U-Mixer: An Unet-Mixer Architecture with Stationarity Correction for Time Series Forecasting

Xiang Ma[1], Xuemei Li[1], Lexin Fang[1], Tianlong Zhao[1], Caiming Zhang[1,2*]

[1]School of Software, Shandong University, Jinan 250101, China
[2]Shandong Provincial Laboratory of Future Intelligence and Financial Engineering, Yantai 264005, China
{xiangma, fanglexin, tianlongzhao}@mail.sdu.edu.cn, {xmli, czhang}@sdu.edu.cn

Abstract

Time series forecasting is a crucial task in various domains. Caused by factors such as trends, seasonality, or irregular fluctuations, time series often exhibits non-stationary. It obstructs stable feature propagation through deep layers, disrupts feature distributions, and complicates learning data distribution changes. As a result, many existing models struggle to capture the underlying patterns, leading to degraded forecasting performance. In this study, we tackle the challenge of non-stationarity in time series forecasting with our proposed framework called U-Mixer. By combining Unet and Mixer, U-Mixer effectively captures local temporal dependencies between different patches and channels separately to avoid the influence of distribution variations among channels, and merge low- and high-levels features to obtain comprehensive data representations. The key contribution is a novel stationarity correction method, explicitly restoring data distribution by constraining the difference in stationarity between the data before and after model processing to restore the non-stationarity information, while ensuring the temporal dependencies are preserved. Through extensive experiments on various real-world time series datasets, U-Mixer demonstrates its effectiveness and robustness, and achieves 14.5% and 7.7% improvements over state-of-the-art (SOTA) methods.

Introduction

Time series forecasting plays a vital role in various domains such as finance (Ma et al. 2022), weather forecasting (Liu et al. 2022a), and sensor data analysis (Zhao et al. 2023). Extracting meaningful patterns, understanding the underlying dynamics of time series to forecast future trends are crucial for informed decision-making and effective problem-solving (Zhang, Guo, and Wang 2023). With the advent of deep learning, convolutional neural networks (CNNs) (Fukushima 1980) and Transformers (Vaswani et al. 2017) have shown remarkable progress in capturing temporal dependencies and extracting features from time series. Recently, the Mixer architecture (Tolstikhin et al. 2021), initially introduced for vision tasks, has gained attention for its ability to model complex relationships within sequential data. However, the application of Mixer to time series forecasting also presents challenges. Time series data often exhibits non-stationary caused by factors such as trend, seasonality, or irregular fluctuations. Such property can hinder accurate modeling and prediction, thereby limiting the effectiveness of the Mixer architecture. In dealing with non-stationary, Mixer mainly have the following three problems: (1) The deep structure of Mixer leads to unstable propagation of shallow features. When information flows through multiple layers, the transmission of low-level features becomes non-stationary. (2) The mixed feature extraction of different channels leads to non-stationary feature distributions, due to significant distribution variations among channels. (3) Model training cannot intuitively learn changes in data distribution, resulting in shifted and non-stationary distribution of predicted values.

In this article, we present U-Mixer to address the issue of non-stationarity in time series forecasting. U-Mixer combines the advantages of both Unet and Mixer architecture to capture local temporal patterns of different levels while separately handles the temporal and channel interactions. The key contribution of U-Mixer lies in the novel stationarity correction technique, which explicitly restores the distribution of data by correcting the stationarity of data.

Specifically, we divide the time series into some patches and process them independently using the Mixer architecture. This patch-based processing allows localized analysis of temporal patterns and captures fine-grained details within the data. Mixer effectively captures the dependencies between different patches and channels without disrupting the overall temporal sequence through the multi-layer perceptron (MLP) block, and learns meaningful representations from the entire time series. Simultaneously, it separately handles the temporal interactions and channel interactions, avoiding feature instability caused by significant variations in data distributions across different channels. Different from MLP-Mixer, U-Mixer adopts a Unet architecture to merge the low- and high-level features and obtain a more comprehensive and richer data representation, thereby improving the ability to understand and model data. Additionally, in order to improve model performance, time series often undergoes a process of stationarization. However, due to the loss of non-stationary information, the model cannot intuitively learn the changes in data distribution, resulting in a shift in the distribution of predicted values. Learning the mapping of data distribution through training is inher-

*Corresponding author.

ently a challenge. Therefore, we propose a stationarity correction method by constraining the difference in stationarity between the data before and after model processing to restore the non-stationary information, while consider the temporal dependencies of the data.

Our contributions are summarized as

- We propose U-Mixer a new time series forecasting framework that can capture local temporal patterns of different levels and handle the temporal and channel interactions separately.
- We propose a stationarity correction method that explicitly restores the non-stationarity information of data by constraining the stationarity differences before and after model processing, while ensuring temporal dependencies within the data.
- U-Mixer adopts the Unet architecture to merge the low- and high-level features, which enables more comprehensive and richer representations of the data.
- We demonstrate the effectiveness and robustness of U-Mixer through extensive experiments on various real-world time series datasets.

Related Work

Time Series Prediction

Time series forecasting is the task to predict future values for one or more variables based on a set of historical observations. Traditional statistical methods and machine learning methods have been widely used but struggle with complex nonlinear patterns, although they are simple and interpretable. In recent years, deep learning models have been widely investigated for this task. Especially long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997), a well-known variant of recurrent neural networks (RNNs), is widely employed in time series forecasting. However, they may suffer from vanishing or exploding gradients, limiting their ability to capture long-term dependencies. Convolutional Neural Networks (CNNs) (Zhao et al. 2023) have also been adapted for this task, by extracting local patterns and identify relevant features across different time steps. CNN-based models have shown promising results in capturing spatial and temporal dependencies within the data.

More recently, Transformer-based models (Zhou et al. 2021; Wu et al. 2021; Zhou et al. 2022) have gained attention in time series forecasting. Transformers leverage self-attention mechanisms to effectively capture global dependencies and long-range interactions within the data. They have achieved state-of-the-art (SOTA) results in various time series forecasting tasks. In contrast to expectations, recent studies (Zeng et al. 2023) have revealed that even a basic univariate linear model can outperform complex multivariate Transformer models by a margin on widely-used long-term prediction benchmarks. Despite the significant advancements in Transformer-based models, this unexpected finding highlights the effectiveness of linear models. In this paper, we introduce the Mixer architecture for time series forecasting to fully utilize the performance of linear models, which is designed by stacking MLPs. And we combine Mixer with the Unet architecture to integrate different levels of features to build more comprehensive richer representations

Stationarization for Time Series Forecasting

Stationarization serves as a fundamental assumption for time series analysis, allowing us to apply various models to efficiently capture patterns in the data and enhence the robustness of models. To stabilize time series data, traditional methods employ various preprocessing approaches, such as differencing, to remove trends, seasonality, and non-stationarity in the data. As for deep models, the presence of non-stationarity and the accompanying variation in data distributions pose significant challenges to time series forecasting. To address this, stationarization are commonly explored and employed as a preprocessing step for deep model inputs (Liu et al. 2022b). By transforming the data into a more stationary form, these methods aim to mitigate the difficulties associated with non-stationarity and enable more effective training and prediction with deep models. Normalization is a widely adopted stationarization method, aiming to mitigate the negative effects of non-stationary features on the learning process by transforming the data into a range that is better aligned with the model.

While applying normalization can address the issue of non-stationarity, it introduces a potential preblem (Kim et al. 2021). Normalization can inadvertently remove non-stationary information that may hold valuable insights for predicting future values. Because normalization changes the distribution of features, potentially hindering the model's ability to capture the nuanced dynamics of the time series. Some research (Kim et al. 2021; Liu et al. 2022b) explicitly return the information deleted through input normalization to the model, eliminating the need for the model to reconstruct the original distribution, thereby reducing the difficulty of modeling. However, directly stationarizing time series will damage the model's capability of modeling specific temporal dependency. Therefore, we propose a stationarity correction method by constraining the difference in stationarity between the data before and after model processing. This method explicitly conveys statistical features of data distribution while maintaining temporal dependencies whthin the data.

Methodology

The architecture of U-Mixer is shown in Figure 1, and the details are described in the following sections.

Notations

Define the historical data as $X \in \mathbb{R}^{C \times L} = \{x_i \mid i \in [1, L]\}$. Here L means the length of the input sequence and C is the number of channels (or variables). x_i is a vector of dimension C at time step i. Let the ground truth future values be $Y \in \mathbb{R}^{C \times H} = \{x_i \mid i \in [L+1, L+H]\}$. Here H means the length of forecasting sequence. We focus on using a learning model $\mathcal{M}$ to analysis X and achieve predicting the values of Y, and the process can be expressed as $\hat{Y} = \mathcal{M}(X)$. $\hat{Y}$ means the prediction results.

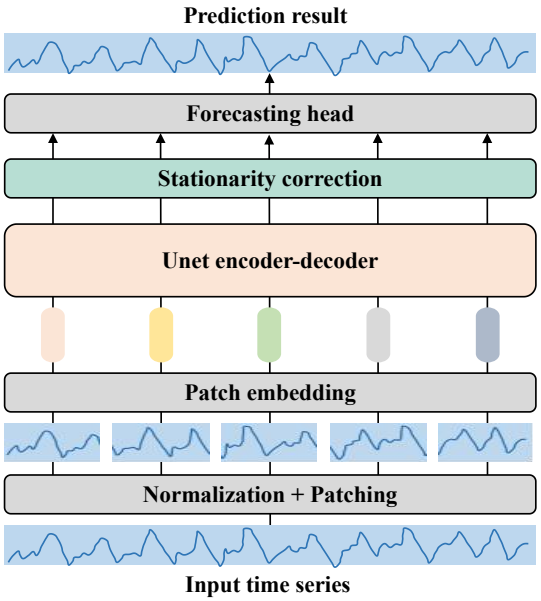


Figure 1: The architecture of U-Mixer, which consists of per-patch embeddings, Unet encoder-decoder, stationarity correction and a forecasting head.

Normalization and Patch Embedding

The input data X go through a normalization process, which is: $X = \frac{X-\mu_{in}}{\sigma_{in}}$. Here μ_{in} and σ_{in} is the vectors composed of the mean and variance of all channels in X, respectively. Through this process, the range of data variation is adjusted to a more suitable scale, contributing to enhanced stability and performance of the model.

After normalization, X is divided into overlapping or non-overlapping patches. Define the patch length as P and the stride between two consecutive patches as S. We can obtain the patch sequence $X_p \in \mathbb{R}^{(C\times N)\times P}$. Here $N = \lfloor \frac{L-P}{S} \rfloor + 2$ is the number of patches. $\lfloor \cdot \rfloor$ is floor function. We repeat the last column of X S times and pad it to the original sequence before patching.

Patches are mapped to the embeddings $X_d \in \mathbb{R}^{(C\times N)\times D}$, where D is the latent space dimension. We use a linear projection $W_{val} \in \mathbb{R}^{P\times D}$ to learn the mapping relationship and a additive position encoding $W_{pos} \in \mathbb{R}^{(C\times N)\times D}$ to provide the information about the relative position of patches. Thus, $X_d = X_p W_{val} + W_{pos}$. Then X_d will be feeded into the Unet encoder-decoder to capture the dependencies between different patches and channels. Because model processing will cause the distribution in the patches to change, here we also need to record the mean μ_x, the variance σ_x and auto-correlation matrix $R(X_d)$ of X_d, so that the distribution of the model's output will be restored in the subsequent stationarity correction operation.

Unet Encoder-decoder

As shown in Figure 2, U-Mixer introduces a novel time series prediction network that combines the Unet architecture with the Mixer architecture.

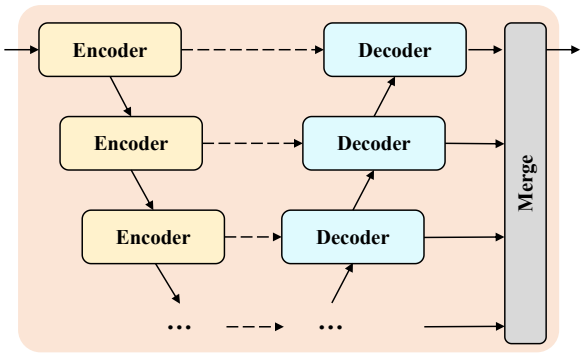


Figure 2: The Unet encoder-decoder of U-Mixer. Encoders and decoders are both MLP blocks. The term "Merge" refers to the combining process of features from different levels.

U-Mixer adopts the encoder-decoder structure of Unet with multiple levels. Encoders adopt a hierarchical structure, progressively extracting features of low- and high-levels from embeddings X_d. Each encoder is responsible for transforming the input embeddings into high-dimensional representations that captures key features and contextual information. Define the input of encoders as:

$$X_{in,i} = \begin{cases} X_d & i = 1 \\ X_{out,i-1} & i \in (1, M] \end{cases} \tag{1}$$

$X_{in,i}$ means the input of the i-level encoder, and M is the number of levels. The output of encoders can be expressed as:

$$X_{out,i} = \mathcal{M}_{en,i}(X_{in,i}), \ \ i \in [1, M] \tag{2}$$

$X_{out,i}$ is the output of the i-level encoder and $\mathcal{M}_{en,i}$ refers to the i-level encoder.

Decoders also adopt a hierarchical structure, progressively analyzing representations generated by encoders. Each decoder is responsible for generating the parsed representations by analyzing the representations from the output of the previous decoder. During the parsing process, each decoder also need to consider the output of the same level encoder to preserve and utilize the features of the same level. This process is achieved through the skip connections at the corresponding level. The input of decoders can be formalized as:

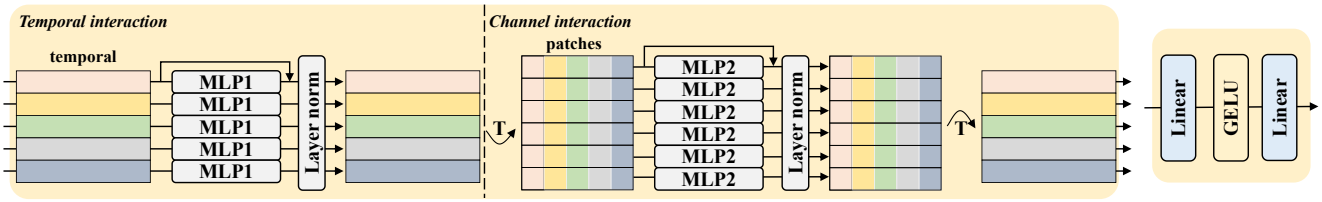
$$Y_{in,i} = \begin{cases} Y_{out,i+1} & i \in [1, M) \\ X_{out,i} & i = M \end{cases} \tag{3}$$

$Y_{in,i}$ means the input of the i-level decoder. The output of decoders is defined as:

$$Y_{out,i} = \begin{cases} W_y(\mathcal{M}_{de,i}(Y_{out,i+1}) + \mathcal{M}_{de,i}(Y_{in,i})) & i \in [1, M) \\ \mathcal{M}_{de,i}(X_{out,M}) & i = M \end{cases} \tag{4}$$

$Y_{out,i}$ is the output of the i-level decoder and $\mathcal{M}_{de,i}$ refers to the i-level decoder. W_y is a simple linear layer to merge the features generated from the same level encoder and the previous decoder.

For ease of description, the final output of the encoder-decoder structure is defined as $Y_d \in \mathbb{R}^{(C\times N)\times D} = Y_{out,1}$.



(a) MLP block (Encoder / Decoder) (b) MLP

Figure 3: The MLP block. (a) is the MLP block, which contains one temporal MLP layer and one channel MLP layer. (b) is the specific structure of MLP, which consists of two fully-connected layers and a GELU nonlinearity.

MLP Block

Encoders and decoders are both MLP blocks, as shown in Figure 3(a). MLP block contains two MLP layers, which are used to implement temporary interaction and channel interaction, respectively. One MLP layer is first used to perform interactions on the input data in the temporal dimension, which contains several MLPs for independently processing each channel. This process has no interactions between different channels to avoid affecting temporal interactions due to differences in distribution between channels. After layer normalization, the output is transposed. Another MLP layer is used to perform interaction on the channel dimension. The output is transposed back to its previous shape after layer normalization. During the interaction process, we also employ skip connections to concatenate the input and output features of MLP layers, reducing the information loss introduced by feature transformation and providing a more comprehensive feature representation. An MLP consists of two linear layers, a GELU activation function and a dropout, as shown in Figure 3(b). As the core components of MLP, linear layers perform linear transformations on the inputs to learn linear combinations of features. GELU achieves nonlinear mapping by applying a Gaussian error linear transformation to the inputs, which is widely used in time series forecasting. Dropout improves the model's robustness by reducing the sensitivity of the network to specific features or input patterns.

Stationarity Correction

Due to the removal of non-stationary information from the input, the distribution of the data goes through a significant shift. Existing methods for restoring the data distribution primarily focus on statistical measures such as mean and variance, without considering the temporal dependencies within the data. As a result, essential features like trends and seasonal in the original data may be affected. Therefore, we introduce a novel stationarity correction method that constraining the relationship between the stationarity of the time series before and after model processing to rectify the data distribution while preserving data dependencies.

The stationarity of time series is mainly characterized by two aspects: the mean and the covariance. The mean is mainly responsible for constraining the distribution from the perspective of statistics, while covariance constrains the distribution from the perspective of temporal dependencies. The change in the mean of a data distribution is typically achieved through global addition or subtraction operations and does not affect the covariance. However, adjusting the covariance can lead to changes in the data distribution's mean. Therefore, we first adjust the covariance of the data. The covariance $Cov(X_d, X_d^i)$ can represent the dependence between the series X_d and its i lag series, which is not sufficient to fully describe the temporal dependencies of the entire time series. Therefore, we introduce autocorrelation matrice to provide a more comprehensive constraint on the time series dependencies.

Define $R(X_d) \in \mathbb{R}^{L\times L} = \{R_{i,j}(X_d) \mid i,j \in [1,L]\}$ as the autocorrelation matrice of X_d, and $R_{i,j}(X_d) = \frac{Cov(X_d^i, X_d^j)}{\sqrt{\sigma_x^i \sigma_x^j}}$. Here $Cov(\cdot)$ is the computation of covariance. X_d^i means the i lag series of X_d, and σ_d^i represents the variance of X_d^i. We perform an affine transformation on the output of model Y_d using matrix α, such that the autocorrelation matrix $R(\alpha Y_d)$ approximates $R(X_d)$. Here $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_L] \in \mathbb{R}^{1\times L}$ is a matrix, and α_i is a scalar. The constraint between $R(X_d)$ and $R(\alpha Y_d)$ can be expressed as:

$$\mathcal{M}_\alpha = \|R(X_d) - R(\alpha Y_d)\|_F^2 \tag{5}$$

Y_d is a non-stationary time series, multiplying Y_d by α will result in the changes between data points in Y_d being scaled by a factor of α^2. So $\mathcal{M}_\alpha$ is equivalent to:

$$\mathcal{M}_\alpha = \|R(X_d) - \alpha^2 R(Y_d)\|_F^2 \tag{6}$$

Here:

$$\alpha_i = \sqrt{\frac{\sum_{j=1}^{L} R_{i,j}(X_d) R_{i,j}(Y_d)}{\sum_{j=1}^{L} R_{i,j}^2(X_d)}} \tag{7}$$

According to the Wiener-Khinchin theorem (Cohen 1998), we can accelerate the computation of α_i by the Fast Fourier Transform (FFT):

$$\alpha_i = \sqrt{\frac{\sum_{j=1}^{L} \mathcal{F}^{-1}(\mathcal{F}(\overline{X_d^i})\overline{\mathcal{F}(\overline{X_d^j})})\mathcal{F}^{-1}(\mathcal{F}(\overline{Y_d^i})\overline{\mathcal{F}(\overline{Y_d^j})})}{\sum_{j=1}^{L} \mathcal{F}^{-1}(\mathcal{F}(\overline{X_d^i})\overline{\mathcal{F}(\overline{X_d^j})})^2}} \tag{8}$$

The means of X_d and Y_d need to be zero to satisfy the Wiener-Khinchin theorem. Here $\overline{X_d^i} = X_d^i - \mu_x^i$ and $\overline{Y_d^i} = Y_d^i - \mu_y^i$. μ_x^i and μ_y^i are the means of X_d^i and Y_d^i, respectively. α is determined by $\overline{X_d}$ and $\overline{Y_d}$.

The output Y_d can be update to $\hat{Y}_d \in \mathbb{R}^{(C\times N)\times D} = \alpha Y_d + \bigtriangledown_\mu$, $\bigtriangledown_\mu = \mu_x - \mu_y$ is used to adjust the difference in mean between Y_d and X_d.

Models		U-Mixer		TimesNet		ETSformer		LightTS		DLinear		FEDformer		Autoformer		LSSL	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	96	**0.317±2e^{-3}**	**0.349±2e^{-3}**	0.338	0.375	0.375	0.398	0.374	0.400	0.345	0.372	0.379	0.419	0.505	0.475	0.450	0.477
	192	**0.369±2e^{-3}**	**0.376±2e^{-3}**	0.374	0.387	0.408	0.410	0.400	0.407	0.380	0.389	0.426	0.441	0.553	0.496	0.469	0.481
	336	**0.395±3e^{-3}**	**0.393±4e^{-3}**	0.410	0.411	0.435	0.428	0.438	0.438	0.413	0.413	0.445	0.459	0.621	0.537	0.583	0.574
	720	**0.443±2e^{-3}**	**0.424±1e^{-3}**	0.478	0.450	0.499	0.462	0.527	0.502	0.474	0.453	0.543	0.490	0.671	0.561	0.632	0.596
	Avg	**0.381±2e^{-3}**	**0.386±3e^{-3}**	0.400	0.406	0.429	0.425	0.435	0.437	0.403	0.407	0.448	0.452	0.588	0.517	0.533	0.532
ETTm2	96	**0.178±2e^{-3}**	**0.256±2e^{-3}**	0.187	0.267	0.189	0.280	0.209	0.308	0.193	0.292	0.203	0.287	0.255	0.339	0.243	0.342
	192	**0.243±3e^{-3}**	**0.301±2e^{-3}**	0.249	0.309	0.253	0.319	0.311	0.382	0.284	0.362	0.269	0.328	0.281	0.340	0.392	0.448
	336	0.331±1e^{-3}	0.355±2e^{-3}	0.321	**0.351**	**0.314**	0.357	0.442	0.466	0.369	0.427	0.325	0.366	0.339	0.372	0.932	0.724
	720	0.434±3e^{-3}	0.413±2e^{-3}	**0.408**	**0.403**	0.414	0.413	0.675	0.587	0.554	0.522	0.421	0.415	0.433	0.432	1.372	0.879
	Avg	0.291±3e^{-3}	**0.331±2e^{-3}**	**0.291**	0.333	0.293	0.342	0.409	0.436	0.350	0.401	0.305	0.349	0.327	0.371	0.735	0.598
ETTh1	96	**0.370±9e^{-4}**	**0.390±1e^{-3}**	0.384	0.402	0.494	0.479	0.424	0.432	0.386	0.400	0.376	0.419	0.449	0.459	0.548	0.528
	192	0.423±1e^{-3}	**0.421±1e^{-3}**	0.436	0.429	0.538	0.504	0.475	0.462	0.437	0.432	**0.420**	0.448	0.500	0.482	0.542	0.526
	336	0.470±2e^{-3}	**0.442±2e^{-3}**	0.491	0.469	0.574	0.521	0.518	0.488	0.481	0.459	**0.459**	0.465	0.521	0.496	1.298	0.942
	720	**0.500±3e^{-3}**	**0.473±4e^{-3}**	0.521	0.500	0.562	0.535	0.547	0.533	0.519	0.516	0.506	0.507	0.514	0.512	0.721	0.659
	Avg	**0.440±3e^{-3}**	**0.432±3e^{-3}**	0.458	0.450	0.542	0.510	0.491	0.479	0.456	0.452	0.440	0.460	0.496	0.487	0.777	0.664
ETTh2	96	**0.290±4e^{-3}**	**0.335±3e^{-3}**	0.340	0.374	0.340	0.391	0.397	0.437	0.333	0.387	0.358	0.397	0.346	0.388	1.616	1.036
	192	**0.366±4e^{-3}**	**0.386±2e^{-3}**	0.402	0.414	0.430	0.439	0.520	0.504	0.477	0.476	0.429	0.439	0.456	0.452	2.083	1.197
	336	**0.423±2e^{-3}**	**0.428±2e^{-3}**	0.452	0.452	0.485	0.479	0.626	0.559	0.594	0.541	0.496	0.487	0.482	0.486	2.970	1.439
	720	**0.446±2e^{-3}**	**0.445±3e^{-3}**	0.462	0.468	0.500	0.497	0.863	0.672	0.831	0.657	0.463	0.474	0.515	0.511	2.576	1.363
	Avg	**0.381±4e^{-3}**	**0.399±3e^{-3}**	0.414	0.427	0.439	0.452	0.602	0.543	0.559	0.515	0.437	0.449	0.450	0.459	2.311	1.259
Electricity	96	**0.151±2e^{-4}**	**0.240±1e^{-3}**	0.168	0.272	0.187	0.304	0.207	0.307	0.197	0.282	0.193	0.308	0.201	0.317	0.300	0.392
	192	**0.163±1e^{-3}**	**0.250±1e^{-3}**	0.184	0.289	0.199	0.315	0.213	0.316	0.196	0.285	0.201	0.315	0.222	0.334	0.297	0.390
	336	**0.179±1e^{-3}**	**0.264±1e^{-3}**	0.198	0.300	0.212	0.329	0.230	0.333	0.209	0.301	0.214	0.329	0.231	0.3383	0.317	0.403
	720	**0.210±1e^{-3}**	**0.294±1e^{-3}**	0.220	0.320	0.233	0.345	0.265	0.360	0.245	0.333	0.246	0.355	0.254	0.361	0.338	0.417
	Avg	**0.176±1e^{-3}**	**0.294±1e^{-3}**	0.192	0.295	0.208	0.323	0.229	0.329	0.212	0.300	0.214	0.327	0.227	0.338	0.313	0.401
Traffic	96	**0.451±3e^{-3}**	**0.280±3e^{-3}**	0.593	0.321	0.607	0.392	0.615	0.391	0.650	0.396	0.587	0.366	0.613	0.388	0.798	0.436
	192	**0.458±2e^{-3}**	**0.277±2e^{-3}**	0.617	0.336	0.621	0.399	0.601	0.382	0.598	0.370	0.604	0.373	0.616	0.382	0.849	0.481
	336	**0.477±2e^{-3}**	**0.278±2e^{-3}**	0.629	0.336	0.622	0.396	0.613	0.386	0.605	0.373	0.621	0.383	0.622	0.337	0.828	0.476
	720	**0.520±3e^{-3}**	**0.288±2e^{-3}**	0.640	0.350	0.632	0.396	0.658	0.407	0.645	0.394	0.626	0.382	0.660	0.408	0.854	0.489
	Avg	**0.477±3e^{-3}**	**0.281±2e^{-3}**	0.620	0.336	0.621	0.396	0.622	0.392	0.625	0.383	0.610	0.376	0.628	0.379	0.832	0.471
Weather	96	**0.160±8e^{-4}**	**0.198±9e^{-4}**	0.172	0.220	0.197	0.281	0.182	0.242	0.196	0.255	0.217	0.296	0.266	0.336	0.174	0.252
	192	**0.203±9e^{-4}**	**0.239±1e^{-3}**	0.219	0.261	0.237	0.312	0.227	0.287	0.237	0.296	0.276	0.336	0.307	0.367	0.238	0.313
	336	**0.252±1e^{-3}**	**0.276±1e^{-3}**	0.280	0.306	0.298	0.353	0.282	0.334	0.283	0.335	0.339	0.380	0.359	0.395	0.287	0.355
	720	**0.326±1e^{-3}**	**0.328±1e^{-3}**	0.365	0.359	0.352	0.288	0.352	0.386	0.345	0.381	0.403	0.428	0.419	0.428	0.384	0.415
	Avg	**0.235±1e^{-3}**	**0.260±1e^{-3}**	0.259	0.287	0.271	0.334	0.261	0.312	0.265	0.317	0.309	0.360	0.338	0.382	0.271	0.334
Exchange	96	0.087±2e^{-4}	0.206±1e^{-4}	0.107	0.234	**0.085**	**0.204**	0.116	0.262	0.088	0.218	0.148	0.278	0.197	0.323	0.395	0.474
	192	**0.171±2e^{-4}**	**0.295±1e^{-4}**	0.226	0.344	0.182	0.303	0.215	0.359	0.176	0.315	0.271	0.380	0.300	0.369	0.776	0.698
	336	**0.285±1e^{-4}**	**0.389±2e^{-4}**	0.367	0.448	0.348	0.428	0.377	0.466	0.313	0.427	0.460	0.500	0.509	0.524	1.029	0.797
	720	**0.578±2e^{-4}**	**0.574±2e^{-4}**	0.964	0.746	1.025	0.774	0.831	0.699	0.839	0.695	1.195	0.841	1.447	0.941	2.283	1.222
	Avg	**0.280±2e^{-4}**	**0.366±2e^{-4}**	0.416	0.443	0.410	0.427	0.385	0.447	0.354	0.414	0.519	0.500	0.613	0.539	1.121	0.798
Improvement		—		14.8%	7.7%	19.1%	14.3%	24.3%	18.5%	19.4%	13.8%	20.8%	16.0%	29.2%	20.8%	62.3%	45.6%
1^{st} Count		**56**		3		3		0		0		2		0		0	

Table 1: Comparing U-Mixer with SOTA benchmarks on large-scale real-world time-series datasets in long-term forecasting.

Instance Normalization and Learning Objective

We flatten $\hat{Y_d}$ into $\hat{Y_p} \in \mathbb{R}^{C\times(L+H)}$ by a linear layer. Then we use instance normalization to mitigate the distribution shift effect between the input X and forecasting result. So the output of U-Mixer is $\hat{Y} \in \mathbb{R}^{C\times H} = \hat{Y_p}[:,\tau] \times \sigma_{in} + \mu_{in}$, $\tau = L : L + H$. We choose L_1 Loss function to measure the discrepancy between the prediction and the ground truth, and the loss is propagated back from the outputs across the entire model. Compared to the commonly used MSE loss function in time series forecasting tasks, L_1 loss function is less sensitive to outliers, which allows the model to achieve more robust performance. The loss of our model is:

$$\mathcal{L}_{U-Mixer} = \frac{1}{C}\sum_{i=1}^{C}|Y[i,:] - \hat{Y}[i,:]| \tag{9}$$

Experiments

Datasets

We evaluate U-Mixer mainly on six large-scale real-world time-series datasets in long-term forecasting. (1) **Electricity transformer temperature (ETT)** (Wu et al. 2021) data contains the power load features and oil temperature collected from electricity transformers, consisting of seven features. Following the same protocol as Informer (Zhou et al. 2021), we split the data into four datasets: ETTh1, ETTh2, ETTm1, and ETTm2. (2) **Electricity Consuming Load (ECL)** (Wu et al. 2021) data contains the hourly electricity consumption of 321 customers from 2012 to 2014. (3) **Traffic** (He et al. 2022) data is a collection of hourly data from California Department of Transportation and describes the occupancy rate of different lanes measured by different sensors on San Fran-

Models		**U-Mixer**	TimesNet	N-HiTS	N-BEATS	ETS.	LightTS	FED.	Stationary	Pyra.	In.	LogTrans	Re.	LSTM	TCN
Yearly	SMAPE	**13.317**$\pm$**2e**$^{-3}$	13.387	13.418	13.436	18.009	14.247	13.728	13.717	15.530	14.727	17.107	16.169	176.040	14.920
	MASE	3.006$\pm$1e^{-2}	**2.996**	3.045	3.043	4.487	3.109	3.048	3.078	3.711	3.418	4.177	3.800	31.033	3.364
	OWA	**0.786**$\pm$**2e**$^{-4}$	**0.786**	0.793	0.794	1.115	0.827	0.803	0.807	0.942	0.881	1.049	0.973	9.290	0.880
Quart.	SMAPE	**9.956**$\pm$**2e**$^{-4}$	10.100	10.202	10.124	13.376	11.364	10.792	10.958	15.449	11.360	13.207	13.313	172.808	11.122
	MASE	**1.156**$\pm$**3e**$^{-3}$	1.182	1.194	1.169	1.906	1.328	1.283	1.325	2.350	1.401	1.827	1.775	19.753	1.360
	OWA	**0.873**$\pm$**1e**$^{-4}$	0.890	0.899	0.886	1.302	1.000	0.958	0.981	1.558	1.027	1.266	1.252	15.049	1.001
Others	SMAPE	**4.858**$\pm$**3e**$^{-4}$	4.891	5.061	4.925	7.267	15.880	4.954	6.302	24.786	24.460	23.236	32.491	186.282	7.186
	MASE	**3.195**$\pm$**1e**$^{-2}$	3.302	3.216	3.391	5.240	11.434	3.264	4.064	18.581	20.960	16.288	33.355	119.294	4.677
	OWA	**1.015**$\pm$**1e**$^{-4}$	1.035	1.040	1.053	1.591	3.474	1.036	1.304	5.538	5.879	5.013	8.679	38.411	1.494
Avg.	SMAPE	**11.740**$\pm$**1e**$^{-3}$	11.829	11.927	11.851	14.718	13.525	12.840	12.780	16.987	14.086	16.018	18.200	160.031	13.961
	MASE	**1.575**$\pm$**1e**$^{-2}$	1.585	1.613	1.599	2.408	2.111	1.701	1.756	3.265	2.718	3.010	4.223	25.788	1.945
	OWA	**0.845**$\pm$**1e**$^{-4}$	0.851	0.861	0.855	1.172	1.051	0.918	0.930	1.480	1.230	1.378	1.775	12.642	1.023

Table 2: Comparing U-Mixer with SOTA benchmarks on M4 datasets in short-term forecasting.

cisco highway. (4) **Exchange** (Lai et al. 2018) data collects the panel data of daily exchange rates from 8 countries ranging from 1990 to 2016. (5) **Weather** (Nie et al. 2023) data is recorded 21 meteorological indicators collected every 10 minutes from the Weather Station of the Max Planck Biogeochemistry Institute in 2020. (6) **M4** (Wu et al. 2023) is a dataset for the short-term forecasting, which involve 6 subsets: M4-Yearly, M4-Quarterly, M4-Monthly, M4-Weakly, M4-Daily, and M4-Hourly. We divide all datasets into training, validation and testing sets according to the chronological order by the ratio of 6:2:2 for ETT dataset and 7:1:2 for the other datasets.

Model Configuration and Metrics

U-Mixer is implemented through Pytorch and trained on an Nvidia A40 GPU (48GB). The following model configuration is used by default for long-term forecasting: Input sequence length $L = 96$, patch length $P = 16$, stride $S = 8$, forecasting sequence length $H \in \{96, 192, 336, 720\}$, the number of levels $M = 3$, and batch size is set to 16. For short-term forecasting the model configuration is: patch length $P = 8$, stride $S = 4$, and batch size is set to 32. The model training runs 10 epochs, and optimization is performed by Adam. To enhance the reproducibility of the implemented results, we fix random seeds. We employ mean square error (MSE) and mean absolute error (MAE) for long-term forecasting evaluation. Following the N-BEATS (Oreshkin et al. 2019), we also employ the symmetric mean absolute percentage error (SMAPE), mean absolute scaled error (MASE) and overall weighted average (OWA) for short-term forecasting.

SOTA Benchmarks

We compare U-Mixer with the following 16 SOTA methods from five different categories: (1) **RNN-based models**: LSTM (Hochreiter and Schmidhuber 1997) and LSSL (Gu, Goel, and Ré 2022). (2) **MLP-based models**: LightTS (Zhang et al. 2022) and DLinear (Zeng et al. 2023). (3) **CNN-based models**: TCN (Bai, Kolter, and Koltun 2018) and TimesNet (Wu et al. 2023). (4) **Transformer-based models**: LogTrans (Li et al. 2019), Reformer (Kitaev, Kaiser, and Levskaya 2020), Informer (Zhou et al. 2021), Pyraformer (Liu et al. 2021), Autoformer (Wu et al. 2021), FEDformer (Zhou et al. 2022), Non-stationary Transformer (Liu et al. 2022b), and ETSformer (Woo et al. 2022). (5) **Decomposition-based models**: N-BEATS (Oreshkin et al. 2019) and N-HiTS (Challu et al. 2023).

Models		**U-Mixer**	w/o UE	w/o SC
ETTh2	MSE	**0.381**	0.405	0.400
	MAE	**0.399**	0.427	0.421
Weather	MSE	**0.235**	0.252	0.254
	MAE	**0.260**	0.285	0.287
M4	SMAPE	**11.740**	11.817	11.808
	MASE	**1.575**	1.582	1.587
	OWA	**0.845**	0.857	0.853

Table 3: Results of ablation study on ETTh2, Weather and M4 datasets.

Overall Comparison

Table 1 presents the long-term forecasting performance measured by MSE and MAE, and Table 2 shows the short-term forecasting performance measured by SMAPE, MASE, and OWA. We can observe that U-Mixer balances short- and long-term forecasing well and achieves the best short- and long-term forecasting performance. Out of the 64 long-term forecastings made on 8 datasets, we achieve the best results in 56 cases and brings 14.5%/7.7% improvements on MSE/-MAE to the existing best results. For short-term forecasting on the M4 dataset, we achieve nearly all optimal outcomes. Specially, we found that RNN-based methods exhibites significantly poorer performance. For long-term forecasting, there is not a substantial difference among MLP-based, CNN-based, and Transformer-based methods. Apart from our model, the Decomposition-based methods demonstrates a leading position in short-term prediction.

Ablation Study

To better evaluate the Unet encoder-decoder (UE) and stationarity correction (SC), we conduct supplementary experiments with ablation consideration. Here **w/o UE** and **w/o SC** are variants of U-Mixer. In w/o UE, we set the Unet encoder-decoder level $M = 0$. In w/o SC, we remove the

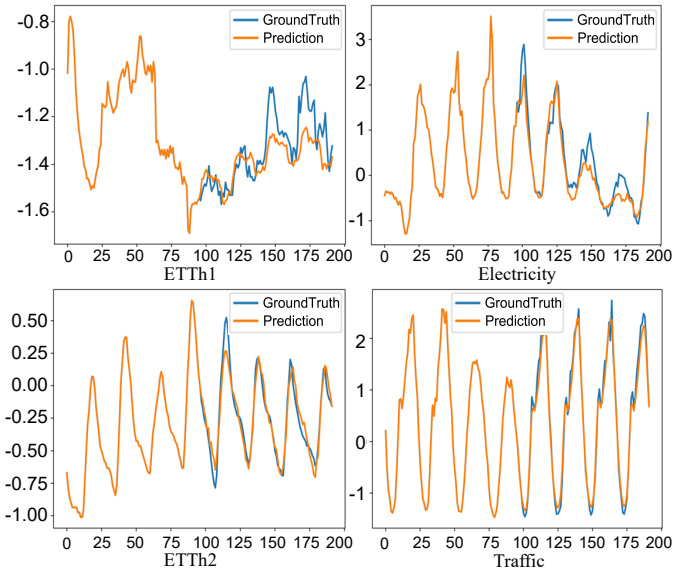


Figure 4: Visualization of forecasting results on multiple datasets by U-Mixer.

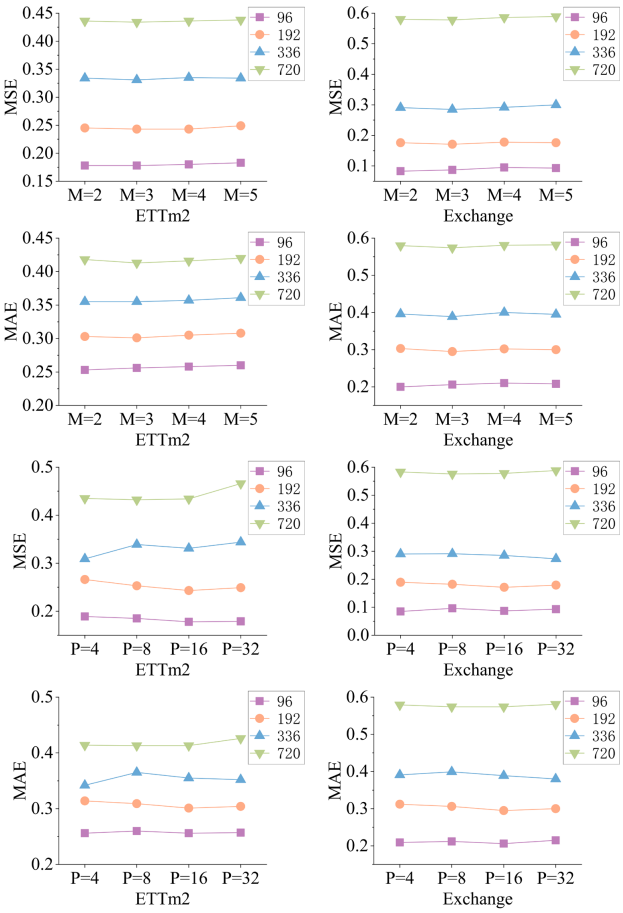


Figure 5: Performance comparison on varying.

stationarity correction process. We set the batch size as 16, input sequence length as 96, horizon in $\{96, 192, 336, 720\}$ and use the same parameter setting in all ablation experiments. The average results are shown in Table 3. All metrics of w/o UE and w/o SC are significantly worsed than U-Mixer. The complete U-Mixer can obtain the best results, demonstrating removing any components of it will affect the effect. The Unet encoder-decoder and stationarity correction are necessary and effective.

Showcases

To further show the forecasting performance of U-Mixer, we visualize the results on ETTh1, Electricity, ETTh2, and Traffic datasets. As shown in Figure 4, the ETTh2 and Traffic datasets exhibit more clear periodicity patterns, which our method can efficiently forecasting the ground truth. This demonstrates the strong capability of U-Mixer in capturing period. In contrast, the patterns of ETTh1 and Electricity are relatively less obvious. Our method manages to capture their periodicity to a reasonable extent, and also makes certain forecastings on their trend. It is verified the robustness of U-Mixer performance among various data characteristics.

Parameter Sensitivity

We choose the perform the sensitivity analysis of U-Mixer on ETTm2 and Exchange datasets by varying the number of level M and the patch length P. As shown in Figure 5, it can be concluded that our model is not highly sensitive to parameter M. Overall, the optimal results are achieved when M is set to 3. Additionally, with an increase in M, the training time of the model also increases. Therefore, we choose $M = 3$ as the parameter for our model. For the parameter P, we can tell our model exhibits a higher sensitivity to parameter P on the ETTm2 dataset, while the sensitivity is lower on the Exchange dataset. The value of P also influences the training time. When P is set to 8 or 16, the training time is relatively shorter, with the outcome of $P = 16$ surpassing that of $P = 8$. Consequently, we set $P = 16$ as the designated value.

Conclusion

In this paper, we study the non-stationarity challenge in time series forecasting task and propose a novel forecasting model U-Mixer. We combine the Unet and Mixer architecture to capture the local dependencies between patches and channels separately of different levels to obtain the comprehensive representations of time series. More importantly, we propose a stationarity correction method to handle the distribution shift caused by non-stationarity while preserving the temporal dependencies. Our model demonstrates state-of-the-art short- and long-term forecasting performance on six real-world datasets, and the overall superiority of various experiments further verifies the effectiveness and robustness of it.

Acknowledgements

Supported by the National Natural Science Foundation of China (NSFC) Joint Fund with Zhejiang Integration of Informatization and Industrialization under Key Project (Grant No.U22A2033) and NSFC (Grant No.62072281).

References

Bai, S.; Kolter, J. Z.; and Koltun, V. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Challu, C.; Olivares, K. G.; Oreshkin, B. N.; Ramirez, F. G.; Canseco, M. M.; and Dubrawski, A. 2023. NHITS: Neural Hierarchical Interpolation for Time Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 6989–6997.

Cohen, L. 1998. The generalization of the Wiener-Khinchin theorem. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 3, 1577–1580. IEEE.

Fukushima, K. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4): 193–202.

Gu, A.; Goel, K.; and Ré, C. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In *The International Conference on Learning Representations (ICLR)*.

He, H.; Zhang, Q.; Bai, S.; Yi, K.; and Niu, Z. 2022. CATN: Cross attentive tree-aware network for multivariate time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 4030–4038.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Kim, T.; Kim, J.; Tae, Y.; Park, C.; Choi, J.-H.; and Choo, J. 2021. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2020. Reformer: The Efficient Transformer. In *International Conference on Learning Representations*.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, 95–104.

Li, S.; Jin, X.; Xuan, Y.; Zhou, X.; Chen, W.; Wang, Y.-X.; and Yan, X. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32.

Liu, S.; Yu, H.; Liao, C.; Li, J.; Lin, W.; Liu, A. X.; and Dustdar, S. 2021. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*.

Liu, T.; Ma, X.; Li, S.; Li, X.; and Zhang, C. 2022a. A stock price prediction method based on meta-learning and variational mode decomposition. *Knowledge-Based Systems*, 252: 109324.

Liu, Y.; Wu, H.; Wang, J.; and Long, M. 2022b. Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting.

Ma, X.; Zhao, T.; Guo, Q.; Li, X.; and Zhang, C. 2022. Fuzzy hypergraph network for recommending top-K profitable stocks. *Information Sciences*, 613: 239–255.

Nie, Y.; H. Nguyen, N.; Sinthong, P.; and Kalagnanam, J. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*.

Oreshkin, B. N.; Carpov, D.; Chapados, N.; and Bengio, Y. 2019. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*.

Tolstikhin, I. O.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Woo, G.; Liu, C.; Sahoo, D.; Kumar, A.; and Hoi, S. 2022. Etsformer: Exponential smoothing transformers for time-series forecasting. *arXiv preprint arXiv:2202.01381*.

Wu, H.; Hu, T.; Liu, Y.; Zhou, H.; Wang, J.; and Long, M. 2023. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In *International Conference on Learning Representations*.

Wu, H.; Xu, J.; Wang, J.; and Long, M. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34: 22419–22430.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 11121–11128.

Zhang, F.; Guo, T.; and Wang, H. 2023. DFNet: Decomposition fusion model for long sequence time-series forecasting. *Knowledge-Based Systems*, 277: 110794.

Zhang, T.; Zhang, Y.; Cao, W.; Bian, J.; Yi, X.; Zheng, S.; and Li, J. 2022. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. *arXiv preprint arXiv:2207.01186*.

Zhao, T.; Ma, X.; Li, X.; and Zhang, C. 2023. MPR-Net:Multi-Scale Pattern Reproduction Guided Universality Time Series Interpretable Forecasting. arXiv:2307.06736.

Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; and Zhang, W. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 11106–11115.

Zhou, T.; Ma, Z.; Wen, Q.; Wang, X.; Sun, L.; and Jin, R. 2022. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, 27268–27286. PMLR.