1 Appendix Overview

In this appendix, we provide additional detailed explanation and supplementary materials to support the main text. Specifically, we include the following sections:

2 Notation Table

Table 7 lists and describes the key notations used in the main text.

3 The Necessity of Designing Lightweight Models

Although GPU clusters powered by state-of-the-art hardware such as A100 and H100 have been widely adopted for high-performance computing, the design of lightweight models remains imperative in addressing critical challenges across diverse applications. This necessity is underscored by the following considerations:

3.0.1 **Resource Constraints in Edge Computing and IoT Applications** Many real-world scenarios—including smart home devices, industrial monitoring, autonomous driving, and aerospace systems—rely on edge computing and embedded hardware (e.g., FPGA boards, ESP32 devices). These platforms are often constrained by limited computational power, storage capacity, and strict requirements for low power consumption and real-time performance. Lightweight models are essential to ensure feasible deployment and reliable operation in such environments.

3.0.2 **Scalability and Cost Efficiency** While modern high-performance hardware (e.g., A100/H100 GPUs) supports large-scale models, computational costs grow non-linearly with data volume, especially for architectures like transformers with quadratic complexity. In applications involving large-channel data or distributed systems, deploying oversized models can lead to prohibitive operational expenses, including hardware procurement, maintenance, and energy consumption. A lightweight design enables scalable deployment while significantly reducing these costs.

3.0.3 **Integration in Multi-Module Systems** In complex systems such as autonomous driving or power grid management, time-series prediction often serves as one component within a broader framework. A compact model preserves prediction accuracy while freeing computational resources for other critical modules (e.g., decision-making or safety protocols), thereby enhancing the overall system's real-time responsiveness and stability.

Critically, our approach demonstrates that lightweight design does not necessitate a trade-off in accuracy. Instead, we achieve optimal performance in both model efficiency and predictive capability, addressing the dual demands of resource-constrained environments and high-performance applications.

4 Key Innovations of LightTF

The goal of this paper is to design a lightweight and efficient time series forecasting model. To achieve this, LightTF is primarily based on three key ideas:

4.0.1 **Weight Sharing**: We promote weight sharing by splitting the complete sequence into multiple subsequences and applying the same operations to each subsequence or to the frequency points in parallel.

4.0.2 **Weight Sparsification**: The spectral leakage and the natural correlation (see Section) between frequency points in time series indicate sparse correlations between these frequency points, leading us to apply group sparsification to the weight matrix in the frequency domain.

4.0.3 **Patch-Scale Prediction**: To reduce the number of parameters, while at the same time capturing the temporal variations of frequency components, we predict the frequency points over an interval at the patch scale.

5 Limitations

Overall, LightTF is a model that is robust to hyperparameters. However, the parameters that achieve the best performance still require careful selection. Comparatively, the parameter that has the greatest impact is the patch size. Based on experimental results, the optimal patch size varies greatly across different datasets (ranging from 4 to 48), and the selection of individual patch sizes may lead to extreme results (see 9.3). This outcome is related to the different inherent periods of each dataset. LightTF currently does not have the ability to automatically optimize, and this is also the focus of our future work.

Another important limitation is that our model does not possess strong zero-shot capabilities. Unlike models based on large language models or pre-trained Transformers, LightTF requires training on specific datasets to achieve good prediction performance. Although we have tested on multiple datasets and achieved good results, it must be acknowledged that LightTF trades its dependence on specific datasets for its minimal parameter count and efficient computational capabilities. In summary, the model must be trained on specific datasets to achieve good prediction performance.

6 LightTF and FITS: A Detailed Comparison

In terms of methodology, FITS is essentially a frequency-domain interpolation model. It interpolates the frequency domain from the original L-point sequence of the look-back window to an $(L + H)$-point sequence in the frequency domain, and then obtains the prediction through an inverse transform. In other words, FITS reconstructs both the look-back window and the prediction sequence, but only uses the latter in the loss function. Specifically, the similarities and differences between FITS and LightTF are as follows:

6.0.1 FITS is a pure frequency-domain model, treating the input sequence and the input+output sequence as a whole for frequency-domain interpolation. While this method leverages the global characteristics of the frequency domain, it overlooks the fact that the FFT operation tends to blur time information associated with

Table 7: Notation Table. The table lists and describes the key notations used in the main text.

Notation	Description
$\boldsymbol{X}$	Time sequence with multiple channels.
$\boldsymbol{x}$, $\boldsymbol{x}_{a:b}^{(i)}$	Input time sequence with a single channel.
$\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{x}}_{a:b}^{(i)}$	Predicted time sequence.
$\boldsymbol{X}_{\text{patch}}$	The matrix that contains the input sequence of each patch.
$\boldsymbol{X}_{\text{samp}}$	The matrix that contains the downsampled input sequence of each patch.
$\boldsymbol{X}_{\text{RFFT}}$	The tensor that contains the RFFT result of each sub-sequence in each patch.
$\boldsymbol{X}_{\text{SFM}}$	The result after sparse frequency mixing (SFM) operation.
$\hat{\boldsymbol{X}}$	The tensor that contains the predicted RFFT result of each sub-sequence in each patch.
$\hat{\boldsymbol{X}}_{\text{IRFFT}}$	The tensor that contains the IRFFT result of each sub-sequence in each patch.
L	The length of the input sequence.
H	The prediction horizon.
P	The number of patches.
M	The downsampling factor.
f_c	The cutoff frequency.
K	The number of sparse groups.
$\text{Concat}(\cdot)$	The operation that concatenates each element.
$\text{RFFT}(\cdot)$	The operation that performs the real-valued Fast Fourier Transform.
$\text{IRFFT}(\cdot)$	The operation that performs the real-valued Inverse Fast Fourier Transform.
$\text{SFM}(\cdot)$	The Sparse Frequency Mixing module, or the SFM operator.
$\mathcal{L}_k$	The linear transformation that applies to the k-th group of frequency points.
$\text{PatchPredictor}(\cdot)$	Module for cross-patch frequency point prediction.
$\text{ZeroPad}(\cdot)$	The operation that pads the tensor with zeros.
$\text{Reshape}(\cdot)$	The operation that reshapes the tensor.

specific frequency points. To better handle potential changes in frequency values over different time periods, we propose LightTF, a model that fuses both time and frequency domains. Unlike FITS, LightTF does not reconstruct the input+output sequence as a whole. Instead, it predicts the frequency points across patches, using the frequency values from different patches in the look-back window to predict the frequency values in the output sequence across different patches.

6.0.2 In FITS, the interpolation layer derives the frequency points of the input+output sequence from the input frequency points, whereas in LightTF, the SFM (Sparse Frequency Mixer) mixes the frequency points within each patch without involving interpolation or prediction between different frequency points.

6.0.3 FITS does not involve any downsampling operations.

6.0.4 Both FITS and LightTF use f_c for frequency-domain filtering, which is a common operation in signal processing. However, in LightTF, since the downsampling operation decreases the upper limit of frequency representation, the purpose of filtering is less prominent. The main reason for filtering in LightTF is to ensure that the number of frequency points can be divided evenly by the sparse grouping number K.

In summary, although both FITS and LightTF utilize the RFFT operation, their fundamental principles and motivations are quite different. Additionally, in LightTF, we employ techniques such as cross-patch weight sharing, weight sparsification, and cross-frequency-point weight sharing to reduce the number of parameters, which the authors believe is another key innovation of LightTF.

7 LightTF and SparseTSF: A Detailed Comparison

SparseTSF is a purely temporal-domain model that employs weight-shared linear layers for cross-cycle temporal point prediction. Its performance heavily depends on the periodicity parameter ω and is effective only for sequences with fixed periods. For non-stationary sequences, such as those in the Weather dataset, SparseTSF exhibits poor performance. In contrast, LightTF captures the dominant frequency information within distinct patches and models the frequency point variation trends across patches, enabling robust effectiveness even for non-stationary sequences.

8 Supplementary theoretical explanations and proofs

8.1 The DFT for Real-Valued Sequences

Property 3. *The Discrete Fourier Transform (DFT) of a real-valued sequence $x(n)$ is Hermitian symmetric, i.e., $X(k) = X^*(N-k)$.*

Proof. We want to prove that the Discrete Fourier Transform (DFT) of a real-valued sequence $x(n)$ is Hermitian symmetric, i.e., $X(k) = X^*(N-k)$.

The DFT of the sequence $x(n)$ is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}$$

where $k = 0, 1, 2, \ldots, N-1$.

The complex conjugate of $X(k)$, denoted as $X^*(k)$, is:

$$X^*(k) = \left(\sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \right)^*$$

Since the sum of the conjugates is equal to the conjugate of the sum, and $e^{-j\theta}$ is conjugated to $e^{j\theta}$, we have:

$$X^*(k) = \sum_{n=0}^{N-1} x^*(n) e^{j\frac{2\pi}{N}kn}$$

Given that $x(n)$ is a real-valued sequence, $x^*(n) = x(n)$. Thus:

$$X^*(k) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}kn}$$

Now, let's express $X(N-k)$ using the definition of the DFT:

$$X(N-k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}(N-k)n}$$

Simplifying the exponent:

$$X(N-k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}Nn} e^{j\frac{2\pi}{N}kn}$$

Since $e^{-j\frac{2\pi}{N}Nn} = e^{-j2\pi n}$ and $e^{-j2\pi n} = 1$ for any integer n, we have:

$$X(N-k) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}kn}$$

Notice that:

$$X^*(k) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}kn}$$

and:

$$X(N-k) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}kn}$$

Thus, we conclude:

$$X(N-k) = X^*(k)$$

We have shown that:

$$X(k) = X^*(N-k)$$

This result confirms that the DFT of a real-valued sequence $x(n)$ is Hermitian symmetric, completing the proof. □

8.2 Auto-Correlation of Time Series

Taking the ETTh1 dataset as an example, Figure 8 shows the normalized autocorrelation results for this dataset. These results are computed using the following formula:

$$\text{ACF}(k) = \frac{1}{L} \sum_{t=0}^{L-k-1} \frac{(x_t - \bar{x})(x_{t+k} - \bar{x})}{\sigma^2}, \tag{9}$$

where L is the length of the time series, x_t is the value at time t, $\bar{x}$ is the mean of the time series, and σ^2 is the variance of the time series. As can be observed, there exists significant correlation between time points in the time series data. This correlation introduces temporal information redundancy, which our downsampling strategy leverages to achieve effective data compression.

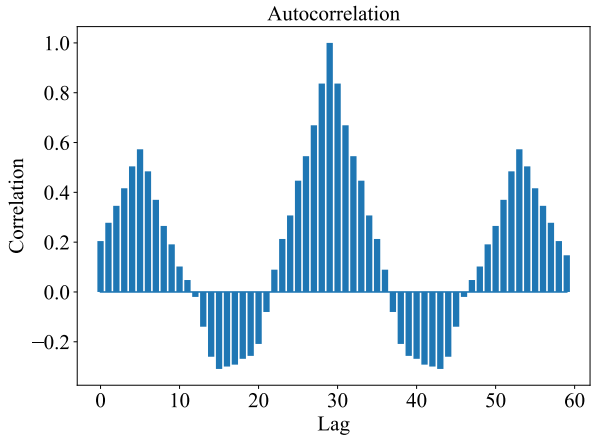


Figure 8: Auto-correlation of ETTh1 dataset. The x-axis represents the lag, and the y-axis represents the auto-correlation value.

8.3 Nyquist Sampling Theorem

Theorem 4 (Nyquist Sampling Theorem (Nyquist 2002))**.** *Let $x(t)$ be a continuous-time signal that is bandlimited, meaning its Fourier transform $X(f)$ is zero for $|f| > f_{max}$, where f_{max} is the maximum frequency component in the signal. If this signal is sampled at a uniform rate f_s, then the original signal $x(t)$ can be perfectly reconstructed from its samples if and only if the sampling frequency f_s is greater than twice the maximum frequency f_{max}:*

$$f_s > 2f_{max}$$

The minimum sampling rate, $2f_{max}$, is called the Nyquist rate. If the sampling rate is less than the Nyquist rate ($f_s < 2f_{max}$), aliasing will occur, where higher frequencies in the original signal will appear as lower frequencies in the sampled signal, leading to information loss and distortion.

The vast majority of real-world time series exhibit low-pass characteristics. This implies redundancy between consecutive time steps within the series, which in turn provides a theoretical basis for patch-wise downsampling.

8.4 Asymptotic Decorrelation of Spectral Components

Theorem 5 (Asymptotic Decorrelation of Spectral Components)**.** *Let $\{X_t\}$ be a discrete-time, zero-mean wide-sense stationary process with duration T. For any two distinct normalized frequencies $\omega_k = \frac{2\pi k}{T}$ and $\omega_{k'} = \frac{2\pi k'}{T}$ where $k \neq k'$, the corresponding discrete Fourier transform (DFT) coefficients D_k and $D_{k'}^*$ satisfy:*

$$\lim_{T\to\infty} \mathbb{E}\left[D_k D_{k'}^*\right] = 0.$$

Proof. We assume the DFT coefficients D_k are normalized as follows, which is standard for such asymptotic theorems:

$$D_k = \frac{1}{\sqrt{T}} \sum_{t=0}^{T-1} X_t e^{-j\omega_k t}$$

where $\omega_k = \frac{2\pi k}{T}$. The complex conjugate $D_{k'}^*$ is then:

$$D_{k'}^* = \frac{1}{\sqrt{T}} \sum_{s=0}^{T-1} X_s^* e^{j\omega_{k'} s}$$

The expected value of the product $D_k D_{k'}^*$ is:

$$\mathbb{E}[D_k D_{k'}^*] = \mathbb{E}\Big[\big(\tfrac{1}{\sqrt{T}} \sum_{t=0}^{T-1} X_t e^{-j\omega_k t}\big)\big(\tfrac{1}{\sqrt{T}} \sum_{s=0}^{T-1} X_s^* e^{j\omega_{k'} s}\big)\Big] \tag{10}$$

$$= \tfrac{1}{T}\mathbb{E}\Big[\sum_{t=0}^{T-1}\sum_{s=0}^{T-1} X_t X_s^* e^{-j\omega_k t + j\omega_{k'} s}\Big] \tag{11}$$

$$= \tfrac{1}{T} \sum_{t=0}^{T-1}\sum_{s=0}^{T-1} R_X[t-s] e^{-j\omega_k t + j\omega_{k'} s}. \tag{1}$$

We interchanged the order of summation and expectation. Since $\{X_t\}$ is a zero-mean wide-sense stationary process, its autocorrelation function is $R_X[t-s] = \mathbb{E}[X_t X_s^*]$. Substituting this into (1):

$$\mathbb{E}[D_k D_{k'}^*] = \tfrac{1}{T} \sum_{t=0}^{T-1}\sum_{s=0}^{T-1} R_X[t-s] e^{-j\omega_k t + j\omega_{k'} s}$$

Let $\tau = t - s$. Then $s = t - \tau$. The summation limits for τ range from $-(T-1)$ to $T-1$. For a fixed τ, t ranges from $\max(0,\tau)$ to $\min(T-1, T-1+\tau)$.

$$\mathbb{E}[D_k D_{k'}^*] = \tfrac{1}{T} \sum_{\tau=-(T-1)}^{T-1} \sum_{t=\max(0,\tau)}^{\min(T-1,T-1+\tau)} R_X[\tau] e^{-j\omega_k t + j\omega_{k'}(t-\tau)} \tag{12}$$

$$= \tfrac{1}{T} \sum_{\tau=-(T-1)}^{T-1} R_X[\tau] e^{-j\omega_{k'}(-\tau)} \sum_{t=\max(0,\tau)}^{\min(T-1,T-1+\tau)} e^{-j(\omega_k - \omega_{k'})t} \tag{13}$$

$$= \tfrac{1}{T} \sum_{\tau=-(T-1)}^{T-1} R_X[\tau] e^{j\omega_{k'}\tau} S_{in}(\tau, \Delta\omega) \tag{2}$$

where $\Delta\omega = \omega_k - \omega_{k'} = \frac{2\pi(k-k')}{T}$, and $S_{in}(\tau, \Delta\omega)$ is the inner sum over t:

$$S_{in}(\tau, \Delta\omega) = \sum_{t=\max(0,\tau)}^{\min(T-1,T-1+\tau)} e^{-j\Delta\omega t}$$

Since $k \neq k'$, $m = k - k'$ is a non-zero integer. We assume $0 \leq k, k' \leq T-1$, so m is not a multiple of T. Consider the full sum of the complex exponentials over $t \in [0, T-1]$:

$$\sum_{t=0}^{T-1} e^{-j\Delta\omega t} = \sum_{t=0}^{T-1} e^{-j\frac{2\pi m}{T} t} = 0$$

because m is a non-zero integer and not a multiple of T. The sum $S_{in}(\tau, \Delta\omega)$ is a partial sum. We can write it as the negative of the sum over the "missing" terms from the full range $[0, T-1]$:

$$S_{in}(\tau, \Delta\omega) = \left(\sum_{t=0}^{T-1} e^{-j\Delta\omega t}\right) \tag{14}$$

$$- \left(\sum_{t=0}^{\max(0,\tau)-1} e^{-j\Delta\omega t} + \sum_{t=\min(T-1,T-1+\tau)+1}^{T-1} e^{-j\Delta\omega t}\right) \tag{15}$$

Since the first term is zero:

$$S_{in}(\tau, \Delta\omega) \tag{16}$$

$$= -\left(\sum_{t=0}^{\max(0,\tau)-1} e^{-j\Delta\omega t} + \sum_{t=\min(T-1,T-1+\tau)+1}^{T-1} e^{-j\Delta\omega t}\right) \tag{17}$$

The number of terms in the first of these "missing" sums is $\max(0,\tau)$. The number of terms in the second "missing" sum is $(T-1) - (\min(T-1, T-1+\tau))$. The total number of terms in these two "missing" sums is $\max(0,\tau) + (T-1) - \min(T-1, T-1+\tau)$. If $\tau \geq 0$, $t_{start} = \tau, t_{end} = T-1$. Missing: $t = 0, \ldots, \tau-1$. Count is τ. If $\tau < 0$, $t_{start} = 0, t_{end} = T-1+\tau$. Missing: $t = T-1+\tau+1, \ldots, T-1$. Count is $T-1-(T-1+\tau)+1 = -\tau = |\tau|$. So, the total number of terms in the exponentials that constitute $S_{in}(\tau, \Delta\omega)$ (when written as the negative of the sum of missing terms) is exactly $|\tau|$. Using the trivial bound $|e^{-j\theta}| = 1$, the magnitude $|S_{in}(\tau, \Delta\omega)|$ is bounded by the number of terms in these "missing" sums:

$$|S_{in}(\tau, \Delta\omega)| \leq |\tau|$$

Substituting this bound into the expression for $\mathbb{E}[D_k D_{k'}^*]$:

$$|\mathbb{E}[D_k D_{k'}^*]| \leq \frac{1}{T} \sum_{\tau=-(T-1)}^{T-1} |R_X[\tau] e^{j\omega_{k'}\tau}||S_{in}(\tau, \Delta\omega)|$$

$$|\mathbb{E}[D_k D_{k'}^*]| \leq \tfrac{1}{T} \sum_{\tau=-(T-1)}^{T-1} |R_X[\tau]||\tau| \tag{18}$$

For many well-behaved stationary processes, the condition $\sum_{\tau=-\infty}^{\infty} |\tau||R_X[\tau]| < \infty$ holds. Let this sum be C_R'. Under

this condition, the sum $\sum_{\tau=-(T-1)}^{T-1} |R_X[\tau]||\tau|$ is bounded by C'_R for all T. Then, from (3):

$$|\mathbb{E}[D_k D_{k'}^*]| \leq \frac{C'_R}{T}$$

As $T \to \infty$, $\frac{C'_R}{T} \to 0$. Therefore, for $k \neq k'$:

$$\lim_{T\to\infty} \mathbb{E}[D_k D_{k'}^*] = 0.$$

The proof is therefore complete under the assumption that $\sum_{\tau=-\infty}^{\infty} |\tau||R_X[\tau]| < \infty$. This condition ensures sufficient decay of the autocorrelation function for the asymptotic decorrelation. □

8.5 Spectral Leakage

Spectral leakage refers to the phenomenon in the Discrete Fourier Transform (DFT) where, when the frequencies of a time series do not align with the frequency components of the DFT, the signal's energy leaks into adjacent frequency components. This phenomenon causes peaks in the spectrum to become blurred and prevents accurate representation of the signal's actual frequency components. Figure 9 demonstrates that when a single-frequency continuous signal is sampled to become a time series, the frequency components do not align with the DFT frequency components, causing energy to leak into adjacent frequency components.

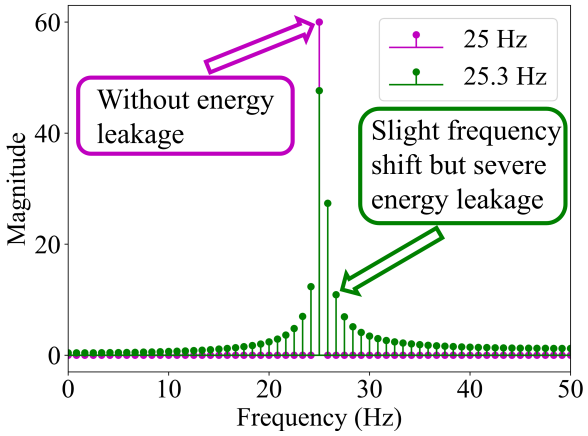


Figure 9: Spectral leakage example. The figure shows the spectrum of two signals: the spectrum of a time series sampled from a 25Hz sine wave, and the spectrum of a time series sampled from a 25.3Hz sine wave. After performing DFT, the spectrum of the 25Hz sine wave has a sharp peak at 25Hz, while the spectrum of the 25.3Hz sine wave has a broader peak at 25Hz, with energy leaking to adjacent frequency points.

Furthermore, the following theorem describes the mathematical principles underlying spectral leakage.

Proposition 6 (Spectral Leakage due to Misalignment with DFT Bins)**.** *Let $x[n]$ be a discrete-time signal of length N, sampled at a rate f_s. The Discrete Fourier Transform (DFT) computes frequency components at specific frequency bins given by:*

$$f_k = \frac{k f_s}{N}, \quad k = 0, 1, \ldots, N-1.$$

If the signal $x[n]$ contains a sinusoidal component of frequency f_0, such that f_0 does not exactly match any of the DFT frequency bins f_k, i.e.,

$$f_0 \neq f_k \quad \textit{for any } k,$$

then the energy of the sinusoidal component at f_0 will leak into adjacent frequency bins. This phenomenon is known as ***spectral leakage****.*

Proof. The phenomenon of spectral leakage occurs when a sinusoidal component within a signal does not align perfectly with the frequency bins of the Discrete Fourier Transform (DFT). The DFT calculates the frequency content of a discrete-time signal $x[n]$ over a finite interval, producing frequency components at specific frequencies $f_k = \frac{k f_s}{N}$, where k ranges from 0 to $N-1$ and f_s is the sampling frequency.

When the frequency of a sinusoidal component f_0 in the signal does not coincide with any of these discrete DFT frequency bins f_k, the DFT cannot represent f_0 as a single frequency component. Instead, the energy that should ideally be concentrated at f_0 is distributed among several neighboring frequency bins, a phenomenon known as spectral leakage.

To understand why this occurs, consider the DFT of a sinusoidal signal $x[n] = A\cos(2\pi f_0 n + \phi)$. If f_0 aligns perfectly with one of the DFT bins f_k, the DFT will represent this component as a peak at f_k, with no energy in the other bins. However, if f_0 does not align with any f_k, we can express f_0 as $f_0 = f_k + \Delta f$, where Δf is the mismatch between f_0 and the nearest DFT bin frequency f_k. The sinusoid can no longer be represented by a single complex exponential at f_k, and instead, its energy spreads across multiple bins.

Mathematically, this is due to the finite length of the signal. The DFT implicitly assumes that the signal is periodic with a period equal to the length of the signal N. When f_0 does not match any f_k, the assumption of periodicity introduces discontinuities at the boundaries of the signal, creating artifacts in the frequency domain. These artifacts manifest as energy in frequency bins that are not directly associated with f_0, leading to the appearance of spectral leakage.

To illustrate, consider the DFT of the signal $x[n] = A\cos(2\pi \frac{k_0+\delta}{N} n)$, where k_0 is an integer, and δ is a small fractional frequency component such that $f_0 = \frac{(k_0+\delta) f_s}{N}$. The DFT will show a peak not only at k_0 but also in adjacent bins $k_0 \pm 1, k_0 \pm 2, \ldots$, depending on the value of δ. The energy spreads over several bins, reducing the sharpness of the spectral peak, which would otherwise be concentrated if $\delta = 0$.

In practice, spectral leakage can be mitigated by techniques such as windowing, where the signal is multiplied by a window function that tapers the signal to zero at the boundaries, reducing the discontinuities and therefore the leakage. However, even with windowing, some degree of leakage is

typically unavoidable when f_0 does not exactly match a DFT bin.

□

8.6 Proof of Theorem 5.1

Theorem 7 (parameter count)**.** *Given the look-back window length L, the number of patches P, the output horizon H, and the cutoff frequency f_c, the number of frequency group K in SFM, the total number of parameters in LightTF can be calculated as:*

$$\textit{Parameters} = \frac{{f_c}^2}{K} + \frac{HP^2}{L} \tag{19}$$

Specifically, without cutting off the frequency points, given the the down sampling factor M, the number of parameters in LightTF can be calculated as:

$$\textit{Parameters} = \left(\lfloor\frac{L}{2PM}\rfloor + 1\right)^2 / K + \frac{HP^2}{L} \tag{20}$$

Proof. The parameter count can be divided into two parts: the number of parameters in the SFM and the patch predictor. Given the input sequence length L, the number of patches P, and the output horizon H, the SFM can be viewed as a mapping from f_c frequency components to f_c frequency components (i.e., $\mathbb{C}^{f_c} \to \mathbb{C}^{f_c}$) with each sparse group conducting a linear transformation from $\frac{f_c}{K}$ to $\frac{f_c}{K}$ frequency components. The number of parameters in the SFM can therefore be calculated as:

$$Parameters_{\text{SFM}} = \frac{f_c}{K} \times \frac{f_c}{K} \times K = \frac{f_c^2}{K} \tag{21}$$

The output number of patches can be calculated as $\frac{PH}{L}$, so that the patch predictor can be viewed as a mapping from $\mathbb{C}^P$ to $\mathbb{C}^{\frac{PH}{L}}$, and the number of parameters in the patch predictor can be calculated as:

$$Parameters_{\text{PatchPredictor}} = P \times \frac{PH}{L} = \frac{P^2H}{L} \tag{22}$$

Therefore, the total number of parameters in LightTF is the sum of the parameters in the SFM and the patch predictor:

$$Parameters = \frac{f_c^2}{K} + \frac{P^2H}{L} \tag{23}$$

Specifically, after downsampling we get PM subsequences with length $L_{sub} = \frac{L}{PM}$, and then the RFFT operation is applied to each subsequence, yielding another PM subsequences with length $\lfloor\frac{L}{2PM}\rfloor + 1$. By setting $f_c = \lfloor\frac{L}{2PM}\rfloor + 1$, the number of parameters in the SFM can be calculated as:

$$Parameters_{\text{SFM}} = \frac{f_c^2}{K} = \frac{\left(\lfloor\frac{L}{2PM}\rfloor + 1\right)^2}{K}, \tag{24}$$

and therefore the total number of parameters in LightTF is:

$$Parameters = \frac{\left(\lfloor\frac{L}{2PM}\rfloor + 1\right)^2}{K} + \frac{P^2H}{L}. \tag{25}$$

□

9 Additional Experimental Results

9.1 Training Settings

In this section, we provide additional details on the training settings used in the experiments. The training process was conducted on a single NVIDIA RTX 4090 GPU with 24GB memory. The optimizer used was Adam with a learning rate of 0.008, a batch size of 256, and a maximum epoch count of 100. The model was implemented using PyTorch and trained using the Mean Squared Error (MSE) loss function. The hyperparameter table for each dataset is shown in Table 8. Specifically, the MSE loss was calculated as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - \hat{\boldsymbol{x}}_i)^2, \tag{26}$$

where N is the number of samples, $\boldsymbol{y}_i$ is the true value (ground truth), and $\hat{\boldsymbol{x}}_i$ is the predicted value.

During training, we adopt an *early stop* strategy based on the validation loss. Specifically, the training process will be terminated if the validation loss does not decrease for 6 consecutive epochs. The model parameters at the epoch with the lowest validation loss are saved as the final model. Besides, we also adopt the *learning rate decay* strategy. Specifically, the learning rate will be reduced by a factor of 0.6 for every 10 epochs.

In the result table in the main text (Table 1), results of FEDformer, TimesNet, PatchTST, DLinear, FITS, and SparseTSF are reported from (Lin et al. 2024b), others are from the original papers.

9.2 Ablation Study

The ablation study result is shown in Table 9. The ablation study is conducted on different datasets and prediction horizons to evaluate the impact of different components of LightTF. Specifically, we conduct the ablation study on the following components:

- **LightTF-SFM:** This model removes the Sparse Frequency Mixer (SFM) component. Removing SFM means that the model predicts each frequency point independently.

- **LightTF-Patch:** This model removes the patch division step. Removing the patch division means the model simply uses the RFFT of the original input sequence as the input of the model and predicts the RFFT of the output sequence directly.

- **LightTF-RevIn:** This model removes the RevIn strategy (Kim et al. 2021), which is a common strategy that normalizes the input sequence and denormalize the output sequence.

The results clearly demonstrate that both the SFM and patch division components in LightTF significantly improve performance. However, while the model uses the RevIn strategy, this approach shows no statistically significant impact on the model's performance.

Table 8: Hyperparameters for different datasets and prediction horizons. Here, *cnt* represents the total number of parameters for the configuration. Here *PS* denotes the patch size.

Dataset	96					192					336					720				
	PS	M	f_c	K	cnt	PS	M	f_c	K	cnt	PS	M	f_c	K	cnt	PS	M	f_c	K	cnt
ETTh1	48	2	12	2	**102**	48	2	12	2	**132**	48	2	12	2	**177**	48	2	12	2	**297**
ETTh2	6	1	4	2	1928	6	1	4	2	3848	6	1	4	2	6728	6	1	4	2	14408
Electricity	4	1	3	1	4329	4	1	3	1	8649	4	1	3	1	15129	4	1	3	1	32409
Traffic	24	2	6	2	138	24	2	6	2	258	24	2	6	2	438	24	2	6	2	918
Weather	12	2	4	2	648	12	2	4	2	1128	12	2	4	2	1848	12	2	4	2	3768

Table 9: Comprehensive ablation study results for different datasets and prediction horizons. *LightTF* denotes the full model, *LightTF-SFM* represents the model without the Sparse Frequency Mixer component, *LightTF-Patch* represents the model without patch division, and *LightTF-RevIn* represents the model without the RevIn strategy.

Dataset	Ablation	Horizon			
		96	192	336	720
ETTh1	LightTF	**0.350**	**0.388**	**0.419**	**0.416**
	LightTF-SFM	0.396	0.425	0.448	0.439
	LightTF-Patch	0.376	0.415	0.441	0.430
	LightTF-RevIn	0.351	0.387	0.419	0.417
ETTh2	LightTF	**0.268**	**0.330**	**0.351**	**0.376**
	LightTF-SFM	0.292	0.346	0.364	0.387
	LightTF-Patch	0.275	0.355	0.362	0.393
	LightTF-RevIn	0.270	0.331	0.352	0.378
Electricity	LightTF	**0.132**	**0.149**	**0.165**	**0.200**
	LightTF-SFM	0.202	0.198	0.242	0.270
	LightTF-Patch	0.139	0.152	0.171	0.207
	LightTF-RevIn	0.133	0.149	0.166	0.202
Traffic	LightTF	**0.387**	**0.403**	**0.410**	**0.446**
	LightTF-SFM	0.464	0.473	0.476	0.542
	LightTF-Patch	0.401	0.415	0.427	0.463
	LightTF-RevIn	0.387	0.405	0.411	0.447
Weather	LightTF	**0.140**	**0.182**	**0.232**	**0.301**
	LightTF-SFM	0.191	0.232	0.272	0.335
	LightTF-Patch	0.151	0.192	0.243	0.321
	LightTF-RevIn	0.140	0.182	0.233	0.303

9.3 Hyperparameter Searching Details

We searched for the optimal M and K for LightTF on the ETTh1, ETTh2, Traffic, and Weather datasets, and the results are shown in Tables 10, 11, 12, and 13, respectively. We also conducted a hyperparameter search for the patch size PS on the 5 datasets, and the results are shown in Table 14.

It must be noted that, although the results vary across different hyperparameters, the performance of LightTF is generally robust with respect to hyperparameters, as the model consistently achieves competitive results across different settings. This robustness is a key advantage of LightTF, making it easy to deploy in practice without extensive hyperparameter tuning. Specifically, for resource constrained environments, the model can be deployed with a relatively small number of parameters while maintaining strong forecasting performance.

9.4 How to Choose Hyperparameters?

In this section, we explore the principles that should guide the selection of hyperparameters. The primary hyperparameters for LightTF include the downsampling factor M, the frequency group number K, and the patch size PS. In practical applications, these hyperparameters are typically chosen empirically rather than through rigorous mathematical proof. Nevertheless, their selection can be guided by the following principles:

- **Patch Size** PS: Common public datasets are mainly in the fields of electricity (ETT, ELC), traffic (Traffic), meteorology (Weather), and finance (Exchange), which are the main application scenarios. Through time and spectral analysis, we found that the cycles of these datasets generally follow hourly, daily, or weekly periods. For the ETT and ELC datasets, the most significant periods are 168 (weekly), 24 (daily), and 12 (half-day). For the

Table 10: Parameter search results for ETTh1. The number in the parenthesis represents the corresponding parameter count. Here the patch size PS is set to 48.

Horizon	96				720			
M \ K	1	2	3	6	1	2	3	6
1	0.350(655)	0.350(318)	0.352(222)	0.351(126)	0.418(850)	0.416(513)	0.417(417)	0.416(321)
2	0.351(199)	**0.350**(102)	0.353(78)	0.355(54)	0.416(394)	**0.416**(297)	0.420(273)	0.419(249)
4	0.353(79)	0.355(48)	0.369(42)	–	0.421(274)	0.420(243)	0.433(237)	–
8	0.359(46)	0.362(38)	–	–	0.425(241)	0.417(233)	–	–

Table 11: Parameter search results for ETTh2. Here the patch size PS is set to 6.

Horizon	96				720			
M \ K	1	2	3	6	1	2	3	6
1	0.268(1936)	**0.268**(1928)	–	–	0.376(14416)	**0.376**(14408)	–	–
2	0.270(1924)	–	–	–	0.378(14404)	–	–	–
4	–	–	–	–	–	–	–	–
8	–	–	–	–	–	–	–	–

Table 12: Parameter search results for Traffic. Here the patch size PS is set to 24.

Horizon	96				720			
M \ K	1	2	3	6	1	2	3	6
1	0.390(289)	0.388(192)	0.390(168)	0.389(156)	0.447(1069)	0.446(972)	0.448(948)	0.456(924)
2	0.389(169)	**0.387**(138)	0.393(132)	–	0.447(949)	**0.446**(918)	0.449(912)	–
4	0.391(136)	0.393(128)	–	–	0.451(916)	0.452(908)	–	–
8	0.396(124)	–	–	–	0.455(904)	–	–	–

Table 13: Parameter search results for Weather. Here the patch size PS is set to 12.

Horizon	96				720			
M \ K	1	2	3	6	1	2	3	6
1	0.142(529)	0.141(498)	0.143(492)	–	0.302(3649)	0.301(3618)	0.306(3612)	–
2	0.142(496)	**0.140**(488)	–	–	0.303(3616)	**0.301**(3608)	–	–
4	0.145(484)	–	–	–	0.306(3604)	–	–	–
8	–	–	–	–	–	–	–	–

Table 14: Hyperparameter search results for the patch size PS on different datasets. Here M and K are both set to 1, and the filtering step is omitted.

Dataset	ETTh1		ETTh2		Electricity		Traffic		Weather	
Horizon	96	720	96	720	96	720	96	720	96	720
$PS = 48$	0.350	0.418	0.273	0.377	0.150	0.213	0.146	0.309	0.421	0.474
$PS = 24$	0.360	0.426	0.276	0.378	0.136	0.204	0.143	0.309	0.390	0.447
$PS = 12$	0.369	0.425	0.272	0.378	0.135	0.203	0.142	0.302	0.389	0.448
$PS = 8$	0.371	0.425	0.271	0.377	0.133	0.201	0.140	0.304	0.391	0.452
$PS = 4$	0.372	0.430	0.268	0.376	0.132	0.200	0.141	0.307	0.394	0.455

Traffic dataset, the most significant periods are 168, 24, and 12. For the Weather dataset, the most significant period is 144 (6 days). When choosing patch division, using multiples or factors of these periods as patch lengths can better capture the changing characteristics. In most cases, frequency-domain patch division is more robust than time-domain patch division regarding patch length, as this is determined by the data compression properties of time-frequency variations. However, in some cases, patch length significantly affects the model's performance, which may be due to the dispersion of frequency information.

- **Downsampling Factor** M: In our experiments, we also employed time-domain correlation analysis to determine a suitable downsampling factor, M. By **computing the autocorrelation of the sequence, we ensure that the M-th autocorrelation value of the original sequence exceeds a certain threshold so that the sequence's information is not overly compressed or distorted**. In practice, the downsampling factor M is **usually set to 2 or 4**. Figure 24 shows the autocorrelation results, while Figure 25 illustrates the overall shape of the sequence at different sampling rates. When the downsampling factor is small, the subsequences remain largely similar to one another and closely resemble the original sequence. However, when the downsampling factor is large, the differences between subsequences become more pronounced, and they no longer resemble the original sequence. This decay in correlation leads to a decline in model performance, as discussed in Section 5.5 and Table 5 of the main text. An interesting exception occurs when **we sample the same positions across different cycles** (i.e., setting the downsampling factor equal to the cycle length). In this case, the model's performance still improves. This has been well discussed in the original paper of SparseTSF (Lin et al. 2024b).

- **Frequency Group Number** K: In our experiments, we used frequency-domain correlation analysis to select an appropriate value for K. If the high values in the frequency correlation matrix are concentrated near the diagonal, it indicates that the **correlations between different frequency points are weak**. In such cases, a **larger** K **can be chosen** to reduce the number of model parameters. Conversely, **if the high values are more dispersed—indicating stronger correlations between different frequency points—a smaller** K **is preferable to maintain model performance and prevent significant information loss**. Generally, a K value **between 2 and 4** works well. Please note that K must evenly divide f_c, which can be further ensured by adjusting the size of f_c.

9.5 Results on More Datasets

In this section we provide additional experimental results on *ETTm1*, *ETTm2*, and *Exchange* datasets. The introduction of these datasets can be found in Section 11, and the results are summarized in Table 16.

9.6 Critical Difference Diagram

The critical difference diagram is a statistical tool used to compare multiple methods across different datasets and horizons. It is based on the Friedman test, a non-parametric statistical test that determines whether there are significant differences between the methods' performance. The critical difference diagram visualizes the average ranks of the methods and indicates which methods are significantly different from each other based on the critical difference value. The critical difference value is calculated using the Nemenyi test, which takes into account the number of datasets and methods being compared. The result is shown in Figure 10.

9.7 Training Detail

In the final part of our experimental analysis, we focus on the training details. Figure 11 presents various training specifics of LightTF on the ETTh1 dataset.

Figures 11a and 11b illustrate the training loss and validation loss curves of LightTF, along with a comparison to DLinear and FITS. The results indicate that LightTF converges more rapidly and with greater stability, with a significantly lower convergence value on the validation set compared to DLinear and FITS. In contrast, FITS and DLinear both exhibit unstable fluctuations during the early stages of training.

Figure 11c shows the predictive performance of LightTF. LightTF is able to better capture the peaks, troughs, and inherent periodicity within the sequence, demonstrating a significantly superior performance compared to DLinear and FITS.

9.8 Additional Prediction Results

Additional prediction results in the training set are shown in Figures 12, 13, 14, 15, 16, 17, 18, 19, 20, 21. The results show that LightTF can effectively capture the underlying patterns in the time series data and make accurate predictions across different datasets and prediction horizons.

10 The Code Bug

In Dec. 2023, an anonymous researcher pointed out a long-existing bug in the source code of a series of time series forecasting models. The bug can be traced back to the implementation of Informer (Zhou et al. 2021), and has already affected a series of subsequent works, including DLinear, Autoformer, Fedformer, PatchTST, Koopa, FITS, and TimeMixer, etc. The bug is related to the calculation of the settings of the test dataloader, **where the *drop_last* parameter is incorrectly set to *True* by default.** This setting causes the last incomplete batch of the test dataloader to be dropped, leading to incorrect evaluation results. Empirically, the bug significantly improved the performance of the models on the ETT datasets, and the impact on other datasets is relatively marginal (Qiu et al. 2024). The bug has been fixed in the latest version of the source code, and the corrected results are presented in this paper.

Table 15: Complete form of statistical information for the datasets used in the experiments.

Dataset	Weather	Traffic	Exchange	Electricity	ETTh1	ETTh2	ETTm1	ETTm2
Dataset Size	52696	17544	7207	26304	17420	17420	69680	69680
Variable Number	21	862	8	321	7	7	7	7
Sampling Frequency	10 mins	1 hour	1 day	1 hour	1 hour	1 hour	15 mins	15 mins

Table 16: **Comparison of different methods across various datasets and horizons.** The best 3 results are highlighted **bold**, underlined, *italic*, respectively.

Dataset	ETTm1				ETTm2				Exchange		
Horizon	96	192	336	720	96	192	336	720	96	192	336
FEDformer (2022b)	0.326	0.365	0.392	0.446	0.180	0.252	0.324	0.410	0.139	0.256	0.426
TimesNet (2023)	0.338	0.371	0.410	0.478	0.187	0.249	0.321	0.497	0.107	0.226	0.367
PatchTST (2023)	**0.290**	**0.332**	**0.366**	0.416	0.165	0.220	0.274	0.362	0.093	0.192	0.350
DLinear (2023)	*0.299*	*0.335*	0.369	*0.425*	*0.167*	*0.224*	*0.281*	0.397	0.081	**0.157**	*0.305*
Koopa (2024)	0.294	0.337	0.380	0.426	0.171	0.226	0.283	0.394	*0.083*	0.184	0.331
MICN (2023)	0.314	0.359	0.398	0.459	0.178	0.245	0.295	*0.389*	0.102	0.172	**0.272**
LightTF (ours)	0.302(4th)	0.334	*0.372*	**0.415**	**0.162**	**0.215**	**0.266**	**0.349**	**0.080**	0.167	0.304

11 Detail of the Public Datasets

11.0.1 **Weather**: This dataset contains 21 meteorological indicators such as humidity and air temperature for the year 2020 in Germany.

11.0.2 **Traffic**: Contains road occupancy rates measured by 862 different sensors across San Francisco Bay Area freeways over a span of two years. The data is sourced from the California Department of Transportation.

11.0.3 **Electricity**: Comprises hourly electricity consumption data of 321 clients, recorded between 2012 and 2014.

11.0.4 **Exchange**: The Exchange dataset is a collection of daily exchange rates of eight countries relative to the US dollar. The countries include Australia, UK, Canada, Switzerland, China, Japan, New Zealand, and Singapore. The data spans from 1990 to 2016.

11.0.5 **ETT (Electricity Transformer Temperature)**: Contains data collected from electricity transformers using seven sensors, capturing variables such as load, oil temperature, etc. The dataset is split into two sub-datasets labeled as 1 and 2, corresponding to two different electric transformers from separate counties in China. Each sub-dataset includes two different time resolutions: 15 minutes and 1 hour, denoted as *m* and *h* respectively. Thus, there are four ETT datasets: ETTh1, ETTh2, ETTm1, and ETTm2.

12 The Individual Configuration

In practical applications, time series datasets are often multi-channel. Let C denote the number of input channels, then a look-back window of data can be represented as $\boldsymbol{X} \in \mathbb{R}^{C \times L}$. For channel-independent models like FITS and DLinear, there are generally two training strategies:

12.0.1 Train a separate single-channel model for each channel, with independent weights across channels, referred to as the *Individual* configuration.

12.0.2 Use a shared set of weights across all channels.

For LightTF, we also employ the *Individual* configuration on the Weather dataset, but with a key difference: we train a SFM for each channel independently, while the patch predictor shares weights across channels. The motivation behind this strategy is to reduce the model's parameter count and mitigate overfitting. Specifically, the SFM integrates frequency information over short time intervals, where different channels exhibit varying short-term behavior in the Weather dataset. Conversely, the patch predictor captures long-term trend changes across patches, which tend to be consistent across different channels.

13 Parameter Table for LightTF

The detailed parameter table of patch predictor and SFM in LightTF is shown in Table 17 and Table 18, respectively. The parameter count of the patch predictor is calculated based on the patch size, prediction horizon, and look-back window, while the parameter count of the SFM is determined by the patch size, downsampling factor, and sparse group number.

14 Detailed Complexity Analysis

In main text, we focus on calculating the parameter count of LightTF, which is a key indicator of model complexity. However, the computational complexity of LightTF is also an important aspect to consider, as it directly affects the model's training and inference efficiency. Here, we provide a detailed analysis of the computational complexity of LightTF, focusing on the forward pass of the model.

Lemma 8. *One complex multiplication requires at least 3 real multiplications.*

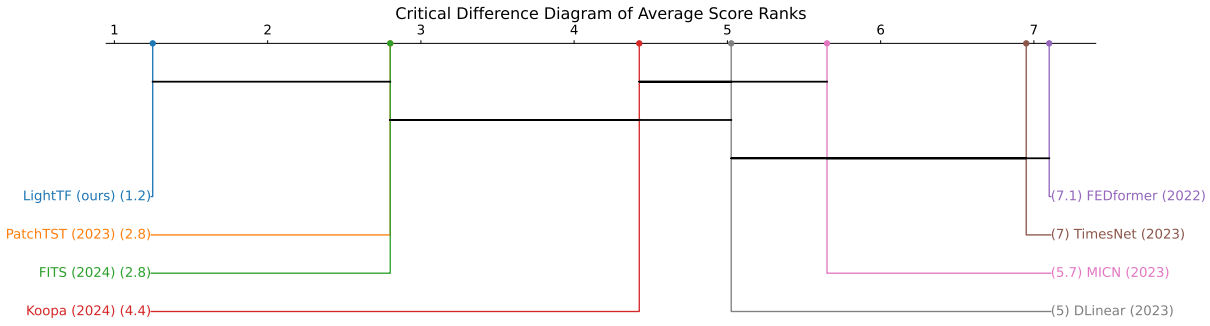
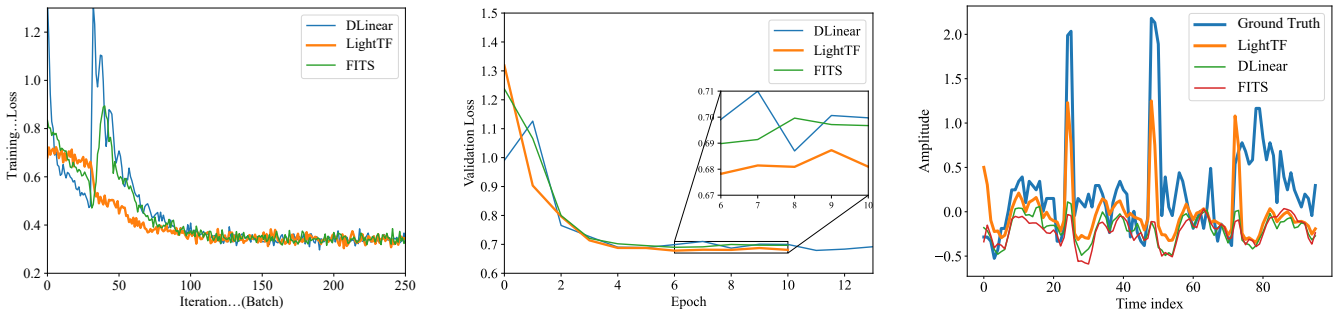


Figure 10: Critical difference diagram for the comparison of different methods across various datasets and horizons. The methods are ranked based on the average rank across all datasets and horizons.



(a) (b) (c)

Figure 11: Training detail and visualization of LightTF on the ETTh1 dataset. (a) Training loss curve, where learning rate is set to 0.008. (b) Validation loss curve. (c) Prediction results.

Proof. for $z_1 = a + bj$, $z_2 = c + dj$, $z = z_1 + z_2$, let $p = (a + b)(c - d), q = ac, r = bd$, then the real part of z is $q - r$, and the imaginary part of z is $p - q + r$. To calculate p, q and r, we need 3 real multiplications, therefore one complex multiplication requires at least three real multiplications. □

Theorem 9 (The number of multiplications in LightTF)**.** *Given the input sequence length L, the number of patches P, the downsampling factor M, the cutoff frequency f_c, the number of sparse groups K, the number of multiplication operations in N-point RFFT (denoted as M^N_{RFFT}) and N-point IRFFT (denoted as M^N_{IRFFT}), the total number of multiplication operations in LightTF (to process a single channel input sequence $\boldsymbol{x} \in \mathbb{R}^L$) can be calculated as:*

$$\begin{aligned} Mul &= PM \times M^{L/PM}_{RFFT} + \frac{MHP}{L} \times M^{L/PM}_{IRFFT} \\ &\quad + PMf_c^2/K + f_c MP^2 H/L \end{aligned} \tag{27}$$

Specifically, the number of real-valued multiplication operations is 3 times the number of complex-valued multiplication operations, so the total number of real-valued multiplication operations in LightTF can be calculated as:

$$\begin{aligned} Mul_{real} &= 3 \times Mul \\ &= 3(PMM^{L/PM}_{RFFT} + \frac{MHP}{L} M^{L/PM}_{IRFFT} \\ &\quad + PMf_c^2/K + f_c MP^2 H/L) \end{aligned} \tag{28}$$

If we omit the RFFT and IRFFT operations and only consider the multiplication operations in the neural network part, the number of multiplication operations in LightTF can be simplified as:

$$\begin{aligned} Mul_{nn,real} &= 3PMf_c^2/K(SFM) \\ &\quad + 3f_c MP^2 H/L(\text{Patch Predictor}) \end{aligned} \tag{29}$$

Proof. The number of multiplication operations in the RFFT and IRFFT operations can be calculated based on the input sequence length L, the downsampling factor M, and the cutoff frequency f_c. Specifically, the RFFT operation is applied to PM subsequences with length $L_{sub} = \frac{L}{PM}$, and the IRFFT operation is applied to PM subsequences. The number of multiplication operations in the RFFT and IRFFT operations can be calculated as $PM \times M^{L/PM}_{\text{RFFT}}$ and $\frac{MHP}{L} \times M^{L/PM}_{\text{IRFFT}}$, respectively.

The number of multiplication operations in the SFM and patch predictor can be calculated based on the number of

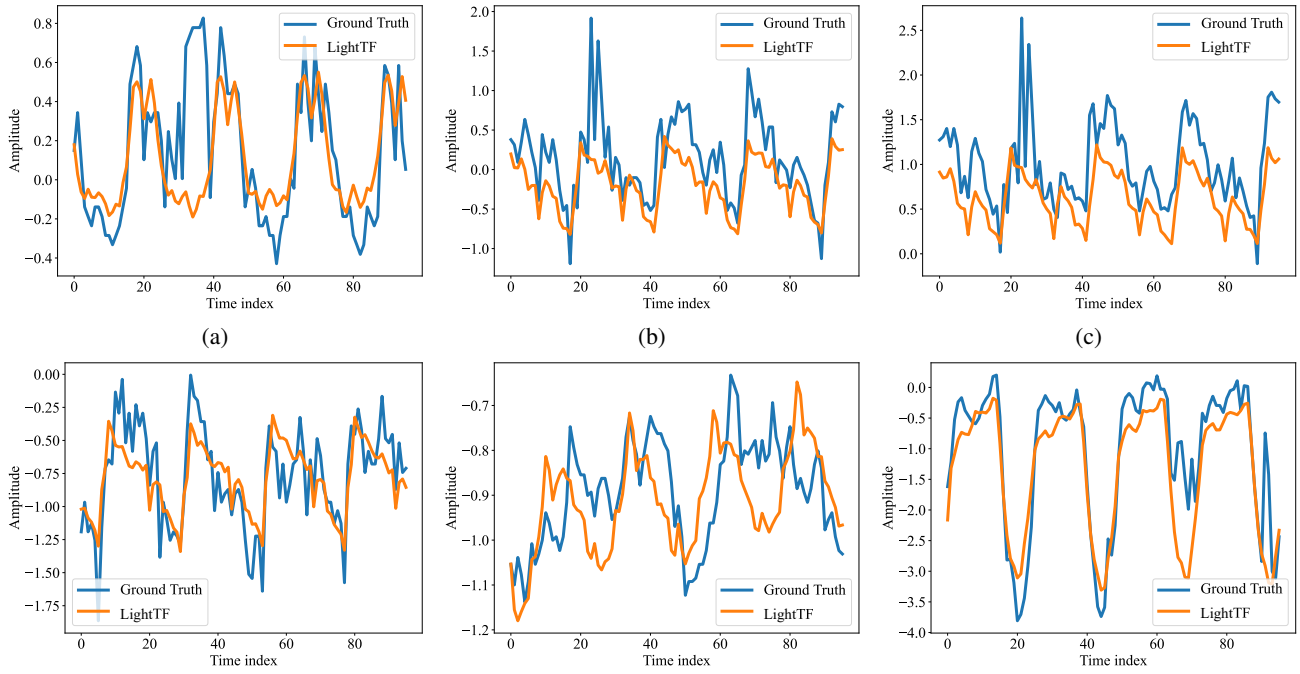


Figure 12: Additional prediction results on the ETTh1 dataset with $H = 96$.

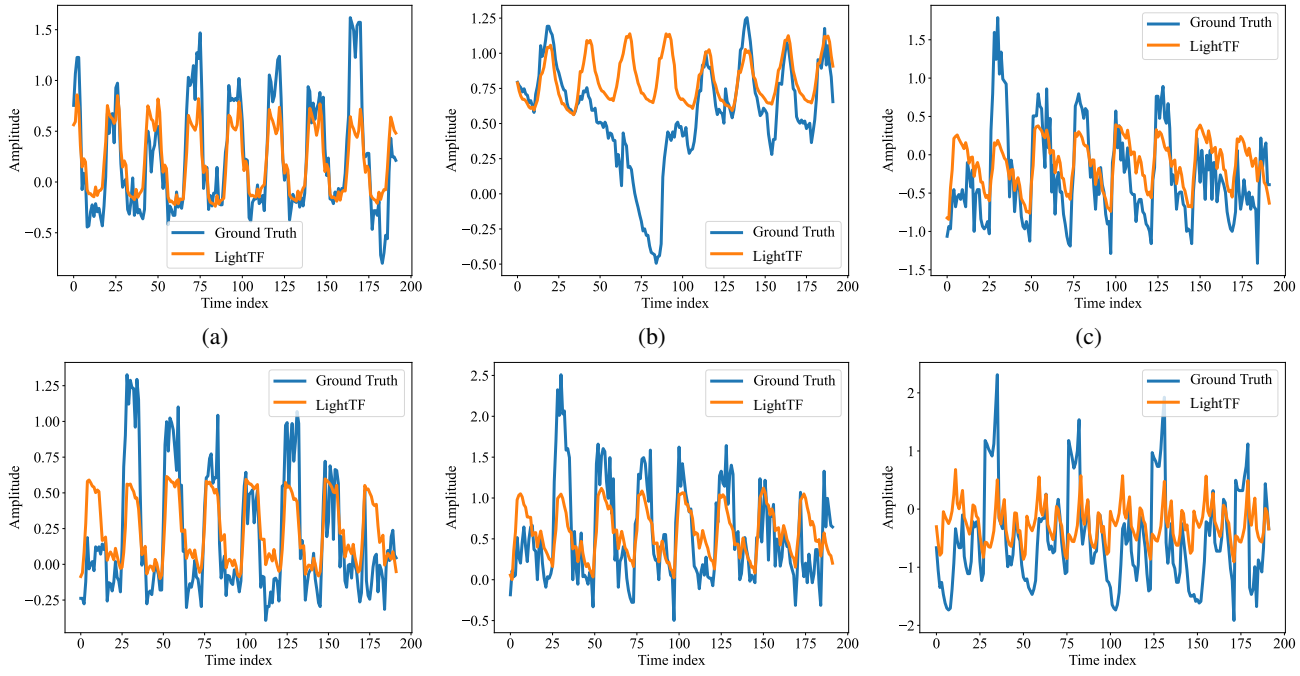


Figure 13: Additional prediction results on the ETTh1 dataset with $H = 192$.

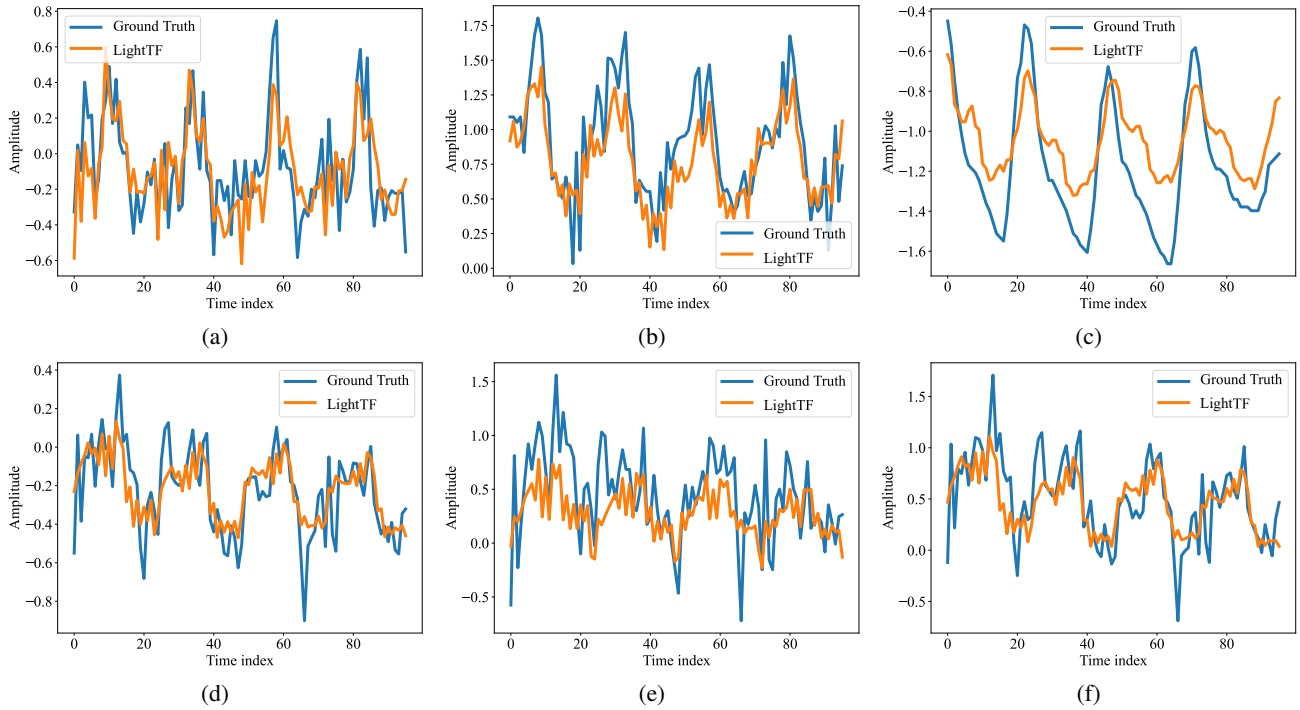


Figure 14: Additional prediction results on the ETTh2 dataset with $H = 96$.

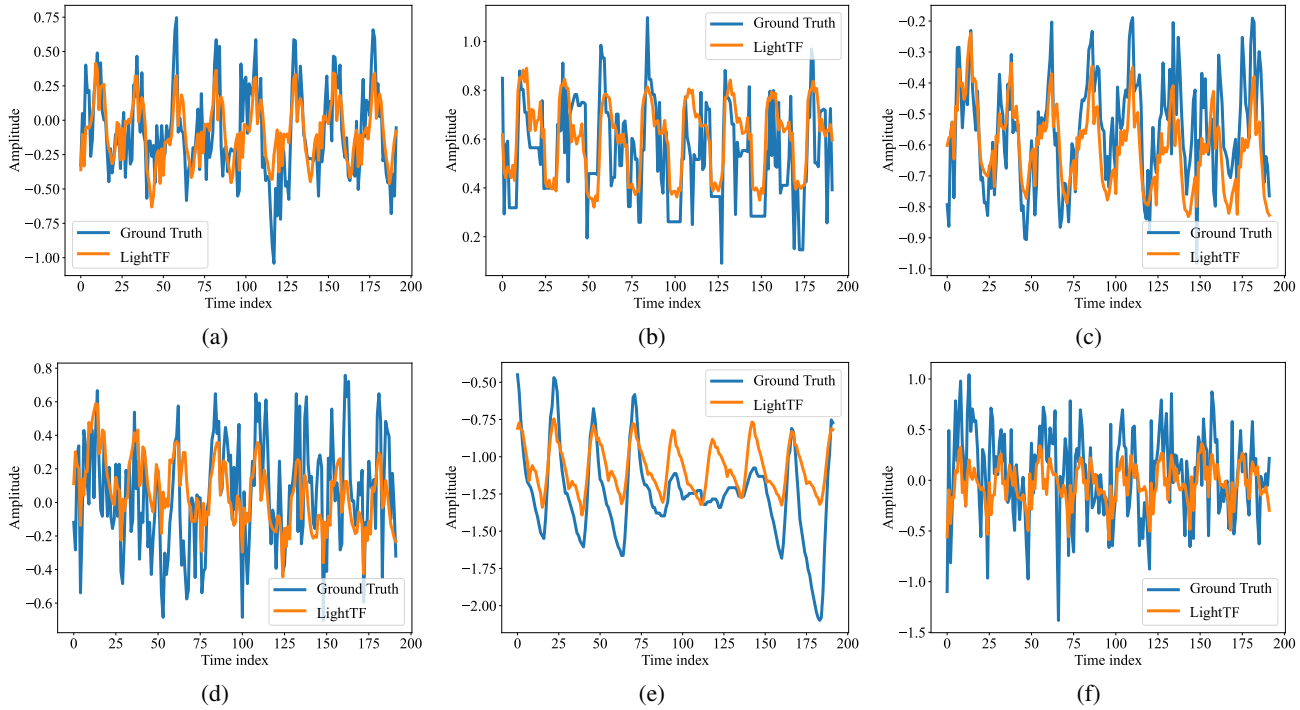


Figure 15: Additional prediction results on the ETTh2 dataset with $H = 192$.

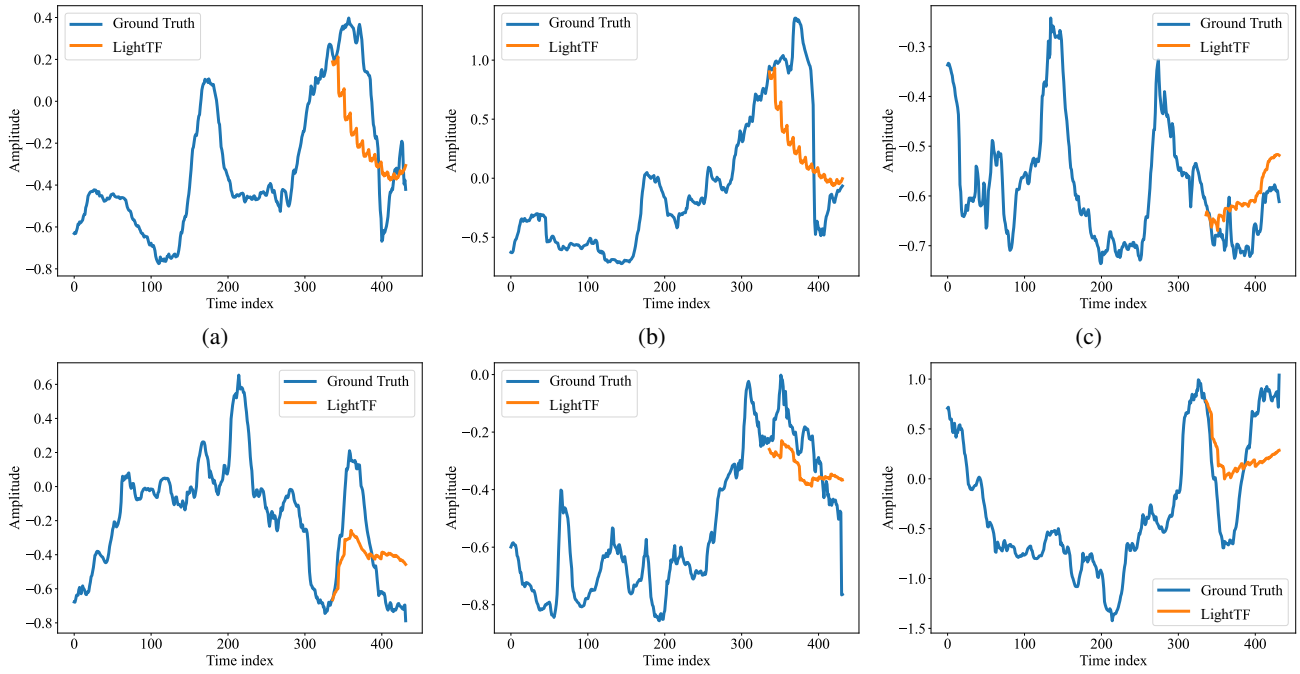


Figure 16: Additional prediction results on the Weather dataset with $H = 96$.

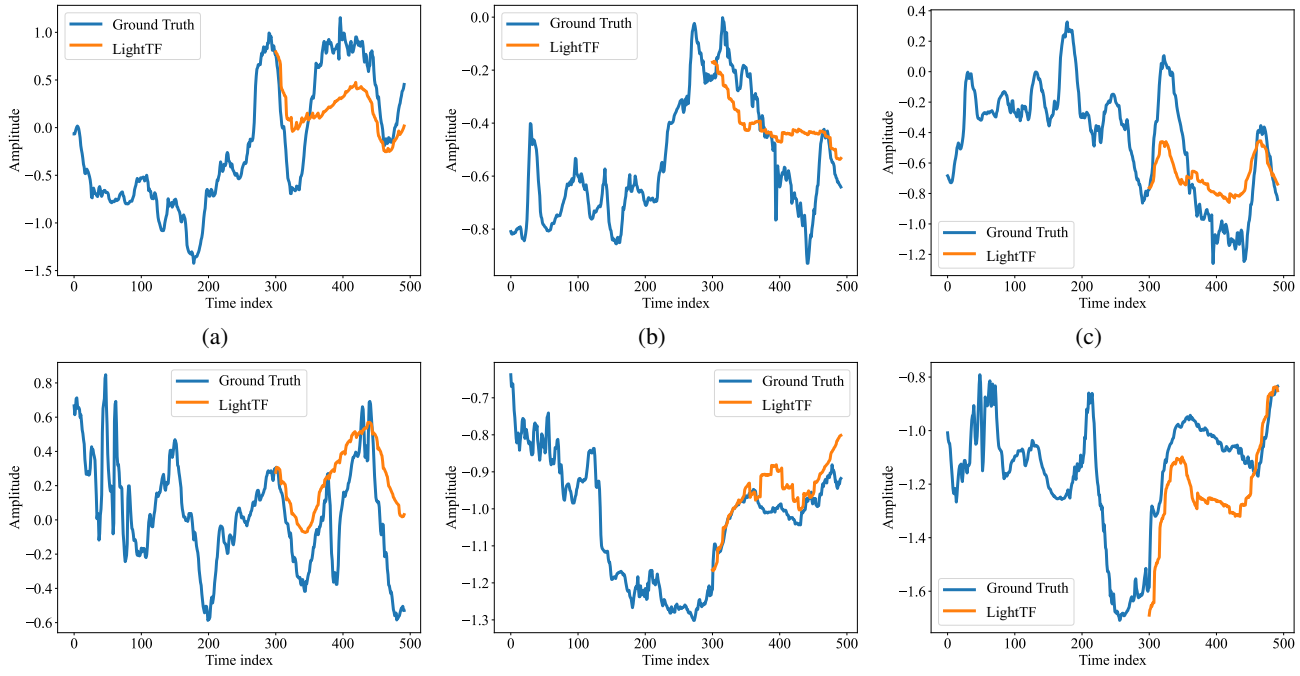


Figure 17: Additional prediction results on the Weather dataset with $H = 192$.

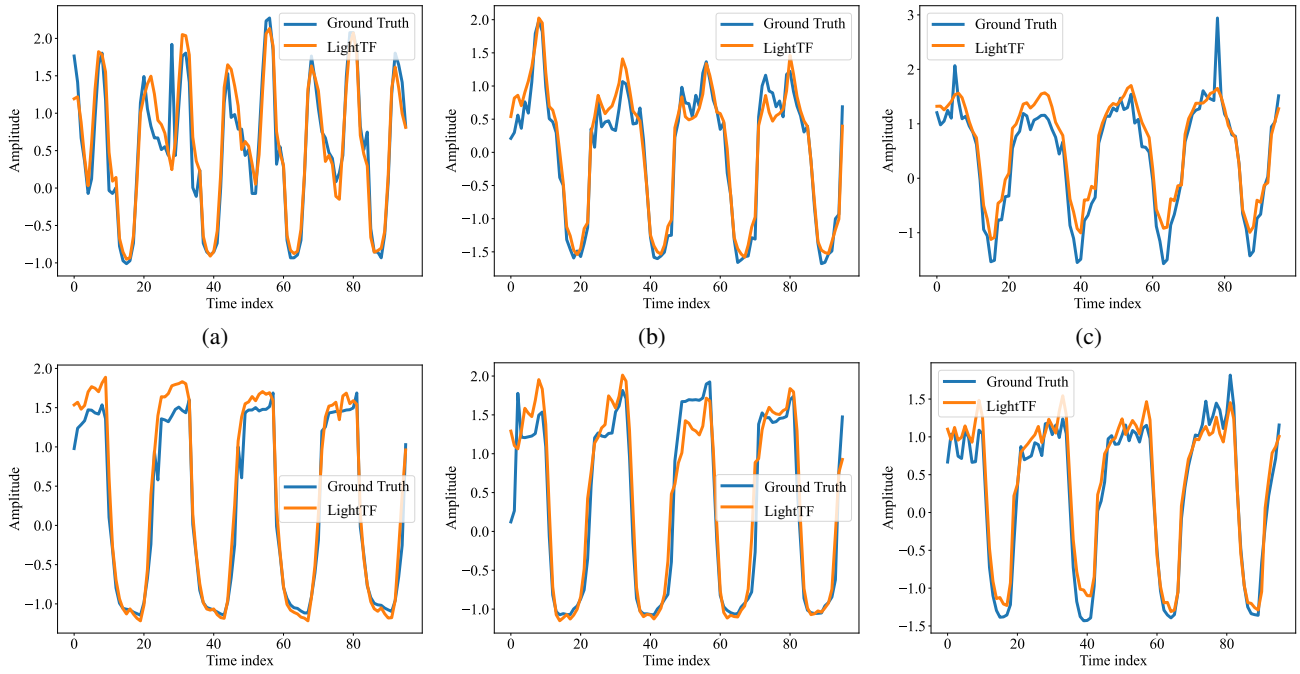


Figure 18: Additional prediction results on the Electricity dataset with $H = 96$.

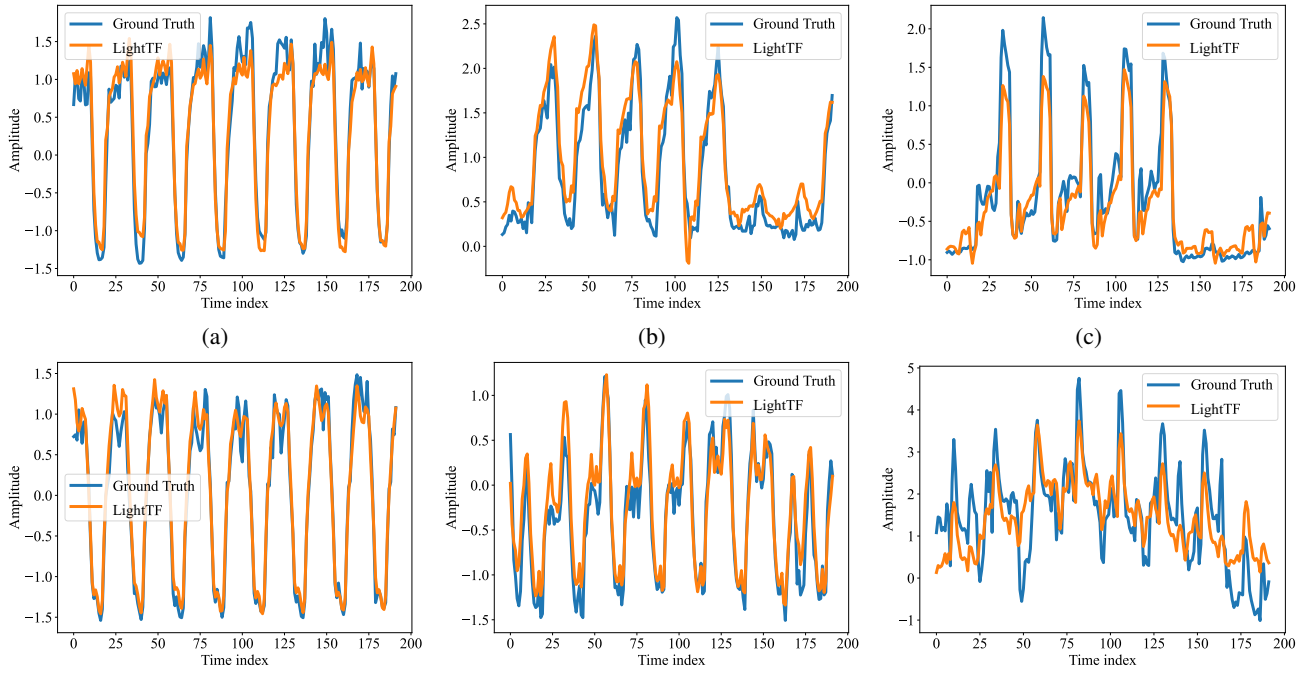


Figure 19: Additional prediction results on the Electricity dataset with $H = 192$.

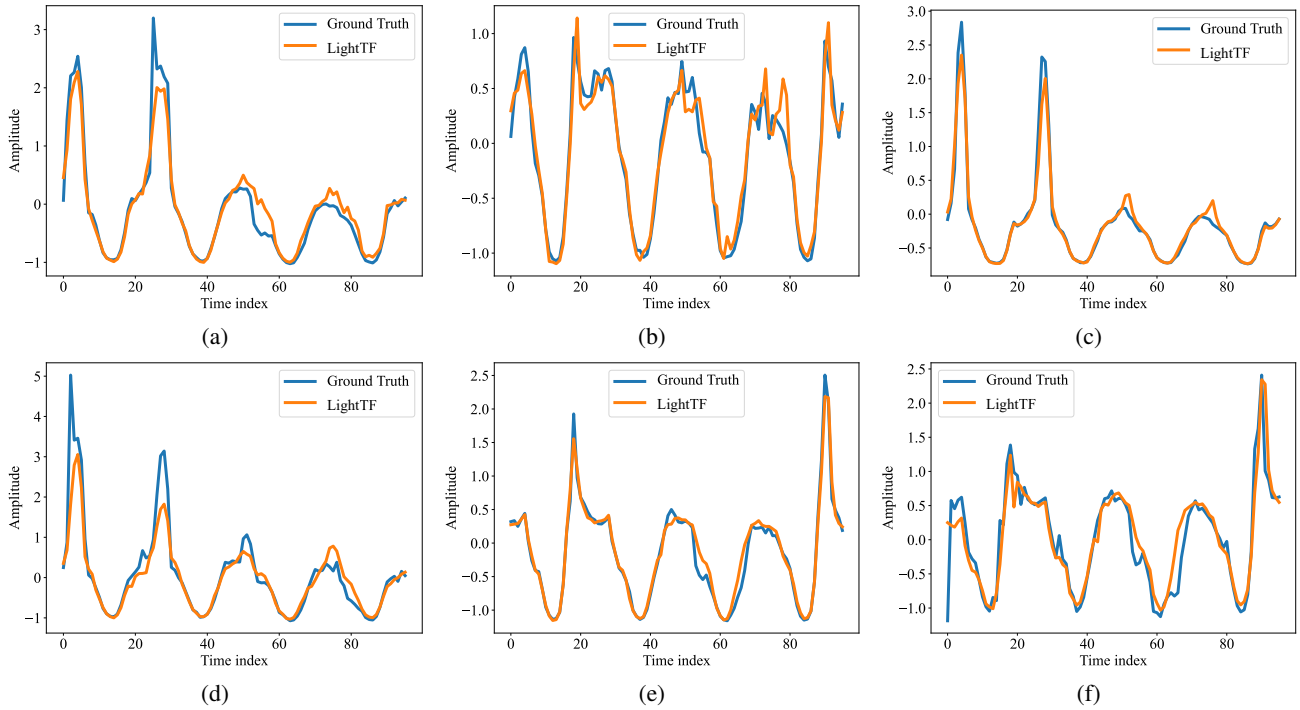


Figure 20: Additional prediction results on the Traffic dataset with $H = 96$.

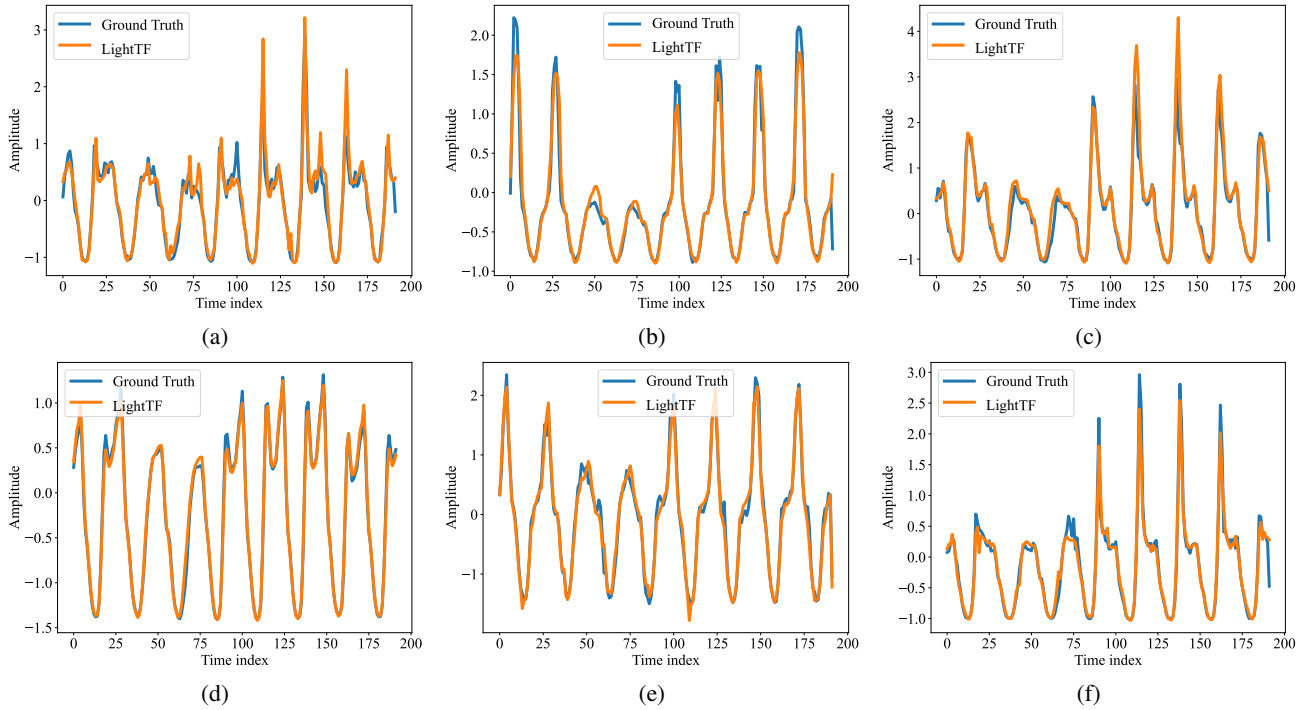


Figure 21: Additional prediction results on the Traffic dataset with $H = 192$

Table 17: The parameter count of **patch predictor** in LightTF with different patch sizes, prediction horizons and look-back windows.

patch size	$L/P = 12$					$L/P = 24$					$L/P = 48$				
Horizon \ Look-back	96	192	336	672	720	96	192	336	672	720	96	192	336	672	720
96	64	128	224	448	480	16	32	56	112	120	4	8	14	28	30
192	128	256	448	896	960	32	64	112	224	240	8	16	28	56	60
336	224	448	784	1568	1680	56	112	196	392	420	14	28	49	98	105
720	480	960	1680	3360	3600	120	240	420	840	900	30	60	105	210	225

Table 18: The parameter count of **SFM** in LightTF with different patch sizes, downsampling factors and sparse group numbers. Note that the values in parentheses represent the corresponding cutoff frequency.

patch size	$L/P = 12$				$L/P = 24$						$L/P = 48$					
K \ M	1	2	4	6	1	2	4	6	8	12	1	2	4	8	12	24
1	49(7)	6(4)	4(2)	4(2)	169(13)	49(7)	16(4)	9(3)	4(2)	4(2)	625(25)	169(13)	49(7)	16(4)	9(3)	4(2)
2	18(6)	8(4)	–	–	72(12)	18(6)	8(4)	–	–	–	288(24)	72(12)	18(6)	8(4)	–	–
3	12(6)	–	–	–	48(12)	12(6)	–	–	–	–	192(24)	48(12)	12(6)	–	–	–
4	–	–	–	–	36(12)	–	–	–	–	–	144(24)	36(12)	–	–	–	–
6	–	–	–	–	24(12)	–	–	–	–	–	96(24)	24(2)	–	–	–	–

sparse groups K, the cutoff frequency f_c, the number of patches P, and the prediction horizon H. Specifically, the SFM applies $\mathbb{C}^{f_c} \to \mathbb{C}^{f_c}$ transformations to each subsequence, results in Mf_c^2/K complex-valued multiplication operations. And similarly, the number of multiplication operations in the patch predictor is $f_c MP^2 H/L$. The total number of multiplication operations in LightTF is the sum of the multiplication operations in the RFFT, IRFFT, SFM, and patch predictor, which gives the formula for Mul.

The number of real-valued multiplication operations is 3 times the number of complex-valued multiplication operations, as each complex multiplication operation can be decomposed into 3 real-valued multiplication operations. Therefore, the total number of real-valued multiplication operations in LightTF is 3 times Mul, which gives the formula for Mul_{real}.

If we omit the RFFT and IRFFT operations and only consider the multiplication operations in the neural network part, the number of multiplication operations in LightTF can be simplified as the sum of the multiplication operations in the SFM and patch predictor, which gives the formula for $\text{Mul}_{\text{nn,real}}$. □

Theorem 10 (The number of multiplications in DLinear and FITS)**.** *We only consider the multiplication operations in the neural network part, so that given the input sequence length L, the prediction horizon H, and the cutoff frequency f_c^{FITS}, the length ratio η and the kernel size for the average pooling K_{avg}, the number of multiplication operations in DLinear can be calculated as:*

$$Mul_{DLinear} = 2LH + K_{avg}L \tag{30}$$

The number of multiplication operations in FITS can be calculated as:

$$Mul_{FITS,nn,real} = 3f_c^{FITS}\lfloor f_c^{FITS}\eta \rfloor \tag{31}$$

Proof. The proof is trivial and omitted here. □

The comparison of the number of multiplication operations in LightTF, DLinear, and FITS is shown in Table 19. The results show that LightTF has a significantly lower number of multiplication operations compared to DLinear and FITS. Specifically, for 720-720 prediction horizon and look-back window, LightTF required only around 2% and 10% of the multiplication operations in DLinear and FITS, respectively.

15 Training Acceleration

To further reduce the training time of LightTF, a preprocessing step can be applied to the input data to accelerate the training process. Specifically, the input data can be downsampled and transformed using the RFFT operation in advance, and the resulting data can be stored for training. This preprocessing step can significantly reduce the computational cost of the RFFT operation during training, as the RFFT operation is computationally intensive and can be a bottleneck in the training process. The detailed algorithm for preprocessing and training with LightTF is shown in Algorithm 1.

16 Development on FPGA Chips

16.1 Brief Introduction to FPGA Chips

FPGA, or Field-Programmable Gate Array, is a type of integrated circuit that allows users to configure hardware functionality after manufacturing. This versatility enables the implementation of custom digital circuits tailored to specific applications, making FPGAs ideal for tasks such as signal processing, data processing, and system control. FPGAs feature a matrix of programmable logic blocks, interconnections, and input/output pins, allowing for high parallel processing capabilities and real-time operation. They are widely used in various industries, including telecommunications, automotive, and aerospace, due to their low la-

Table 19: The number of multiplication operations in LightTF, DLinear, and FITS with different prediction horizons and look-back windows. The kernel size in DLinear is set to 25, the cutoff frequency for FITS is set to half of the RFFT sequence length, the patch size is set to 48, the f_c is set to 12, the downsampling factor M is set to 2, and the sparse group number K is set to 3.

patch size	DLinear					FITS					LightTF				
Horizon \ Look-back	96	192	336	672	720	96	192	336	672	720	96	192	336	672	720
96	20.8K	41.7K	72.9K	145.8K	156.2K	3.5K	10.4K	27.2K	96.8K	110.2K	864	1728	3024	6048	6480
192	39.3K	78.5K	137.4K	274.8K	294.5K	5.2K	13.8K	33.3K	108.9K	123.1K	1152	2304	4032	8064	8640
336	66.9K	133.8K	234.2K	468.4K	501.8K	7.8K	19.0K	42.3K	127.0K	142.6K	1584	3168	5544	11088	11880
720	140.6K	281.3K	492.2K	984.5K	1054.8K	14.7K	32.9K	66.5K	175.4K	194.4K	2736	5472	9576	19152	20520

Algorithm 1: Preprocess and Train with LightTF

Require: Training data $\boldsymbol{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$, Downsample factor M, cutoff frequency f_c, Batch size B
Ensure: Trained model θ
Preprocessing:
for $i = 1$ to N **do**
  $\boldsymbol{x}^{(i)}_{\text{downsampled}} \leftarrow \text{Downsample}(\boldsymbol{x}^{(i)}, M)$
  $\boldsymbol{X}^{(i)}_{\text{RFFT}} \leftarrow \text{RFFT}(\boldsymbol{x}^{(i)}_{\text{downsampled}})$
  $\boldsymbol{X}^{(i)}_{\text{preprocessed}} \leftarrow \text{Cut}(\boldsymbol{X}^{(i)}_{\text{RFFT}}, f_c)$
end for
Store preprocessed data $\boldsymbol{X}_{\text{preprocessed}}$
Training:
for each training epoch **do**
  for each mini-batch of size B **do**
    Sample a batch $\boldsymbol{X}_{\text{batch}} \subset \boldsymbol{X}_{\text{preprocessed}}$
    Forward pass: Predict $\hat{\boldsymbol{X}} \leftarrow \text{LightTF}(\boldsymbol{X}_{\text{batch}})$
    Compute loss $L(\hat{\boldsymbol{X}}, \boldsymbol{Y})$
    Backpropagate gradients
    Update model parameters θ
  end for
end for

tency, low power consumption, and adaptability to changing requirements.

16.2 Zynq UltraScale+ RFSoC ZCU208 Evaluation Kit

The Zynq™ UltraScale+™ RFSoC ZCU208 Evaluation Kit is an ideal platform for out-of-the-box RF evaluation and cutting-edge application development. It features the Zynq UltraScale+ RFSoC ZU48DR, which integrates eight 14-bit 5GSPS ADCs, eight 14-bit 10GSPS DACs, and eight soft-decision forward error correction (SD-FEC) cores, making it suitable for RF-class applications. The key features of the ZCU208 Evaluation Kit are shown in Table 20.

The board features are shown in Figure 22.

16.3 Usage Details

The resource utilization of LightTF on the ZCU208 Evaluation Kit is shown in Figure 23. We now give a brief introduction for clock, BRAM, and DSP, shown below:

Table 20: Key Parameters of Zynq UltraScale+ RFSoC ZCU208

Parameter	Value
14-bit, 5.0 GSPS RF-ADC Count	8
14-bit, 10.0 GSPS RF-DAC Count	8
SD-FEC Cores	8
System Logic Cells (K)	930
Memory (Mb)	60.5
DSP Slices	4272
33G Transceivers	16
Maximum I/O Pins	347

- **Clock** The clock in an FPGA provides the timing signal that synchronizes all operations within the device. It controls the flow of data by determining when actions such as data processing or memory reads/writes occur. Multiple clock domains can exist in an FPGA design to drive different parts of the logic at different frequencies, which helps optimize performance and reduce power consumption.

- **BRAM (Block RAM)** BRAM (Block Random Access Memory) is a dedicated on-chip memory resource available within the FPGA. It provides high-speed memory that can be used for data storage, buffering, or caches. BRAM blocks are widely used in applications requiring temporary data storage, such as signal processing, image processing, and communication systems. Unlike external memory, BRAM is tightly integrated into the FPGA fabric, making data access faster and more efficient.

- **DSP (Digital Signal Processing) Slices** DSP slices are specialized hardware units in an FPGA that are optimized for performing arithmetic operations such as multiplication, addition, and accumulation, which are common in digital signal processing tasks. These slices allow FPGAs to efficiently handle operations like filtering, fast Fourier transforms (FFT), and other real-time processing tasks. By leveraging DSP slices, designers can offload critical signal processing operations from general-purpose logic, improving performance and reducing resource usage.

Please note that the results may vary depending on the specific FPGA chip and the implementation details, for example, the algorithm for FFT.

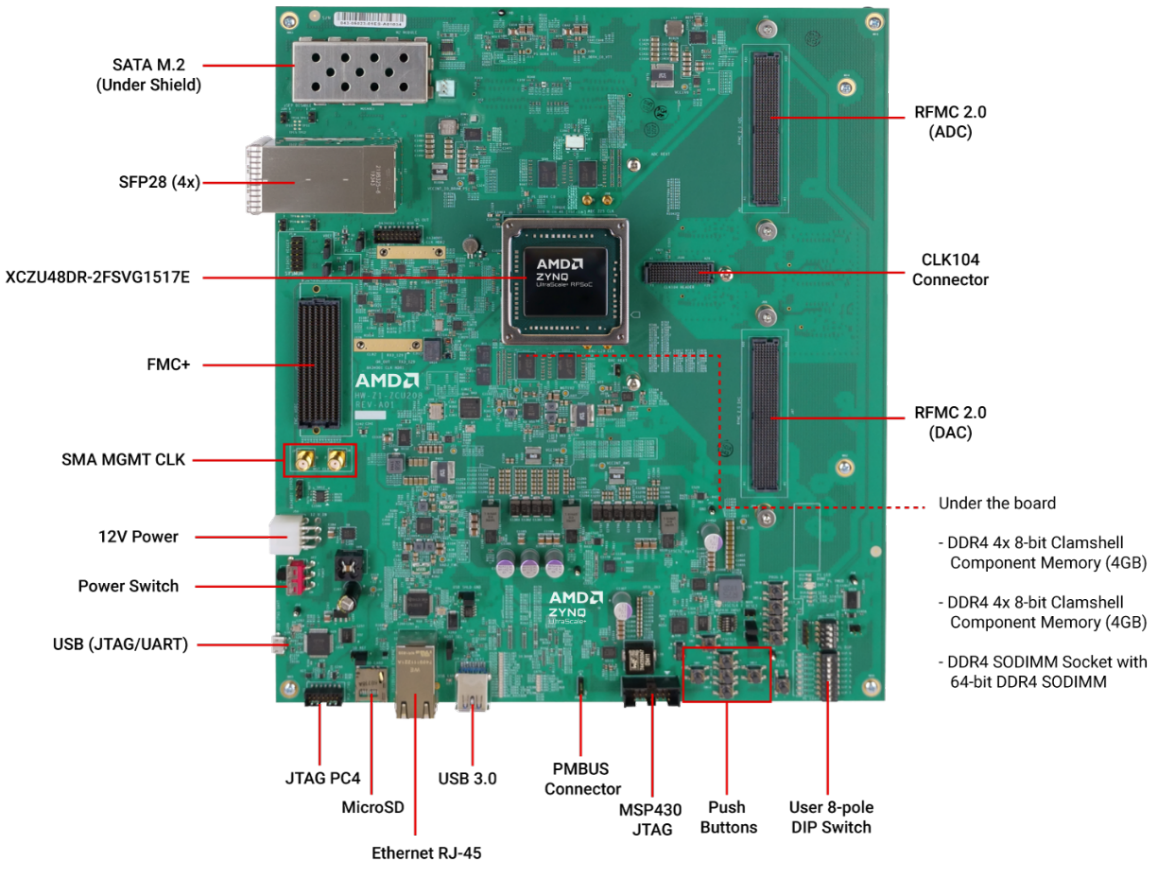
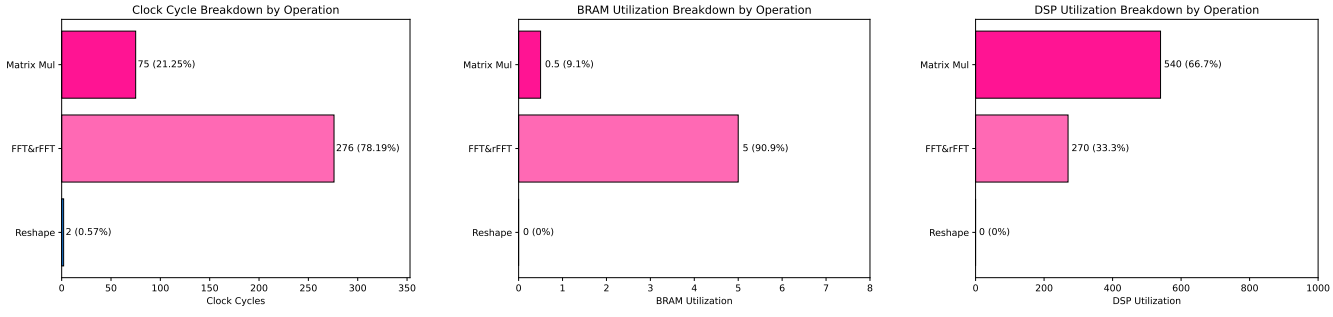


Figure 22: Zynq UltraScale+ RFSoC ZCU208 Evaluation Kit



(a) (b) (c)

Figure 23: Resource utilization of LightTF on ZCU208 Evaluation Kit. (a) Break down by clock cycles (b) Break down by BRAM utilization. (c) Break down by DSP utilization.

16.4 Future Work

In this paper, we only present a hardware implementation scheme. Future work can be further explored in the following aspects:

- **A more streamlined hardware implementation of LightTF** Theoretically, LightTF requires significantly less computation than other algorithms, which presents opportunities for reducing the computational resources required on the FPGA.

- **Integration with other hardware accelerators** Implementation on additional hardware platforms: In the near future, we aim to deploy this algorithm on a range of other hardware devices, including embedded systems such as Raspberry Pi, ESP32, and STM32, to fully verify the performance of the proposed algorithm.