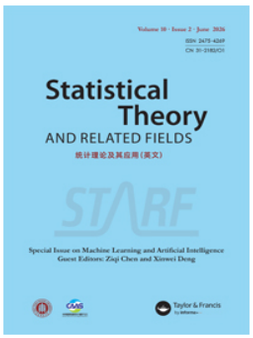


Statistical Theory and Related Fields

ISSN: 2475-4269 (Print) 2475-4277 (Online) Journal homepage: www.tandfonline.com/journals/tstf20

Addressing challenges in time series forecasting: a comprehensive comparison of machine learning techniques

Seyedeh Azadeh Fallah Mortezanejad & Ruochen Wang

To cite this article: Seyedeh Azadeh Fallah Mortezanejad & Ruochen Wang (2026) Addressing challenges in time series forecasting: a comprehensive comparison of machine learning techniques, Statistical Theory and Related Fields, 10:2, 184-231, DOI: 10.1080/24754269.2026.2633813

To link to this article: https://doi.org/10.1080/24754269.2026.2633813

Published online: 18 May 2026.

STATISTICAL THEORY AND RELATED FIELDS
2026, VOL. 10, NO. 2, 184–231
https://doi.org/10.1080/24754269.2026.2633813

Taylor & Francis
Taylor & Francis Group

Addressing challenges in time series forecasting: a comprehensive comparison of machine learning techniques

Seyedeh Azadeh Fallah Mortezanejad and Ruochen Wang

School of Automotive and Traffic Engineering, Jiangsu University, Zhenjiang, People's Republic of China

ABSTRACT

The explosion of time series (TS) data, driven by advancements in technology, necessitates sophisticated analytical methods. Modern management systems increasingly rely on analyzing this data, highlighting the importance of efficient processing techniques. State-of-the-art machine learning (ML) approaches for TS analysis and forecasting are becoming prevalent. In this paper, we provide an overview of suitable algorithms for TS regression tasks, comparing their performance with each other and with the traditional autoregressive integrated moving average (ARIMA) method across diverse datasets–including synthetic data, long-term recorded data, data containing outliers, and data with missing values. Our focus is on forecasting accuracy, especially for long-term predictions. A key strength of our work is the comprehensive collection of various ML methods tailored for TS data, along with their evaluation across different datasets that include various challenging scenarios. The results show that tree-based ensemble methods outperform other algorithms in most cases. This study aims to assist researchers and practitioners in selecting the most appropriate algorithm based on specific forecasting requirements and data characteristics.

ARTICLE HISTORY
Received 3 April 2025
Revised 2 December 2025
Accepted 12 February 2026

KEYWORDS
Time series (TS); machine learning (ML); regression task; outliers; missing data

1. Introduction

A time series (TS) is a collection of data points indexed in chronological order. The time intervals between these data points are typically consistent, such as hourly, daily, monthly, or yearly. Multivariate TS involves multiple TS variables measured over the same time period, where the variables are likely interrelated. While regular, evenly spaced multivariate TS are popular, there is also a type characterized by irregular time steps. Irregular multivariate TS consist of data points recorded at uneven intervals over time, involving multiple variables. Analyzing these series is more challenging due to the gaps and varying time gaps between observations. Their significance lies in accurately capturing real-world phenomena that do not follow a strict schedule–such as patient health records, financial transactions, or sensor data from unpredictable environments. For instance, trades happen at random times in finance, and understanding these patterns can aid better decision-making.

CONTACT Ruochen Wang wrc@ujs.edu.cn School of Automotive and Traffic Engineering, Jiangsu University, No. 301, Xuefu Road, Jingkou District, Zhenjiang City, Jiangsu Province 212013, People's Republic of China

Y. Chen et al. (2023) developed a model that extends transformer architecture to handle irregular TS by explicitly capturing continuous-time dynamics, and demonstrated its superior performance through experiments on synthetic and real-world datasets. W. Zhang et al. (2024) introduced a method that transforms irregular multivariate TS into aligned patches to better capture local and global correlations, and demonstrated its superior forecasting performance across diverse real-world datasets. While this work focuses on forecasting for univariate consistent time steps, applying ML to multivariate TS presents distinct challenges and opportunities, especially in integrating domain knowledge. In our complementary research (Mortezanejad et al., 2025), we addressed this by introducing methods to automatically discover governing equations from historical multivariate TS data and incorporate them into physics-informed neural networks (NNs) for probabilistic forecasting, demonstrating their effectiveness on a real-world energy management dataset.

The two most traditional TS datasets are weather data and stock prices. Hourly or daily measurements of temperature, rainfall, wind speed, and air humidity recorded at a weather station assist district management officials in making critical decisions regarding development, flood prevention, and understanding climate changes. For instance, the quality of seasonal agricultural products is directly affected by climatic conditions. Daily opening and closing prices of a particular stock are useful for shareholders to predict future prices, identify trading opportunities, and manage risks. Other examples of TS data include website traffic, sales data, energy consumption, sensor data, medical data, social media activity metrics, economic indicators, and more. Casado-Vara et al. (2021) developed a long short-term memory (LSTM)-based architecture to forecast web traffic TS using a newly created dataset, successfully extracting features and hidden patterns to achieve highly accurate predictions despite limited data and the unpredictable nature of human behavior. Pavlyshenko (2019) applied ML techniques, including stacking ensembles, to forecast sales data TS, demonstrating that combining models improved prediction accuracy, especially when dealing with complex patterns, noise, and limited historical data. Dash et al. (2020) developed a workflow to generate synthetic medical TS data that incorporated static patient covariates, demonstrating that the synthetic data closely resembled real data and maintained its utility for healthcare research. Each of these data points plays an important role in the management of various trades, industries, and managements, enhancing the quality of their services and products.

Torres et al. (2021) reviewed deep learning (DL) architectures, including feed-forward networks (FFNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs) such as Elman, LSTM, gated recurrent units (GRUs), and bidirection for TS forecasting. They discussed the advantages and limitations of these architectures, hyper-parameter tuning, suitable frameworks, and identified research gaps across various application domains.

Since the importance of TS data has grown, many libraries in MATLAB, Python, and other programming languages have been developed to implement state-of-the-art methods for analyzing TS data. Tavenard et al. (2020) introduced the tslearn library, a Python package designed for ML tasks such as clustering, classification, and regression specifically tailored for TS data. PyTS is an open-source Python package for TS classification, as established in Faouzi and Janati (2020). It provides a variety of algorithms, preprocessing tools, and is utilized for loading datasets, all built on standard scientific Python libraries, and is available on GitHub. Schirmer and Mporas (2024) explored TS modeling, emphasizing five key applications: denoising, forecasting, nonlinear transient modeling, outlier detection, and degradation modeling. The study categorizes modeling approaches into statistical, linear

algebra, and ML methods while reviewing relevant literature. A key contribution of the article is the introduction of PyDTS, a Python-based toolkit that integrates various TS modeling techniques, complete with practical examples and benchmark comparisons across diverse datasets.

There are several survey papers on TS forecasting using ML. For instance, Lim and Zohren (2021) discussed many different algorithms, but a drawback is the lack of implementation of example datasets, so they compare the networks analytically. Z. Liu et al. (2021) classified several existing TS forecasting methods, analyzed their strengths and weaknesses, compared different approaches, and outlined future research directions in data preprocessing, incremental modeling, and parallel computing. Chiarot and Claudio (2023) surveyed the state-of-the-art in TS compression for IoT and smart devices, proposed a comprehensive taxonomy, and evaluated how various methods perform under different data characteristics and constraints.

In this paper, we provide an overview of suitable algorithms for TS regression tasks, comparing their performance against each other and the traditional ARIMA method across diverse datasets–including long-term recorded data, data containing outliers, and data with missing values. Our focus is on forecasting accuracy, especially for long-term predictions. A key strength of our work is the comprehensive collection of various ML methods tailored for TS data, along with their evaluation across different datasets that include challenging scenarios. To ensure a thorough comparison, we systematically analyze the impact of data irregularities on model performance and identify which methods are more robust under such conditions. The results show that tree-based ensemble methods outperform other algorithms in most cases, highlighting their potential for accurate and reliable forecasting. Importantly, this work provides practical insights for selecting suitable algorithms based on specific data challenges and forecasting goals. By bridging the gap between theoretical development, simulation studies, and real-world application, our study offers valuable guidance for both researchers and practitioners seeking effective solutions for complex TS forecasting problems.

The contribution of this paper is outlined as follows. In Section 2, we provide a brief explanation of the TS concept and forecasting using ARIMA models. In Section 3, we discuss feature selection relevant to TS analysis. In Section 4, we supply a historical overview of ML algorithms for TS analysis that have emerged. In Section 5, we compile algorithms suitable for TS regression tasks. In Section 6, we examine algorithms appropriate for handling outliers. In Section 7, we address the issue of missing data and the corresponding algorithms. In Section 8, we introduce a systematically generated dataset and analyze the ML algorithm's performance. In Section 9, we present the results of three real case studies. Finally, in Section 10, we share the insights derived from this paper.

2. Traditional TS concepts

A fundamental concept in TS analysis is stationarity. A stationary TS exhibits consistent statistical properties, such as mean, variance, and autocorrelation, over time. The simplest method to check for stationarity is the TS plot. In this plot, several key features must be examined, including trends, seasonal components, and any malformed behaviors such as sharp points and outliers. For more information, let's get into some mathematical definitions. A TS model represents a specific joint distribution, or at the very least, the means and covariances of a randomly ordered chronological variable $\{X_t\}$, with observations denoted

as $\{x_t\}$. The mean, variance, and covariance functions for a TS $\{X_t\}$ with $E(X_t)^2 < \infty$, are defined as follows:

$$\begin{aligned}\mu_X(t) &= E(X_t),\\ \mathrm{Var}_X(t) &= E[(X_t - \mu_X(t))^2],\\ \mathrm{Cov}(X_t, X_s) &= E[(X_t - \mu_X(t))(X_s - \mu_X(s))].\end{aligned}$$

A TS $\{X_t\}$ is said to be strictly stationary if the joint distribution of $(X_{t_1}, \ldots, X_{t_k})$ is the same as the joint distribution of $(X_{t_1+h}, \ldots, X_{t_k+h})$ for all integers h and $k \geq 1$. Also, $\{X_t\}$ is considered as a weakly stationary TS if it satisfies the following conditions:

- $\mu_X(t)$ and $\mathrm{Var}_X(t)$ are independent of time *t;*
- $\mathrm{Cov}(X_{t+h}, X_t)$ is independent of t for every integer h.

In practice, the definition of weak stationarity is commonly used and referred to simply as stationarity. A TS is considered non-stationary if any of these conditions are violated. Common indicators of non-stationarity include the following.

- Trending: the mean of the series changes systematically over time;
- Seasonality: the series exhibits periodic fluctuations;
- Heteroskedasticity: the variance of the series changes over time.

In traditional modeling and forecasting methods, the first and most crucial step is to transform a non-stationary TS into a stationary one. The presence of a trend is indicated by a consistent increase or decrease in values over time. A key seasonality characteristic is the recurrence of specific patterns at regular intervals. Heteroskedasticity can be identified by visualizing variations in the frequency of patterns across different periods. Common techniques to address these issues include first-order differencing, defined as $Y_t = X_t - X_{t-1}$, which removes trends; seasonal differencing of lag h, represented as $Z_t = Y_t - Y_{t-h}$, where h corresponds to the fluctuation period; and log transformation to address heteroskedasticity. There are some other methods like seasonal-trend decomposition using locally estimated scatterplot smoothing (LOESS) that help finding the trends and seasonal components. STL–seasonal and trend decomposition using Loess–uses locally fitted regression models to decompose a TS into trend, seasonal, and the remaining components. Furthermore, there are several statistical tests, such as the augmented Dickey-Fuller, Kwiatkowski-Phillips-Schmidt-Shin, and Mann-Kendall tests, used to analyze unit roots and trends. These can be helpful for beginners in diagnosing non-stationarity in TS analysis.

There are several TS models to forecast a stationary TS such as moving average (MA), autoregressive (AR), ARIMA, seasonal autoregressive integrated moving average (SARIMA), generalized autoregressive conditional heteroskedasticity (GARCH), and vector autoregression (VAR). An MA(q) model represents a TS with a weighted average of current and past error terms like $X_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1} + \cdots + \theta_q\epsilon_{t-q}$, where random error terms ϵ_ts are independent and identically distributed (*i.i.d.*) and θ_is are the model parameters. An AR(p) models a TS as a linear combination of its past values and an error term $X_t = \phi_0 + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \epsilon_t$, where ϕ_is are the model parameters. An ARIMA(p, d, q) model combines

AR and MA components and incorporates differencing d. The value of d in ARIMA represents the number of differences needed to achieve stationarity. A SARIMA$(p, d, q)(P, D, Q)$ model extends ARIMA to account for seasonality with period s. The capital letters (P, D, Q) represent the seasonal AR and MA components. A GARCH model captures TS data with changing volatility by modeling the variance of the current error term based on past errors and past variances. It is often used for financial data where volatility clustering occurs. A VAR model extends the autoregressive approach to multiple TS, modeling each variable as a linear combination of past values of itself and other variables in the system. It is useful for understanding interdependencies among multiple related TS. Only the ARIMA model from the traditional methods is applied in this paper, as our primary focus is on exploring modern techniques for TS analysis. We intentionally choose to limit the scope and do not include other classical models.

Choosing the appropriate model necessitates careful consideration of the data's characteristics, including trends, seasonality, and the autocorrelation function (ACF) as well as the partial autocorrelation function (PACF). Generally, the values of q and p can be approximately determined based on the number of significant ACF and PACF lags, respectively. Model selection generally involves conducting diagnostic checks to ensure a proper fit and to evaluate the model's residuals for stationarity. If stationarity is not attained through preprocessing, directly applying these models may result in inaccurate or unreliable forecasts (Brockwell & Davis, 2002). The traditional models perform effectively with data that exhibits significant values in the ACF and PACF only for the most recent lags. Consequently, these models may struggle to manage data with large values at higher lags. Additionally, TS data with persistent patterns are intractable for making predictions using the classical models.

3. Feature extraction for TS analysis

ML approaches offer significant practical advantages over traditional TS methods, particularly regarding preprocessing requirements and modeling flexibility. Classical models like ARIMA demand strict stationarity, necessitating extensive transformations that grow complex with multivariate data. In contrast, ML algorithms exhibit inherent robustness to many common data irregularities.

This shifts the preprocessing focus from enforcing statistical assumptions to feature engineering and data quality refinement. The appropriate strategy depends on the data's underlying mechanics and the chosen model's characteristics. Ultimately, ML's adaptability reduces the need for rigid, algorithm-specific preprocessing, allowing greater emphasis on informative feature creation and model selection. These models also excel at capturing intricate nonlinear patterns and seamlessly incorporating external variables, and they can be updated continuously as patterns evolve.

However, applying ML algorithms to TS data requires careful feature engineering. Raw TS data are often unsuitable as direct input to most ML algorithms. Feature engineering transforms the raw data into a format more amenable to ML model. This process involves creating new features from the existing data that might improve model performance.

A common approach for preparing TS data for ML models is to create lagged features. This involves creating input vectors comprising previous time steps' values. For example, if we want to predict the value at time t, and use a lag of h, the input and output are $(X_{t-h}, \ldots, X_{t-1})$ and X_t, respectively. Another method involves calculating rolling statistics, including rolling means, standard deviations, minimums, maximums, and medians over a specified window

size (e.g., the rolling mean over the past h days). These features effectively capture trends and variations within the data. Additionally, there are several other features that can be selected for a specific problem, such as time-based features, seasonal features, trend indicator, and external predictor variables. Feature selection techniques assist in identifying the features that have a significant impact on the output. Additionally, data normalization can enable the machine to learn more quickly than it can if the input is provided in its original scale. Splitting the data into three sets–raining, validation, and testing–facilitates a more effective evaluation of the machine's performance (Joseph, 2022).

4. Historical development of ML in TS forecasting

The application of ML to TS forecasting has progressed through several distinct phases, each building on prior advances while addressing earlier limitations. It began with traditional statistical methods such as ARIMA and exponential smoothing, which provided a solid mathematical foundation but struggled to capture complex nonlinear relationships and to handle high-dimensional data.

The 1980s and 1990s saw the first major integration of ML with TS, marked by the emergence of early NNs. The Elman RNN, in Bengio et al. (1970), introduced memory through recurrent connections, enabling the modeling of temporal dependencies. However, these early RNNs suffered from vanishing and exploding gradient problems, which limited their effectiveness for long-range forecasting.

A breakthrough followed in the late 1990s with the invention of LSTM networks by Hochreiter and Schmidhuber (1997), and later the GRU in Chung et al. (2014). These architectures address the gradient problem with gating mechanisms, transforming sequence modeling and enabling more reliable learning of long-range dependencies. During this period, tree-based ensemble methods also rose to prominence, including random forest (RF, Breiman, 2001), and later extreme gradient boosting (XGBoost, T. Chen, 2015) and light gradient-boosting (LightGBM, Ke et al., 2017), which offer powerful nonlinear modeling with improved interpretability.

The 2010s brought further specialization for TS. Facebook's Prophet (Taylor & Benjamin, 2017) combined decomposable TS modeling with nonlinear trends, while wavelet-based transformation (WBT) provided multi-resolution analysis. This era also saw advances in attention-based architectures with temporal fusion transformer (TFT, Lim et al., 2019) and the development of interpretable forecasting models such as neural basis expansion methods (N-BEATS Oreshkin et al., 2019 and N-HiTS Cristian et al., 2022).

Concurrently, decomposition techniques evolved from classical methods like singular value decomposition (SVD) to more adaptive approaches such as empirical mode decomposition (EMD) and its Ensemble variant EEMD, which are better suited to non-stationary and nonlinear signals without requiring predetermined basis functions.

More recently, the field has seen a broad surge of DL architectures. Deep feed-forward neural network (DFFNN) laid foundational nonlinear modeling capabilities, while CNNs and TCNs offer parallelizable alternatives to recurrent models. Bayesian neural network (BNN) introduces uncertainty quantification, and a suite of advanced techniques–including short-time Fourier transformation (STFT), reinforcement learning (RL), graph neural network (GNN), and, most recently, large language models (LLM)–are increasingly tackling forecasting scenarios with multiple frequencies, spatial-temporal relationships, and external factors.

5. ML algorithms for sequential data

In this section, we aim to comprehensively collect the ML algorithms that are suitable for TS forecasting and provide a brief explanation of each. A concise overview is provided by highlighting its strengths, weaknesses, and suitability for different types of TS data. The algorithms are categorized for ease of understanding. The hierarchical organization groups algorithms based on their core characteristics and how they model TS data. We start with foundational models RNNs, which are specifically designed for sequence data, and then move to traditional ML methods like tree-based ensembles that often require manual feature extraction. Next, we include specialized TS models that incorporate advanced techniques for forecasting, followed by other NN architectures that offer versatile pattern recognition. Finally, we present advanced hybrid and signal processing methods. This structure helps readers understand the landscape of algorithms from basic to state-of-the-art, making it easier to see their relationships and differences.

5.1. Recurrent neural networks (RNNs)

RNNs are a subset of NNs specifically designed for processing sequential or temporal data, enabling them to effectively capture nonlinear short-term dependencies effectively. RNNs are widely used in processing text, speech, and TS data because they can handle information across multiple time steps. RNN neurons transmit feedback signals to one another, allowing them to maintain a kind of memory for past inputs. RNN architectures are distinguished by its cyclical connections, which allow information to persist across time steps. RNNs incorporate loops that feed the output of a layer back into itself, which creates a form of memory. This enables the network to take past inputs into account when processing current inputs. However, standard RNN architectures suffer from the vanishing gradient problem, limiting their ability to learn long-term dependencies (Tsantekidis et al., 2022). To mitigate this, several variations have been developed.

5.1.1. Elman RNN

Elman (Vanilla) RNN is the simplest type of RNN with a context layer that stores a copy of the previous hidden state. While effective for shorter sequences, they still struggle with very long-term dependencies. Ab Aziz et al. (2021) explored the use of advanced NNs for classification tasks, specifically integrating an improved Elman RNN with a particle swarm optimization algorithm. They developed a new hybrid RNN model aimed at enhancing learning speed and accuracy, demonstrating that this approach generally outperformed traditional NN models, including those using standard back-propagation.

5.1.2. Long short-term memory (LSTM)

LSTM networks are more complex than standard RNNs but are significantly better at capturing long-range dependencies. LSTMs address the vanishing gradient problem through a sophisticated cell state mechanism (Noh, 2021). This allows them to learn long-range dependencies more effectively than standard RNN or Elman networks. The gating mechanisms, including input, forget, and output gates, regulate the flow of information into and out of the cell state. They use sigmoid activations to produce values between 0 and 1, controlling the flow of information. Lindemann et al. (2021) published a survey about different types of RNN, particularly focusing on LSTM for TS. LSTM networks are designed for sequential

data, such as TS, but it is not efficient in extracting spatial relationships. In this context, a CNN is preferable. We discuss about CNN later in this section.

5.1.3. Gated recurrent unit (GRU)

GRUs are a simplified version of LSTM networks. The forget and input gates are combined into a single update gate and the cell state is merged with the hidden state. This design reduces the number of parameters, making GRUs computationally less expensive and often easier to train than LSTMs, while still delivering strong performance across various applications. Mateus et al. (2021) compared the performance of LSTM networks and GRU in a case study where traditional models such as ARIMA and SARIMA struggled to capture the stochastic nature of the data. As anticipated, the performance of the GRU outperformed that of the LSTM in their analysis.

5.1.4. Bidirectional RNN

Bidirectional RNNs process input sequences in positive and negative time directions known as forward-backward states. By combining the information from both directions, bidirectional RNNs can capture contextual information from both the past and the future, leading to improved performance in tasks where understanding the entire sequence is crucial. This is particularly beneficial for TS forecasting, where knowledge of future data points within a limited window can enhance predictions. The networks can be implemented using two RNN layers, including LSTMs and GRUs. The concept behind bidirectional RNNs is to separate the state neurons into forward-backward states, with the inputs of each component remaining unconnected to the other.

These models can process data in both forward-backward sequential directions, increasing their ability to understand and interpret textual information more effectively than traditional RNNs (Hindarto, 2023). The model's architecture exhibits symmetry in its directional flow, which is identified as a drawback in textual problems. This limitation arises because word positions are determined by grammar rather than by temporal context. Another issue with bidirectional RNNs is the concatenation of the forward-backward states without any specific considerations (Rozenberg et al., 2021). Zhao et al. (2020) introduced an enhanced hybrid algorithm that combines bidirectional LSTM networks and CNNs to effectively process sequential and spatial data.

5.1.5. Deep RNN

Instead of using a single or two recurrent layers, deep RNNs stack multiple recurrent layers on top of each other. This allows the network to learn hierarchical representations of the input sequence, capturing both low-level and high-level features. Deep RNNs can be more powerful than shallow RNNs, particularly for complex TS with intricate patterns. However, they are also more computationally expensive and require more data for effective training. There are several papers that have worked on deep RNNs for different tasks. For instance, Ahn and Park (2021) developed deep RNNs for short-term forecasting of fluctuating photovoltaic power output. The model achieved high accuracy for forecasts made 5 and 15 min in advance. However, the accuracy decreased slightly for longer forecasts ranging from 1 to 3 h. The deep RNN-based forecasting algorithm demonstrated superior accuracy in predicting short-term photovoltaic power generation.

5.1.6. Deep autoregressive RNN (DeepAR)

DeepAR (Salinas et al., 2020) is a probabilistic forecasting model developed by Amazon. It is designed for TS forecasting, particularly when dealing with many independent TS. DeepAR leverages LSTMs, to capture temporal dependencies within each TS. Crucially, it is a probabilistic model, meaning it produces not just a point forecast but also a probability distribution over future values. This allows for quantifying the uncertainty in the forecast (Salinas et al., 2020).

DeepAR uses an autoregressive RNN architecture. This means it predicts future values based on past values of the same TS. The LSTM processes the historical data, and the model learns to map past observations to a probability distribution over future values. A significant feature is its ability to handle many TS simultaneously by learning shared patterns across them. This is achieved by using an embedding layer that represents each TS as a vector.

Like many other algorithms, it involves costly computations and requires tuning of hyperparameters for improved performance. It struggles with very long-range and noisy data, similar to LSTM, and presents challenges in interpretation.

5.2. Tree-based ensemble methods

Tree-based ensemble methods have become increasingly popular for TS forecasting due to their ability to handle nonlinear relationships, automatically capture feature interactions, and provide robust performance across diverse datasets (Mir et al., 2022; Rady et al., 2021). These methods build multiple decision trees and combine their predictions to improve accuracy and reduce over-fitting.

The strength of these models is in their ability to capture complex patterns commonly found in temporal data, such as seasonality, trends, and cyclical behavior. Common algorithms used in this context include RF, XGBoost, and LightGBM. For regression tasks in TS forecasting, the output of the ensemble is a continuous value that represents the predicted future value of the TS. Feature engineering such as lagged values, rolling statistics, and external regressors, plays a crucial role in enhancing forecast accuracy.

5.2.1. Random forest (RF)

RF, illustrated in Breiman (2001), is an ensemble learning method used for both classification and regression tasks. RF constructs multiple decision trees during training to enhance accuracy and control over-fitting. Its outputs represent the prediction mode and mean of the individual trees for classification and regression tasks, respectively. A random subset of features is selected when splitting nodes to further reduce correlation among trees and enhance model robustness. RF is a rapid algorithm that effectively manages noise and outliers, outperforming many other algorithms. Additionally, it does not require data scaling. By increasing the number of trees, over-fitting may occur, and inadequate features can result in suboptimal performance.

Paul et al. (2018) developed an improved RF classifier that automatically selected the optimal number of trees and important features by iteratively removing unimportant features and adding trees based on a theoretical upper bound. This approach achieved high classification accuracy with fewer trees and features, and was demonstrated to be effective across various datasets, including biomedical and industrial applications.

5.2.2. Extreme gradient boosting (XGBoost)

XGBoost, explained in T. Chen (2015), is another highly popular gradient boosting library known for its performance and robustness. It builds an ensemble of decision trees sequentially, with each new tree learning from the mistakes of the previous ones. Each tree is built by splitting nodes to maximize a gain function. This function considers both the improvement in accuracy and $L1$ and $L2$ regularization to avoid over-fitting. XGBoost cleverly handles missing data and uses parallel processing for faster tree construction. However, it is computationally expensive and memory-intensive, and requires careful tuning of its hyper-parameters.

There are several applications of XGBoost. For example, C. Zhang et al. (2021) proposed a cause-aware failure detection scheme for optical transport network boards using the interpretable XGBoost algorithm with both balanced and unbalanced datasets. Sheridan et al. (2016) compared XGBoost algorithm to RF and deep NNs for making predictions in the pharmaceutical industry. They found that, with a standard set of parameters, XGBoost was faster and generally produced more accurate results than RF, closely rivaling deep NNs.

5.2.3. Light gradient-boosting (LightGBM)

LightGBM is a gradient boosting framework that stands out due to its speed and efficiency, particularly for dealing with large TS datasets. It achieves this through several key optimizations in its algorithm structure. LightGBM is the optimized version of XGBoost algorithm. The training data is partitioned into several bins to approximate the data distribution by histograms. This significantly reduces the memory usage and speeds up the training process. Each leaf node stores a histogram containing the sum of gradients and the sum of Hessians for each data point.

Trees are constructed iteratively. At each iteration, LightGBM evaluates the gain for each possible split for each leaf and selects the leaf with the largest gain at each step, leading to deeper trees with potentially higher accuracy. However, this can also lead to over-fitting if not carefully controlled. To speed up the training process and reduce the impact of data instances with smaller gradients, gradient-based one-side sampling (GOSS) technique randomly samples a subset of data with larger gradients and keeps all data with smaller gradients. This focuses computational effort on the more informative data points. To reduce the dimensionality of the data, exclusive features with minimal overlap are gathered. LightGBM incorporates $L1$ and $L2$ regularization to prevent over-fitting. LightGBM requires careful hyper-parameter tuning to achieve optimal performance.

So many papers used this algorithm for different TS problems. Cao et al. (2023) proposed a LightGBM tree to predict greenhouse temperature TS. LightGBM significantly outperformed other models, such as support vector machine, radial basis function NN, back-propagation NN, and multiple linear regression model, in terms of MSE and R-squared.

5.3. Specialized TS models

This subsection explores specialized TS models that offer distinct advantages over general-purpose algorithms. These advantages include effectively managing strong seasonality and trend components, handling missing data, incorporating multiple exogenous variables, and accounting for temporal dependencies.

5.3.1. Facebook prophet

Facebook Prophet is a TS forecasting tool designed for business applications, particularly those with strong seasonality and trend components, as it was used in Daraghmeh et al. (2021). It is known for its ease of use and ability to handle missing data and outliers. It decomposes the TS into three components: trends, seasonalities, and vacation irregularities. The trend component represents long-term developments, while the seasonal component accounts for periodic variations.

A nonlinear function captures the overall direction of the TS trends. The Prophet utilizes the Fourier series for specified periods, such as daily, weekly, or yearly, to create an effective model that incorporates periodic effects. Prophet is an effective tool for TS analysis, accommodating multiple levels of seasonality, which makes it suitable for a variety of data types. The model can automatically identify and learn these patterns. The effects of holidays and other special events are presented as a list of dates. Additionally, it is equipped with external regressors that can impact the TS. Tuning certain hyper-parameters can help achieve optimal solutions. Furthermore, Prophet is intended for long-term TS analysis, rendering it ineffective for short-term predictions.

5.3.2. Wavelet-based transformation (WBT)

Wavelets are mathematical functions that can transform data into different scales or resolutions, and they are commonly used for TS decomposition. Unlike traditional Fourier transforms, which decompose signals into sine and cosine functions, wavelets can capture both frequency and location information, making them particularly useful for non-stationary TS. TS data can be decomposed into various components, such as trends, seasonality, and noise, using WBTs. The coefficients obtained from WBTs can serve as features for ML models.

However, wavelet analysis can be mathematically complex, which may make it challenging to extract useful features efficiently. The performance of WBTs heavily depends on the choice of wavelet function (e.g., Haar, Daubechies, Coiflets). Additionally, wavelet features can lead to over-fitting, particularly if model complexity is not managed appropriately. Many other factors must be considered when using WBTs, such as parameter tuning and potential information loss during decomposition. While WBTs are effective for non-stationary data, they may struggle with very complex non-stationarity patterns and are susceptible to boundary effects, which can impact forecasting accuracy.

5.3.3. Temporal fusion transformer (TFT)

TFT (Lim et al., 2021) is a DL model designed for multi-horizon forecasting TS data by selecting features that exhibit strong interactions with the target variable. Its architecture incorporates LSTM networks to effectively capture long-term dependencies. Additionally, TFT combines the advantages of attention mechanisms with a thoughtfully constructed architecture that explicitly addresses various aspects of TS data, including known future inputs and static covariates. The attention mechanism enhances interpretability by highlighting which past time steps and features are most influential in making predictions. As a result, it provides greater insight compared to many black-box methods. Also, gated residual networks in TFT filter out unrelated algorithm nodes to enhance implementation speed.

TFT can handle various types of features, including numerical and categorical data, to gain deeper insights into the TS. Besides all these effective performances, it has high computational costs due to its complex architecture. Additionally, the decision-making process can be quite challenging. López Santos et al. (2022) explained TFT and compared its results with

those of simple ARIMA and other NNs, such as LSTM, multilayer perceptron, and XGBoost, for day-ahead photovoltaic power forecasting.

5.3.4. Neural basis expansion analysis for TS (N-BEATS)

N-BEATS is designed for univariate TS and features a dense architecture incorporating forward-backward residual links. It can be applied to a wide range of data structures without requiring any modifications to the architecture. Each fully connected block learns a distinct component of TS data, such as a specific pattern or trend. These blocks are referred to as basis functions, and their outputs are aggregated to generate the final forecast. Despite the dense and deep layers, N-BEATS offers fast performance that is user-friendly and achieves high accuracy in predictions. Although it is designed for TS purposes, it can be effectively applied to non-TS data (Oreshkin et al., 2019).

The decomposition into basis functions offers a degree of interpretability, enabling an understanding of which components contribute most significantly to the forecast. The fully connected nature makes it relatively straightforward to scale to longer TS and larger datasets. Compared to RNNs and LSTMs, this architecture is simpler to implement and train. It may have difficulty handling very complex or irregular seasonal patterns when compared to specialized models that are specifically designed for these scenarios. Fully understanding the learned basis functions can be challenging, particularly when dealing with a large number of blocks. Like many DL models, the performance is sensitive to hyper-parameter tuning. Finding the optimal configuration may necessitate considerable experimentation. While some extensions exist, the fundamental N-BEATS architecture does not directly incorporate external regressors or additional features that influence the TS.

5.3.5. Neural hierarchical interpolation for TS (N-HiTS)

N-HiTS (Challu et al., 2023) is another DL model designed for TS forecasting, particularly effective for handling long-term dependencies and high-frequency data. Unlike N-BEATS which uses a stack of fully connected blocks, N-HiTS utilizes a hierarchical structure. This structure decomposes the forecasting problem into multiple levels, each responsible for predicting a specific temporal resolution or frequency. Jeffrey et al. (2023) applied N-HiTS to forecast stock prices using multivariate TS analysis and demonstrated the effectiveness of N-HiTS for short-, medium-, and long-term stock price forecasting.

The decomposition into various levels facilitates the efficient processing of high-frequency TS, which can be computationally intensive for other models. It naturally offers forecasts at various resolutions, enabling both coarse-grained long-term predictions and fine-grained short-term forecasts. In scenarios involving complex long-term dependencies and multi-resolution requirements, N-HiTS can outperform simpler methods. The hierarchical structure adds more complexity than simpler models, such as N-BEATS, resulting in longer training times. Comprehending the internal mechanisms of each level poses challenges. As a result of increased complexity, there is a larger hyper-parameter space that requires more careful tuning to achieve optimal performance. Optimal performance often necessitates a significant amount of data to effectively train the model at each hierarchical level.

5.3.6. Decomposition techniques

Decomposition methods enhance TS forecasting by decomposing complex signals into interpretable components, thereby improving pattern recognition and predictive accuracy. SVD

uses matrix factorization to decompose a TS matrix into orthogonal components that capture the dominant patterns. Its strengths include noise reduction and effective dimensionality reduction. However, SVD assumes linearity and approximate stationarity, which can limit its performance on non-linear or highly non-stationary data. EMD adaptively decomposes non-stationary signals into Intrinsic Mode Functions (IMFs) through a data-driven sifting process. While well suited to non-linear data, EMD can suffer from mode mixing and endpoint effects, which may distort the extracted components. EEMD mitigates EMD's limitations by performing multiple noise-assisted decompositions and averaging the results. This ensemble approach reduces mode mixing and enhances component stability at the cost of increased computational complexity. Collectively, these decomposition techniques provide valuable preprocessing for TS analysis, yielding enriched feature representations that can improve downstream ML or forecasting models.

Jang et al. (2018) introduced a new version called Zoom-SVD, presented as a fast, memory-efficient method for uncovering latent patterns in multiple TS over an arbitrary time range. It compressed data by block during storage and stitched the compressed SVD results during querying, achieving up to about 15× faster performance and 15× less space usage than existing methods. Maiti et al. (2024) applied EEMD to river flow data to generate intrinsic mode functions, then built and compared multiple DL and statistical models on these decomposed datasets, evaluating performance with cross-validation and missing-data scenarios. They found that, while DL models on the original data performed robustly, incorporating EEMD-processed components did not consistently improve predictions, and DL methods generally outperformed Prophet and SARIMA across the experiments. In this paper, we apply decomposition methods to the dataset to extract components, and then implement a RF model for training and prediction.

5.4. Other NN architectures for TS

In this subsection, we explore various famous algorithms that are utilized for a range of problems, including their application to TS forecasting tasks. This requires a deep understanding of TS data and the relative features affecting future outcomes. Therefore, the features must be carefully selected for inputs.

5.4.1. Deep feed-forward neural network (DFFNN)

FFNs are well-known alongside ML tools, which process data in a single pass. DFFNNs for TS forecasting provide a straightforward approach by utilizing multiple layers of fully connected neurons, making them relatively easy to implement. Unlike RNNs or specialized architectures such as N-HITS, it does not inherently possess mechanisms for managing long-range dependencies or multi-resolution data. Instead, it relies on the representational power of deep layers to learn complex patterns from the input data. For TS analysis, this often involves preparing the input data as a sequence of lagged values and rolling statistics.

DFFNNs are relatively simple to implement and comparable to more complex architectures. Although the computational cost increases, they can be scaled to manage large datasets and longer TS. DFFNNs can be adapted for various forecasting tasks by modifying the input preparation and output layers. Standard DFFNNs often struggle to capture long-range dependencies. Information from distant past time steps can be lost or diluted as it propagates through the network. DFFNNs do not inherently provide multi-resolution forecasts; so obtaining predictions at different frequencies requires additional processing. This requires

the use of separate models or post-processing techniques. The performance is highly dependent on the preparation of the input data. With numerous layers and parameters, DFFNNs are susceptible to over-fitting if not properly regularized. Other drawbacks of traditional training algorithms, such as back-propagation, include slow convergence speed and a tendency to fall into local minima, which can diminish the performance of the classifier or regressor (Ab Aziz et al., 2021).

5.4.2. Bayesian neural network (BNN)

Bayesian ML algorithms are especially effective in TS forecasting because they can integrate prior beliefs and handle uncertainty in both parameters and predictions. This makes them well-suited for dynamic and noisy environments, such as those found in TS data. Bayesian methods can effectively model complex relationships and incorporate hierarchical structures. They tend to be more resilient to over-fitting, particularly when regularization is applied through prior distributions. Common Bayesian methods for TS forecasting include Bayesian linear regression, Gaussian processes, Bayesian structural TS, Bayesian hierarchical models, dynamic linear models, and BNNs.

Bayesian inference can be computationally intensive and may have slow implementations, especially when dealing with large datasets or complex models. Setting up Bayesian models can be more complex than traditional methods, as they require a deeper understanding of the underlying probabilistic frameworks. The selection of prior distributions can significantly impact the results. Inappropriately chosen priors may lead to biased predictions. Some Bayesian methods struggle to scale effectively with large datasets, which makes them less suitable for very high-dimensional or large-scale TS data.

In this paper, we focus on BNNs. In BNNs, rather than optimizing a single set of weights, distributions over weights are inferred. This approach enables the model to express uncertainty regarding its predictions. Consequently, BNNs provide a posterior distribution over weights, allowing the model to adapt based on the data it encounters.

5.4.3. Convolutional neural network (CNN)

CNNs are primarily recognized for their effectiveness in image processing, particularly when spatial relationships exist between neighboring points. Consequently, they are applied to TS forecasting. In this context, the TS data is treated as a one-dimensional image, with each data point representing a pixel. The convolutional filters traverse the TS, identifying local patterns and features.

CNNs are exceptional at identifying local patterns and features within TS data, which can be crucial for short-term forecasting. Compared to fully connected networks of similar capacity, CNNs typically require fewer parameters. This reduction minimizes the risk of over-fitting and accelerates the training process. Many well-established libraries offer efficient implementations of CNNs, making them user-friendly. Unlike some recurrent models, CNNs can be adapted to handle TS data of varying lengths by utilizing padding techniques.

5.4.4. Temporal convolutional network (TCN)

TCNs are a specialized type of CNN designed explicitly for sequential data, such as TS. They address some of the limitations of standard CNNs in handling long-range dependencies by employing techniques like dilated convolutions and residual connections. Dilated convolutions allow TCNs to have a large receptive field without an excessive number of parameters

or computational costs, enabling them to capture long-range dependencies more effectively than standard CNNs.

Unlike RNNs, TCNs can be parallelized during training and inference, leading to significantly faster processing times, especially for long sequences. The residual connections help stabilize the training process, mitigating the vanishing gradient problem often encountered in deep networks. Similar to CNNs, TCNs can handle variable-length sequences with appropriate padding. The optimal configuration of dilated convolution rates and filter sizes can significantly impact performance and require careful tuning. Although more efficient than RNNs, processing extremely long sequences can still be computationally demanding, especially with a large number of layers and wide filters. In certain scenarios, other specialized architectures like transformers or N-HITS might still outperform TCNs, particularly for very complex TS.

5.5. Advanced techniques

In addition to all suitable algorithms for TS forecasting, including those specifically designed for this purpose, we include a subsection on four additional models that are recognized as advanced methods in ML for various types of data. Here, our focus is on the aspects related to TS analysis.

5.5.1. Short-time Fourier transformation (STFT)

STFT is a method for analyzing the frequency content of non-stationary signals or TS. It divides data into shorter overlapping segments and applies the Fourier transform to each one, allowing for the examination of how frequency components change over time. The Fourier transform converts the time-domain signal into the frequency domain to extract features from the resulting spectrogram. These invisible features can be utilized in other forecasting models, such as LSTM and GRU. This makes STFT particularly useful in TS forecasting with varying patterns. Additionally, STFT can help filter out noise by isolating certain frequency components.

Low-frequency signals capture long-term trends rather than short-term fluctuations. The choice of window size and overlap significantly affects results, and often requires experimentation. The STFT can be computationally intensive, especially with large datasets and small window sizes. While shorter windows improve time resolution, they may reduce frequency resolution, and vice versa. Furthermore, the extracted features can sometimes lack interpretability, and STFT may struggle to capture complex non-linear relationships in the data without advanced ML techniques.

5.5.2. Reinforcement learning (RL) for TS

RL offers a unique approach to TS forecasting, framing the problem as a sequential decision-making task. Instead of directly predicting future values, an RL agent learns a policy to optimize a reward function that reflects the accuracy of its predictions or some other desired objective. An autonomous agent analyzes its environment to gather information about its current state and take appropriate actions. In contrast, the environment offers a reward signal, which can be either positive or negative. The agent aims to maximize the expected cumulative reward signal throughout the interaction (Ernst & Louette, 2024). An agent's policy dictates how it decides to act based on the information available to it. A policy can be deterministic or stochastic, and it may be fixed or contingent upon historical data.

RL agents can adapt to changing patterns in TS data by learning from their past actions and rewards. This capability is especially beneficial in non-stationary environments, where the underlying dynamics of the TS evolve over time. In principle, RL manages complex and non-linear dependencies within TS data, as it learns to map observations to actions that optimize the reward function. RL allows incorporating external information or side information into the forecasting process. This can involve incorporating macroeconomic indicators, weather data, or other relevant factors that affect the target TS. RL can be designed to optimize multiple objectives simultaneously, such as prediction accuracy and risk management. Training RL agents can be computationally expensive, especially for complex problems and large datasets. The process often requires extensive simulation and exploration of the state-action space. Carefully designing the reward function is critical. A poorly designed reward function can lead to suboptimal or unexpected agent behavior. This requires deep domain expertize. RL algorithms typically require a large amount of training data to learn effectively. Understanding why an RL agent makes a particular prediction can be challenging, making it a black-box approach in many cases. Balancing exploration of new actions with exploitation of known good actions is crucial in RL, and finding this balance can be difficult.

5.5.3. Graph neural network (GNN) for TS

GNNs offer a novel approach to TS forecasting by representing the TS data as a graph. This allows for capturing relationships between different time points or incorporating external information represented as nodes and edges in the graph. GNNs can effectively capture complex, non-linear relationships between different parts of the TS or between the TS and external factors represented as nodes in the graph. This is particularly advantageous when there are dependencies that are not easily captured by sequential models.

External information, including sensor readings, geographic data, and social media sentiment, can be directly integrated into the graph structure to enhance the predictive model. GNNs can manage TS data with irregular sampling more effectively than some other methods, as the graph structure can represent temporal relationships regardless of the specific sampling intervals. GNNs are inherently well-suited for managing multivariate TS, where multiple TS are interconnected. This is because graphs can effectively represent the relationships among different series. Spatial information can be incorporated into the GNN to enhance its efficiency, as demonstrated in numerous studies, including Liang et al. (2022) and Bui et al. (2022)

Defining the appropriate graph structure is crucial. The selection of nodes and edges significantly impacts performance. There is no universally optimal approach; it often requires domain expertize. Training GNNs can be computationally intensive, particularly when dealing with large graphs and intricate architectures. Understanding the learned representations within a GNN can be challenging, much like other DL models. GNNs may require more data than simpler models, especially to effectively learn the graph structure and relationships. Compared to traditional TS models or RNNs and CNNs, the application of GNNs to TS forecasting is a relatively new field.

5.5.4. Large language models (LLMs) for TS

LLMs, initially developed for natural language processing, are increasingly applied to TS prediction thanks to their capability to model complex, long-range dependencies within sequential data. Their extensive capacity and pre-training on large datasets enable them to capture intricate patterns with minimal feature engineering. This pre-training supports

transfer learning, improving performance on related tasks even with limited labeled data. Additionally, their flexible architecture allows adaptation to various types of TS, including multivariate and irregularly spaced data. However, recent research has explored using LLMs for structured data, including TS, by framing the problems as natural language tasks or serialized sequences. To accomplish this, the numerical data must be transformed into a textual format suitable for LLM input. After processing with a pretrained LLM, the predicted text can then be converted back into numerical values.

LLMs are particularly effective for large-scale, high-dimensional TS with long temporal horizons, such as financial markets, climate modeling, or industrial sensors. However, for smaller or sparser datasets, traditional models like ARIMA or simpler NNs often prove more practical and efficient. Despite their strengths, LLMs demand significant computational resources and large amounts of data, and their limited interpretability remains a challenge, especially in critical applications where understanding model decisions is essential.

X. Zhang et al. (2024) reviewed how researchers have adapted LLMs–originally designed for natural language processing–to analyze TS. It detailed various strategies for transferring knowledge from LLMs to numerical TS, including prompting, quantization, alignment, using vision as a bridge, and tool integration, highlighting their potential and current challenges. Tan et al. (2024) tested whether LLMs truly improve TS forecasting. They found that replacing LLMs with simple attention-based models often led to similar or better performance, while significantly reducing computational costs, suggesting that LLMs offer little advantage for this task.

6. Outlier handling algorithms

There are two ways to interpret outliers. First, outliers might result from measurement errors, such as operator mistakes or equipment malfunction. In this case, the source of the error needs to be identified and corrected. However, if we are confident in the accuracy of our data collection and still observe outliers, we need to consider broader implications. For instance, unusual weather events like unprecedented floods, heatwaves, or cold snaps in meteorological data require careful consideration and should inform future planning and policy decisions.

Outlier predictions encompass various aspects in TS and ML algorithms. Two primary aspects include predicting whether outliers will happen and the prediction based on outliers. The first aspect can be framed as either a classification or regression problem, which involves predicting whether outliers occur in the future and, if they do, estimating their potential values. The second aspect is specifically a regression problem that aims to forecast future outcomes using data that contains outliers. Different algorithms are employed for classification and regression tasks. In this context, we concentrate on forecasting future outcomes using data that contains outliers. Let's now explore some ML algorithms suitable for analyzing this type of TS data.

S. Lin et al. (2020) introduced a modified version of LSTM networks combined with a variational autoencoder for outlier detection in TS data. This algorithm demonstrated the capability to identify anomalies across multiple time scales. Tang et al. (2023) proposed GRU networks to address the challenges associated with multivariate TS data with outliers. They identified GRU as a viable solution for the problems of vanishing and exploding gradients, which often arise when using a high number of hidden layers (HLs) to learn the outliers present in the data. C. Lin et al. (2022) examined LSTM and GRU networks for detecting outliers in TS data. The findings indicated that GRU outperformed LSTM due to its higher

accuracy and simpler structure, which requires fewer learning parameters. Bidirectional algorithms have been studied for outlier detection, and new versions specifically designed for this type of TS data have been introduced, such as those in Ullah et al. (2021) and L. Zhang et al. (2021).

Other algorithms renowned for outlier detection include tree-based ensemble algorithms. C. Zhang et al. (2022) applied XGBoost and LSTM-stacked denoising autoencoders to effectively predict outliers in wind patterns in northeastern China. The study by Hartanto et al. (2023) addressed the approximation of stock price TS using LightGBM and compared the results with XGBoost, AdaBoost, and CatBoost. Their experiment demonstrated the superior capabilities of LightGBM in forecasting stock price data, even when it contained outliers. Y. Zhang et al. (2022) introduced an enhanced version of the N-BEATS network that integrates seasonality with support vector machines to diagnose outliers in TS data related to sewage treatment.

Falchi et al. (2023) involved training a TFT model to predict the vibrational characteristics of the aged Guinigi Tower's experimental frequencies, located in Lucca, Italy, along with various environmental parameters to monitor the structural health of the tower. The primary objective of the TFT model was to identify potential damages caused by the Viareggio earthquake that occurred in 2022. The implemented TFT model demonstrated sensitivity to outliers and provided accurate predictions. This paper tests the algorithms described above on a TS containing outliers. We then analyze and interpret their performance and effectiveness.

7. Missing data handling algorithms

Missing data are common in any kind of datasets, particularly when collected over an extended period. Therefore, missing data is a frequent occurrence in TS analysis. Addressing this issue in ML can be categorized into two approaches: some algorithms can directly handle datasets with missing values, while others necessitate the imputation of these missing values using appropriate methods, such as the mean, median, or interpolation. DeepAR and Prophet can handle data with missing values, while the other algorithms discussed in this paper require the imputation of appropriate values to address these gaps.

Imputation refers to the process of replacing missing values in a dataset. In recent years, generative models and LLMs have been increasingly used to perform advanced imputation by learning complex data distributions and generating plausible replacements. Generative model-based imputation methods include probabilistic Bayesian models, variational autoencoders, deep generative models for tabular data, diffusion models, and graph-based generative techniques. These approaches are particularly effective at capturing dependencies within the data and producing coherent, context-aware values. On the other hand, LLM-based imputation strategies leverage the semantic and contextual understanding of LLMs. These include prompt-based imputation, where carefully designed prompts guide the model to generate appropriate values; fine-tuning or adapter-based methods that specialize pre-trained LLMs for imputation tasks; and multimodal or hybrid prompting techniques that incorporate diverse data types for richer context. Both generative models and LLMs offer powerful, data-driven alternatives to traditional imputation techniques, especially in cases where missingness patterns are complex or data relationships are highly nonlinear.

Qin and Wang (2023) introduced a model to handle missing data for multivariate TS using generative adversarial networks, incorporating an iterative strategy and gradient

optimization. They tested the method by implementing it on three large-scale datasets. As a result, the generative algorithm excelled in accuracy compared to traditional methods. Yang et al. (2024) illustrated frequency-aware generative models for multivariate TS imputation, which improve missing-data handling by integrating frequency-domain information through high- and dominant-frequency filters and by employing two cross-domain representation learning modules. These design choices targeted the residual term, enabling more accurate imputations and better alignment with time-attribute dependencies. Y. Liu et al. (2024) explained Timer, a generative pre-trained TS transformer, and showed how large TS models can tackle missing data imputation by treating forecasting, imputation, and anomaly detection as a unified generative task. They pre-trained Timer on a billion-timestamp dataset using a single-series sequence format and next-token prediction, demonstrating state-of-the-art imputation performance in few-shot and zero-shot settings across diverse TS tasks.

8. Simulation study

This section presents a controlled simulation study designed to complement the validation on real-world datasets in Section 9. While real-world data provides crucial practical insights, a synthetic TS establishes a known ground truth. This allows for a precise quantification of how varying degrees of outliers and missing values influence different models–an effect that is difficult to isolate when analyzing a singular real-world dataset. This approach directly addresses the critical question of how specific data irregularities shape forecasting performance.

We generate a non-stationary TS, $\{X_t\}$, that incorporates components commonly observed in real-world data: a long-term trend, multiple seasonal patterns, and random noise. The model is formally defined as

$$X_t = T_t + S_{1,t} + S_{2,t} + \epsilon_t,$$

where $t = 1, \ldots, N$ is the time index, with $N = 5000$ representing hourly observations. The components are specified as follows.

- T_t is the trend component, modeled as a simple linear function $T_t = \beta \cdot t$, where $\beta = 0.001$ is the slope, imparting a gentle, upward trajectory to the series.
- $S_{1,t}$ is the primary seasonality component, simulating a strong daily cycle (24-hour period). This is modeled using a sinusoidal function $S_{1,t} = A_1 \cdot \sin(\frac{2\pi t}{L_1})$, where $A_1 = 2$ is the amplitude and $L_1 = 24$ is the period length.
- $S_{2,t}$ is the secondary seasonality component, capturing a slower, weekly rhythm (168-hour period) with a larger amplitude $S_{2,t} = A_2 \cdot \sin(\frac{2\pi t}{L_2})$, where $A_2 = 5$ and $L_2 = 168$.
- ϵ_t is the noise component, representing random, unpredictable fluctuations. We assume this noise to be i.i.d., following a normal distribution $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$, where $\sigma = 0.5$.

This additive model produces a clean, realistic-looking TS, $\{X_t\}$, which serves as our ground truth. Subsequently, we systematically introduce two types of data corruption to obtain the observed series $\{Y_t\}$.

Missing values are introduced completely at random. A proportion $\alpha_{\text{Missing}} = 0.05$ of the data points is selected uniformly at random and set to 'NaN'. The mechanism is defined as

$$Y_t = \begin{cases} \text{NaN}, & \text{with probability } \alpha_{\text{Missing}}, \\ X_t, & \text{otherwise.} \end{cases}$$

Table 1. Descriptive statistics for the synthetic and real-world examples.

Set	Mean	Std	Min	25%	Median	75%	Max	Variance	Skewness	Kurtosis
Synthetic TS										
Full Dataset	2.60	5.04	−25.81	−0.76	2.57	5.92	31.71	25.45	0.23	5.33
Training Set	2.09	4.95	−25.81	−1.22	2.10	5.32	30.42	24.52	0.19	5.43
Validation Set	4.59	4.96	−22.09	1.34	4.79	7.57	31.71	24.59	0.41	7.62
Test Set	4.81	4.71	−12.38	1.51	4.33	7.95	30.30	22.22	0.65	4.56
Sunspot TS										
Full Dataset	81.78	67.89	0.0	23.90	67.20	122.50	398.2	4608.95	0.93	0.34
Training Set	80.04	66.75	0.0	23.62	66.50	117.40	398.2	4454.95	0.99	0.65
Validation Set	94.08	72.60	0.0	29.10	76.10	149.90	307.7	5270.84	0.61	−0.65
Test Set	44.52	41.78	0.2	7.55	32.35	76.35	146.1	1745.55	0.67	−0.87
CPU TS										
Full Dataset	89.79	12.08	18.72	89.08	92.45	94.30	99.12	145.90	−4.76	22.55
Training Set	89.17	13.61	18.72	88.83	92.47	94.35	99.12	185.34	−4.19	16.79
Validation Set	91.47	2.85	79.17	89.21	91.88	93.58	97.87	8.11	−0.42	0.01
Test Set	94.59	1.20	91.26	93.75	94.55	95.21	99.04	1.45	0.42	1.05
CO TS										
Full Dataset	2.15	1.45	0.1	1.1	1.8	2.90	11.9	2.11	1.37	2.67
Training Set	2.26	1.41	0.1	1.3	2.0	3.00	11.9	1.98	1.29	2.67
Validation Set	2.05	1.39	0.1	1.0	1.7	2.75	8.7	1.93	1.21	1.46
Test Set	1.71	1.14	0.1	0.9	1.4	2.20	7.5	1.30	1.61	3.37

This scenario mimics random sensor failures or data transmission errors.

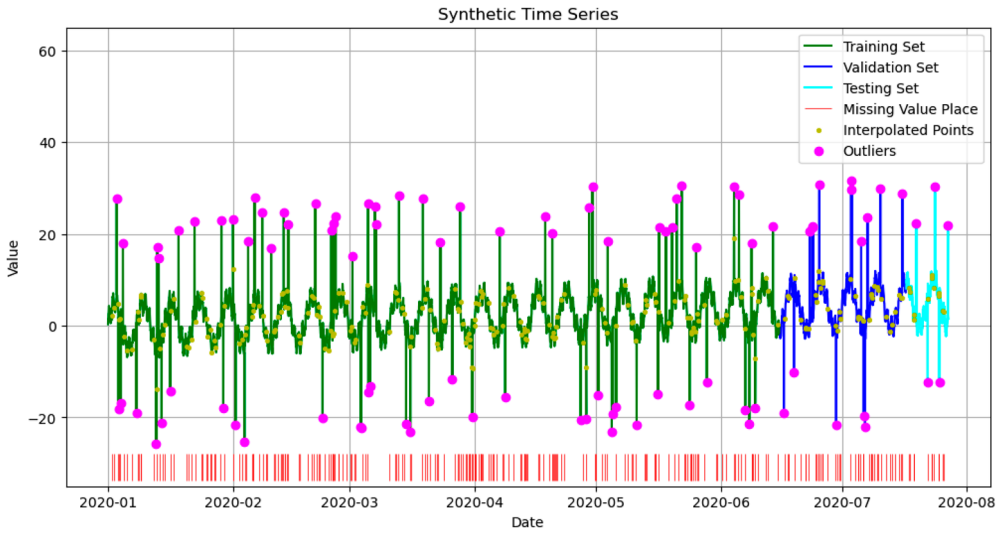
Additive outliers are then superimposed on the partially observed series $\{Y_t\}$. A proportion $\alpha_{\text{Outlier}} = 0.02$ of the remaining data points is selected. For each selected point, a large deviation is added:

$$Y_t = Y_t + \delta_t,$$

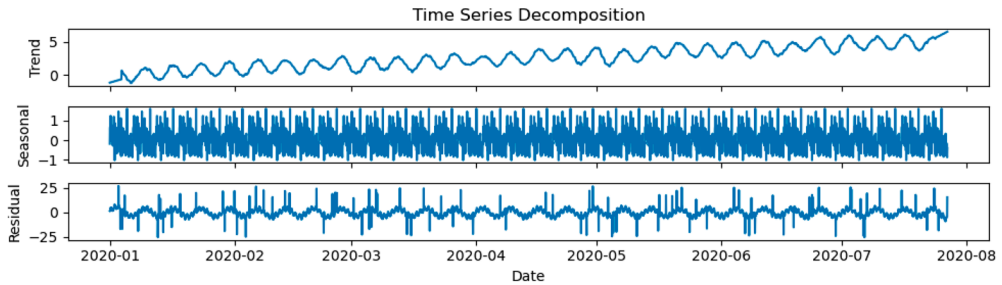
where the outlier magnitude δ_t is given by $\delta_t = \xi \cdot \text{sign} \cdot \sigma_Y$. Here, $\xi = 5$ is the outlier strength multiplier, 'sign' is randomly chosen from $\{-1, +1\}$ (creating both positive and negative spikes), and σ_Y is the standard deviation of the non-missing portion of the data. This method ensures the outliers are severe and statistically significant, simulating sudden sensor spikes or drops.

The final, corrupted series $\{Y_t\}$ used for the simulation experiment is thus a combination of the underlying clean signal $\{X_t\}$, gaps from missing data, and extreme anomalous points. A representative sample from this series is visualized in Figure 1(a), together with its decompositions, as well as the ACF/PACF plots and the histogram, shown in Figure 1(b,c), respectively. The training, validation, and test set sizes are 4000, 750, and 250, respectively. The statistical metrics of the samples are presented in Table 1. Following the feature engineering approach outlined in Section 3, we then create input features from $\{Y_t\}$ using a window of lagged values to prepare the data for ML model training. This controlled setup allows us to dissect model performance in a way that is impossible with real data, where the true underlying process is always unknown. The divergence metrics of the predictions from the ML algorithms, relative to the test-set samples, are provided in Table 2. We utilize mean absolute error (MAE), mean absolute percentage Error (MAPE), mean squared error (MSE), root mean squared error (RMSE), and coefficient of determination R^2.

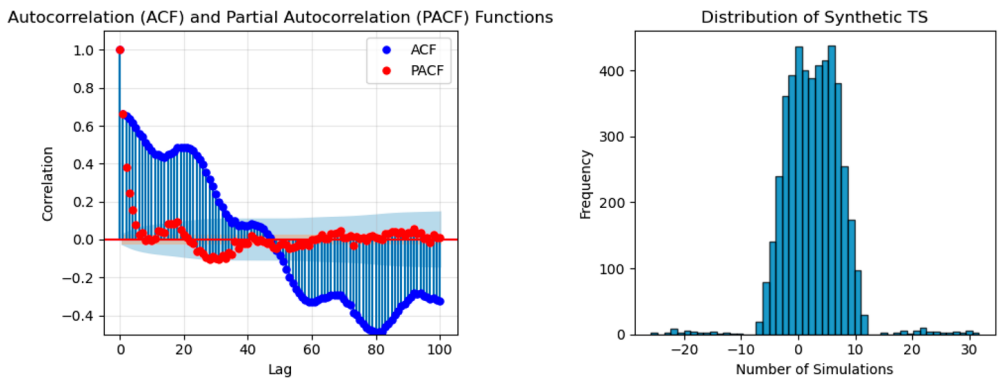
Although ML algorithms do not inherently require stationarity, we first ensure stationarity before applying ARIMA forecasting shown on the left side of Figure 2(a). Additionally, we overlook the normalization process for DeepAR (the left side of Figure 2(d)), tree-based ensemble methods, WBT (the right side of Figure 2(f)), Prophet (the left side of Figure 2(f)),



(a) Synthetic TS.



(b) Decompositions.



(c) ACF/PACF and histogram.

Figure 1. Synthetic TS visualizations. (a) Synthetic TS. (b) Decompositions and (c) ACF/PACF and histogram.

Table 2. Evaluation metrics for test prediction sets.

Method	Synthetic (N, MAE, MAPE, MSE, RMSE, R^2)	Sunspot (N, MAE, MAPE, MSE, RMSE, R^2)	CPU (N, MAE, MAPE, MSE, RMSE, R^2)	CO (N, MAE, MAPE, MSE, RMSE, R^2)
ARIMA	(−, 5.64, 1.22,49.31, 7.02, −1.28)	(−, 48.31, 10.31,3822.78, 61.83, −1.21)	(−, 94.57, 1.00,9018.92, 94.97, −6288.04)	(−, 4.69, 3.83,24.63, 4.96, −18.07)
Elman	(E : 7, 4.30, 1.25,31.87, 5.65, −0.47)	(E : 7, 248.41, 86.65,63434.72, 251.86, −35.72)	(E : 10, 4.32, 0.05,20.07, 4.48, −12.99)	(E : 10, 1.98, 2.16,4.59, 2.14, −2.56)
LSTM	(E : 89, 5.09, 4.75,41.42, 6.44, −0.92)	(E : 81, 47.40, 3.89,3417.84, 58.46, −0.98)	(E : 220, 0.92, 0.01,1.47, 1.21, −0.03)	(E : 100, 1.30, 1.19,2.83, 1.68, −1.19)
GRU	(E : 10, 22.22, 15.88,533.96, 23.11, −23.71)	(E : 10, 69.54, 32.16,6264.28, 79.15, −2.63)	(E : 10, 1.02, 0.01,1.75, 1.32, −0.22)	(E : 10, 0.75, 0.49,1.19, 1.09, 0.08)
Bidirectional LSTM	(E : 10, 5.78, 5.34,49.47, 7.03, −1.29)	(E : 10, 51.99, 25.69,3462.63, 58.84, −1.00)	(E : 10, 0.92, 0.01,1.44, 1.20, 0.00)	(E : 10, 1.50, 1.56,3.34, 1.83, −1.58)
Deep RNN	(E : 10, 4.76, 1.05,37.82, 6.15, −0.75)	(E : 10, 244.91, 85.60,61707.25, 248.41, −34.72)	(E : 10, 4.45, 0.05,21.26, 4.61, −13.83)	(E : 10, 1.50, 1.66,2.78, 1.67, −1.15)
DeepAR	(E : 100, 0.95, 0.30,8.56, 2.93, 0.60)	(E : 100, 13.41, 3.76,354.57, 18.83, 0.79)	(E : 100, 2.55, 0.03,8.43, 2.90, −4.88)	(E : 100, 0.90, 0.65,1.55, 1.25, −0.20)
RF	(T : 100, 1.26, 0.47,10.58, 3.25, 0.51)	(T : 100, 9.84, 1.40,220.04, 14.83, 0.87)	(T : 100, 0.98, 0.01,1.53, 1.24, −0.06)	(T : 100, 0.35, 0.27,0.28, 0.53, 0.78)
XGBoost	(T : 100, 1.74, 0.63,15.52, 3.94, 0.28)	(T : 100, 10.61, 1.49,264.13, 16.25, 0.85)	(T : 100, 1.01, 0.01,1.61, 1.27, −0.12)	(T : 100, 0.36, 0.27,0.29, 0.54, 0.77)
LightGBM	(T : 1000, **1.11**, **0.36**,**3.52**, **1.88**, **0.84**)	(T : 1000, **6.21**, **0.81**,**73.26**, **8.56**, **0.96**)	(T : 1000, **0.18**, **0.00**,**0.07**, **0.27**, **0.95**)	(T : 1000, **0.06**, **0.05**,**0.01**, **0.08**, **1.00**)
Prophet	([C : 4, I : 100], 17.78, 10.46,498.83, 22.33, −22.09)	([C : 4, I : 30], 40.54, 17.91,2026.03, 45.01, −0.17)	([C : 4, I : 100], 1.10, 0.01,1.86, 1.36, −0.30)	(I : 10000, 0.75, 0.56,0.93, 0.96, 0.28)
WBT	([L : 5, T : 100], 1.42, 0.73,10.17, 3.19, 0.53)	([L : 5, T : 100], 11.42, 1.73,252.43, 15.89, 0.85)	([L : 5, T : 100], 0.93, 0.01,1.51, 1.23, −0.05)	([L : 5, T : 100], 0.36, 0.29,0.27, 0.52, 0.79)
TFT	(E : 29, 5.00, 0.99,40.83, 6.39, −0.89)	(E : 25, 40.56, 18.01,2064.00, 45.43, −0.19)	(E : 15, 22.78, 0.24,525.74, 22.93, −365.61)	(E : 26, 6.15, 5.98,39.16, 6.26, −29.27)
N-BEATS	(E : 100, 1.31, 0.54,10.02, 3.17, 0.54)	(E : 100, 10.79, 1.98,231.46, 15.21, 0.87)	(E : 100, 0.96, 0.01,1.56, 1.25, −0.09)	(E : 100, 3.04, 0.25,9.25, 2.93, 0.78)
N-HiTS	(E : 100, 7.06, 4.52,65.45, 8.09, −2.03)	(E : 100, 72.28, 34.29,6852.23, 82.78, −2.97)	(E : 100, 4.92, 0.05,28.82, 5.37, −19.10)	(E : 100, 3.60, 1.47,17.62, 4.20, −1.37)
SVD	(T : 100, 1.29, 0.54,9.64, 3.10, 0.55)	(T : 100, 13.21, 1.51,322.04, 17.95, 0.81)	(T : 100, 2.24, 0.02,6.67, 2.58, −3.65)	(T : 100, 0.75, 0.63,0.91, 0.96, 0.29)
EMD	(T : 100, 10.68, 8.60,138.10, 11.75, −5.39)	(T : 100, 24.72, 9.40,809.20, 28.45, 0.53)	(T : 100, 1.71, 0.02,4.90, 2.21, −2.42)	(T : 100, 2.16, 2.32,5.48, 2.34, −3.24)
EEMD	(T : 30, 9.47, 7.95,113.15, 10.64, −4.24)	(T : 30, 25.24, 8.22,909.08, 30.15, 0.47)	(T : 30, 0.98, 0.01,1.66, 1.29, −0.16)	(T : 30, 1.81, 1.97,3.95, 1.99, −2.06)
DFFNN	(E : 10, 5.55, 2.15,43.73, 6.61, −1.02)	(E : 10, 104.57, 18.13,12952.18, 113.81, −6.50)	(E : 30, 5.12, 0.05,28.32, 5.32, −18.75)	(E : 30, 1.21, 1.22,2.12, 1.46, −0.64)
BNN	([C : 2, B : 1000, I : 2000],3.97, 1.50, 27.43, 5.24, −0.27)	([C : 2, B : 1000, I : 2000],49.72, 23.69, 3144.41, 56.08, −0.82)	([C : 2, B : 1000, I : 2000],4.97, 0.05, 26.18, 5.12, −17.26)	([C : 2, B : 1000, I : 2000],1.05, 1.10, 1.56, 1.25, −0.20)
CNN	(E : 100, 8.34, 1.65,112.52, 10.61, −4.21)	(E : 100, 13.56, 2.41,294.84, 17.17, 0.83)	(E : 100, 1.01, 0.01,1.64, 1.28, −0.14)	(E : 100, 0.56, 0.45,0.88, 0.94, 0.32)
TCN	(E : 30, 1.16, 0.38,9.39, 3.06, 0.57)	(E : 30, 12.04, 3.14,259.91, 16.12, 0.85)	(E : 30, 4.60, 0.05,22.61, 4.75, −14.76)	(E : 100, 0.37, 0.26,0.30, 0.54, 0.77)
STFT	(E : 500, 4.57, 1.61,34.91, 5.91, −0.62)	(E : 500, 49.95, 24.30,3376.51, 58.11, −0.95)	(E : 500, 4.73, 0.05,24.19, 4.92, 15.87)	(E : 500, 1.06, 1.13,1.60, 1.26, −0.24)
RL	(E : 90, 4.29, 1.95,32.02, 5.66, −0.48)	(E : 90, 39.77, 8.73,2344.65, 48.42, −0.36)	(E : 90, 1.37, 0.01,3.07, 1.75, −1.14)	(E : 90, 1.24, 1.05,2.70, 1.64, −1.09)
GNN	(E : 100, **1.03**, **0.49**,**7.11**, **2.67**, **0.67**)	(E : 100, **7.98**, **1.03**,**123.80**, **11.13**, **0.93**)	(E : 100, **0.92**, **0.01**,**1.40**, **1.18**, **0.02**)	(E : 100, **0.50**, **0.25**,**0.46**, **0.68**, **0.78**)
LLM	(P : 101, 5.14, 4.01,88.09, 9.39, −3.61)	(P : 93, 52.95, 5.86,4346.92, 65.93, −1.46)	(P : 121, 1.30, 0.01,2.49, 1.58, −1.27)	(P : 90, 1.44, 1.39,3.87, 1.97, −0.88)

Note: The term N indicates the number of Epochs (E), Trees (T), Chains (C), Iterations (I), Levels (L), Burn-in iterations (B), and Predictions (P). Moreover, the best and second-best metrics are bolded.

N-HITS (the left side of Figure 2(h)), BNN (the right side of Figure 2(j)), RL (the right side of Figure 2(l)), and GNN (the left side of Figure 2(m)). We scale the data for the remaining algorithms to achieve better results with lower computational costs. For the ML purpose, we employ a window size of 30 to generate the input features from TS data. Each window is used as an input for our models, allowing them to learn patterns and make predictions based on a set number of previous values.

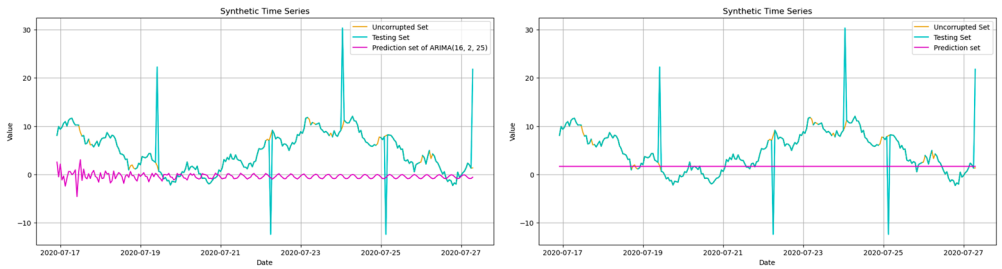
To ensure a fair and practical comparison, we implement all algorithms under consistent experimental conditions regarding data splitting and evaluation metrics. Given the vast differences in their inherent architectures and training paradigms–from the tree-based growth of LightGBM to the gradient-based learning of RNNs–a one-size-fits-all hyper-parameter tuning is neither feasible nor informative. Instead, we evaluate each algorithm using a solid, well-accepted configuration that reflects its typical out-of-the-box use, rather than an exhaustively optimized version. This approach provides a more realistic assessment of their practical utility and computational burden for practitioners facing similar TS forecasting challenges.

The implementation of some algorithms can be significantly time-consuming, such as RNN, TFT (the left side of Figure 2(g)), and BNN. In contrast, some algorithms, like tree-based ensemble methods, Prophet, N-BEATS (the right side of Figure 2(g)), N-HiTS, SVD (the right side of Figure 2(h)), EMD (the left side of Figure 2(i)), RL, and GNN, are notably faster. Others–like EEMD (the right side of Figure 2(i)), STFT (the left side of Figure 2(l)), and LLM (the right side of Figure 2(m))–require more time to implement, but still less than methods such as BNN. The speed of DFFNN (the left side of Figure 2(j)), CNN (the left side of Figure 2(k)), and TCN (the right side of Figure 2(k)) depends on various factors, including the number of HLs, the number of neurons and filters, and the size of the dataset. However, the models we consider here are relatively fast. As the number of layers and the complexity increase, both the data volume and the number of training epochs must be sufficient. This may explain why the results are poor for some algorithms, such as bidirectional LSTM (the left side of Figure 2(c)), Deep Elman (the right side of Figure 2(c)), DFFNN, BNN, and TFT. Some algorithms, like RNNs, can effectively predict initial points, while others, such as DeepAR, tree-based ensemble methods, N-BEATS, TCN, and GNN, have the capability to make accurate predictions for the more distant future as well.

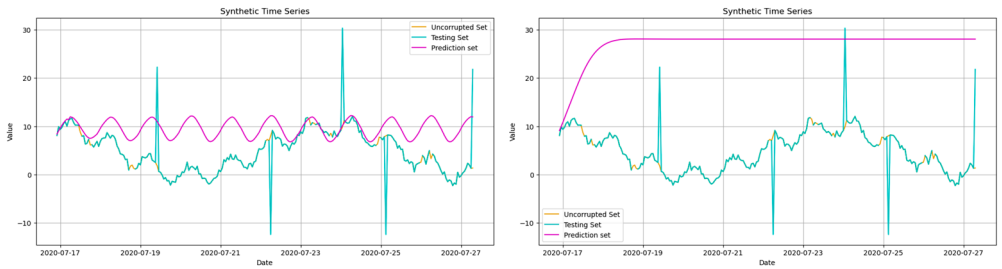
The best predictive performance is achieved by LightGBM (the right side of Figure 2(e)), GNN, DeepAR, TCN, SVD, N-BEATS, WBT, RF (the right side of Figure 2(d)), XGBoost (the left side of Figure 2(e)), and RL, respectively. Although the RL predictions yield negative R^2, they nonetheless trace the general trajectory of the samples fairly well. Some algorithms struggle to accurately predict outliers or example, RF and XGBoost–whereas others, such as DeepAR, TCN, SVD, N-BEATS, and WBT, tend not to predict outliers at all. Nonetheless, LightGBM, GNN, and RL demonstrate strong performance in predicting outliers present in the test set. In the next section, we tackle algorithms in real-world scenarios, along with more details on their architectures.

9. Real-world examples

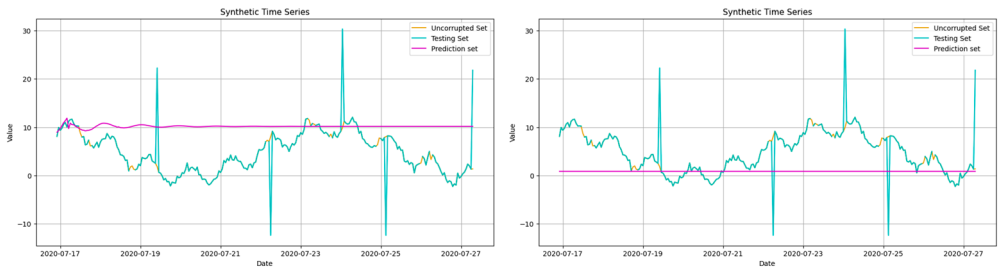
In this section, we examine three real distinct datasets: the average number of sunspots over centuries, CPU usage with outliers, and CO concentration with missing data. Our objective is to compare the performance of the presented ML algorithms in the real examples.



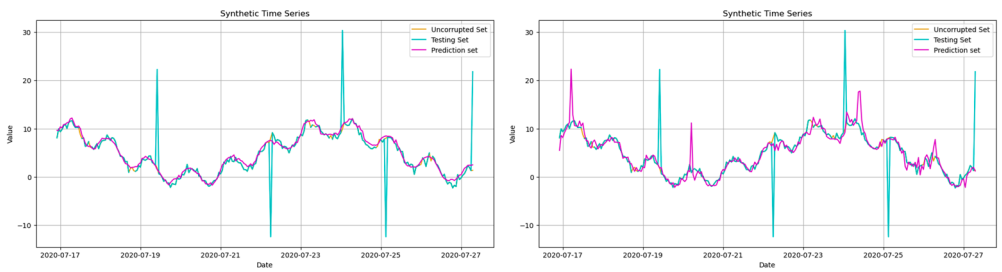
(a) ARIMA and Elman.



(b) LSTM and GRU.

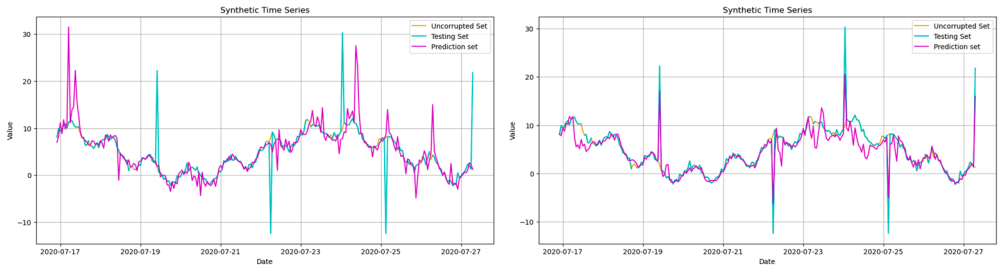


(c) Bidirectional LSTM and Deep Elman.

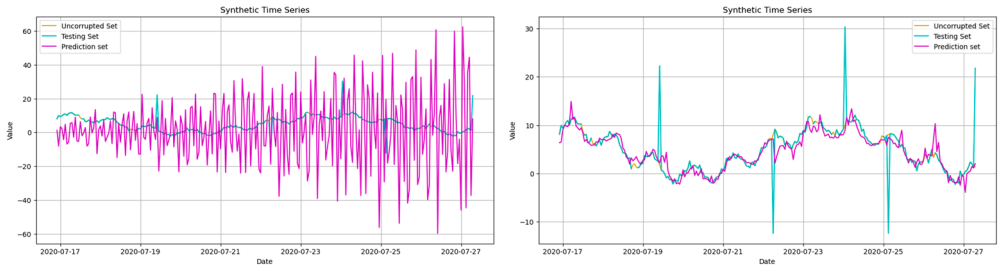


(d) DeepAR and RF.

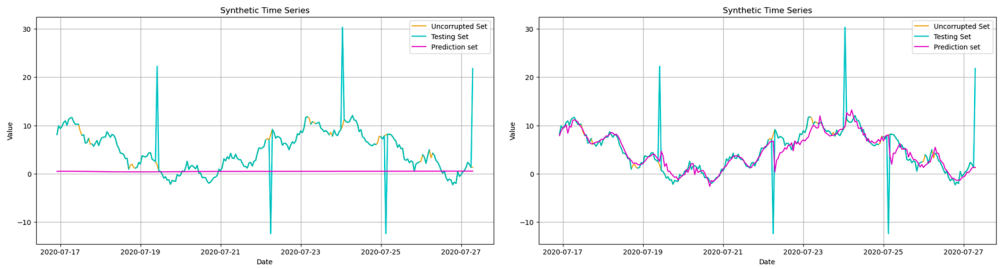
Figure 2. Prediction results for synthetic TS. Each row displays the prediction output for two specific algorithms, with the algorithm name in the corresponding subcaptions. (a) ARIMA and Elman. (b) LSTM and GRU. (c) Bidirectional LSTM and Deep Elman. (d) DeepAR and RF. (e) XGBoost and LightGBM. (f) Prophet and WBT. (g) TFT and N-BEATS. (h) N-HiTS and SVD. (i) EMD and EEMD. (j) DFFNN and BNN. (k) CNN and TCN. (l) STFT and RL and (m) GNN and LLM.



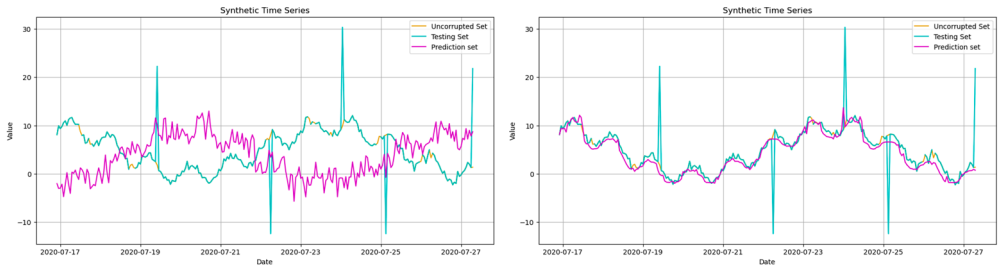
(e) XGBoost and LightGBM.



(f) Prophet and WBT.



(g) TFT and N-BEATS.

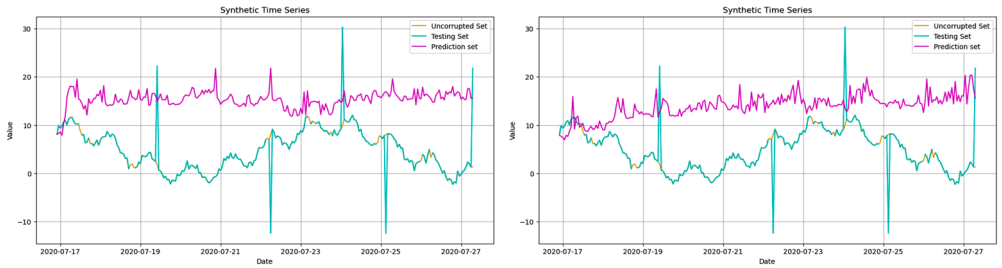


(h) N-HiTS and SVD.

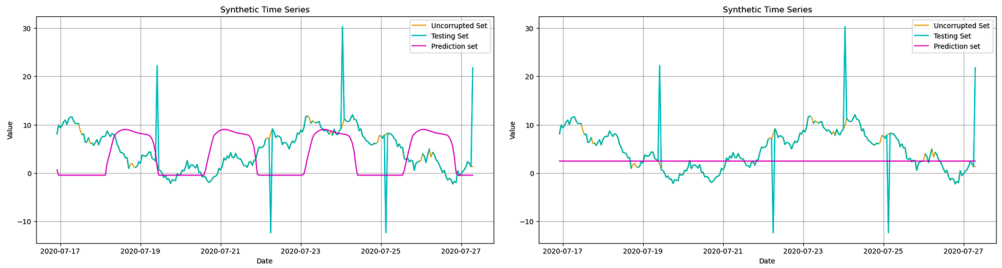
Figure 2. Continued.

9.1. Sunspot TS

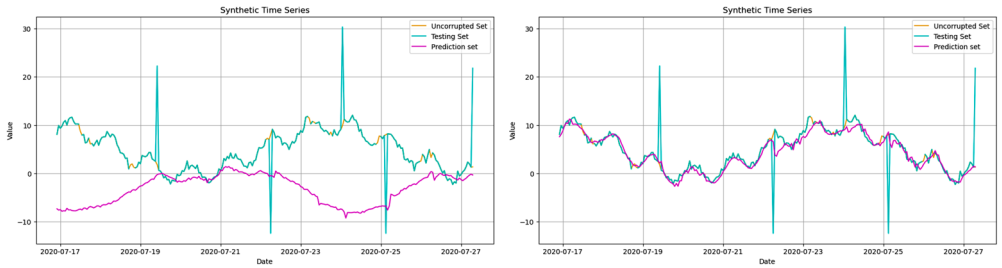
Sunspots from Valt (2021) are cooler areas on the surface of the Sun that are created by massive changes in the Sun's magnetic field. This dataset includes monthly mean of total sunspot number from 1749 to 2021, averaging about 273 years. These variations in the Sun's magnetic field can lead to solar flares and coronal mass ejections, which can have significant effects



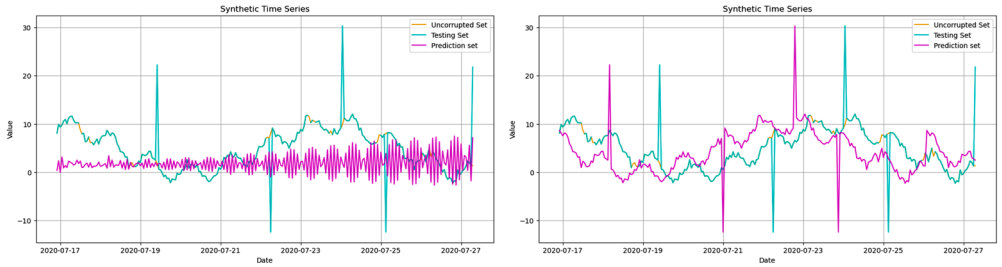
(i) EMD and EEMD.



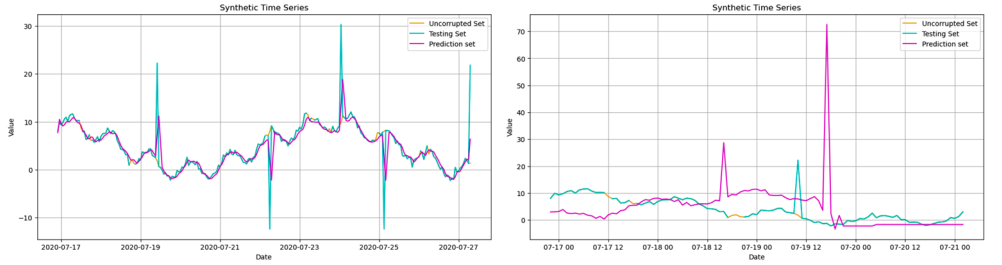
(j) DFFNN and BNN.



(k) CNN and TCN.



(l) STFT and RL.



(m) GNN and LLM.

Figure 2. Continued.

on Earth's atmosphere and climate. We split the data into three setsraining, validation, and testingith sizes 2514, 653, and 98, respectively, as shown in Figure 3(a). Also, there are 17 outliers in the TS.

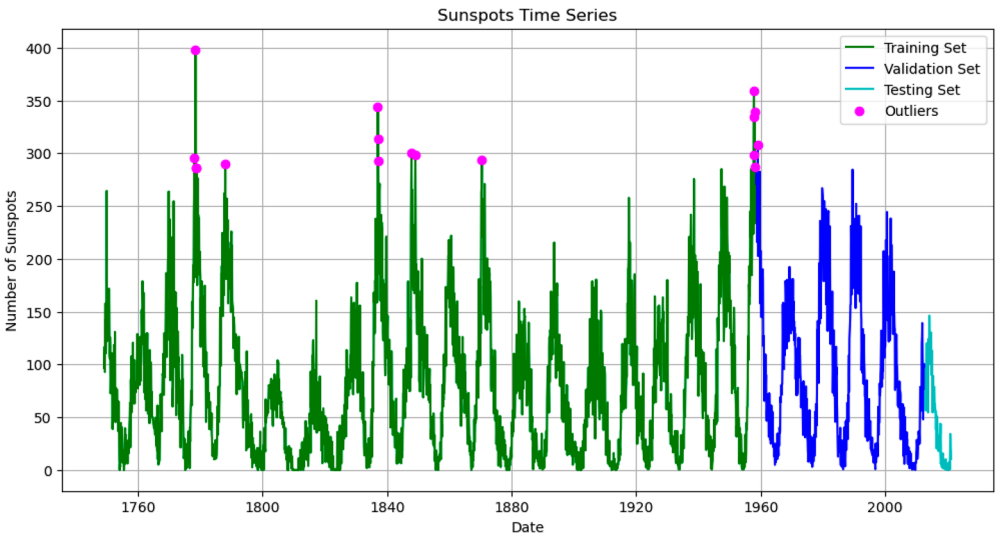
The statistics in Table 1 describe the sunspot TS as highly volatile with a pronounced positive skew, suggesting a process characterized by long quiet spells punctuated by intense, sporadic activity. The full dataset has a mean of 81.78 and a large standard deviation of 67.89, underscoring substantial variability. A notable observation is the clear difference between the training/validation sets and the test set: the training and validation means are 80.04 and 94.08, respectively, while the test-set mean is 44.52. This suggests that the test period corresponds to a solar minimum, a regime fundamentally different from the more active periods used for training. In other words, the reduction of cooler regions on the Sun's surface over recent years suggests an overall increase in solar activity and, in turn, greater solar heating of Earth. Such a shift poses a significant forecasting challenge, requiring the model to extrapolate into a quieter phase it has not previously observed.

The sunspot TS displays classic traits of a long-memory, non-stationary process governed by a strong underlying cycle. The series covers the full data range with few outliers, and decomposition, in Figure 3(b), highlights a dominant, highly regular seasonal component linked to the well-known. This is corroborated by a slow, sinusoidal ACF that remains significant at lags beyond 100, indicating dependence over very long horizons. The histogram on the left side of Figure 3(c), is heavily right-skewed with many zeros, consistent with long periods of low activity punctuated by shorter, intense solar maxima. Notably, there is a clear gap between the training/validation and test sets in both mean and variance, as shown in the table and plot, suggesting the test period corresponds to a solar minimum. This presents a stringent test of model generalizability across different solar regimes.

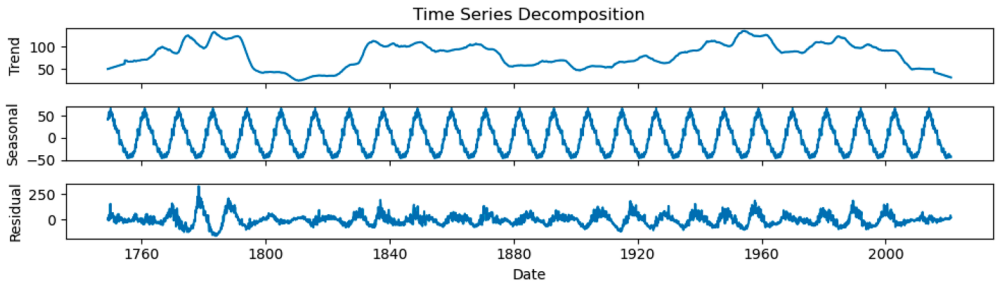
Figure 4, presents the predictions for the sunspot TS. The figure is organized into rows, with each row containing two figures. The left figure corresponds to the algorithm named on the left side of the subcaption, and the right figure corresponds to the algorithm named on the right. This setup allows us to compare pairs of algorithms side by side. This layout makes an easy comparison of the performance methods on the same data, highlighting differences in accuracy, trend capture, and prediction smoothness. By examining these plots side by side, we gain a better understanding of the strengths and weaknesses of each approach in capturing the underlying patterns.

Although we do not observe any significant trends in the data, our experiments indicate that implementing first-order differencing is beneficial followed by a 240th-order differencing. The ARIMA model, shown on the left side of Figure 4(a), has a high computational cost, produces inappropriate predictions, and performs ineffectively, particularly for the initial data points. One reason is that the TS is recorded over centuries, making it difficult for the ARIMA to accurately predict future values.

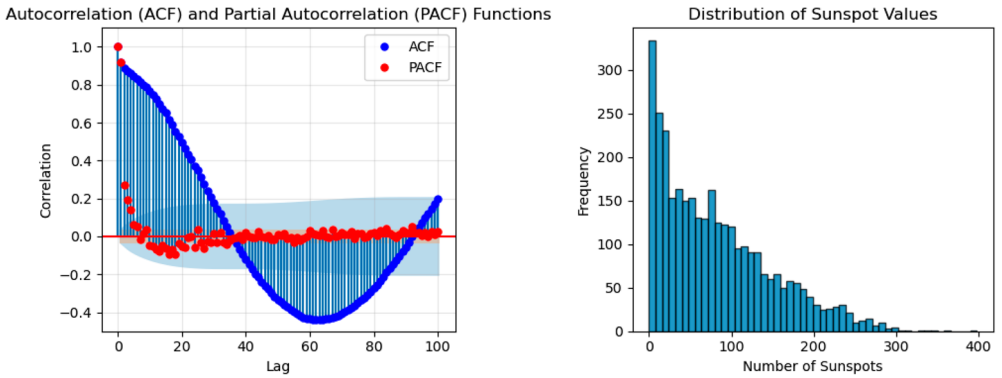
The best RNN predictions are for DeepAR, shown on the left side of Figure 4(d), and LSTM, displayed on the left side of Figure 4(b). Considering the time consuming and accuracy, the DeepAR acts acceptably. Two HLs with 50 neurons and tanh activation are used in the Elman, followed by one Dense Layer (DeLa). In the LSTM, three HLs with 40, 30, and 20 neurons and ReLU activation are applied, followed by global max pooling and one DeLa. For the GRU, shown on the right side of Figure 4(b), two HLs with 50 and 30 neurons are utilized, both utilizing tanh activation, followed by one DeLa. The bidirectional LSTM, on the left side of Figure 4(c), has two HLs with 50 and 30 neurons, using tanh activation, followed by one DeLa. The deep Elman, on the right side of Figure 4(c), has five HLs with 30 neurons each



(a) Sunspot TS.

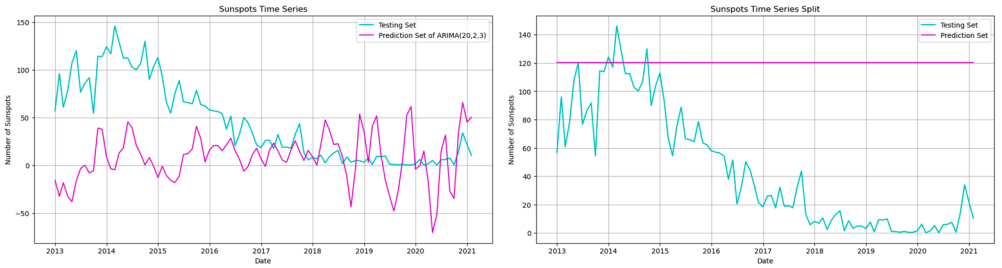


(b) Decompositions.

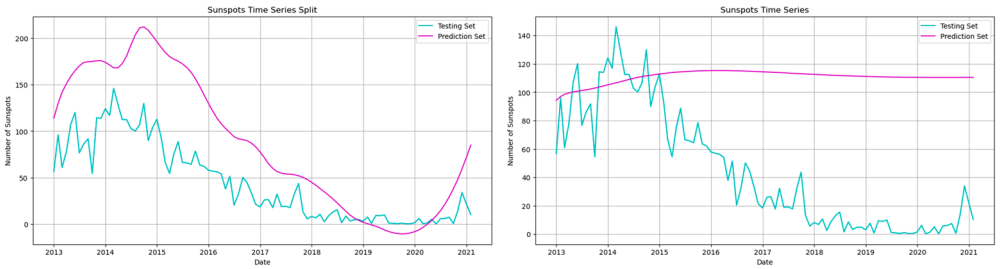


(c) ACF/PACF and histogram.

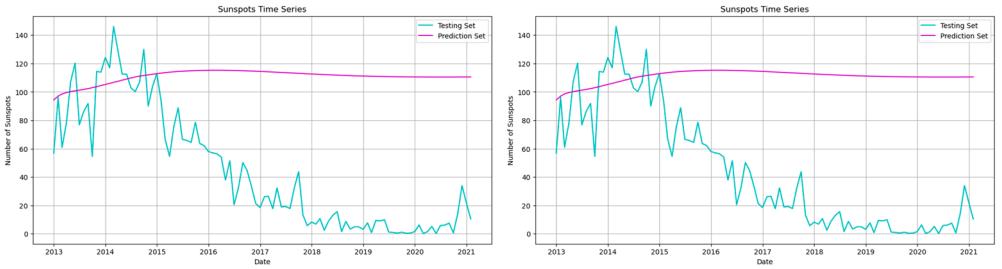
Figure 3. Sunspot TS visualizations. (a) Sunspot TS. (b) Decompositions and (c) ACF/PACF and histogram.



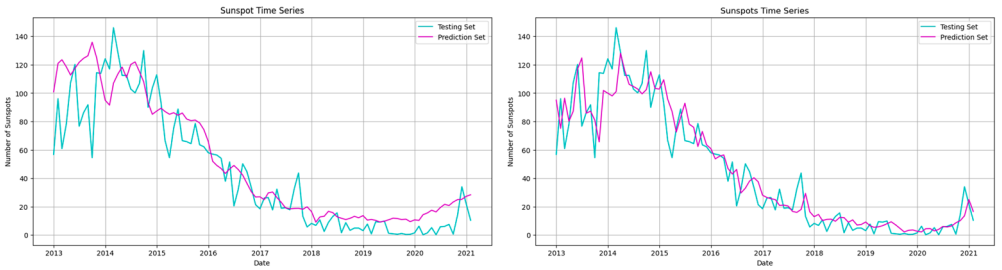
(a) ARIMA and Elman.



(b) LSTM and GRU.

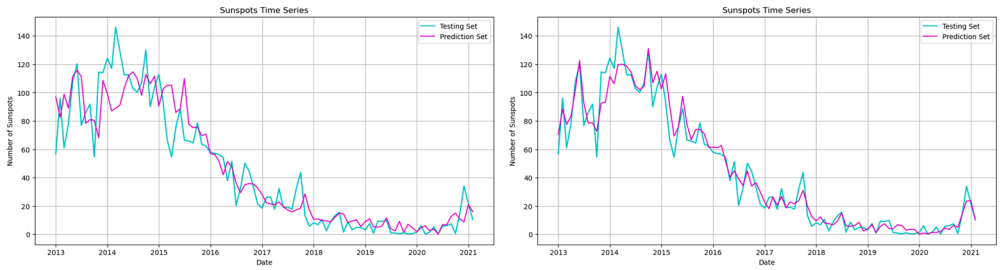


(c) Bidirectional LSTM and Deep Elman.

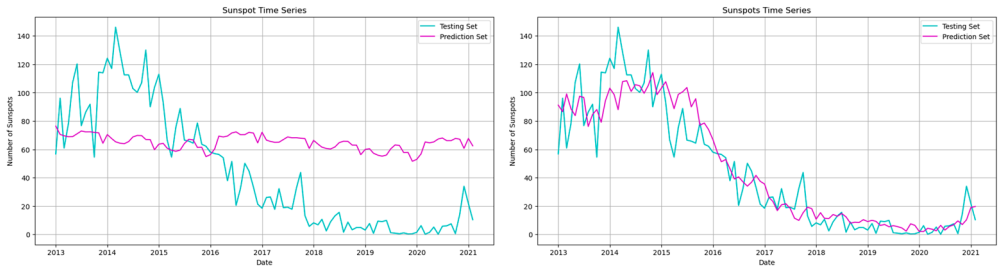


(d) DeepAR and RF.

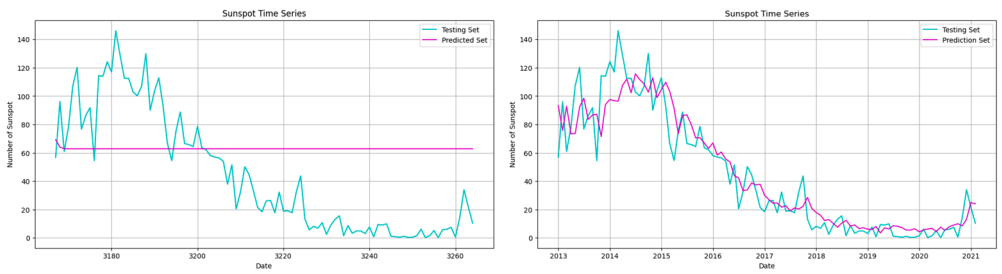
Figure 4. Prediction results for Sunspot TS. (a) ARIMA and Elman. (b) LSTM and GRU. (c) Bidirectional LSTM and Deep Elman. (d) DeepAR and RF. (e) XGBoost and LightGBM. (f) Prophet and WBT. (g) TFT and N-BEATS. (h) N-HiTS and SVD. (i) EMD and EEMD. (j) DFFNN and BNN. (k) CNN and TCN. (l) STFT and RL and (m) GNN and LLM.



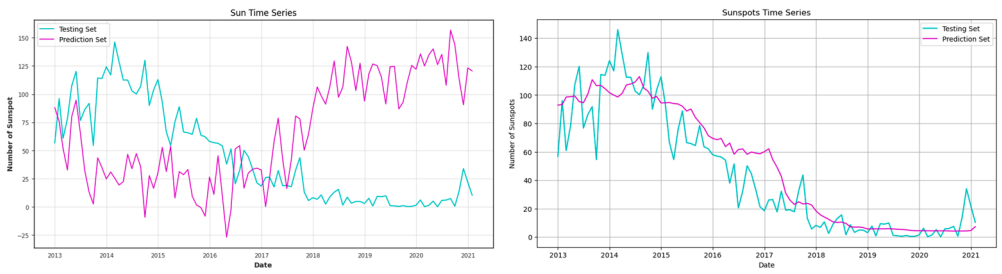
(e) XGBoost and LightGBM.



(f) Prophet and WBT.



(g) TFT and N-BEATS.

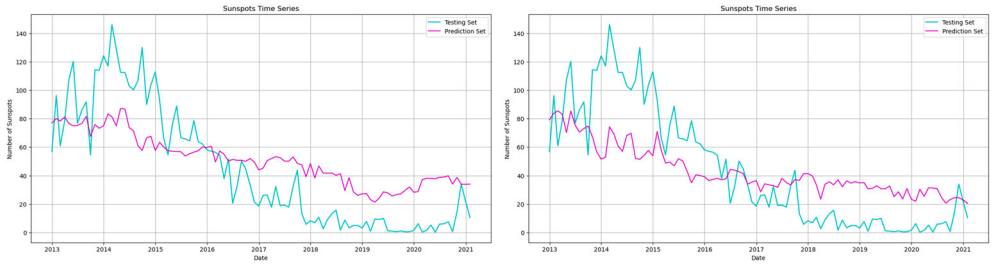


(h) N-HiTS and SVD.

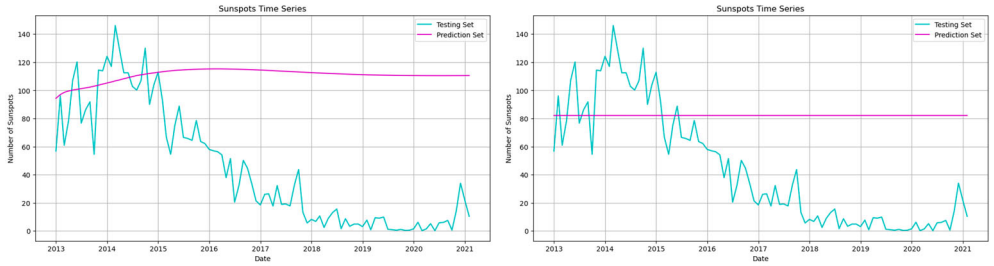
Figure 4. Continued.

using tanh activation, followed by one DeLa. All algorithms utilize Adam optimizer, except for the LSTM, which employs the Adamax variant. Adam and Adamax have slight differences; Adam utilizes the $L2$ norm for scaling, whereas Adamax employs the infinity norm, which provides more stable updates when gradients are large or sparse.

Tree-based ensemble algorithms act exceptional predictive performance across all our datasets. The lagged differences ranging from 1 to 30 are applied for the models of RF (the right side of Figure 4(d)), XGBoost (the left side of Figure 4(e)), and LightGBM (the right

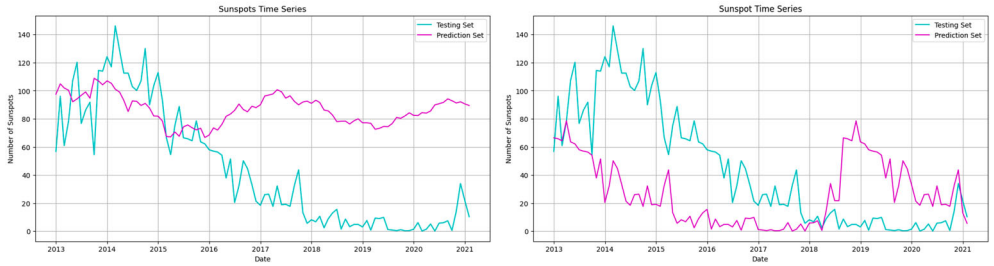


(i) EMD and EEMD.

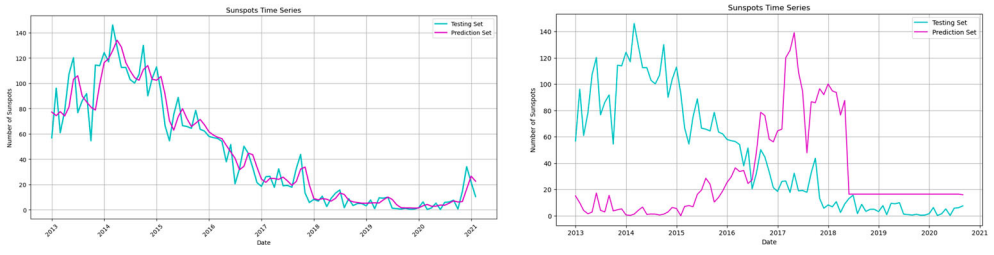


(j) DFFNN and BNN.

(k) CNN and TCN.



(l) STFT and RL.



(m) GNN and LLM.

Figure 4. Continued.

side of Figure 4(e)). For the LightGBM, we add additional features, including the rolling mean with windows of 3 and 6, as well as the rolling standard deviation with a window of 3. Unlike traditional methods, RF, XGBoost, and LightGBM do not require a specified number of epochs for training. Instead, they utilize the number of trees to make decisions. While the accuracy of predictions improves with an increasing number of trees, this also raises the risk of over-fitting.

The Haar functions at level 5 are used to extract the WBT features, with the prediction shown on the right side of Figure 4(f). These features are then incorporated into an RF model, along with lagged differences ranging from 1 to 30. We employ a multiplicative Prophet model, with the prediction plot on the left side of Figure 4(f), using 30 MCMC samples across 4 chains. For the TFT, exhibited on the left side of DeLa 4(g), we apply differencing of lags 1 to 30, along with a rolling mean and standard deviation using a window size of 30. For the decomposition methods, we first decompose the data into several components and then apply RF to predict on the test set. The related plots are the right side of Figure 4(h), the left side of Figure 4(i), and the right side of Figure 4(i), respectively for SVD, EMD, and EEMD. The best model among the specialized TS models is WBT and N-BEATS; the prediction plot for N-BEATS is on the right side of Figure 4(g). The prediction curves roughly resemble the structure of the testing set.

In the DFFNN with plot on the left side of Figure 4(j), two HLs containing 50 neurons each, activated by the ReLU function, are followed by one DeLa utilizing the tanh activation function. Weights, biases, and observations in the BNN are modeled using a normal distribution with ten HLs. To enhance the complexity of the BNN model, we incorporate lagged differencing features with ranges from 1 to 30. The BNN prediction plot is on the right side of Figure 4(j). The CNN, shown in the plot (the left side of Figure 4(k)), has two one-dimensional convolutional layers with 32 and 64 filters followed by the ReLU activation. This is followed by global average pooling and a DeLa. The TCN, with its plot on the right side of Figure 4(k), has two main convolutional HLs, each with 64 filters, a kernel size of 2, and ReLU activation. Each layer is followed by a dropout rate of 0.2. Subsequently, there is a flattening layer and a DeLa.

The STFT features are utilized within a GRU architecture comprising three HLs, containing 40, 30, and 20 neurons, respectively. This architecture is followed by a dropout rate of 0.1 and a DeLa. The STFT prediction plot is on the left side of Figure 4(l).

The architecture of the GNN, exhibited on the left side of Figure 4(m), consists of two graph convolutional layers, with the first layer expanding the initial single node feature into 16 features followed by a ReLU activation, and the second layer reducing the dimensionality back to a single output feature per node. We use GPT-2, whose predictions are shown on the right side of Figure 4(m), which is a basic LLM with approximately 124 million parameters. Designed primarily for natural language generation, it provides only rough and imprecise predictions for TS data. For our experiments, we input the most recent 65 samples from the validation set, constrained by the maximum token limit of 1024. Since the data includes decimal numbers, more tokens are needed to represent them accurately. While constructing a more advanced fine-tuned transformer model could improve performance, this would require a much larger dataset. Generally, more complex models demand greater amounts of training data for effective learning. In this example, the LLM model can predict only 93 samples, while the other algorithms predict 98 samples. The remaining algorithms have been implemented as usual, and we do not define any special characteristics.

Overall, Elman, GRU, bidirectional LSTM, Deep Elman, Prophet, and DFFNN capture the early signal well, but their predictions quickly level off and show limited improvement across the rest of the samples. TFT and BNN behave similarly to Elman, with TFT hampered by long runtimes and BNN requiring careful tuning and substantial training time to reach reasonable accuracy. Although TFT and BNN come with high computational costs, we include their results to illustrate the trade-off between predictive capability and efficiency. N-HiTS, EMD, EEMD, STFT, and RL identify the true values fairly well at the start, but beyond that their predictions are not very accurate, though a few points at midpoints are captured reasonably well by some methods. LSTM can trace the general shape of the test set curve, but the predictions remain imperfect. As a consequence, the most accurate predictions, in terms of the plots and the metrics, are as follows: LightGBM, GNN, RF, XGBoost, N-BEATS, WBT, TCN, SVD, CNN, and DeepAR.

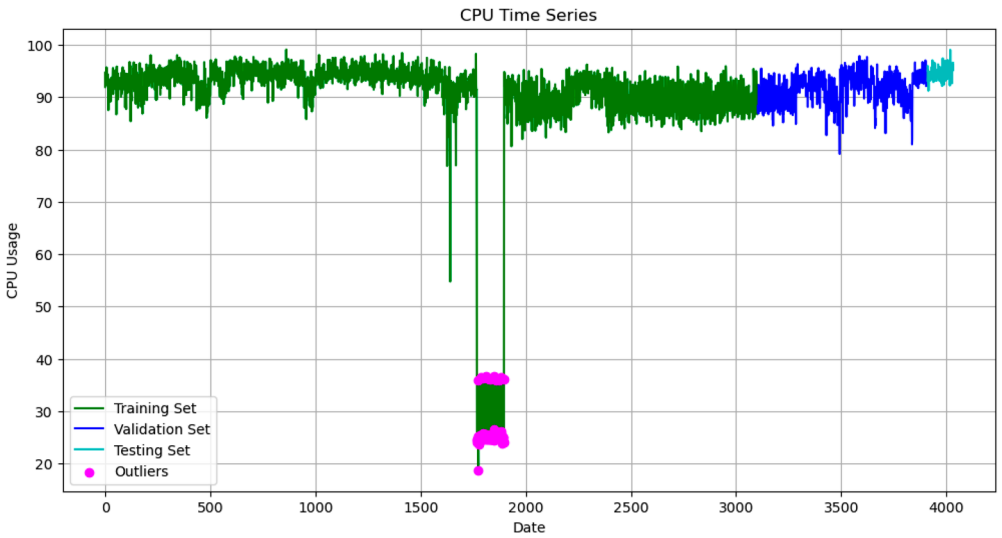
9.2. CPU usage TS with outliers

We select the real CPU utilization TS data from Numenta (2021), network bytes collected by Amazon Watch Server Cloud metrics. The data was recorded every 5 min from April 10 to 24, 2014. This dataset contains outliers, as shown in Figure 5 with 3104, 807, and 121 samples, respectively for training, validation, and test sets. Also, there are 129 outliers in the training data.

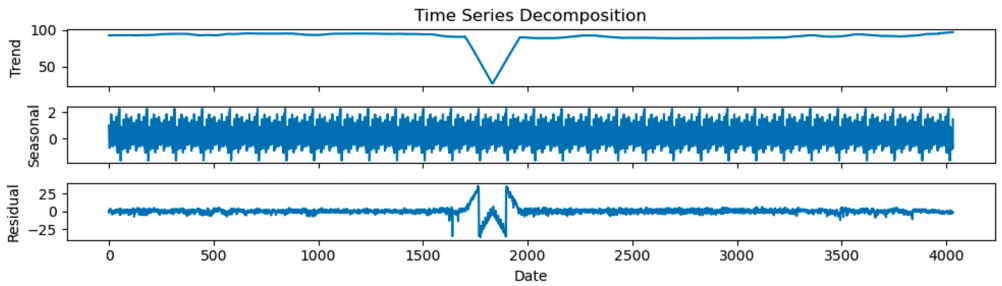
This dataset shows striking differences across its splits, with the training data containing extreme outliers as noted in Table 1. The full dataset exhibits a strong negative skewness (−4.76) and a high kurtosis (22.55), characteristic of a left-skewed distribution with a heavy tail of low values. This is further reflected in the much larger standard deviation for the training set (13.61) compared with the validation (2.85) and test (1.20) sets. The statistics suggest the training set includes anomalous low-value outliers (for example, periods of server downtime or malfunction), while the validation and test sets reflect more stable conditions with high CPU usage (means of 91.47 and 94.59). Consequently, a model's performance depends heavily on its robustness to these extreme training-time anomalies.

The CPU Utilization dataset shows a stark contrast between a stable operating state and a period of severe anomalous activity, confined entirely to the training set in Figure 5(a). The TS plot clearly marks these outliers as drastic dips, which pull the training mean downward, while dramatically increasing its variance and negative skewness, as reported in the statistics table. The trend component in Figure 5(b) features a major dip that aligns with this anomaly, and the residuals plot indicates heightened volatility during the same period. Despite these extremes, a clear, repeating seasonal pattern remains evident in the decomposition plot, suggesting underlying regular usage. The ACF and PACF plots on the left side of Figure 5(c) show significant correlations at short lags, confirming short-term dependence. The histogram on the right side of Figure 5(c) reveals a bi-modal distribution: a soft left tail of anomalies alongside a dense, near-normal body of stable readings. This encapsulates the dataset's main challenge: building a robust model that captures the dominant pattern without being unduly swayed by rare, extreme events.

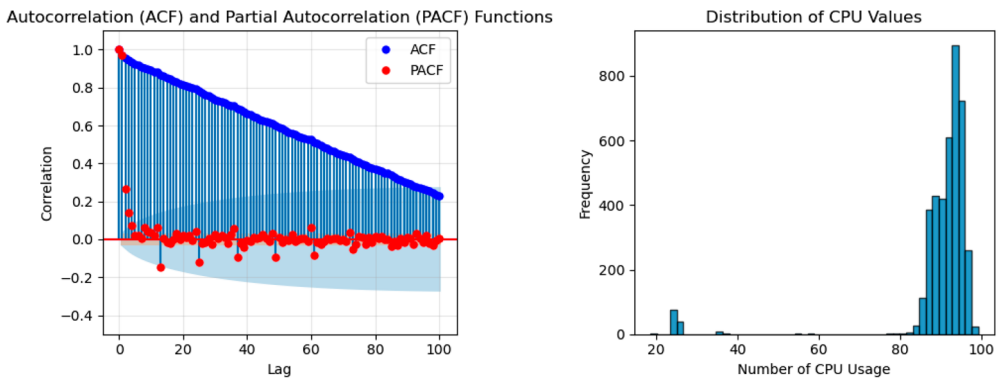
Our goal is to predict the test set using models trained on the training and validation sets, which contain outliers. We want to determine whether the presence of this data affects the prediction results. Some algorithms perform well, so the preparation of outliers is not necessary. Consequently, they facilitate an easy prediction process with high accuracy.



(a) CPU TS.



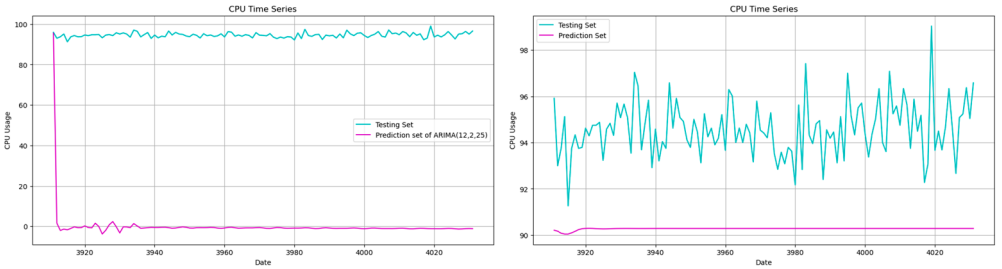
(b) Decompositions.



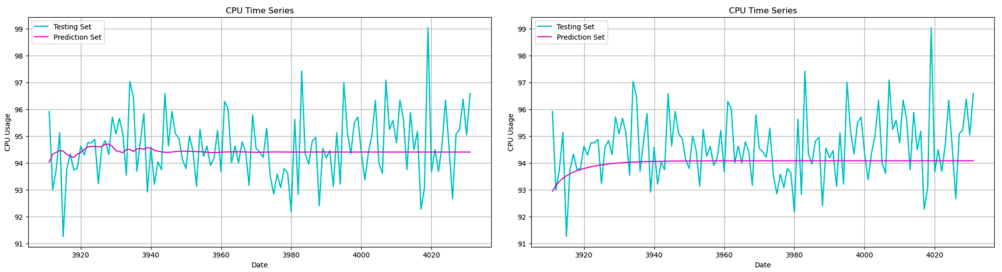
(c) ACF/PACF and histogram.

Figure 5. CPU TS visualization. (a) CPU TS. (b) Decompositions and (c) ACF/PACF and histogram.

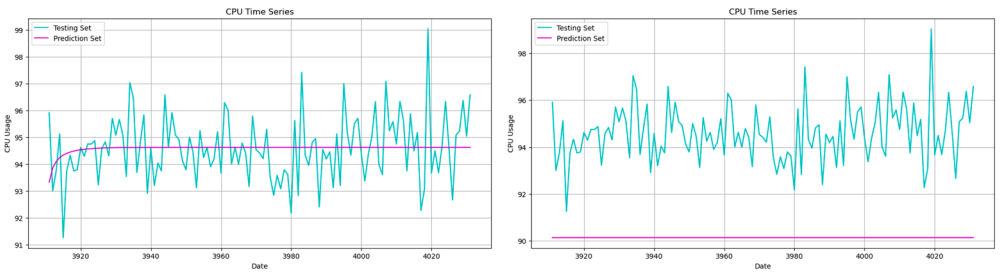
For this example, first and twelfth-order differencing are performed, followed by normalization for the ARIMA model shown on the left side of Figure 6(a). This TS exhibits persistent patterns, making it challenging to predict using ARIMA. Also, we apply similar ML procedures to the sunspot data, with a few exceptions. The LSTM architecture, with the



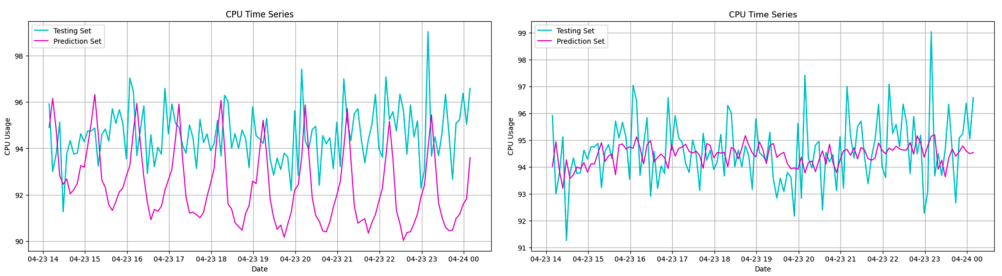
(a) ARIMA and Elman.



(b) LSTM and GRU.



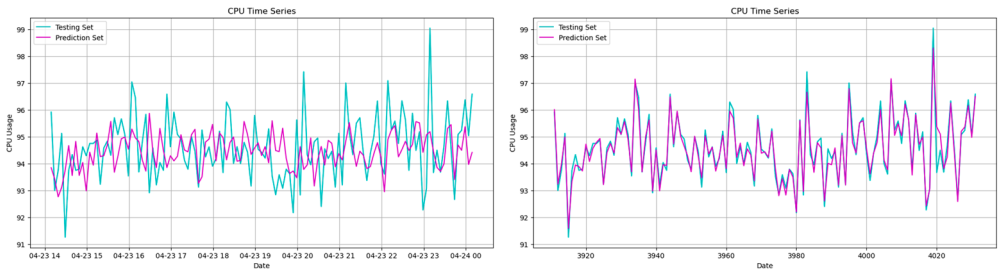
(c) Bidirectional LSTM and Deep Elman.



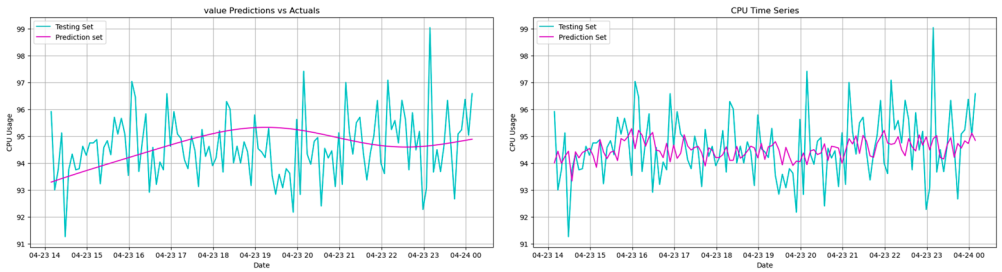
(d) DeepAR and RF.

Figure 6. Prediction results for CPU TS. (a) ARIMA and Elman. (b) LSTM and GRU. (c) Bidirectional LSTM and Deep Elman. (d) DeepAR and RF. (e) XGBoost and LightGBM. (f) Prophet and WBT. (g) TFT and N-BEATS. (h) N-HiTS and SVD. (i) EMD and EEMD. (j) DFFNN and BNN. (k) CNN and TCN. (l) STFT and RL and (m) GNN and LLM.

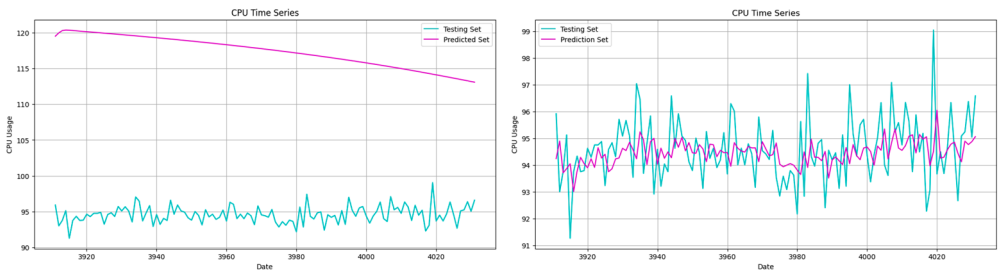
corresponding plot on the left side of Figure 6(b), consists of two HLs containing 20 and 10 neurons, followed by a global max pooling layer and a DeLa. The DFFNN, with the prediction on the left side of Figure 6(j), has three HLs with 50, 40, and 30 neurons, along with a DeLa.



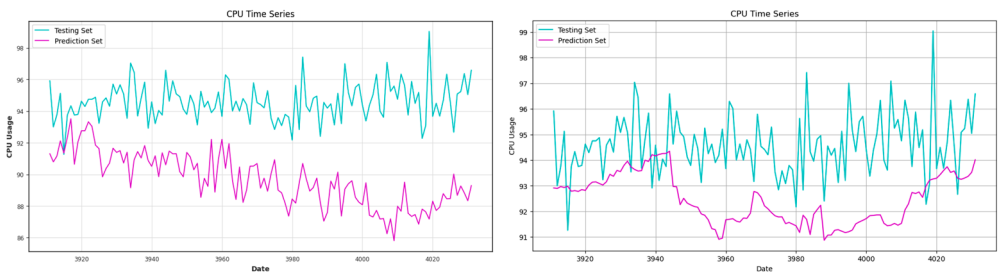
(e) XGBoost and LightGBM.



(f) Prophet and WBT.



(g) TFT and N-BEATS.

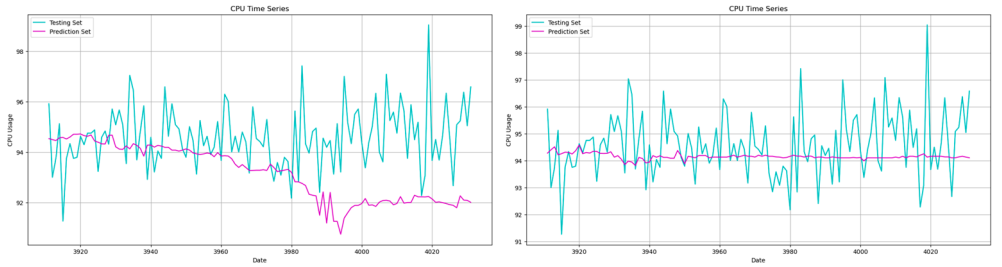


(h) N-HiTS and SVD.

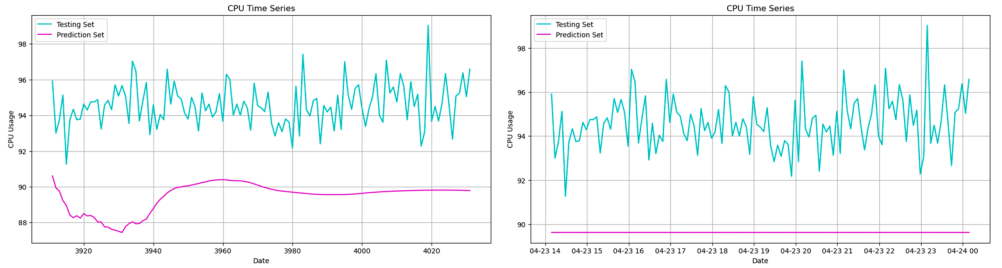
Figure 6. Continued.

The total number of samples that the LLM algorithm can predict for this example is 121, identical to the test set size. However, predictions become more diverse after the first 70 samples. Consequently, the remaining predicted samples are ignored for the plot on the right side of Figure 6(m) and for the metrics in Table 2.

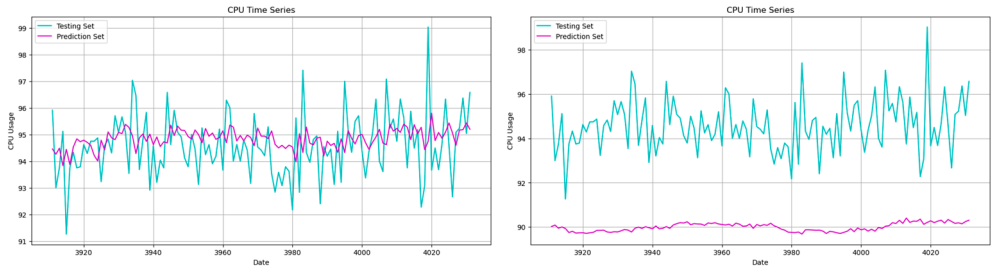
The forecasting plots are presented in Figure 6. The LightGBM model, with the plot on the right side of Figure 6(e), demonstrates excellent predictive accuracy, even in the presence of outliers, while the GNN (the left side of Figure 6(m)), WBT (the right side of Figure 6(f)),



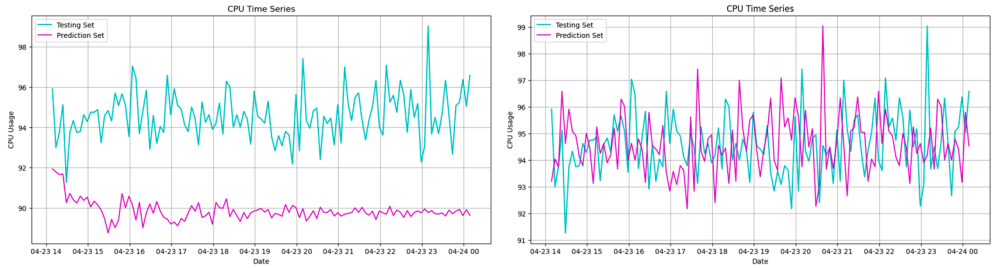
(i) EMD and EEMD.



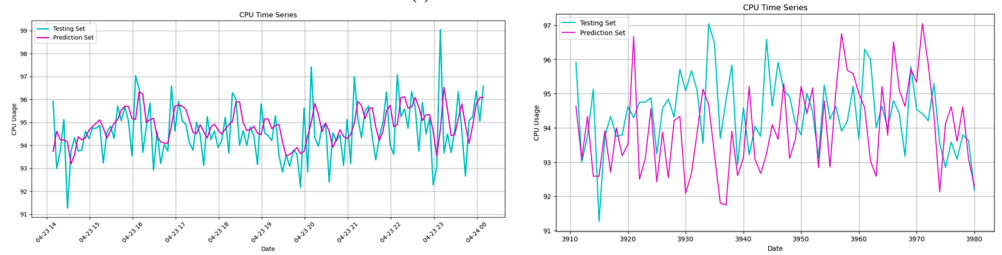
(j) DFFNN and BNN



(k) CNN and TCN.



(l) STFT and RL.



(m) GNN and LLM.

Figure 6. Continued.

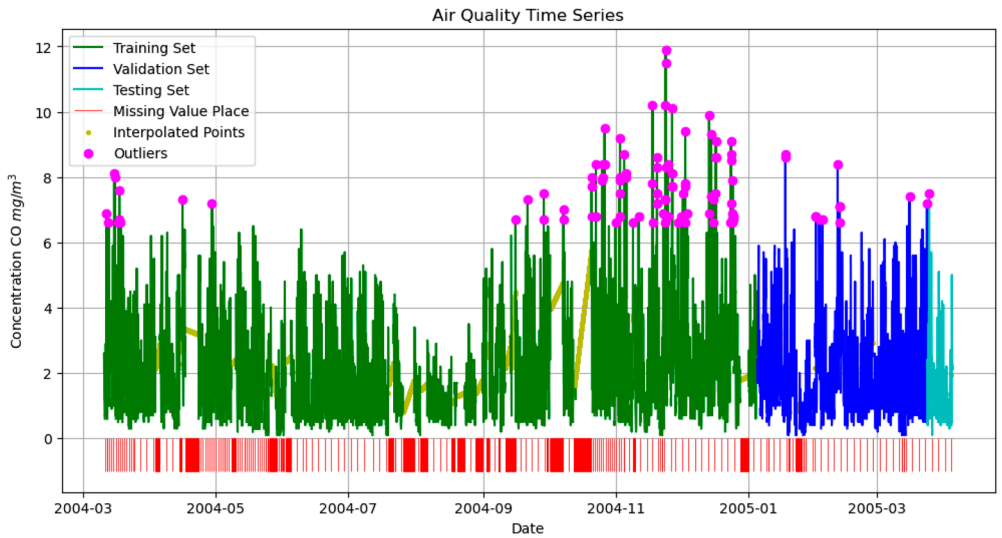
N-BEATS (the right side of Figure 6(g)), RF (the right side of Figure 6(d)), XGBoost (the left side of Figure 6(e)), CNN (the left side of Figure 6(k)), RL (the right side of Figure 6(l)), and DeepAR (the left side of Figure 6(d)) models provide acceptable predictions, ranked respectively in terms of accuracy and error minimization. The error rate of GPT-2 predictions, shown in Figure 6(m), is only slightly higher than that of models with highly accurate predictions. However, this small difference in the error rate does substantially change the overall result accuracy performance. Algorithms such as bidirectional LSTM (the left side of Figure 6(c)), LSTM (the left side of Figure 6(b)), GRU (the right side of Figure 6(b)), Prophet (the left side of Figure 6(f)), and EEMD (the right side of Figure 6(i)) exhibit low error rates; however, their predictions may not be suitable and have no information. ARIMA identifies the first point with low error, but after that, it underperforms, similar to other methods–such as TFT (the left side of Figure 6(g)), BNN (the right side of Figure 6(j)), and TCN (the right side of Figure 6(k))–producing noticeably inaccurate values. N-HiTS (the left side of Figure 6(h)), SVD (the right side of Figure 6(h)), EMD (the left side of Figure 6(i)), and STFT (the left side of Figure 6(l)) accurately predict early samples with low error, but their performance declines over time and they diverge from the actual values.

The presence of severe contextual anomalies poses a critical test of model robustness. We retain these outliers rather than remove them, revealing clear differences in how algorithms handle corrupt training data. Tree-based ensemble methods (LightGBM, RF, XGBoost) and WBT demonstrate superior robustness, likely due to internal feature selection and averaging that prevent single anomalous samples from unduly influencing the model. Similarly, N-BEATS, CNN, and TCN architectures learn to ignore localized noise and emphasize broader temporal patterns. In contrast, models highly sensitive to the exact training data distribution or that minimize error in a deterministic manner struggle. ARIMA's performance collapses because its parameters are skewed by the anomalous period. TFT and BNNs–despite their complexity–likely overfit to the non-stationarity introduced by the outliers. The poor performance of STFT, EMD, and EEMD suggests that the anomalies disrupt the intrinsic frequency components these methods rely on. Overall, this clear stratification shows that for real-world data prone to faults, choosing a robust algorithm is paramount and often outweighs the need for extensive outlier preprocessing.

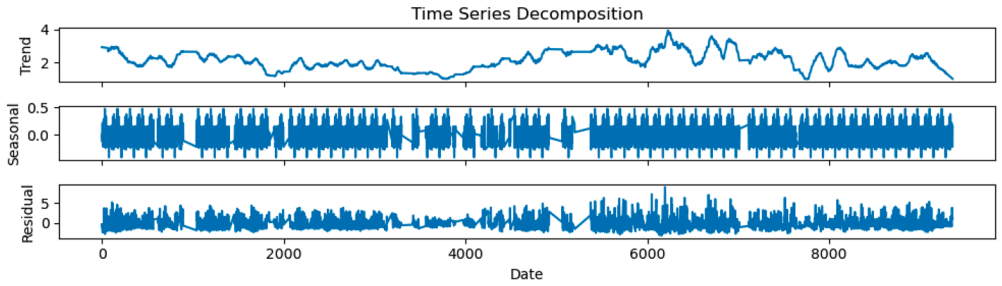
9.3. Air CO concentration TS with missing data

For this subsection, we analyze the real hourly averaged concentration of carbon monoxide (CO) in mg/m^3, as displayed in Figure 7, adapted from Repository (2019). The sensor was located in an Italian city's significantly polluted area at road level. Data were recorded from March 2004 to February 2005 from air quality chemical sensor devices deployed in the field. The training, validation, and testing sets contain 7204, 1872, and 281 samples, respectively. The dataset has 1683 missing values and 102 outliers. The employed filling method is first-order spline interpolation, as higher-order splines produced negative values, which are not acceptable for a positive variable.

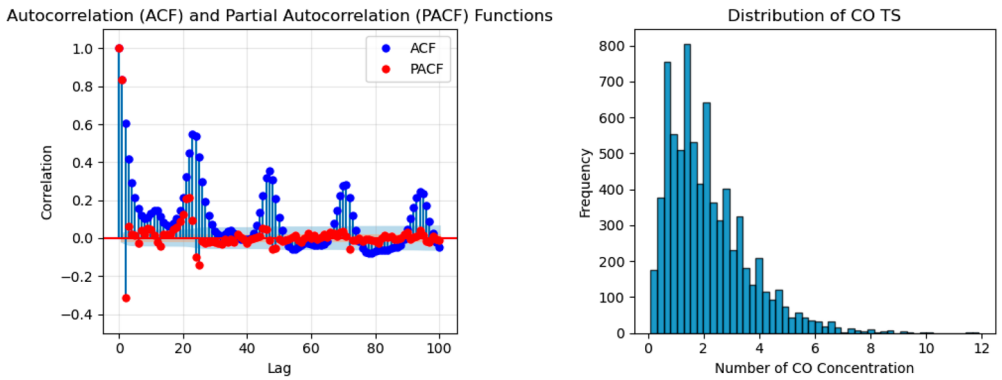
The CO data show a consistent, positively skewed distribution across all splits, which is typical for environmental concentration data that cannot go below zero. The mean decreases slightly from the training set to the test set (2.26 to 1.71) in Table 1, suggesting a modest improving trend or a seasonal pattern where the test period features lower concentration levels. The standard deviations and skewness values are similar across splits, indicating that the overall distribution shape is preserved. As a result, the test set appears to be a plausible,



(a) CO TS.



(b) Decompositions.



(c) ACF/PACF and histogram.

Figure 7. CO Concentration TS visualization. (a) CO TS. (b) Decompositions and (c) ACF/PACF and histogram.

though somewhat less extreme, subset of the training conditions. This relative consistency suggests a more straightforward forecasting task compared with the shifts seen in the sunspot and CPU datasets.

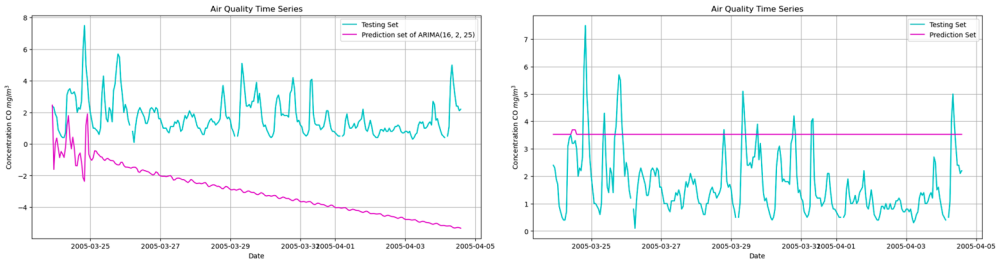
The CO concentration TS in Figure 7(a) highlights the challenge of forecasting a positively skewed environmental variable with substantial missing data, as shown by gaps in the original series that were interpolated. The decomposition in Figure 7(b) reveals a smoothly varying trend with strong, regular seasonality–likely diurnal and weekly cycles–though the seasonal component's continuity is partially maintained by interpolation across missing periods. The ACF on the left side of Figure 7(c) shows a U-shaped pattern, decaying slowly and oscillating, which is typical of a series with a strong trend and seasonal structure. This indicates that past values exert long-lasting, periodic influence on future observations, a fundamental feature that forecasting models must capture to perform well. The histogram on the right side of Figure 7(c) is clearly right-skewed, with most concentrations clustering near 0 to 4 mg/m^3 and a long tail of higher values, consistent with the high skewness and kurtosis reported in the table.

The same preparations to the CPU TS are done for predicting with ARIMA model shown on the left side of Figure 8(a). Furthermore, we apply similar ML algorithms to the sunspot TS, with the exception of the Prophet algorithm, as shown in Figure 8. The additive Prophet model performs effectively on this data without the need for MCMC sampling. The algorithms that accurately predicted the entire testing set, ranked by the lowest error rates, are LightGBM (the right side of Figure 8(e)), GNN (the right side of Figure 8(k)), RF (the right side of Figure 8(d)), WBT (the right side of Figure 8(f)), XGBoost (the left side of Figure 8(e)), TCN (the left side of Figure 8(j)), CNN (the right side of Figure 8(i)), Prophet (the left side of Figure 8(f)), SVD (the right side of Figure 8(h)), GRU (the right side of Figure 8(b)), DeepAR (the left side of Figure 8(d)), DFFNN (the left side of Figure 8(j)), RL (the right side of Figure 8(l)), LSTM (the left side of Figure 8(b)), LLM (the right side of Figure 8(m)), bidirectional LSTM (the left side of Figure 8(c)), and N-BEATS (the right side of Figure 8(g)).

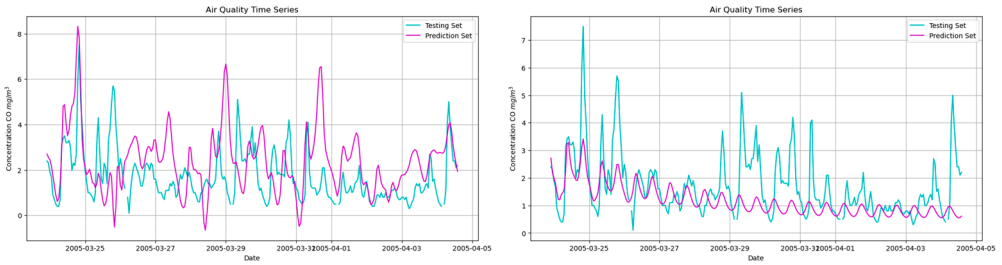
Although the LLM can predict up to 90 samples for this example, we currently use only the first 70 samples of predictions for the plot on the right side of Figure 4(m) and for the metrics reported in Table 2. The reason is that the later predictions diverge significantly from their corresponding real values in the test set. In other words, when we operate only with the maximum token limit of 1024, the model struggles to predict sequences that are very long. In comparison, the other models predict 281 samples, and their errors are calculated accordingly.

Some algorithms like Prophet, GRU, DFFNN, and bidirectional LSTM provide suitable predictions for the initial data points; however, they struggle to predict long-term future values. ARIMA starts the forecast well but later produces inaccurate predictions. Elman (the right side of Figure 8(a)), Deep Elman (the right side of Figure 8(c)), and BNN (the right side of Figure 8(j)) do not perform well from the outset and tend to predict all points uniformly. TFT (the left side of Figure 8(g)) underperforms overall, whereas N-HiTS (the left side of Figure 8(h)) accurately identifies the overall range of change from start to finish, but its values are not sufficiently accurate. EMD (the left side of Figure 8(i)), EEMD (the right side of Figure 8(i)), and STFT (the left side of Figure 8(l)) produce smooth, uniform forecast lines with small fluctuations, yet their predictions remain unreliable.

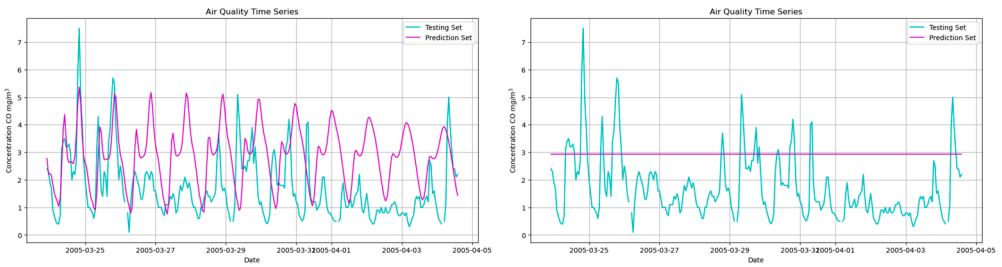
The presence of substantial missing data (1683 values), likely due to sensor malfunctions or maintenance periods, introduces a significant challenge that directly affects model performance. Our choice of first-order spline interpolation effectively preserves the overall



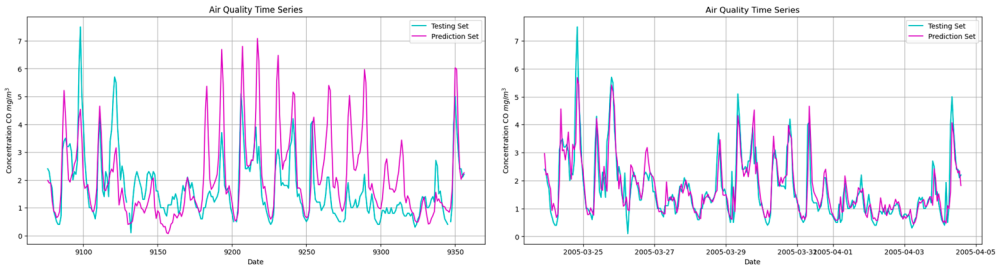
(a) ARIMA and Elman.



(b) LSTM and GRU.



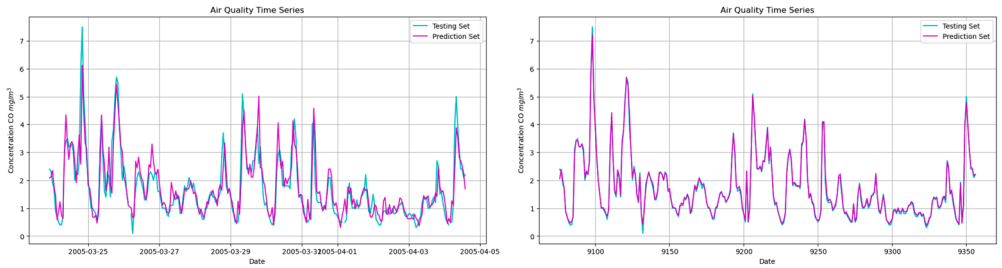
(c) Bidirectional LSTM and Deep Elman.



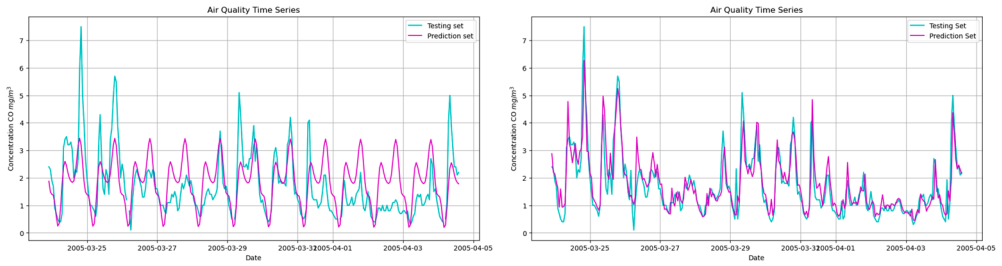
(d) DeepAR and RF.

Figure 8. Prediction results for air quality TS. (a) ARIMA and Elman. (b) LSTM and GRU. (c) Bidirectional LSTM and Deep Elman. (d) DeepAR and RF. (e) XGBoost and LightGBM. (f) Prophet and WBT. (g) TFT and N-BEATS. (h) N-HiTS and SVD. (i) EMD and EEMD. (j) DFFNN and BNN. (k) CNN and TCN. (l) STFT and RL and (m) GNN and LLM.

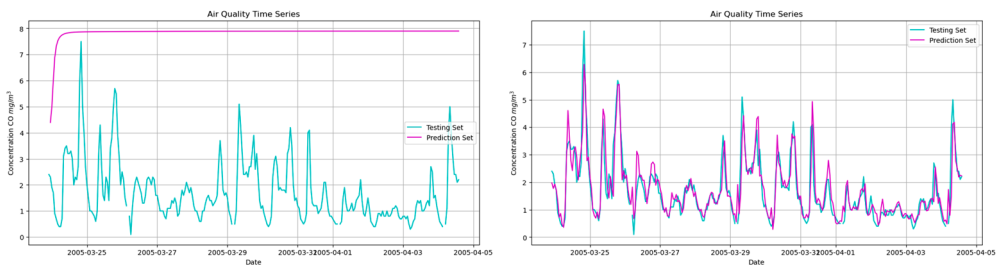
trend and seasonality without adding negative artifacts, but it naturally smooths over the true volatility that occurs during missing periods. This imputation strategy benefits models like LightGBM, RF, and Prophet, which are robust to smooth, interpolated values and can leverage strong seasonal signals. In contrast, models that rely heavily on the authenticity of the



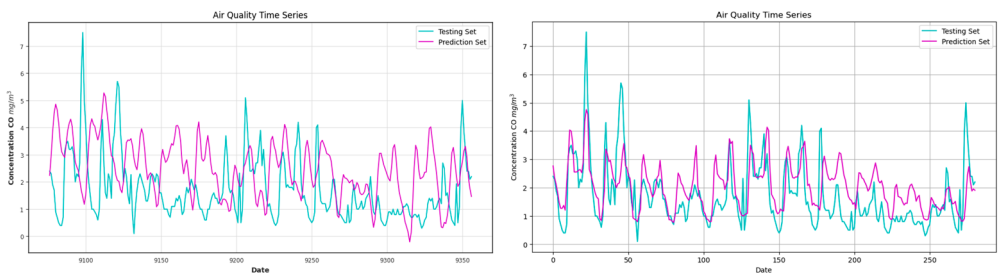
(e) XGBoost and LightGBM.



(f) Prophet and WBT.



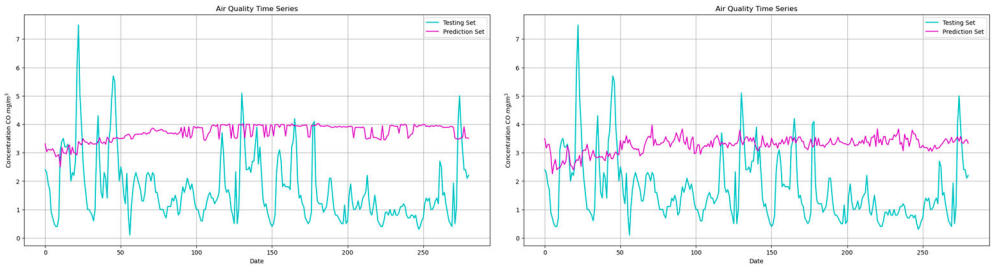
(g) TFT and N-BEATS.



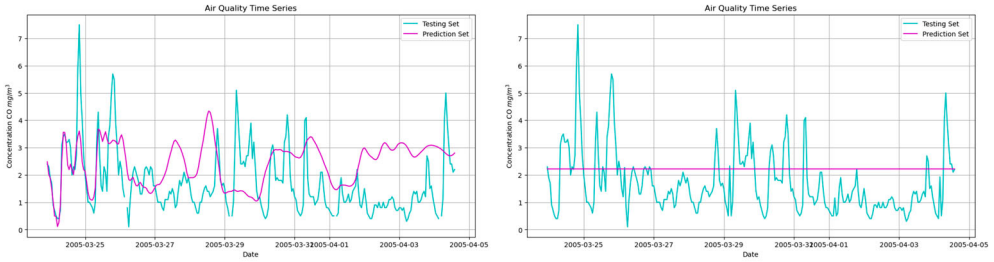
(h) N-HiTS and SVD.

Figure 8. Continued.

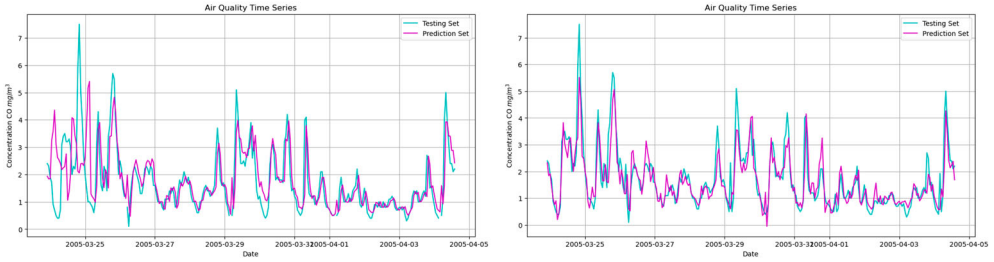
input signal or require a precise underlying distribution–such as BNNs and TFT–struggle. The decomposition-based methods (EMD, EEMD, STFT) falter due to their dependence on the intrinsic data structure, which the interpolation process alters. This analysis highlights a key practical insight: the best forecasting model is determined not only by the algorithm itself but also by its compatibility with data preprocessing steps, especially how missing values are handled.



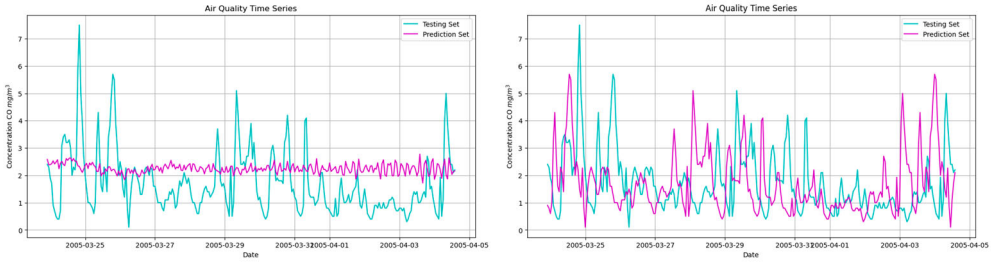
(i) EMD and EEMD.



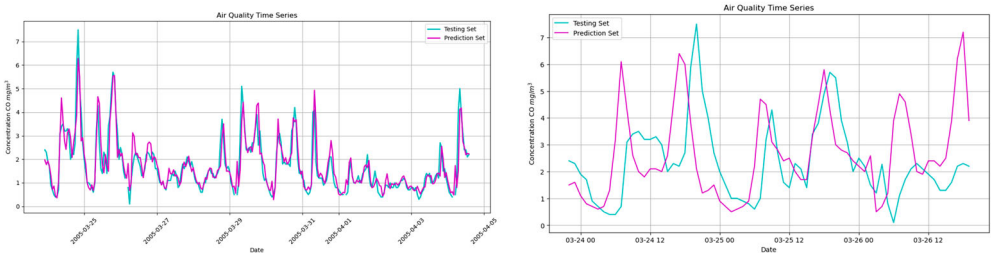
(j) DFFNN and BNN.



(k) CNN and TCN.



(l) STFT and RL.



(m) GNN and LLM.

Figure 8. Continued.

10. Conclusion

In this paper, we conduct a comprehensive review of ML algorithms suitable for TS forecasting. Ultimately, we generate a synthetic TS and select three real-world TS datasets for our analysis: sunspot data with long-term cyclical patterns, CPU utilization data containing outliers, and CO concentration data with missing values.

Hyper-parameters are determined through systematic experimentation for each algorithm, ensuring that the settings reflect the behavior of the methods on the given data. Across the datasets, tree-based ensemble methods–LightGBM, RF, and XGBoost–consistently deliver superior performance, effectively addressing TS-specific challenges without the need for extensive preprocessing or normalization. These methods prove robust to outliers and missing data while maintaining computational efficiency, and neural architectures such as GNN, N-BEATS, TCN, CNN, and DeepAR also demonstrate strong predictive capabilities, particularly in capturing complex temporal patterns.

There are notable differences in computational requirements among the approaches. RNN variants, TFT, and BNN demand substantial computational resources with only modest gains in performance, whereas Prophet, N-BEATS, N-HiTS, and decomposition techniques like SVD and EMD offer a more favorable balance between accuracy and efficiency. EEMD, STFT, and LLM require moderate computational time, placing them in a middle ground between the most demanding and the most efficient methods.

The study highlights that the specific characteristics of the TS data profoundly influence model performance. Tree-based ensembles repeatedly exhibit robustness and accuracy across challenging scenarios due to internal feature selection, resistance to over-fitting, and their ability to model non-linear relationships without being misled by outliers or interpolated missing values. In contrast, more complex models such as TFT and BNNs show high sensitivity to data quality and require extensive tuning, which can reduce reliability in practice. Decomposition methods (EMD, EEMD, STFT) and simpler RNNs (Elman, GRU) struggle when missing values or outliers disrupt the intrinsic data structure or when extrapolation to regime shifts–such as the sunspot solar minimum–becomes necessary.

The study also highlights algorithm-specific strengths and limitations. While some methods (Elman, GRU, bidirectional LSTM, Deep Elman, Prophet, and DFFNN) capture initial patterns effectively, they often fail to maintain accuracy over longer forecasting horizons. Other approaches (N-HiTS, EMD, EEMD, STFT, and RL) identify general trends but lack precision in point predictions.

A key takeaway is that algorithmic complexity does not guarantee better performance; robustness, efficiency, and compatibility with data imperfections often determine real-world forecasting success. The strengths and limitations by method reveal that early-pattern models (Elman, GRU, bidirectional LSTM, Deep Elman, Prophet, and DFFNN) excel at capturing initial patterns but may lose accuracy over longer horizons, while general-trend methods (N-HiTS, EMD, EEMD, STFT, RL) can identify broader trends but may lack precision in point forecasts.

This paper targets practitioners working with TS data and offers guidance on selecting an appropriate analysis method. Our results show that tree-based ensembles, GNN, N-BEATS, CNN, TCN, DeepAR, and RL models deliver strong predictive performance on testing sets, including those with outliers and missing data. Other algorithms are not inherently unsuitable; they simply require careful hyper-parameter tuning and feature engineering. For practitioners dealing with similar TS data, tree-based ensembles are recommended as

a robust starting point due to their consistent performance, computational efficiency, and resilience to data quality issues. When complex temporal patterns are present, neural architectures such as GNN, N-BEATS, and TCN provide additional capabilities, though with higher computational costs. This study contributes to the TS forecasting literature by offering empirical evidence across diverse data characteristics and provides practical guidance on algorithm selection based on data properties, computational constraints, and accuracy needs.

To build on these findings, future research could pursue several directions. Controlled simulation studies that vary properties such as noise levels, missing data mechanisms, outlier types, and trend-seasonality interactions would offer a more granular understanding of each algorithm's strengths and weaknesses and isolate the impact of individual data characteristics on performance. Additionally, developing automated, data-centric pipelines that recommend both the optimal algorithm and the necessary preprocessing steps based on initial data diagnostics would be a significant advancement for practitioners. Finally, exploring hybrid models that blend the robustness of tree-based ensembles with the pattern-recognition capabilities of DL could unlock new performance frontiers for the most complex forecasting scenarios.

Author contributions

Seyedeh Azadeh Fallah Mortezanejad: Methodology, Conceptualization, Validation, Writing-original draft, Writing-review and editing; **Ruochen Wang**: Supervision, Methodology, Investigation, Writing-original draft, Writing-review and editing.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Innovative Research Group Project of the National Natural Science Foundation of China [grant number 51975253].

ORCID

Seyedeh Azadeh Fallah Mortezanejad http://orcid.org/0000-0002-4358-0768
Ruochen Wang https://orcid.org/0000-0003-2014-4282

References

Ab Aziz, M. F., Mostafa, S. A., Foozy, C. F. M., Mohammed, M. A., Elhoseny, M., & Abualkishik, A. Z. (2021). Integrating Elman recurrent neural network with particle swarm optimization algorithms for an improved hybrid training of multidisciplinary datasets. *Expert Systems with Applications*, *183*, 115441. https://doi.org/10.1016/j.eswa.2021.115441

Ahn, H. K., & Park, N. (2021). Deep RNN-based photovoltaic power short-term forecast using power IoT sensors. *Energies*, *14*(2), 436. https://doi.org/10.3390/en14020436

Bengio, Y., Paolo, F., & Marco, G. (1970). Recurrent neural networks for adaptive temporal processing. In *Proceedings of the 6th Italian Workshop on Parallel Architectures and Neural Networks WIRN93* (pp. 85–117). World Scientific Publishing Co.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Brockwell, P. J., & Davis, R. A. (2002). *Introduction to time series and forecasting*. Springer.

Bui, K. H. N., Cho, J., & Yi, H. (2022). Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Applied Intelligence*, *52*(3), 2763–2774. https://doi.org/10.1007/s10489-021-02587-w

Cao, Q., Wu, Y., Yang, J., & Yin, J. (2023). Greenhouse temperature prediction based on time-series features and LightGBM. *Applied Sciences*, *13*(3), 1610. https://doi.org/10.3390/app13031610

Casado-Vara, R., Martin del Rey, A., Pérez-Palau, D., & Corchado, J. M. (2021). Web traffic time series forecasting using LSTM neural networks with distributed asynchronous training. *Mathematics*, *9*(4), 421. https://doi.org/10.3390/math9040421

Challu, C., Olivares, K. G., Oreshkin, B. N., Ramirez, F. G., Canseco, M. M., & Dubrawski, A. (2023). N-HiTS: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 37, pp. 6989–6997). Association for the Advancement of Artificial Intelligence.

Chen, T. (2015). *XGBoost: Extreme gradient boosting*. R Package Version 0.4-2 1 (4) 1–4.

Chen, Y., Ren, K., Wang, Y., Fang, Y., Sun, W., & Li, D. (2023). Contiformer: Continuous-time transformer for irregular time series modeling. In *Advances in Neural Information Processing Systems 36-37th Conference on Neural Information Processing Systems, NeurIPS 36* (pp. 47143–47175). Curran Associates, Inc.

Chiarot, G., & Claudio, S. (2023). Time series compression survey. *ACM Computing Surveys*, *55*(10), 1–32. https://doi.org/10.1145/3560814

Chung, J., Caglar, G., KyungHyun, C., & Yoshua, B. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555.

Cristian, C., Kin G, O., Boris N, O., Federico, G., Max, M. C., & Artur, D. (2022). N-HiTS: Neural hierarchical interpolation for time series forecasting. arXiv:2201.12886.

Daraghmeh, M., Agarwal, A., Manzano, R., & Zaman, M. (2021). Time series forecasting using facebook prophet for cloud resource management. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)* (pp. 1–6). IEEE (Institute of Electrical and Electronics Engineers).

Dash, S., Yale, A., Guyon, I., & Bennett, K. P. (2020). Medical time-series data generation using generative adversarial networks. In *Artificial Intelligence in Medicine: 18th International Conference on Artificial Intelligence in Medicine, AIME 2020, Minneapolis, MN, USA, August 25–28, 2020, Proceedings 18* (pp. 382–391). Springer Nature.

Ernst, D., & Louette, A. (2024). *Introduction to reinforcement learning*. University of Liège (ULiège).

Falchi, F., Girardi, M., Gurioli, G., Messina, N., Padovani, C., & Pellegrini, D. (2023). Deep learning and structural health monitoring: A TFT-based approach for anomaly detection in masonry towers. SSRN: 4679906.

Faouzi, J., & Janati, H. (2020). PyTS: A Python package for time series classification. *Journal of Machine Learning Research*, *21*(46), 1–6.

Hartanto, A. D., Kholik, Y. N., & Pristyanto, Y. (2023). Stock price time series data forecasting using the light gradient boosting machine (LightGBM) model. *JOIV: International Journal on Informatics Visualization*, *7*(4), 2270–2279. https://doi.org/10.62527/joiv.7.4.1740

Hindarto, D. (2023). Comparison of RNN architectures and non-RNN architectures in sentiment analysis. *Sinkron: Jurnal Dan Penelitian Teknik Informatika*, *7*(4), 2537–2546. https://doi.org/10.33395/sinkron.v8i4

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Jang, J. G., Dongjin, C., Jinhong, J., & Kang, U. (2018). Zoom-SVD: Fast and memory efficient method for extracting key patterns in an arbitrary time range. In *In Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 1083–1092). Association for Computing Machinery (ACM).

Jeffrey, N., Gunawan, A. A. S., & Kurniawan, A. (2023). Development of multivariate stock prediction system using N-Hits and N-BEATS. In *Proceedings of the Computational Methods in Systems and Software* (pp. 50–63). Springer.

Joseph, M. (2022). *Modern time series forecasting with Python: Explore Industry-ready time series forecasting using modern mchine learning and deep learning*. Packt Publishing, Limited.

Ke, G., Qi, M., Thomas, F., Taifeng, W., Wei, C., Weidong, M., & Tie-Yan, L. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30 – Proceedings of the 2017 Conference 2017*. Curran Associates, Inc.

Liang, F., Qian, C., Yu, W., Griffith, D., & Golmie, N. (2022). Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, *2022*(1), 9261537. https://doi.org/10.1155/wcm.v2022.1

Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748–1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

Lim, B., Sercan Omer, A., Nicolas, L., & Tomas, P. (2019). Temporal fusion transformers for interpretable multi-horizon time series forecasting. arXiv:1912.09363.

Lim, B., & Zohren, S. (2021). Time-series forecasting with deep learning: A survey. *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200209. https://doi.org/10.1098/rsta.2020.0209

Lin, C., Weng, K., Lin, Y., Zhang, T., He, Q., & Su, Y. (2022). Time series prediction of dam deformation using a hybrid STL–CNN–GRU model based on sparrow search algorithm optimization. *Applied Sciences*, *12*(23), 11951. https://doi.org/10.3390/app122311951

Lin, S., Clark, R., Birke, R., Schönborn, S., Trigoni, N., & Roberts, S. (2020). Anomaly detection for time series using VAE-LSTM hybrid model. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Vol. 2025, pp. 4322–4326). IEEE (Institute of Electrical and Electronics Engineers).

Lindemann, B., Müller, T., Vietz, H., Jazdi, N., & Weyrich, M. (2021). A survey on long short-term memory networks for time series prediction. *Procedia Cirp*, *99*, 650–655. https://doi.org/10.1016/j.procir.2021.03.088

Liu, Y., Zhang, H., Chenyu, L., Xiangdong, H., Jianmin, W., & Mingsheng, L. (2024). Timer: Generative pre-trained transformers are large time series models. arXiv:2402.02368.

Liu, Z., Zhengtong, Z., Jing, G., & Xu, C. (2021). Forecast methods for time series data: A survey. *IEEE Access*, *9*, 91896–91912. https://doi.org/10.1109/ACCESS.2021.3091162

López Santos, M., García-Santiago, X., Echevarría Camarero, F., Blázquez Gil, G., & Carrasco Ortega, P. (2022). Application of temporal fusion transformer for day-ahead PV power forecasting. *Energies*, *15*(14), 5232. https://doi.org/10.3390/en15145232

Maiti, R., Balagopal G, M., & Anand, A. (2024). Ensemble empirical mode decomposition based deep learning models for forecasting river flow time series. *Expert Systems with Applications*, *255*, 124550. https://doi.org/10.1016/j.eswa.2024.124550

Mateus, B. C., Mendes, M., Farinha, J. T., Assis, R., & Cardoso, A. M. (2021). Comparing LSTM and GRU models to predict the condition of a pulp paper press. *Energies*, *14*(21), 6958. https://doi.org/10.3390/en14216958

Mir, A. A., Çelebi, F. V., Alsolai, H., Qureshi, S. A., Rafique, M., Alzahrani, J. S., & Hamza, M. A. (2022). Anomalies prediction in radon time series for earthquake likelihood using machine learning-based ensemble model. *IEEE Access*, *10*, 37984–37999. https://doi.org/10.1109/ACCESS.2022.3163291

Mortezanejad, S. A. F., Wang, R., & Mohammad-Djafari, A. (2025). Physics-informed neural networks with unknown partial differential equations: An application in multivariate time series. *Entropy*, *27*(7), 682. https://doi.org/10.3390/e27070682

Noh, S. H. (2021). Analysis of gradient vanishing of RNNs and performance comparison. *Information*, *12*(11), 442. https://doi.org/10.3390/info12110442

Numenta (2021). Numenta Anomaly Benchmark (NAB). Retrieved November 18, 2024, from https://github.com/numenta/NAB?ref=hackernoon.com.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv:1905.10437.

Paul, A., Mukherjee, D. P., Das, P., Gangopadhyay, A., Chintha, A. R., & Kundu, S. (2018). Improved random forest for classification. *IEEE Transactions on Image Processing*, *27*(8), 4012–4024. https://doi.org/10.1109/TIP.2018.2834830

Pavlyshenko, B. M. (2019). Machine-learning models for sales time series forecasting. *Data*, *4*(1), 15. https://doi.org/10.3390/data4010015

Qin, R., & Wang, Y. (2023). ImputeGAN: Generative adversarial network for multivariate time series imputation. *Entropy*, *25*(1), 137. https://doi.org/10.3390/e25010137

Rady, E., Fawzy, H., & Fattah, A. M. A. (2021). Time series forecasting using tree based methods. *Journal of Statistics Applications & Probability*, *10*(1), 229–244. https://doi.org/10.18576/jsap

Repository, U. M. L. (2019). Air quality. Retrieved November 18, 2024, from https://archive.ics.uci.edu/dataset/360/air+quality.

Rozenberg, R., Gesnouin, J., & Moutarde, F. (2021). Asymmetrical Bi-RNN for pedestrian trajectory encoding. arXiv:2106.04419.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

Schirmer, P. A., & Mporas, I. (2024). PyDTS: A Python toolkit for deep learning time series modelling. *Entropy*, *26*(4), 311. https://doi.org/10.3390/e26040311

Sheridan, R. P., Wang, W. M., Liaw, A., Ma, J., & Gifford, E. M. (2016). Extreme gradient boosting as a method for quantitative structure–activity relationships. *Journal of Chemical Information and Modeling*, *56*(12), 2353–2360. https://doi.org/10.1021/acs.jcim.6b00591

Tan, M., Merrill, M., Gupta, V., Althoff, T., & Hartvigsen, T. (2024). Are language models actually useful for time series forecasting? In *Advances in Neural Information Processing Systems 37-38th Conference on Neural Information Processing Systems 37* (pp. 60162–60191). Curran Associates, Inc.

Tang, C., Xu, L., Yang, B., Tang, Y., & Zhao, D. (2023). GRU-based interpretable multivariate time series anomaly detection in industrial control system. *Computers & Security*, *127*, 103094. https://doi.org/10.1016/j.cose.2023.103094

Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., & Holtz, C. (2020). Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, *21*(118), 1–6.

Taylor, S. J., & Benjamin, L. (2017). Prophet: Forecasting at scale. *PeerJ Preprints*, *5*, e3190v2.

Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., & Troncoso, A. (2021). Deep learning for time series forecasting: A survey. *Big Data*, *9*(1), 3–21. https://doi.org/10.1089/big.2020.0159

Tsantekidis, A., Passalis, N., & Tefas, A (2022). Recurrent neural networks. In *Deep Learning for Robot Perception and Cognition* (pp. 101–115). Elsevier.

Ullah, W., Ullah, A., Haq, I. U., Muhammad, K., Sajjad, M., & Baik, S. W. (2021). CNN features with bi-directional LSTM for real-time anomaly detection in surveillance networks. *Multimedia Tools and Applications*, *80*(11), 16979–16995. https://doi.org/10.1007/s11042-020-09406-3

Valt, R. (2021). Sunspots Dataset. Retrieved November 18, 2024, https://www.kaggle.com/datasets/robervalt/sunspots?ref = hackernoon.com.

Yang, X., Yu, S., & Xinyang, C. (2024). Frequency-aware generative models for multivariate time series imputation. *Advances in Neural Information Processing Systems*, *37*, 52595–52623.

Zhang, C., Hu, D., & Yang, T. (2022). Anomaly detection and diagnosis for wind turbines using long short-term memory-based stacked denoising autoencoders and XGBoost. *Reliability Engineering & System Safety*, *222*, 108445. https://doi.org/10.1016/j.ress.2022.108445

Zhang, C., Wang, D., Wang, L., Guan, L., Yang, H., Zhang, Z., & Zhang, M. (2021). Cause-aware failure detection using an interpretable XGBoost for optical networks. *Optics Express*, *29*(20), 31974–31992. https://doi.org/10.1364/OE.436293

Zhang, L., Liu, P., Zhao, L., Wang, G., Zhang, W., & Liu, J. (2021). Air quality predictions with a semi-supervised bidirectional LSTM neural network. *Atmospheric Pollution Research*, *12*(1), 328–339. https://doi.org/10.1016/j.apr.2020.09.003

Zhang, W., Yin, C., Liu, H., Zhou, X., & Xiong, H. (2024). Irregular multivariate time series forecasting: A transformable patching graph neural networks approach. In R. Salakhutdinov et al. (Eds.), *Proceedings of the 41st International Conference on Machine Learning* (Vol. 235, pp. 60179–60196). PMLR.

Zhang, X., Chowdhury, R. R., Gupta, R. K., & Shang, J. (2024). Large language models for time series: A survey. arXiv:2402.01801.

Zhang, Y., Suzuki, G., & Shioya, H. (2022). Prediction and detection of sewage treatment process using N-BEATS autoencoder network. *IEEE Access*, *10*, 112594–112608. https://doi.org/10.1109/ACCESS.2022.3216924

Zhao, C., Huang, X., Li, Y., & Yousaf Iqbal, M. (2020). A double-channel hybrid deep neural network based on CNN and BiLSTM for remaining useful life prediction. *Sensors*, *20*(24), 7109. https://doi.org/10.3390/s20247109