverse order, thereby reducing dependencies.

In terms of accuracy, since rdfft only reformulates the memory layout to support true in-place execution, it introduces no information loss and preserves the mathematical equivalence of the Fourier transform. Tab.3 shows that both absolute and relative errors remain at the level of floating-point numerical noise, confirming that rdfft faithfully reproduces the FFT spectrum.

5.2.2 Model-Level Evaluation

At the model level, Tab. 4 reports both training throughput and downstream task accuracy. While our method exhibit lower throughput compared to fft and rfft, they eliminate all intermediate buffer allo-

Table 3: Standalone operator runtime and numerical accuracy of FFT variants. Runtime (RT, in ms) is measured on an A800 GPU with FP32 precision, averaged over 1000 runs. Each operator is evaluated for both forward and inverse transforms, as shown in the table. Operator-level numerical accuracy of rfft and ours is evaluated against the `torch.fft.fft` baseline, with errors reported as absolute and relative values. Entries marked as "N/A" indicate that the baseline fft is used for reference, and thus its self-comparison is unnecessary.

Method		p=512			p=1024			p=4096		
		fft	rfft	ours	fft	rfft	ours	fft	rfft	ours
RT	forward	0.0246	0.0195	**0.0279**	0.0249	0.0197	**0.0319**	0.0252	0.0199	**0.0687**
	inverse	0.0325	0.0450	**0.0233**	0.0322	0.0421	**0.0272**	0.0327	0.0470	**0.0503**
Acc.	absolute	N/A	1.88e-07	**5.99e-07**	N/A	1.92e-07	**5.75e-07**	N/A	2.55e-07	**5.84e-07**
	relative	N/A	0.0001	**0.0008**	N/A	0.0001	**0.0005**	N/A	0.0012	**0.0018**

Table 4: Runtime throughput and MRPC accuracy of different fine-tuning methods. Token-level throughput (Thr., in k tokens/sec) is measured on LLaMA-2-7B using the GSM8K dataset with one A800 GPU, while MRPC classification accuracy (Acc., %) is evaluated on RoBERTa-large. Accuracy results are reported from the Block-Circulant Adapter (BCA) work [10], where only limited configurations were provided, leading to some "N/A" entries. All lora experiments use rank $r = 32$. For circulant-based methods, p denotes the block size.

Method	FF	lora	p=512			p=1024			p=4096		
			fft	rfft	ours	fft	rfft	ours	fft	rfft	ours
Thr. (k)	3.29	3.36	1.45	1.77	**0.92**	1.45	1.77	**0.93**	1.45	1.77	**0.93**
Acc. (%)	90.9	90.2	N/A	90.7	**90.0**	N/A	89.7	**90.3**	N/A	N/A	N/A

cations during both forward and backward passes, resulting in substantial GPU memory savings—a key advantage for large-scale fine-tuning under limited hardware.

In downstream evaluation on the MRPC benchmark, all experiments were repeated with multiple random seeds for consistency. Our method achieves task-level accuracy on par with `rfft` and full fine-tuning, indicating no degradation in learning quality. Together with the operator-level evidence, these results verify that **rdfft** is a reliable drop-in replacement for real-input FFT computations, offering strong memory efficiency while maintaining runtime competitiveness and numerical fidelity.

6 Conclusion

In this work, we present rdFFT, a novel real-domain, fully in-place Fourier transform framework designed for memory-efficient neural computation. Our method enables seamless integration into modern deep learning pipelines and supports consistent forward and backward passes entirely in the real domain. Extensive experiments on NLU benchmarks demonstrate that rdFFT significantly reduces memory consumption. Our results highlight the potential of operator-level memory optimization as a complementary and lossless strategy to existing model compression methods. In future work, we plan to extend rdFFT to support broader classes of structured transformations and explore its integration with hardware-aware training frameworks for edge deployment.

Limitations. While our method enables real-valued inputs to undergo Fourier transformation and remain in real-valued storage with a corresponding inverse transform, it inherently encodes frequency-domain information in an implicit form. This design is well-suited for use cases that do not require direct manipulation of the complex spectrum. However, for scenarios where explicit access to the complex-valued frequency representation is needed—such as spectral filtering or custom frequency-domain operations—additional logic is required to decode the real-valued encoding into a usable complex form. Once this conversion is performed, it typically involves casting the data to a complex type, which breaks the in-place memory symmetry and incurs additional memory overhead. As such, our framework is most effective in applications where complex-domain access is not strictly required.

Acknowledgments

This work was financially supported by the National Key R&D Program of China (Grant No. 2024YFA1211400) and the Key Project of Shenzhen Basic Research Program (Grant No. JCYJ20241206180301003).

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[3] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[4] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, 1988, pp. 21–28.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[6] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision arithmetic for deep learning." in *ICLR (Workshop)*, 2015.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[8] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2857–2865.

[9] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan *et al.*, "Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 395–408.

[10] X. Ding, M. Wang, S. Liao, and Z. Wang, "Block circulant adapter for large language models," *arXiv preprint arXiv:2505.00582*, 2025.

[11] Z. Gao, Q. Wang, A. Chen, Z. Liu, B. Wu, L. Chen, and J. Li, "Parameter-efficient fine-tuning with discrete fourier transform," *arXiv preprint arXiv:2405.03003*, 2024.

[12] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[13] NVIDIA Corporation, "cufft documentation: Data layout," https://docs.nvidia.com/cuda/cufft/index.html#data-layout, 2025, accessed: 2025-05-01.

[14] S. R. Bulo, L. Porzi, and P. Kontschieder, "In-place activated batchnorm for memory-optimized training of dnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5639–5647.

[15] L. Liu, B. Fu, M. D. Wong, and E. F. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1309–1314.

[16] Y. Sun, T. Chen, Q. V. H. Nguyen, and H. Yin, "Tinyad: Memory-efficient anomaly detection for time-series data in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 824–834, 2023.

[17] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[18] H. Pratt, B. Williams, F. Coenen, and Y. Zheng, "Fcnn: Fourier convolutional neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 786–798.

[19] C. Zhao, Y. Chen, Y. Feng, and S. Yang, "Efficient synthesis of large-scale time-modulated antenna arrays using artificial neural networks and inverse fft," *IEEE Transactions on Antennas and Propagation*, vol. 72, no. 2, pp. 1568–1580, 2023.

[20] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[21] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "Fnet: Mixing tokens with fourier transforms," in *Proceedings of the 2022 Conference of the north American chapter of the Association for Computational Linguistics: human language technologies*, 2022, pp. 4296–4313.

[22] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[23] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & systems*. Pearson Educación, 1997.

[24] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state that our method enables a fully in-place Fourier transform in the real domain, effectively reducing memory usage compared to standard FFT and RFFT implementations. These claims are well-supported by the theoretical analysis and experimental results presented in the paper, accurately reflecting the scope and contributions.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper explicitly discusses limitations in Sec.6, particularly the constraint that the method encodes frequency information implicitly, which may require additional decoding for tasks needing complex-domain access. It also notes that the method is naturally suited for input sizes that are powers of two, though this can be mitigated via zero-padding.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper includes theoretical derivations related to the in-place real-valued Fourier transform. All assumptions are clearly stated, and the derivations are presented in detail within the main text to ensure completeness and correctness.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All experiments in the paper are described with detailed settings, including model architecture, training hyperparameters, datasets, and evaluation metrics, ensuring that the results can be independently reproduced.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will release anonymized code and scripts in the supplementary material to reproduce the main experimental results. The provided package includes our in-place FFT implementation, baseline methods, model loading procedures (e.g., for LLaMA-7B and RoBERTa-Large), memory profiling scripts, and all commands needed to reproduce the bar charts and analyses reported in the paper. Instructions on environment setup and dependencies are also included for reproducibility.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all relevant training and testing details, including dataset splits, optimizer choices, learning rates, batch sizes, and other hyperparameters. These details are presented in the main text.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [NA]

Justification: The experiments measure GPU memory usage, which is deterministic under fixed settings and does not involve sources of randomness such as weight initialization or data sampling. Therefore, statistical significance measures such as error bars are not applicable.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments were conducted on a single NVIDIA A100 GPU with 80 GB memory. Each experimental run (e.g., memory measurement for one model variant) typically required less than 5 minutes. The total compute used across all experiments was under 5 GPU-hours. All experiments were performed on a local server. No additional large-scale pretraining or fine-tuning was performed beyond the reported experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Justification: The research complies with the NeurIPS Code of Ethics. It does not involve human subjects, sensitive data, dual-use concerns, or other ethically sensitive areas. The work focuses solely on algorithmic and system-level improvements for efficient model training.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The paper focuses on foundational research in memory-efficient training techniques for large-scale machine learning models. It does not directly involve any deployment scenarios or applications that would raise societal concerns. While such techniques can be broadly useful for improving the accessibility and efficiency of ML, the work itself does not have immediate positive or negative societal impacts.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper does not release any models or datasets that pose a high risk of misuse. The contribution is methodological in nature, focusing on improving memory efficiency during training, and does not involve any data or model release that would require safeguards.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All third-party assets used in the paper, including models such as LLaMA2-7B and RoBERTa-Large, are properly cited in the paper. We have respected and acknowledged their original licenses (e.g., Meta's non-commercial license for LLaMA2 and Apache 2.0 for RoBERTa).

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces a new in-place FFT implementation for memory-efficient training, which is a novel software asset. We provide well-documented source code, including usage instructions, environment setup, and descriptions of each component. The documentation accompanies the submission in anonymized form.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

Justification: This work does not involve any crowdsourcing experiments or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This work does not involve any human subjects or crowdsourcing experiments, and thus does not require IRB or equivalent ethical approval.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The research does not involve the use of large language models (LLMs) as important, original, or non-standard components of the core methodology. Any LLM usage was limited to writing assistance and does not affect the scientific contribution of the work.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.