The Predictions and Detections of Time Series in Wireless Communications Using Machine Learning Approaches

by

Sheng Liu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan-Dearborn
2023

Doctoral Committee:

Professor Weidong Xiang Chair
Associate Professor Jinhua Guo
Associate Professor Sridhar Lakshmanan
Associate Professor Paul Watta

Sheng. Liu

shengl@umich.edu

ORCID iD: 0000-0003-4625-6450

Dedication

I dedicate this work to my parents, whose unwavering love, support, and belief in my abilities have been the foundation upon which I have built my dreams. Thank you for instilling in me the value of hard work, perseverance, and the pursuit of knowledge. Your sacrifices have not gone unnoticed, and I am eternally grateful for everything you have done for me.

Acknowledgements

The journey of completing this dissertation has been both challenging and rewarding, and I would like to take this opportunity to express my deepest gratitude to those who have been instrumental in bringing this project to fruition .First and foremost, I extend my heartfelt appreciation to my advisor, Professor Weidong Xiang, for their invaluable guidance, expertise, and unwavering support throughout this process. Your mentorship has been instrumental in shaping my academic and professional growth, and I am eternally grateful for the opportunity to learn from you. My sincerest thanks go to the University of Michigan-Dearborn for providing me with the resources and facilities necessary for the completion of this research. Finally, I would like to acknowledge the countless others who have contributed to my personal and academic growth, and who have touched my life in various ways. Your support, encouragement, and belief in my abilities have made this journey possible. Thank you all for being a part of my journey and for helping me bring this dissertation to fruition.

Table of Contents

List of Tables

List of Figures

Abstract

Wireless channel prediction has been the subject of significant research interest in recent years; however, it remains an open topic. Both conventional mathematical and statistical models have proven incapable of forecasting time-varying channel variables within coherent time windows. This is due to the fact that the mechanisms of these models are designed to fit a range of scenarios for specific categories, targeting minimal mean square errors over the entire dataset, rather than time-consequent prediction within coherent windows. Thus, these models are not well-suited for predicting time-consequent patterns within coherent windows. Conversely, artificial neural network (ANN) based approaches have been proposed and investigated for wireless channel prediction. In principle, ANN can learn and identify patterns that are hidden in data. However, extracting and predicting wireless channel variables using ANN presents significant challenges for researchers. Additionally, the commonly utilized long-short term memory (LSTM) approach has not yielded satisfactory results due to the presence of interferences, outliers, and noise that cannot be avoided. The impact of noise on the performance of machine learning models in predicting future values in time series data can be substantial. In the context of long-term predictions, noise can be particularly problematic as small errors in the prediction can accumulate over time, leading to significant deviations between predicted and actual values. Noise is characterized by random variations or errors in the data that can obscure the patterns that machine learning models are designed to recognize and learn. Additionally, the noise can lead to overfitting of the model, whereby it fits not only the underlying patterns in the data but also the noise. This can result in poor generalization performance, where the model performs well on the training data but poorly on new, unseen data. In response to these concerns, two novel machine learning models have been designed. One model targets long-term time series predictions, while the other is designed for time series pattern detections. For the prediction model, we reconfigure the conventional LSTM cell by introducing new Infinite impulse response (IIR) gates into the LSTM cells, which are specifically designed to remove noise and interferences. These gates are self-adaptive through backpropagation during the training phase to optimize the model. To validate the proposed IIR-LSTM prediction model, experimental Global Position Systems (GPS) distance error datasets were utilized to assess the effectiveness of the model. In addition, the proposed model was evaluated alongside several commonly used time series forecast models, and the results demonstrate its ability to perform well on diverse datasets, with the best performance of long-term predictions achieved. For the pattern detection model, we proposed a novel model that utilizes the position information of the softmax layer to estimate the confidence of prediction and determine the patterns, if there were. To increase detection tolerance to noise, the positions of the most likely class and its adjacent classes were utilized. A dedicated standard deviation layer was introduced to calculate the position variance of the softmax outputs while assigning a set of weights to them to optimize the model prediction performance under various noise conditions. The model was tested by using both a simulated dataset and real wireless sensor data. The results demonstrate that the presented method achieves a higher detection accuracy, particularly under high noise conditions, when compared to the one-dimensional Convolutional Neural Network (1D-CNN) Auto-encoder.

Chapter 1 Introduction

Wireless communications have become an indispensable part of modern life, with a growing number of devices and applications relying on this technology to transmit information over long distances. As wireless networks continue to expand and evolve, it is becoming increasingly important to develop accurate and reliable time series prediction models that can forecast future patterns of network traffic, signal strength, and other critical metrics.

The importance of time series prediction in wireless communications lies in its ability to provide operators with valuable insights into how their networks are performing and how they can optimize them to meet the demands of an increasingly connected world. By forecasting trends and identifying patterns in network activity, operators can take proactive steps to improve the reliability, speed, and efficiency of their networks, while also reducing downtime and minimizing the risk of service disruptions.

However, time series prediction in wireless communications is not without its challenges. One of the primary challenges is dealing with the complex and dynamic nature of wireless networks, which are subject to a wide range of external factors, such as environmental interference, network congestion, and user behavior. These factors can make it difficult to accurately model network behavior and predict future trends. Another challenge is the need to process large volumes of data in real-time. Wireless networks generate vast amounts of data, which must be collected, processed, and analyzed in real-time to provide accurate predictions. This requires sophisticated data processing and analysis tools, as well as powerful computing resources capable of handling the high volumes of data generated by wireless networks.

There are two types of models for wireless channels, mathematical and statistical, but mathematical models are often too oversimplified to fit real-world situations. Many forecasting models targeting time series, such as Autoregressive Moving Average (ARMA) and ARIMA, are statistical models that aim to minimize the average mean square errors. However, these models are not susceptible to noise, interference, and corrupted data, which are critical issues in wireless communications.

In recent years, Artificial Neural Networks (ANN) have been applied and heavily studied in response to the critical need for accurate wireless channel predictions. Recurrent Neural Network (RNN), in particular, is capable of learning and modeling time-varying data series while ignoring outliers and interference contained in the training datasets. Preliminary results show that such models can generalize and predict for unseen data, providing a new and effective approach to predict time-varying data sequences. Long Short-Term Memory (LSTM) is an extension of RNN that has emerged as a powerful model for long-term sequential data forecasting. LSTM replaces traditional artificial neurons with memory cells, each consisting of an input gate, a forget gate, and an output gate. The gates offer RNN the capacity to learn long-term dependencies. By doing so, RNN remembers only valuable information for long periods of time, hence providing high prediction capacity.

1.1 Time Series in Wireless Communications

Time series data refers to a type of data that consists of a sequence of observations recorded over time. This type of data is commonly used in a variety of applications such as finance, economics, and weather forecasting. In wireless communications, time series data is used to represent various network metrics such as signal strength, network traffic, and packet loss. The analysis of time series data in wireless communications can provide valuable insights into the performance and behavior of wireless networks.

One of the key differences between time series data in wireless communications and normal time series data is the underlying process that generates the data. In normal time series data, observations are typically generated by a stationary process, meaning that the statistical properties of the data remain constant over time. In contrast, the observations in time series data in wireless communications are generated by a non-stationary process, resulting in statistical properties that change over time. The non-stationarity of time series data in wireless communications can be attributed to several factors such as changes in network topology, user behavior, and environmental interference. These factors can lead to significant fluctuations in network metrics, making it challenging to accurately model and predict the behavior of wireless networks. Another key difference between time series data in wireless communications and normal time series data is the high dimensionality of the former. Wireless networks generate vast amounts of data, which can include multiple metrics collected at different locations and time intervals. This high dimensionality can make it challenging to analyze and visualize time series data in wireless communications. Moreover, time series data in wireless communication is highly noise-polluted due to the complex and dynamic nature of wireless networks. Wireless networks are subject to various external factors such as environmental interference, network congestion, and user behavior, among others. These factors can lead to significant fluctuations in network metrics, making it challenging to accurately model and predict the behavior of wireless networks.

Despite these challenges, the analysis of time series data in wireless communications can provide valuable insights into the performance and behavior of wireless networks. Time series analysis in wireless communications can be used to identify trends, patterns, and anomalies in network activity. Furthermore, it can be used to predict future network behavior, allowing network operators to optimize network resources and improve network performance. Several techniques have been developed for the analysis of time series data in wireless communications, including time domain analysis, frequency domain analysis, and wavelet analysis, among others. Time domain analysis involves analyzing the statistical properties of the time series data, such as mean, variance, and autocorrelation, to identify trends and patterns. Frequency domain analysis involves transforming the time series data into the frequency domain using techniques such as Fourier transforms, to identify periodicities and other frequency-dependent patterns in the data. Wavelet analysis involves analyzing the time series data at different scales, using a wavelet transform, to identify patterns and trends at different frequencies and time intervals.

1.2 Noise in Wireless Communications

One of the key characteristics of time series in wireless communications is the presence of noise, which is sometimes regarded as inevitable. Noise can significantly impact the accuracy of the data analysis and prediction. Wireless signals can be corrupted by noise that can arise from various sources. Some of the common factors that cause noise in wireless data are listed as follow:

1. Environmental Factors: Wireless signals can be affected by environmental factors such as temperature, humidity, and atmospheric pressure. These factors can cause signal attenuation, interference, and scattering, leading to noise in the data.
2. Hardware Imperfections: Imperfect hardware components such as amplifiers, filters, and oscillators can introduce noise in the signal. These components can be manufactured with a certain level of imperfection, making it impossible to completely eliminate noise from the signal.

3. Interference: Interference from other wireless devices, electrical equipment, or natural phenomena such as lightning can cause noise in the wireless signal. Interference noise can interfere with the signal of interest, leading to errors in the data analysis.
4. Multipath Fading: Multipath fading occurs when the wireless signal reflects off objects in the environment, leading to multiple paths for the signal to reach the receiver. This can cause variations in the signal amplitude and phase, leading to errors in the data analysis.
5. Attenuation: Attenuation is the reduction in signal strength as it travels through the air. Attenuation can be caused by factors such as distance, obstacles, and the frequency of the signal. Attenuation can cause the signal to become weaker, leading to noise in the data.
6. Thermal Noise: Thermal noise is the random noise generated by the thermal agitation of electrons in the conductors and components of the wireless system. Thermal noise can lead to variations in the signal amplitude and phase, leading to errors in the data analysis.
7. Quantization Noise: Quantization noise occurs when the analog signal is converted into a digital signal. This can cause rounding errors, leading to noise in the data.
8. Shot Noise: Shot noise is the random noise generated by the statistical fluctuations in the flow of electrons in the conductors and components of the wireless system. Shot noise can lead to variations in the signal amplitude and phase, leading to errors in the data analysis.

1.3 Problem Definition

Accurate forecasting with these models demands accurate and reliable data sources, which becomes a major challenge for wireless time series forecasting. Wireless channel variables are inherently unstable and are corrupted by noise, interference, and fading. Noise refers to random variations or disturbances in the time series data that do not correspond to any underlying pattern or trend. These variations can be caused by a variety of factors which have been listed above. Noise is a sequence of uncorrelated random data with specific means and variances, and it is not possible to predict noise by any statistical or machine learning forecasting methods. One of the main problems with noise is that it can obscure the underlying patterns in the data, making it difficult to accurately model and predict future trends. For example, if the time series data contains significant levels of noise, it may be difficult to identify any meaningful patterns or trends, as the noise can overwhelm any signal that may be present. In addition, noise can also lead to overfitting in prediction models. Overfitting occurs when a model is too complex and captures the noise in the data, rather than the underlying patterns. This can result in a model that performs well on the training data but fails to generalize well to new, unseen data. Overfitting is particularly problematic in time series analysis, as the noise in the data can be mistaken for meaningful patterns or trends, leading to inaccurate predictions. Another challenge posed by noise is that it can make it difficult to estimate the parameters of prediction models accurately. In many cases, prediction models rely on estimating the parameters of the underlying statistical distribution of the data, such as the mean and variance. However, if the data contains significant levels of noise, these estimates may be inaccurate, leading to poor model performance.

Noisy data is also a significant challenge in machine learning models as it can adversely affect the accuracy and reliability of the predictions made by these models. Overfitting and underfitting are two major issues that arises when machine learning models are trained on noisy data. For machine learning approaches, high frequency noise and interference caused overfitting occurs when a function is fit too closely to training datasets. In the case of overfitting, the machine learning algorithm tries to learn and predict noise patterns, but fails to capture the more valuable patterns. Underfitting occurs when a machine learning model is too simple and fails to capture the underlying patterns in the data. This can result in a model that is too general and does not make accurate predictions. In addition, noisy data can also affect the quality of the features used in the machine learning model. Features are the variables used to train the model, and noisy data can lead to inaccurate or irrelevant features being selected, which can lead to poor model performance. Therefore, it is crucial to understand the types and effects of noise in wireless signal data and use appropriate methods to reduce its impact. The common solution is to filter noise within the dataset before training, but the characteristics of the filter are often pre-known and stationary, and they have a clearly distinguished spectrum. However, such assumptions are not applicable for wireless channels, which are influenced by the change of environmental characteristics.

It is worth noting that other time series may also contain high-frequency elements and demonstrate seasonal or periodic changes, such as seasonal temperature. However, the high-frequency elements with evident patterns are different from Gaussian noise, and thus unlikely to cause overfitting. In summary, wireless channel prediction is challenging due to the presence of noise and interference, and machine learning approaches, such as LSTM, offer promising solutions but require accurate and reliable data sources.

1.4 Contributions

1.4.1 Contributions on Time Series Prediction Model

This part of the study makes a significant contribution by firstly identifying the factors that cause the failure of artificial neural network (ANN) based time series predictions. Through the analysis of GPS distance errors datasets, the study identified that noise causing overfitting is the main concern. In response, the study designed and built a novel LSTM prediction model that introduces a set of forget gates, input gates, and output gates, all with IIR filters. While standard LSTM gates decide how much information will be passed to the next step, the IIR gates learn what frequency contained in the data should be passed, making them effective at eliminating high frequency noise. The standard models examined in the study fail to perform long-term predictions and simply duplicate input data, acting as followers. In contrast, the proposed model was demonstrated to be capable of long-term prediction by minimizing the prediction delay, resulting in the lowest root mean square error (RMSE).

Instead of utilizing filters as separated data preprocess, the proposed model integrates IIR gates into each cell, making the best utilization of IIR filter and LSTM, interactively. Compared to the approach of simply combining an IIR filter with LSTM, the proposed model produces more accurate long-term predictions on average, without requiring prior knowledge of the statistical properties of the input data. Furthermore, the IIR-LSTM cell keeps the same input and output interfaces as a conventional LSTM cell, and can be easily integrated into existing neural network architectures.

In summary, this study contributes by not only identifying the factors that cause the failure of ANN based time series predictions, but also proposing a novel IIR-LSTM prediction model that can effectively address the issue of noise-induced overfitting and improve long-term prediction accuracy.

1.4.2 Contributions on Time Series Pattern Detection Model

This part of the study proposes a new 1D-CNN (one-dimensional convolutional neural network) based anomaly detection model that targets the detection of anomalies in wireless time series data. The proposed model utilizes the positional information of the Softmax output to estimate the confidence of the neural network. Additionally, an error margin window is adopted to create a data buffer and thus increase the noise tolerance capability of the model. A standard deviation layer is also created to provide customized weights on the Softmax output, optimizing the model performance under different environments.

The effectiveness of the proposed model was evaluated using both simulation and experimental datasets. The results demonstrate that the proposed model outperforms the 1D-CNN auto-encoder in terms of anomaly detection accuracy. The proposed model is able to effectively estimate the confidence of the neural network and accurately detect anomalies in wireless time series data, even under high noise conditions.

In summary, the proposed 1D-CNN based anomaly detection model provides a novel approach for detecting anomalies in wireless time series data. By utilizing the positional information of the softmax output and incorporating an error margin window and standard deviation layer, the proposed model is able to effectively detect anomalies even in the presence of high noise levels, thus improving the reliability and accuracy of anomaly detection in wireless communications systems.

Chapter 2 A GPS Distance Error Forecast Model Based on IIR Filter De-noising and LSTM

2.1 Long Short-Term Memory

LSTM is a type of RNN that is designed to handle long-term dependencies and overcome the vanishing gradient problem that affects traditional RNNs. In traditional RNNs, the gradients of the loss function with respect to the network parameters tend to vanish or explode as they propagate through time, which can make it difficult for the network to learn long-term dependencies. LSTM addresses this issue by using memory cells and gating mechanisms that control the flow of information through the network. A memory cell is a unit that stores information over time, and it is connected to input and output gates that regulate the flow of information into and out of the cell. The gates are modeled as sigmoid functions that determine how much of the input and output information should be passed through to the next time step.

In detail, each LSTM unit consists of three gates: the input gate, the forget gate, and the output gate. The input gate controls the amount of new input data that will be added to the cell state, while the forget gate controls the amount of previous data that will be removed from the cell state. The output gate controls how much of the current cell state will be output to the next time step. These gates enable LSTM to selectively remember or forget information over long periods of time, which makes it suitable for time-series data with long-term dependencies.

The weights and biases of an LSTM network are learned using backpropagation through time (BPTT) algorithm. During training, the network is presented with a sequence of input vectors and the corresponding target output vectors. The network outputs a sequence of predicted output vectors, which are compared to the target output vectors using a loss function. The gradients of the loss function with respect to the network parameters are computed using BPTT, and the weights and biases are updated using an optimization algorithm such as stochastic gradient descent (SGD).

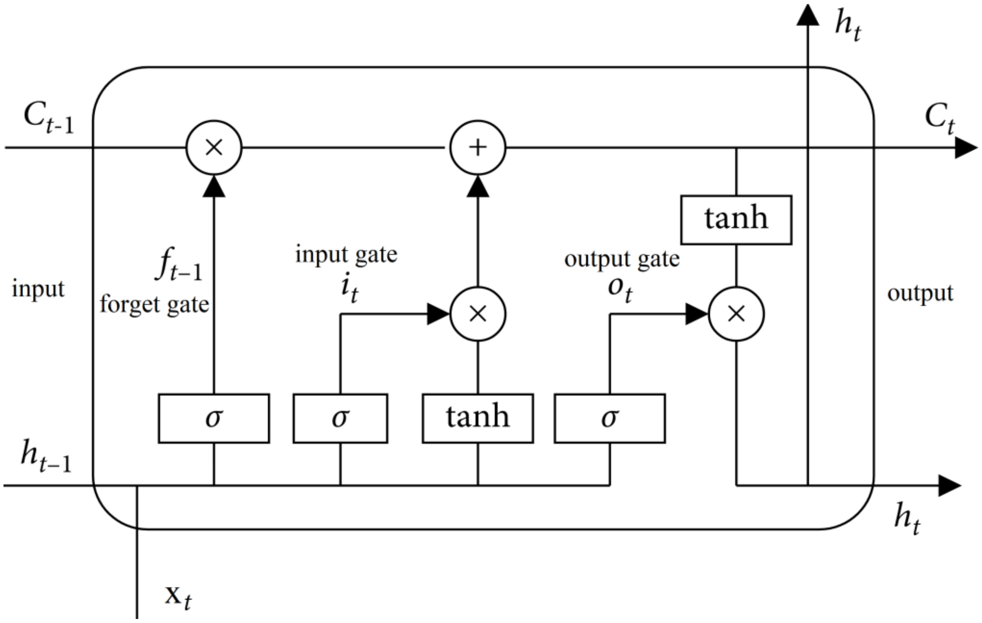


Figure 1. Cell structure of Long Short-term Memory

Figure 1 shows the architecture of Long Short-Term Memory (LSTM) neural network. Unlike a simple recurrent neuron, LSTM reuses two vectors, c_t and h_t. The new input data, x_t, is added to ht, which acts as short-term memory. Meanwhile, c_t is multiplied with the new input value, allowing it to be stored as long-term memory. The three gates, namely the input gate, forget gate, and output gate, regulate how much of the data is retained, forgotten, or outputted. This design enables the recognition of important inputs, which are stored in the long-term memory state via the output of the input gate. Moreover, logistic regression and element-wise multiplication are utilized to determine which elements of the long-term memory should be erased. Finally, the output gate specifies which portion of the new long-term memory is going to be outputted. The vanishing gradient problem is a common issue for most neural network architectures like multilayer perceptron (MLP), CNN, and simple RNN, but LSTM overcomes this problem by using the gated architecture.

2.2 Related Work

Much effort has been invested in improving the predictions and many of them combine different techniques, achieving improved results in particular fields. In general, those models are divided into three categories. The first category is dedicated to adopting multiple stages/models to achieve enhanced performance. A typical example is to integrate ARIMA with wavelet models into a RNN as an additional data preprocessing component. [13-17] proposes this idea and conducts corresponding experiments. The models in this category first considered time series composed of components of linear stationary and a nonlinear variables. ARIMA and RNN were then adopted to predict them separately. Theoretically, ARIMA works well with linear data whereas RNN predicts nonlinear parts better [13]. Wavelet are commonly utilized as a data smoothing tool to eliminate noise and make hidden trends more evident [18]. [19] designed a low-pass filter as its data smoothing tool followed by traditional LSTM training. The dataset came from a wireless channel, and it achieved improved results. The second category mainly focuses on the combination of a digital filter and neural networks for various proposals. [20] applied a nonlinear adaptive filter on RNN to lower its complexity, [21-23] utilized neural network to design and built a digital filter. In such situations, a desired amplitude-frequency response is assumed and is pre-known to the models. The third category uses FIR or IIR as the tool to encoding temporal information for MLP network, thus provide a static neural network with dynamic behavior. MLP is a feedforward neural network and its result does not depend on the order of inputs. In other words, the sequential information is not an intrinsic capability of MLP [26]. Sequential information can be fed into a MLP network through adopting a group of buffered MLP in which tapped delay lines (TDL) of the inputs are used [27]. The TDL can be applied at the network inputs only or at the input of each neuron as in MLP with FIR filter synapses [27], [32-34]. Such an architecture is also called a time-delay neural network (TDNN) [28-29]. Local Feedback Multi-Layered Network (LF-MLN) has the similar architectures but with IIR filter synapses, the additional feedbacks can provide a long and complex temporal dynamics for the system [35]. The main disadvantage of these approaches is that it has fixed and limited buffers to deal with previous inputs [27], [30-31]. On the other hand, RNN is an alternative to overcome it. It always feeds back previous output to input and is able to remember long time dependencies, but not limited by the size of delay units in the buffered MLP. The temporal processing capabilities of RNN make it suitable for time series predictions [26], and it can provide better modeling accuracy compared to TDNN [35]. In the proposed model, an IIR filter is built into each LSTM neuron with the objective of smoothing the inputs. A set of coefficient gates are introduced and applied to the filters to construct them as low-pass filters. In addition, the prediction model is designed for multivariate inputs, which offer enhanced performance in time series prediction than a univariate model.

Data preprocessing is regarded as a common, or even necessary step for neural network (NN) based predictions but is usually done as a separate process [24]. Today, LSTM is often regarded as a classical data processing model and little work has been done to improve it, especially in time series forecasting. In this paper, IIR filtering is first integrated with a set of gates in LSTM cells, instead of cascading them simply. The weights of these cells are merged into both forward and backpropagation processes. By the aid of gates, a dynamic adaptive filter is constructed to remove outliers and interference while enlarging the underline trends for prediction. Most importantly, the proposed model does not require any pre-knowledge of dataset.

2.3 FIR and IIR Filter

A FIR filter is a type of digital filter where the impulse response is finite in length. It works by convolving the input signal with a finite sequence of coefficients, known as the filter's impulse response. FIR filters are linear and time-invariant, meaning that their response to a given input signal is independent of the time at which the input signal is applied. FIR filters are also stable, which means that their output does not oscillate or diverge over time. FIR filters are used extensively in wireless communication data because they are easy to design and implement, and their frequency response can be easily controlled. FIR filters can be used to remove noise from the signal, remove unwanted frequencies, and extract useful information from the signal.

An IIR filter is a type of digital filter where the impulse response is infinite in length. Unlike FIR filters, IIR filters use feedback to create a recursive filter structure. The feedback loop can cause the filter to have an infinite impulse response, leading to a potentially infinite duration response. IIR filters are nonlinear and time-variant, meaning that their response to a given input signal can change over time. IIR filters can be unstable, which means that their output can oscillate or diverge over time if not properly designed. IIR filters are also used extensively in wireless communication data because they can be used to remove noise and extract useful information from the signal. IIR filters are often used in applications where the filter design needs to be highly efficient, such as audio processing, wireless communication, and image processing.

In this paper, the IIR filter is chosen because its framework offers more options. When applied to a dataset, if optimization algorithm determines that FIR works better for certain cells, the training process will adjust the coefficients applied to previous output to close to zero turning an IIR filter into a FIR filter. In another words, the proposed IIR-LSTM includes FIR filtering. Simplicity is another consideration. The relatively simple formulas in both forward and backpropagation largely decreases the complexity of NN while still fulfilling the purpose as a low-pass filter.

2.4 IIR-LSTM

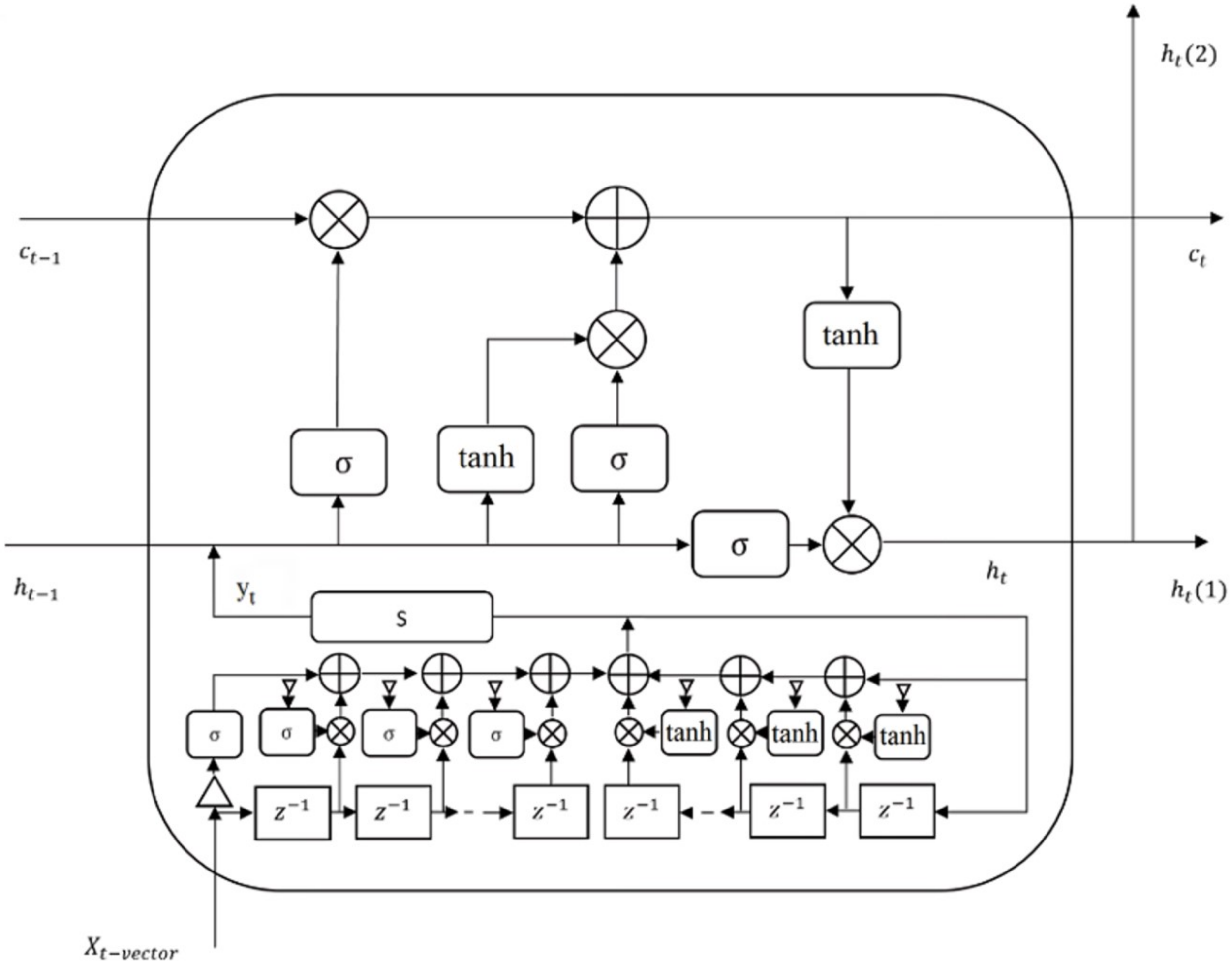


Figure 2. An IIR-LSTM cell structure

The proposed model utilizes a conventional LSTM cell as the basic structure, where the sigma function (denoted as σ in Fig.2) is applied to a forget gate, an input gate, and an output gate, while the tanh function (denoted as tanh) is applied to the active gate. The system is designed with multiple hidden layers followed by an output layer. In contrast to appending an IIR filter as a preprocessing unit, the IIR filter is constructed into each LSTM cell, forming a new cell called IIR-LSTM. The IIR filter is embedded into the LSTM cell, where the one-step delay function is represented as z^{-1}, and the coefficients are denoted as the triangular symbols shown in Fig.2. At the output of the IIR filter, there is a stacking process denoted as S, which forms a multivariate input for the LSTM. The IIR-LSTM cells are only applied to the first hidden layer.

The gradient source of the IIR gates is directly coming from the inputs of LSTM gates. In the standard LSTM cell, during the backpropagation in the training phase, the gradients of input y_t will be transferred back forward to cells in the layers ahead directly. However, in the proposed model, the gradients of the cells in the first hidden layer are further used to tune the weights of the IIR unit, which is equivalent to the coefficients of the IIR filter. The IIR gates are implemented to introduce non-linearity, and during the training phase, the weights of the IIR gates, or the coefficients of the IIR filter, will be adjusted automatically. Throughout the training process, it was found that tuning the IIR filters to be low-pass filters is a prerequisite to achieving accurate prediction.

2.5 Forward Propagation

For each iteration of forward-backward propagation, the first input dataset is built as a vector $X_{t-vector\ 1}$ defined in (2.1). The vector is formed from a segment of continuous data. The second input vector $X_{t-vector\ 2}$ defined in (2.2) is formed by applying the D(.) operator on the $X_{t-vector\ 1}$. D(.) is a delay operator implemented by right shifting one element of time series vector.

Variable n is predetermined as the size of the input window. The whole dataset will be segmented in this fashion. The technique is called forward chain, a commonly used tool used to prevent overfitting in training time series.

$$x_{t-vector\ 1} = [x_{t,}x_{t+1,\dots}\ x_{t+n}] \tag{2.1}$$

$$x_{t-vector\ 2} = [x_{t+1,}x_{t+2,\dots}\ x_{t+n+1}] \tag{2.2}$$

$$x_{t-vector\ 2} = \mathrm{D}(x_{t-vector\ 1}) \tag{2.3}$$

Ordinarily, each element of the input vector is fed into the IIR filter in certain delay, and the corresponding output y is looped back to the filter as input for next round. A delay operation is thereafter applied to the output to configure an IIR filter.

$$y_t = \sum_{i=0}^{N} b_i \bullet x_{t-i} + \sum_{j=1}^{M} a_j \bullet y_{t-j} \tag{2.4}$$

$$y_t = \sum_{i=0}^{N} \sigma(b_i) \bullet x_{t-i} + \sum_{j=1}^{M} \tanh(a_j) \bullet y_{t-j} \tag{2.5}$$

An IIR filter is formulated in (2.4) where i and j are the orders of the feedforward and feedback filter, respectively, and b_i and a_j are their corresponding coefficients. Those coefficients are initially set as random values close to 0. Additionally, non-linear activation functions are applied, for the same reason at the gates of LSTM, which are noted as IIR gates thereafter. Non-linear activation functions are able to learn and perform complex tasks. In this work, the selection of the sigma function as the activation function for the feedforward coefficients b_i is based on the criterion of providing the shortest training time without compromising the optimized performance. The tanh function is further used as the activation function for a_j. The new formula of the IIR filter is construct in (2.5). The output of the filter system is then formed into a new vector $Y_{t-vector}$

given by (2.6), which becomes the new input for the LSTM. The new input vector is further fed into the forget gate, input gate, active gate and output gate within a LSTM cell. The output of forget, input, output and active gates are noted as f_t, i_t, o_t and a_t, which can be calculated through (2.7) to (2.10). The current cell state and the output of the cell are further denoted as c_t (2.11) and h_t (2.12), respectively.

$$y_{t-vector} = [y_{t,}y_{t+1,\dots}\, y_{t+n}] \tag{2.6}$$

$$f_t = \sigma(\hat{f}_t), \hat{f}_t = W_f y_t + U_f h_{t-1} + bias_f \tag{2.7}$$

$$i_t = \sigma(\hat{\imath}_t), \hat{\imath}_t = W_i y_t + U_i h_{t-1} + \ bias_i \tag{2.8}$$

$$o_t = \sigma(\hat{o}_t), \hat{o}_t = W_o y_t + U_o h_{t-1} + \ bias_o \tag{2.9}$$

$$a_t = \tanh(\hat{a}_t), \hat{a}_t = W_a y_t + U_a h_{t-1} + \ bias_a \tag{2.10}$$

$$c_t = f_t \odot c_{t-1} + \ i_t \odot a_t \tag{2.11}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.12}$$

2.6 Backpropagation

The initial loss is measured by taking the difference between predicted output and ground true, after back propagating through the output layer. The initial losses E for the first step of backward propagation are measured, then the gradients at each of the gate, $\delta\hat{f}, \delta\hat{\imath}, \delta\hat{o}$ and $\delta\hat{a}$, are calculated by the chain rule using (2.13), (2.14), (2.15) and (2.16), respectively.

$$\delta\hat{f}(t) = \frac{\partial \mathrm{E}}{\partial \mathrm{f(t)}} . \frac{\partial \mathrm{f(t)}}{\partial \hat{f}\mathrm{(t)}} = \delta f(t) \odot \hat{f}(t) \odot (1 - \hat{f}(t)) \tag{2.13}$$

$$\delta\hat{\imath}(t) = \frac{\partial \mathrm{E}}{\partial \mathrm{i(t)}} . \frac{\partial \mathrm{i(t)}}{\partial \hat{\imath}\mathrm{(t)}} = \delta i(t) \odot \hat{\imath}(t) \odot (1 - \hat{\imath}(t)) \tag{2.14}$$

$$\delta\hat{o}(t) = \frac{\partial \mathrm{E}}{\partial \mathrm{o(t)}} . \frac{\partial \mathrm{o(t)}}{\partial \hat{o}\mathrm{(t)}} = \delta o(t) \odot \hat{o}(t) \odot (1 - \hat{o}(t)) \tag{2.15}$$

$$\delta\hat{a}(t) = \frac{\partial \mathrm{E}}{\partial \mathrm{a(t)}} . \frac{\partial \mathrm{a(t)}}{\partial \hat{a}\mathrm{(t)}} = \delta a(t) \odot (1 - \tanh^2(\hat{a}(t)) \tag{2.16}$$

The last gradients required are at C_t and h_t. Both gradients will be back propagated to the previous recurrent step of the cell at C_{t-1} and h_{t-1}. The gradient of C comes from two sources, expressed by (2.11) and (2.12), hence it is calculated by (2.17). As shown in Fig.1, the incoming gradient at h_t is composed of two parts, one from the next recurrent cell $h(t)_1$, another from the previous layer $h(t)_2$. $h(t)_1$ can be calculated by combining the gradients from all LSTM gates by (2.18). For the last hidden layer, $\delta\mathrm{h(t)}_2$ is the initial lost E, and for the remaining hidden layers, $\delta\mathrm{h(t)}_2$ is the summation of the gradients at the input of the next layer.

$$\delta\mathrm{c(t)} = \delta\mathrm{h(t)} \odot o(t) \odot (1 - \tanh^2\big(\hat{c}(t)\big) + \delta\mathrm{C(t+1)} \odot \mathrm{f(t+1)} \tag{2.17}$$

$$\frac{\partial \mathrm{E}}{\partial \mathrm{h(t-1)}} = \frac{\partial \mathrm{E}}{\partial \hat{f}(t)} \bullet \frac{\partial \hat{f}(t)}{\partial \mathrm{h(t-1)}} + \frac{\partial \mathrm{E}}{\partial \hat{\imath}(t)} \bullet \frac{\partial \hat{\imath}(t)}{\partial \mathrm{h(t-1)}} + \frac{\partial \mathrm{E}}{\partial \hat{a}(t)} \bullet \frac{\partial \hat{a}(t)}{\partial \mathrm{h(t-1)}} + \frac{\partial \mathrm{E}}{\partial \hat{o}(t)} \bullet \frac{\partial \hat{o}(t)}{\partial \mathrm{h(t-1)}}$$

$$\delta h(t) = \delta\hat{f} \bullet u_f(t+1) + \delta\hat{\imath} \bullet u_i(t+1) + \delta\hat{a} \bullet u_a(t+1) + \delta\hat{o} \bullet u_o(t+1) \tag{2.18}$$

$$\delta\mathrm{y(t)} = \delta\hat{f} \bullet w_f(t+1) + \delta\hat{\imath} \bullet w_i(t+1) + \delta\hat{a} \bullet w_a(t+1) + \delta\hat{o} \bullet w_o(t+1) \tag{2.19}$$

As a result, all the gradients of the weights and bias of the LSTM can be calculated by applying the chain rule through (2.7) to (2.10), and they will be used to update the weights and bias to complete backpropagation

To update the coefficients of the IIR filter, the initial loss is taken from $\delta\mathrm{y}(\mathrm{t})$ calculated in (2.19), which is the sum of all LSTM gate gradients with respect to their input vector $Y_{t-vector}$. By applying the chain rule on (2.5), the gradients of the coefficients b and a then can be calculated in (2.20) and (2.21). The same update rule is applied to complete the backpropagation for the IIR-LSTM cells.

$$\frac{\partial \mathrm{E}}{\partial b_i} = \frac{\partial \mathrm{E}}{\partial y_t} \bullet \frac{\partial y_t}{\partial b_i} + \frac{\partial \mathrm{E}}{\partial y_{t+1}} \bullet \frac{\partial y_{t+1}}{\partial b_i} + \cdots + \frac{\partial \mathrm{E}}{\partial y_{t+n}} \bullet \frac{\partial y_{t+n}}{\partial b_i}$$

$$\delta b_i = \delta y_t \bullet x_{t-i+1} \bullet b_i(1-b_i) + \delta y_{t+1} \bullet x_{t-i+2} \bullet b_i(1-b_i) + \cdots +$$

$$\delta y_{t+n} \bullet x_{t-i+n+1} \bullet b_i(1-b_i)$$

$$\delta b_i = b_i(1-b_i) \sum_{j=0}^{n} \delta y_{t+j} \bullet x_{t+j-i+1} \qquad (2.20)$$

$$\frac{\partial \mathrm{E}}{\partial a_i} = \frac{\partial \mathrm{E}}{\partial y_t} \bullet \frac{\partial y_t}{\partial a_i} + \frac{\partial \mathrm{E}}{\partial y_{t+1}} \bullet \frac{\partial y_{t+1}}{\partial a_i} + \cdots + \frac{\partial \mathrm{E}}{\partial y_{t+n}} \bullet \frac{\partial y_{t+n}}{\partial a_i}$$

$$\delta a_i = \delta y_t \bullet y_{t-i} \bullet (1 - tanh^2 a_i) + \delta y_{t+1} \bullet y_{t+1-i} \bullet (1 - tanh^2 a_i) + \cdots +$$

$$\delta y_{t+n} \bullet y_{t+n-i} \bullet (1 - tanh^2 a_i)$$

$$\delta a_i = (1 - tanh^2 a_i) \sum_{j=0}^{n} \delta y_{t+j} \bullet y_{t+j-i} \qquad (2.21)$$

The sizes of each variable are listed in Table I where the cell size of a one hidden layer is denoted as h.

2.7 Neural Network Structural and Model Hyperparameter

Table. I Model parameters of IIR-LSTM.

Symbol	size
x_t	$\mathbb{R}^{N\times n}$
y_t	$\mathbb{R}^{n}$
$f_t, i_t, o_t, a_t, h_t, c_t$ and bias	$\mathbb{R}^{h}$
W_f, W_i, W_o, W_a	$\mathbb{R}^{h\times n}$
U_f, U_i, U_o, U_a	$\mathbb{R}^{h\times h}$
b_i	$\mathbb{R}^{h\times N}$
a_i	$\mathbb{R}^{h\times M}$

Table. II Model hyperparameters of IIR-LSTM.

Hidden layer Depth	2
IIR-LSTM layer size, *h*	32
LSTM layer size, *h*	32
Input length, *N*	10
Prediction length, *M*	5
Feedforward filter order, *i*	10
Feedback filter order, *j*	10

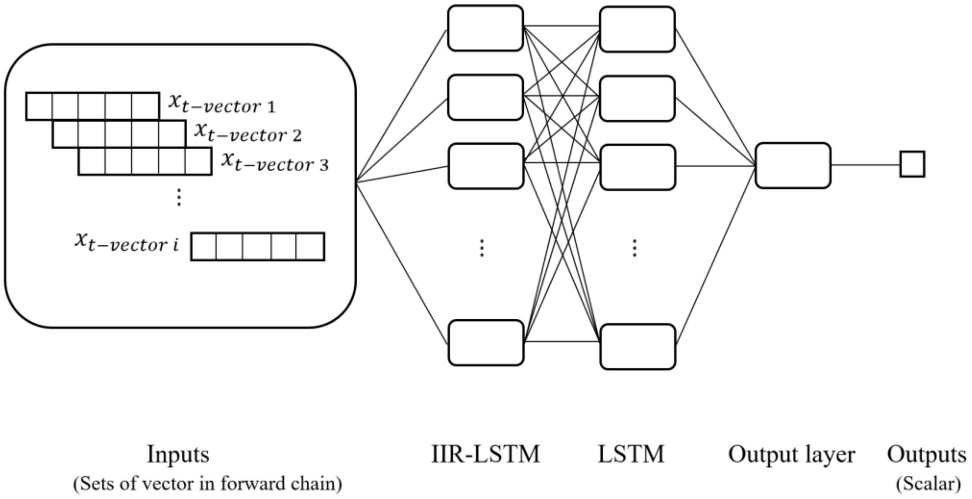


Figure 3. IIR-LSTM neural network structure.

The structure of the neural network is illustrated in Fig.3. Forward chain is utilized in the data processing to create delays. Each chunk of input includes i $x_{t-vector}$ vectors, which are used to output one unique $y_{t-vector}$through the IIR unit in each IIR-LSTM cell shown in Fig.2

The parameters of the model used in this study are presented in Table II. This specific set of hyperparameters was selected to balance the performance and complexity of the model, ensuring that the model can make accurate predictions with relatively short training times. A time lag of 10 was chosen as the input length for the model for the same reason, as increasing the time lag did not necessarily improve the accuracy of predictions. Multiple tests were conducted, setting the time lags between 2 and 15, with the lag of 10 delivering the lowest RMSEof prediction on average. All the trainings were performed on a personal computer with an i7-6700K CPU. The 30 epochs training of all tests based on the proposed model took less than 5 minutes to complete, demonstrating the efficiency of the proposed model. These findings suggest that the selected hyperparameters and time lag are suitable for making accurate predictions while minimizing the complexity of the model and training times.

2.8 Testing

The forecasting scheme was tested using a multi-step input and single-step output approach. Figure 4 demonstrates how the multi-step forecasting can be achieved. A detailed data flow for one prediction is provided in Fig.5, which shows the sequence of operations involved in generating a single forecast output. To simplify the visualization of the process, both Fig.4 and Fig.5 show the testing process after the IIR unit. Specifically, the input vector in both figures is represented as $y_{t-vector}$. The forecasting process for the model involved first performing a single-step forecast for each new incoming input data point. The single-step output prediction was then used as input for the subsequent prediction step. And the process was repeated iteratively. In practice, At the beginning of the prediction process, the true data vector (represented by white blocks in Fig.5) was loaded into the model. In the subsequent prediction steps, the current output (represented by red blocks) was appended to the vector, and a shift procedure was applied to the new vector to ensure that the input data had the same length as the original vector. The aforementioned process was repeated iteratively until the nth step, at which point the output prediction (represented by blue blocks) was the forecast for the nth step with respect to the first input data. At the beginning of the next iteration, the second true data point was loaded as the starting input for the prediction at the (n+1)th step. This process was then repeated iteratively to generate additional predictions for future time steps. The index of the true input data is denoted as B in Fig.4. As a result, the method only requires new inputs every n steps, when performing the n^{th} step forecasting. The input data that directly response to the forecasting output is purely formed by the pervious predicted data. And the closest true input for the prediction at $(B+n)^{th}$ step is at B^{th} step, thus, the nth step prediction is achieved. At the first step of the recurrent process, the previous cell and hidden states are set to 0, and forward propagation is carried out until new data is loaded. Once new data is loaded, the previous cell and hidden states from the last data are retrieved and used as the initial states for the current step. The current cell and hidden states are then stored for the next step. Since the loaded data are in sequence, this process ensures a continuous flow of cell and hidden states. The importance of this retrieve/store process is further discussed.

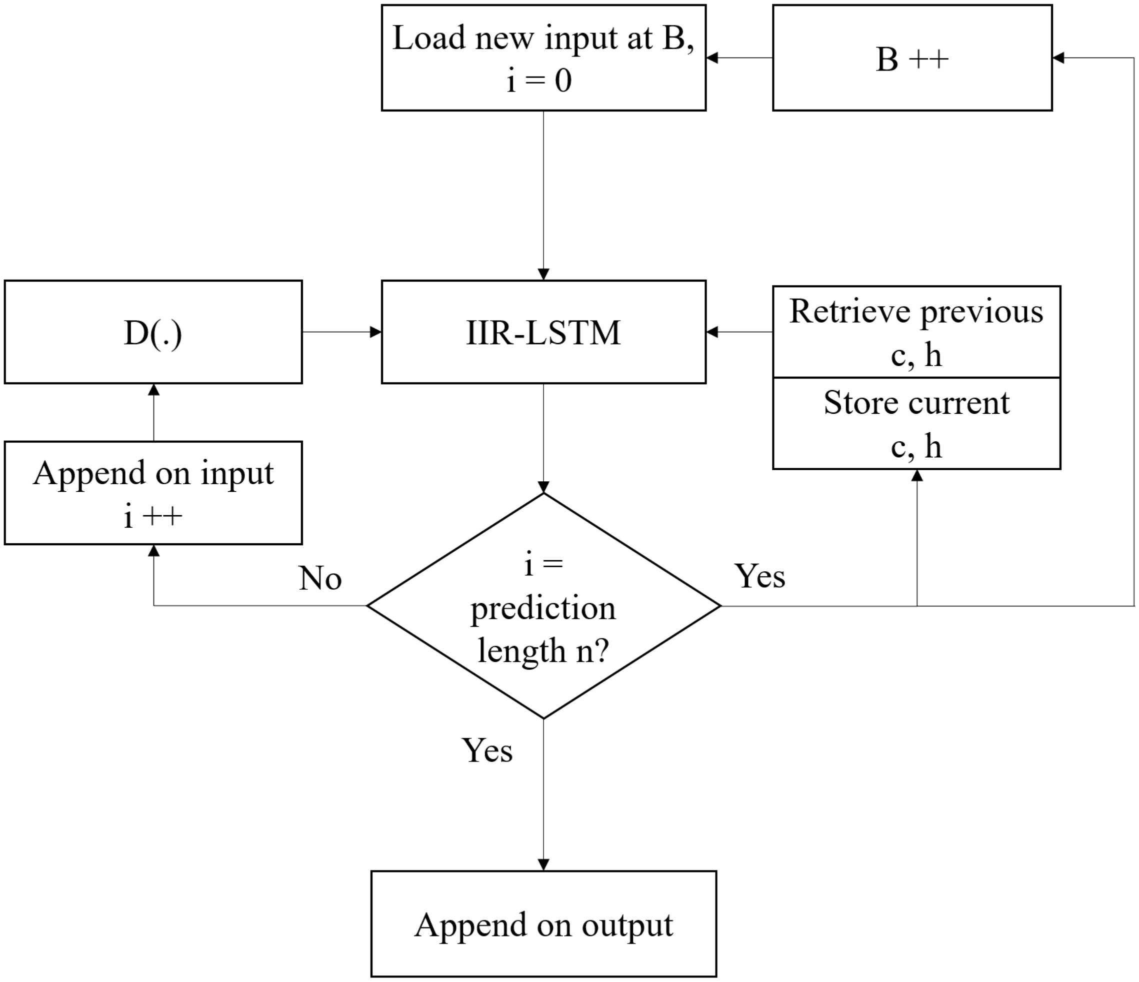


Figure 4. The flowchart of the multi-step forecasting scheme.

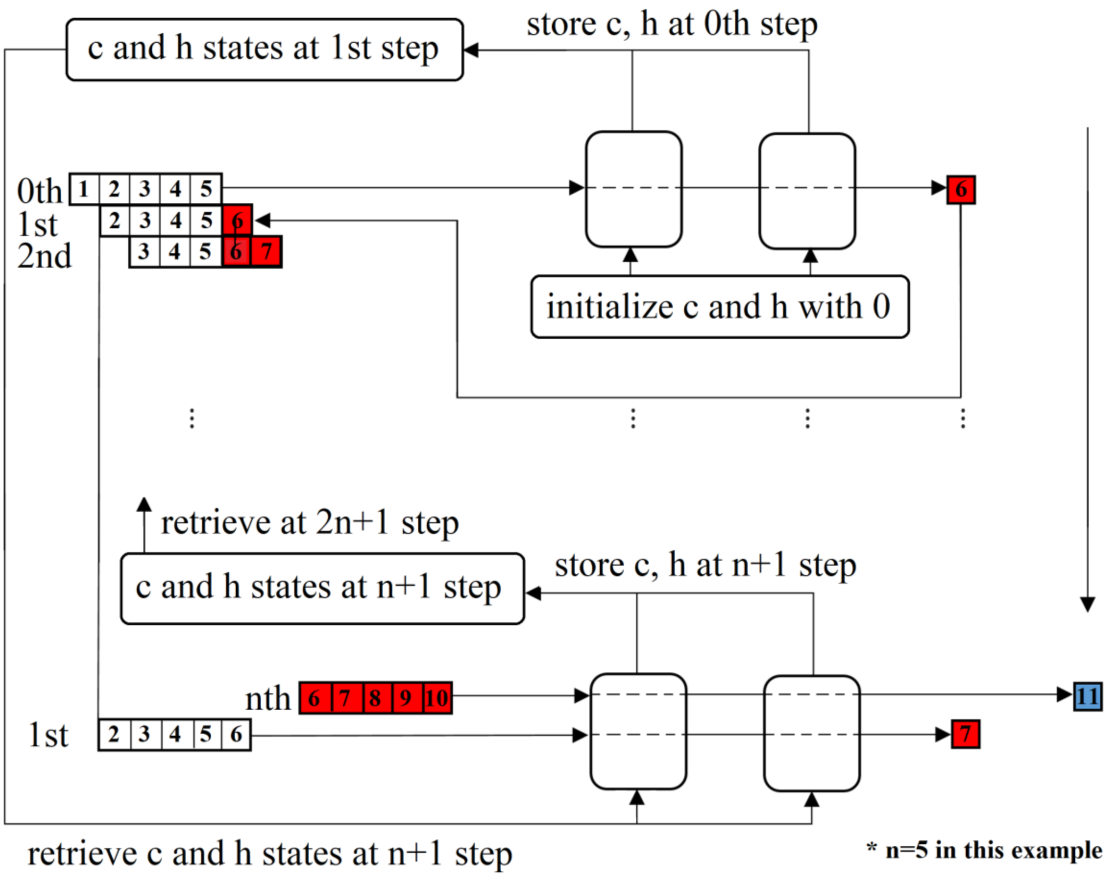


Figure 5. The data flow of one prediction of the multi-step forecasting.

2.9 GPS Data Collection

The data collected in the experiments was obtained from two identical GPS sensors (GARMIN, model GLO2), with each sensor providing position information at a rate of 10Hz and an accuracy of 3 meters. The GPS data is in National Marine Electronics Association (NMEA) format, only a portion of the information was extracted as the analytical target, which includes GPS time stamp, latitude, longitude, ground speed, and the number of satellites in use.

2.9.1 Experiment One: Stationary GPS Error

The objective of this experiment is to identify the critical factors that contribute to GPS errors. The methodology employed for the experiment was straightforward; a GPS receiver was positioned at a stationary location, which served as the reference point with known GPS coordinates. The data was then collected for a specific period of time. The Haversine formula was used to calculate the distances between the reference GPS coordinates and the received GPS coordinates, with the distance being identified as the GPS distance error. To ensure universality in the results, multiple experiments were conducted at several locations under various weather conditions.

To improve the precision and accuracy of GPS measurements, Carrier-Smoothed-Code (CSC) methods were utilized. These methods combine the advantages of carrier phase measurement and pseudorange measurement to obtain a smoothed, less noisy, and unambiguous estimate of pseudoranges. Most CSC algorithms are based on the Hatch filter, which is a type of recursive filter that depends on the current measurement and previous estimate. However, phase measurement is susceptible to cycle slips, which occur when the receiver Phase Locked Loop (PLL) locks to a new stable point. Consequently, these algorithms need to be reinitialized every time a cycle slip occurs.

An example of the observed pattern is illustrated in Fig. 6, based on the observations, a distinguishable pattern was observed; the distance error remained constant for a specific period before suddenly jumping to another level. The constant periods were due to the CSC method, while the distance jumps were caused by cycle slips. Based on our observations, we conclude that the distance error is not related to the number of satellites in use, and the magnitude of the distance jump is unpredictable. However, the duration time between the jumps is relatively constant, which is around 30 seconds.

2.9.2 Experiment Two: GPS Error in the Movement

In this experiment, two GPS sensors were mounted on the roof of a vehicle, 1.5 meters apart from each other. The position data of the sensors were collected separately, and the GPS timestamps were extracted from the data to pair the data. The distance errors were assumed to be the true distance, which is a constant 1.5 meters subtracted from the measured distances. In this case, the distance errors were the combination of the errors from two GPS sensors. As the sensors were placed relatively close to each other, the noise was assumed to come from wireless fading channels with similar characteristics, the reason is further explained.

Identical experiments were conducted at the same location four times, but the traffic situations on the road were uncontrollable, leading to uncertainties in the wireless mobile environment, the real-time distance error, and speed for each experiment, which are shown in Fig.7. The distinguishable jump pattern from experiment one was no longer observed, instead, a continuously varying waveform with the majority of energy carried at lower frequencies was obtained. The shapes of the waveform results were dissimilar to each other, suggesting no strong correlation between GPS distance errors and local mobile environments. Additionally, the potential relationship between speed and GPS distance error was investigated, and it was observed that there were no clear-cut correlations between the ground speed of the GPS sensors and the distance errors while they were in motion. However, it was noticed that whenever the test vehicle was stopped, the distance error began the pattern observed in experiment one.

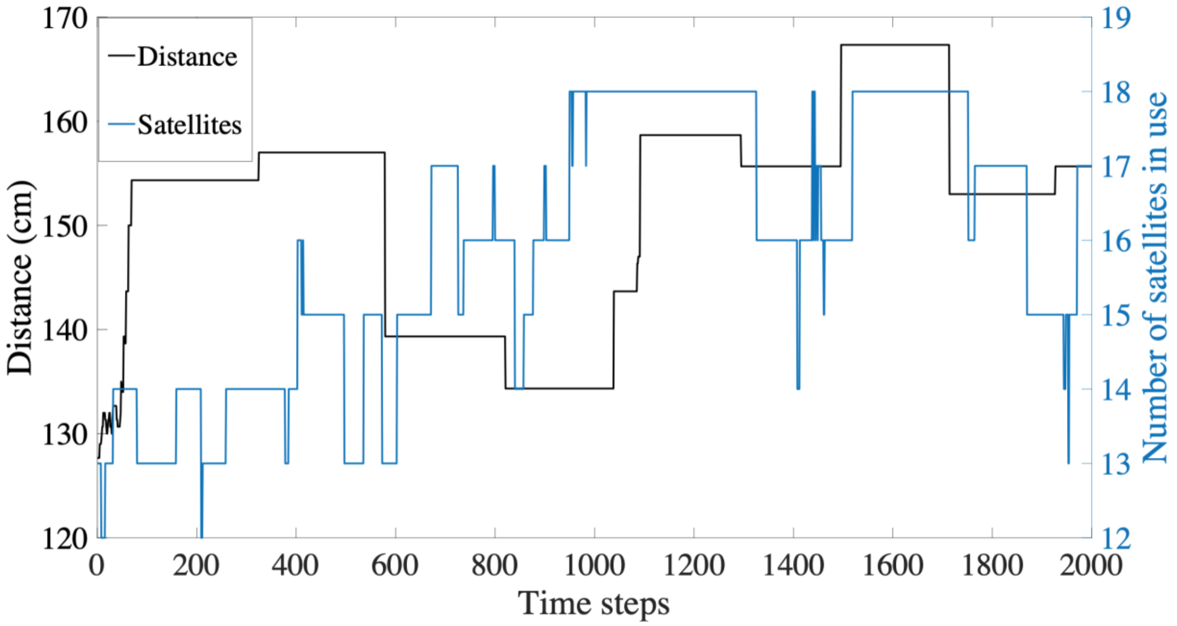


Figure 6. Stationary GPS distance error versus number of satellites in use.

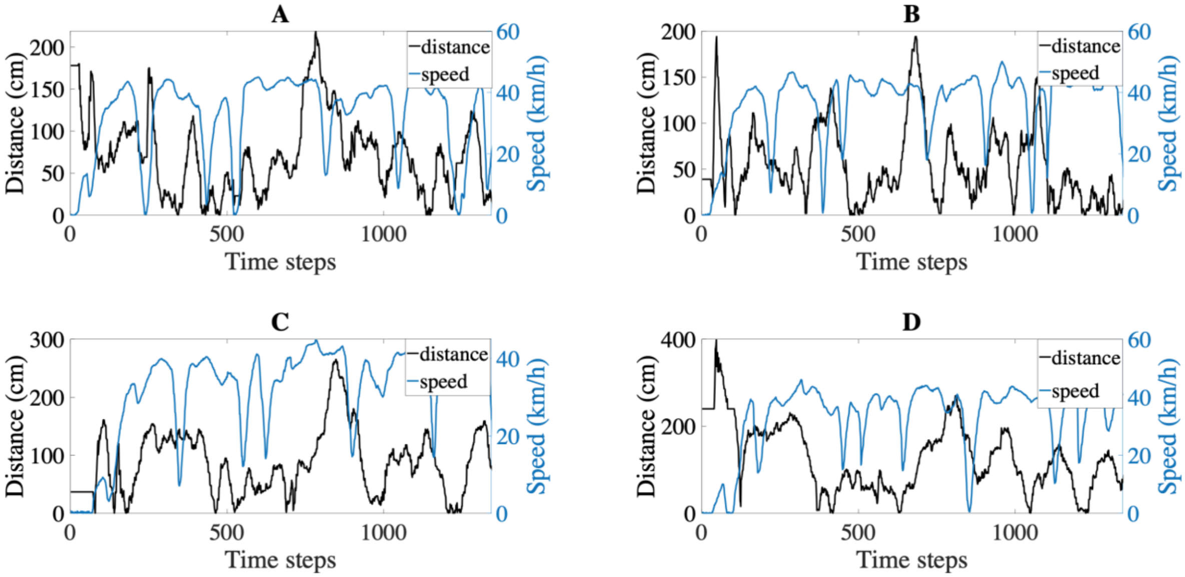


Figure 7. The GPS distance errors versus speeds.

Based on the observations of the results of the two experiments, we conclude that the distance error is not highly correlated to neither the speed of the car nor the number of satellites in use. Therefore they are not be consider as the valuable input features for our training model.

2.9.3 The Cause and the Definition of the GPS Error

Despite the unprecedented accuracy that satellite navigation provides for global positioning, GPS trajectories are still prone to errors. GPS errors can be caused by a variety of factors, including:

1. Satellite and receiver clock errors: GPS signals are sent from satellites with accurate atomic clocks, but the GPS receiver's clock is not as precise, and any difference between the receiver's clock and the satellite's clock can result in an error in position estimation.
2. Ionospheric and tropospheric effects: GPS signals can be affected by the ionosphere and troposphere in the atmosphere, which can lead to signal delays and phase shifts. This can cause errors in position estimation, especially at low elevations and high frequencies.
3. Multipath effects: GPS signals can be reflected off surfaces such as buildings, trees, or water, causing multiple signals to arrive at the receiver at different times. This can result in errors in position estimation, especially in urban areas.
4. Signal blockage: GPS signals can be blocked or weakened by obstructions such as buildings, trees, or even your body. This can result in inaccurate or no position estimation.
5. Atmospheric conditions: The Earth's atmosphere can cause GPS signal degradation due to factors such as solar flares, radio interference, and magnetic storms.
6. User error: GPS errors can also be caused by user error, such as incorrect positioning of the receiver or incorrect interpretation of the GPS readings.
7. Selective availability: In the past, the US government used selective availability to intentionally introduce errors into the GPS signal to prevent its use for military purposes. Although selective availability has been turned off, it can still cause errors in older GPS receivers.

It is challenging to construct a mathematical forecasting model for specific locations, but it is possible to effectively model these locations using the latest machine learning methods. The patterns observed in these locations are influenced by factors such as weather, satellite rotation, and local environments, making them suitable for validating the proposed prediction model.

The movement data recorded by GPS can be affected by two types of errors: measurement error and interpolation error, which are inherent to any kind of movement data and cannot be avoided [40]. Therefore, these errors also affect trajectories that are recorded with GPS. Measurement error occurs when it is not possible to determine the actual position of an object due to the limitations of the measurement system. In the context of satellite navigation, this refers to the spatial uncertainty that is associated with each position estimate. Interpolation error pertains to the limitations of representing the actual motion between consecutive positions through interpolation. The accuracy of this process is affected by the temporal sampling rate at which GPS records positions. In cases where movement is recorded at high frequencies, such as 1Hz, the interpolation error can typically be neglected [41].

Real GPS data exhibits both spatial and temporal autocorrelation. In another word, GPS measurement error is not independent of space and time, and position estimates obtained at similar locations and times will have similar errors due to similar atmospheric conditions and satellite constellation. This has been extensively studied and is also the underlying principle behind the Differential Global Positioning Systems (DSPS). [41] shows measurement error causes a systematic bias in distances recorded with a GPS, and they are strongly spatially and temporally auto correlated.

In our experiments, we utilized two GPS receivers operating over the same firmware version and at a sample rate of 3Hz. This helped to further minimize the effects of interpolation error. The distance between the two GPS receivers is calculated by the position estimates obtained at the closest time from the two GPS receivers to maximize the temporal autocorrelation of the GPS measurement error, which has been proved in [41]. [41] also demonstrated that the distance errors between the two GPS receivers increase as the distance between them increases, which is attributed to a decrease in the spatial autocorrelation of GPS measurement error. In their car movement experiments, they found that the measurement error increased by approximately 2 times when the two position estimates were one meter apart compared to when they were at the same location.

During our experiments, two GPS sensors were positioned at 1.5 meters apart. The true positions of the two GPS sensors at time t are noted as $R1_t$ and $R2_t$, respectively. The estimate positions of two GPS sensors are expressed by $E1_t = R1_t + r1_t$ and $E2_t = R2_t + r2_t$ where $r1_t$ and $r2_t$ is the measurement errors from the two GPS that are correlated with each other and have identical distributions to the GPS positioning errors. It is practically hard to get the ground true values of $R1_t$ and $R2_t$. Instead, we define and study the GPS distance error r' as shown in (2.22) where the distance between the GPS estimate positions and between their true positions are expressed by $d(E1_t, E2_t)$ and $d(R1_t, R2_t)$, respectively.

$$r^{'} = d(E1_t, E2_t) - d(R1_t, R2_t) = d(E1_t, E2_t) - 1.5 = r1_{t,} - r2_t \tag{2.22}$$

2.9.4 Data Collection and Processing

Four tests were conducted in the same location at different time periods. In each test, the vehicle was driven around a local mall twice to ensure that error patterns caused by the local environment existed in both the testing and training datasets. A GPS route and the corresponding distance errors are displayed in Fig.8. The raw data was down sampled to 3Hz, and each dataset was resized to 1400 data points. The datasets were further divided into training and testing datasets, where the last 180 data points (~13%) of each dataset were designated as the test set, and the remaining data points (~87%) were labeled as the training set.

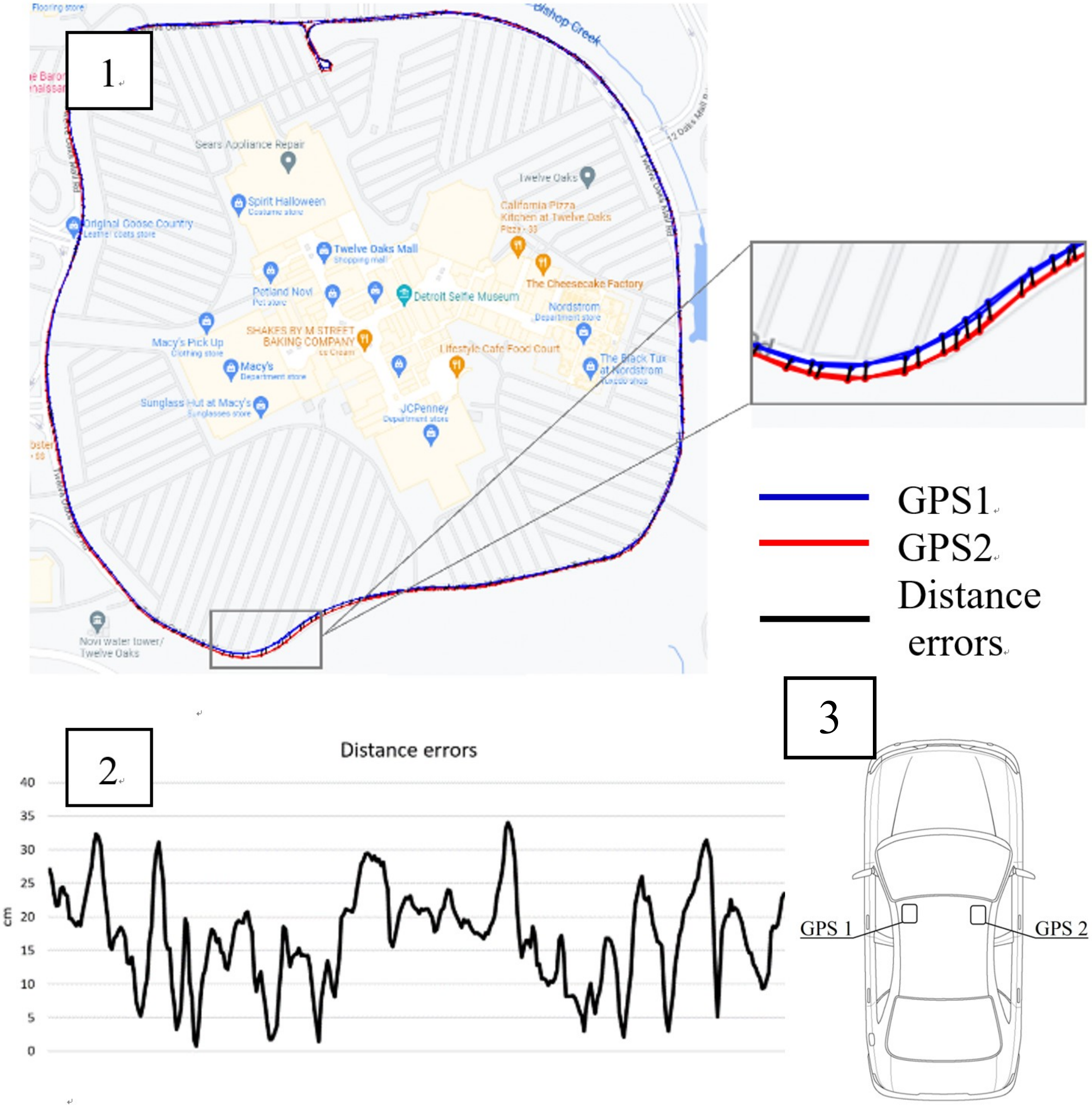


Figure 8. GPS route map (1), corresponding distance errors (2) and the test setup (3).

2.10 Performance Evaluations

Six commonly used time series prediction models were applied to the same datasets to validate the proposed model. Four of these models were derived from artificial neural networks (ANN): the RNN, the LSTM, the Gated Recurrent Unit (GRU) [37], and the MLP neural network. Two models were statistical models: the Autoregressive Integrated Moving Average (ARIMA) and the Exponential Smoothing (ETS), in addition to the persistence model. The persistence model [38] simply copies the current value as a prediction, which is straightforward but works well in short-term forecasts when data change slowly. The result is an n-step delay of input, making it challenging to forecast multi-step futures. The same prediction scheme was applied to all models, and the RMSE between the prediction results and true values were used as the metric to evaluate their performance.

2.10.1 Gated Recurrent Unit

GRU is a type of RNN architecture that is used for processing sequential data. It was introduced as an alternative to the LSTM network, which is another popular type of RNN architecture.

Like the LSTM, the GRU is designed to handle the vanishing gradient problem that can occur in deep neural networks that process sequential data. The GRU achieves this by using gating mechanisms that allow it to selectively update and forget information from past time steps. Specifically, the GRU has two gates, an update gate and a reset gate, that control the flow of information through the network. The update gate controls how much of the past information should be retained, while the reset gate controls how much of the current input should be forgotten. The GRU also has a hidden state that is updated at each time step, which contains a summary of the information learned so far.

Compared to the LSTM, the GRU has fewer parameters, which makes it faster to train and requires less memory. Additionally, the GRU is often reported to perform similarly or even better than the LSTM on various tasks, one reason for this could be that the GRU has a simpler architecture that is easier to optimize, which allows it to achieve good performance with fewer parameters. In some cases, the LSTM may outperform the GRU, particularly when the dataset has long-term dependencies that are difficult to capture. Additionally, the LSTM has a more explicit memory mechanism that allows it to store and retrieve information over longer time periods, which can be beneficial for tasks that require long-term memory.

2.10.2 Multilayer Perceptron

MLP is a type of feedforward neural network that is commonly used for classification and regression tasks. Unlike RNN such as LSTM, MLPs do not have memory cells or any form of temporal processing. Instead, they process input data through a series of layers, with each layer consisting of a set of neurons that transform the input data into a higher-level representation.

In an MLP, the input layer receives the input data, which is then passed through one or more hidden layers before reaching the output layer. Each hidden layer contains multiple neurons, and each neuron applies a nonlinear transformation to the input data. The weights and biases of the neurons are learned through backpropagation, which involves computing the gradients of the loss function with respect to the weights and biases and updating them accordingly.

Compared to LSTMs, MLPs are simpler and faster to train, and they require less memory. They are also more interpretable, as the output of each neuron can be directly linked to the input features. MLPs are well-suited for tasks that involve high-dimensional input data, such as image or speech recognition, and they have been successfully used in various applications, including financial forecasting, recommendation systems, and fraud detection. However, MLPs are not designed to handle sequential data, and they cannot model temporal dependencies between input data points.

2.10.3 Autoregressive Integrated Moving Average

ARIMA is a type of statistical model used for time series analysis and forecasting. It is a popular method for modeling the temporal dependencies and patterns in time series data. ARIMA models are based on the idea of decomposing a time series into its trend, seasonal, and residual components. The model consists of three parameters: p, d, and q, where p represents the autoregressive term, d represents the integrated term, and q represents the moving average term. The p term represents the number of past values of the time series that are used to predict the current value, the q term represents the number of past forecast errors that are used to predict the current value, and the d term represents the number of times the time series needs to be differenced to achieve stationarity.

Compared to LSTM neural networks, ARIMA models are simpler and easier to interpret. ARIMA models are also less computationally expensive and require less data to train than LSTMs. However, ARIMA models may not perform as well as LSTMs on tasks that involve complex temporal dependencies, such as natural language processing or music generation. LSTMs are specifically designed to handle sequential data and can model complex temporal patterns, making them more suitable for these types of tasks.

2.10.4 Exponential Smoothing State Space Model

ETS is another popular statistical time series forecasting model. It is similar to Autoregressive Integrated Moving Average (ARIMA) models in that it models the underlying structure of a time series, but it uses a different approach to modeling the time series data.

ETS models are based on the principle of exponential smoothing, which involves taking a weighted average of past observations to predict future values. The weights are determined by a smoothing parameter, which controls how much weight is given to recent observations versus past observations. ETS models extend the concept of exponential smoothing to incorporate seasonal and trend components, as well as other complex features such as cycles and shocks.

Compared to ARIMA models, ETS models are simpler and more intuitive to interpret. They are also faster to compute and require fewer parameters than ARIMA models, making them easier to train and more suitable for real-time forecasting applications. However, ETS models may not perform as well as ARIMA models on data that has complex temporal patterns or unusual behavior, such as sudden spikes or changes in trend.

2.10.5 Test Results

The RNN, LSTM, GRU, and MLP models were all built based on the Tensorflow library [39], where the same configuration and hyperparameters were applied. The hidden layer size of the MLP model was set to 5 instead of 2. An ARIMA (10,1,10) model was applied, where 10 is the order of the autoregressive (AR) component and the order of the moving average (MA) component, which matches the number of time lags in the IIR-LSTM model. Furthermore, an ETS (A,N,N) model, also known as simple exponential smoothing, was applied. It is a commonly used time series model with an underlying state space model consisting of a level component, a trend component (T), a seasonal component (S), and an error term (E). The (A,N,N) represents additive errors, no trend, and no seasonality, which matches the characteristics of our datasets.

Figure 9 depicts the forecasting performances of various models on a dataset, reflecting Test A. The training epochs for all ANN-based models are selected at 30, except for LSTM, which displays its results at both 30 and 100 epochs. Upon comparing the forecasting waveforms, it can be concluded that the predictions at the 10th step of all conventional ANN-based models and statistical models fail, as they exhibit unacceptable lags of true data. Specifically, the results of the ARIMA model are almost identical to those of the persistence model. The ANN-based models are negatively affected by overfitting caused by noise, as mentioned earlier. Taking LSTM as an example, the impact of overfitting becomes more noticeable as the number of epochs increases. In contrast, the IIR-LSTM model shows a promising forecasting result. Initially, the prediction waveform is smoother than the true data, indicating that the IIR filter gates play a vital role in removing high-frequency noise. Secondly, the convergence rate of this model is much faster than that of conventional LSTM models. Finally, the lags between the predictions and true data are much smaller. The same experiments were conducted on three other datasets collected in the same environment, and their best forecasting results are presented in Figures 10 to 12, respectively. By comparing the results with other methods, the IIR-LSTM model captures underlying patterns contained in the training data and predicts the trend at the 10th step successfully. IIR-LSTM outperforms all other methods with a lowest average RMSE (12.5), followed by the persistence model (23.0), ARIMA (23.8), MLP (24.2), RNN (25.2), GRU(25.8), ETS (24.6), LSTM at 30 epochs (26.9), and LSTM at 100 epochs (32.7). Their individual RMSEs are listed in Table III. It is noteworthy that the GPS distances are in centimeters, and all the RMSEs are also in centimeters.

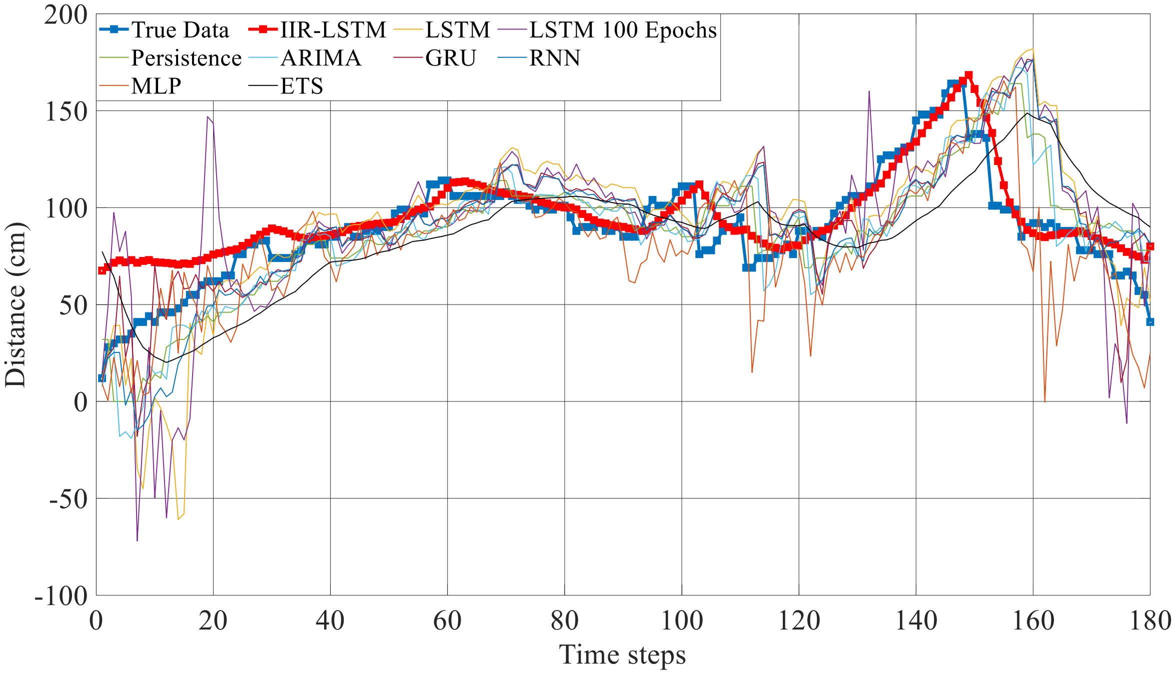


Figure 9. The 10th step predictions of test A.

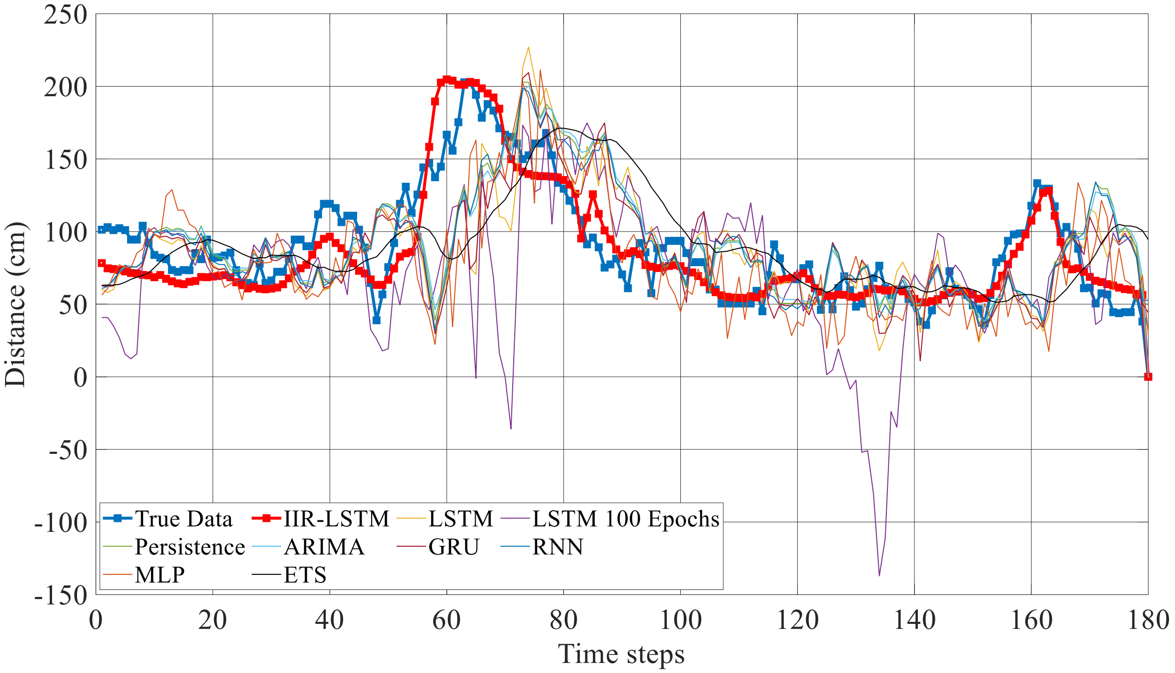


Figure 10. The 10th step predictions of test B.

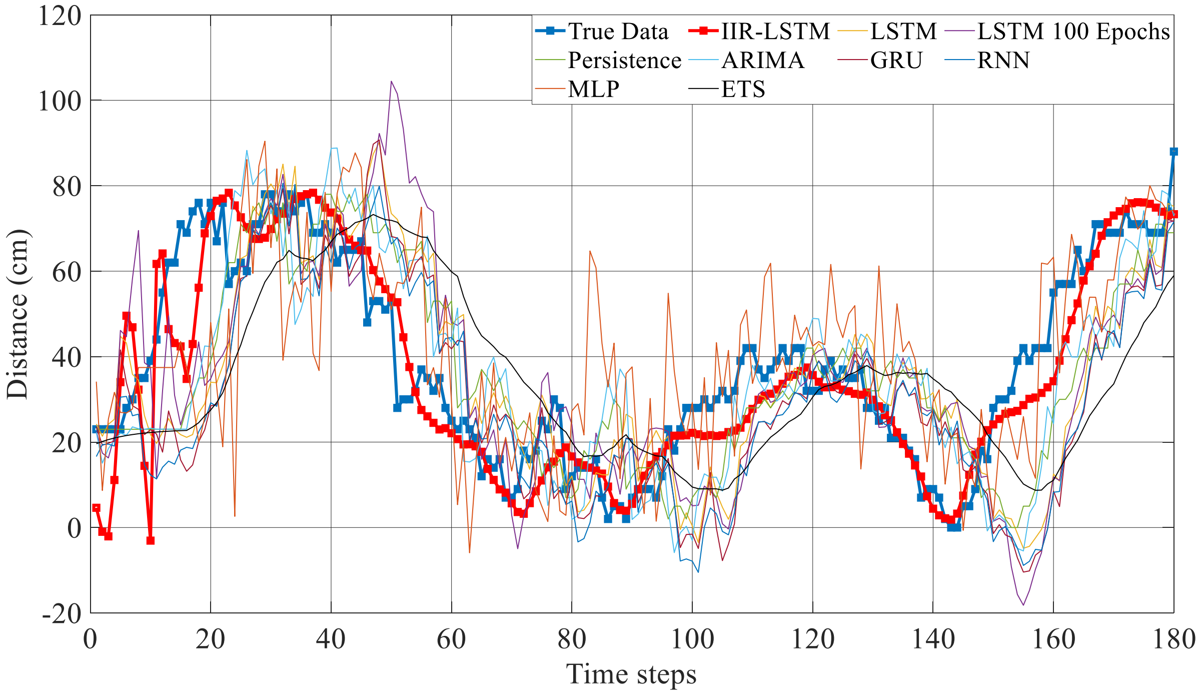


Figure 11. The 10th step predictions of test C.

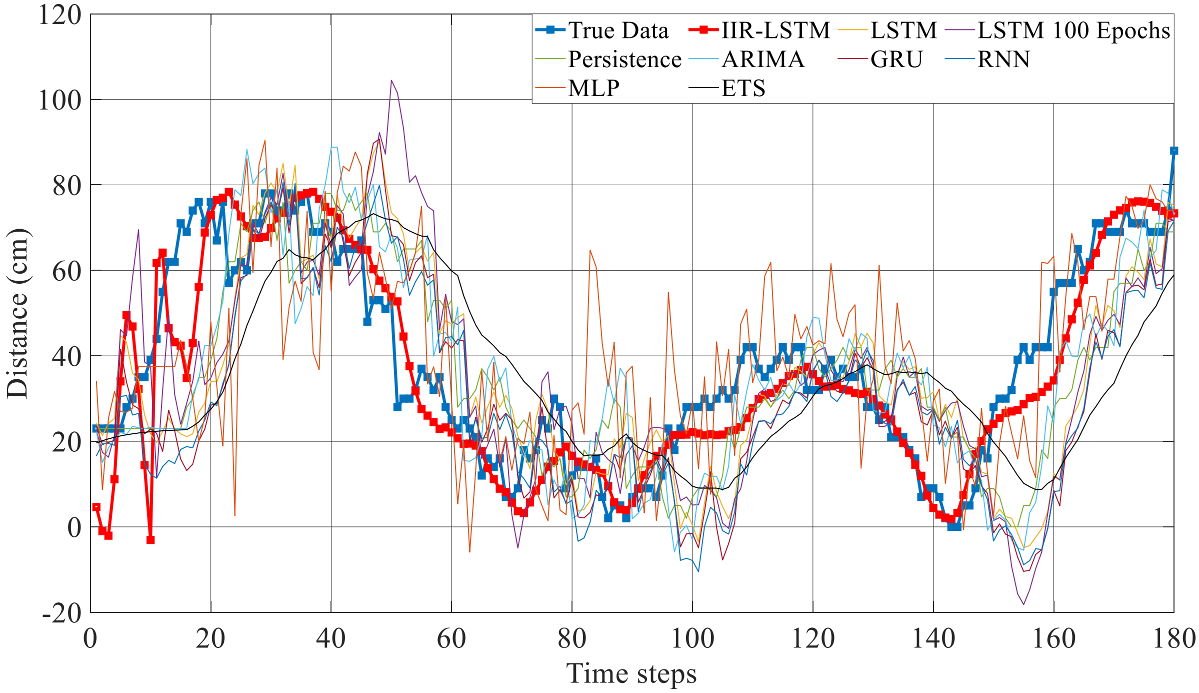


Figure 12. The 10th step predictions of test D.

Table III. The lowest, means and variances of the RMSEs of 20 times predictions over dataset of Test A, B, C and D.

	Test A	Test B	Test C	Test D	**Avg.**
IIR-LSTM *L*	**13.7**	**17.0**	**10.1**	**9.2**	**12.5**
μ	**16.2**	**20.7**	**13.2**	**10.6**	**15.2**
σ	2.0	3.0	2.5	1.1	2.2
LSTM *L* (30)	31.0	40.7	21.6	14.4	26.9
μ	33.1	43.3	23.5	16.3	29.1
σ	1.7	2.1	1.6	1.5	1.7
LSTM *L* (100)	34.3	57.5	23.3	15.7	32.7
μ	38.8	63.1	28.3	18.8	37.3
σ	3.6	4.5	4.0	2.5	3.7
RNN *L*	26.4	38.1	22.8	13.4	25.2
μ	29.0	40.6	24.6	14.5	27.2
σ	2.1	2.0	1.4	**0.9**	1.6
GRU *L*	25.8	40.2	23.0	14.0	25.8
M	27.7	41.6	21.6	15.5	26.6
σ	**1.5**	**1.1**	**1.1**	1.2	**1.2**
MLP *L*	24.9	40.1	19.7	12.0	24.2
μ	27.1	43.0	21.2	13.2	26.1
σ	1.8	2.3	1.2	1.0	1.6
Persistence	23.0	38.3	18.7	12.0	23.0
ARIMA	24.5	38.2	19.7	12.8	23.8
ETS	27.6	39.9	23.8	12.7	26.0
RMSE decrease %	30.0	45.8	29.4	11.7	33.9

The performance of all Artificial Neural Network (ANN) models on all datasets was tested repeatedly 20 times. The variability in the results was due to the random initial weights used in each test. The outcomes of these tests were recorded and are presented in Table III, which includes the best case (indicated as L), the mean (μ) and the standard deviation (σ). The numbers in parentheses indicate the training epochs used for each test. The results showed that the IIR-LSTM model outperformed all other ANN models in terms of prediction accuracy, with much lower average Root Mean Squared Error (RMSE) across all four tests. The percentages of the RMSE decrease, when compared to the second best model, are also listed in the last row of the table. However, it was also observed that the IIR-LSTM model delivered the second least consistent performances, with relatively large standard deviations in its results.

During the testing, it was discovered that few training runs were unable to converge within 30 epochs, resulting in failure of the prediction and large RMSE. This issue requires further investigation to enhance the robustness and fast training of the IIR-LSTM model in future studies.

After analyzing the results of the experiments conducted on various models, we have concluded that the IIR-LSTM model is the only model that can make successful predictions at the 10th step. The following reasons support our conclusion: First, the IIR-LSTM model produced the most accurate predictions in terms of RMSE. In fact, it is the only model that can produce results with significantly lower RMSE than the persistence model. Second, based on the shape of the prediction waveforms, the IIR-LSTM model is the only model that has the capability to overcome or at least minimize the lagging issue that occurred between the predictions and true data. This issue is demonstrated in the examples shown in Fig.9 to 12. In conclusion, the IIR-LSTM model is the most effective model for predicting the 10th step in the time series data. It outperforms all other models in terms of accuracy and has the capability to minimize the lagging issue observed in other models.

To validate the proposed model, several field tests were conducted under different environments, including a mall area (Test E), a freeway area (Test F), and a residential area (Test G). For each test, separate training datasets and validation datasets were collected. In other words, these datasets come from similar environments but different tests. The proposed model was trained using the training datasets and the 10th step predictions were made on the corresponding validation datasets. Each model was trained 10 times to ensure robustness and reliability of the results. Fig.13, Fig.14, Fig.15 compares the predictions made by the proposed model with those made by a standard LSTM model. The corresponding RMSEs of the predictions are listed in Table IV. The results are consistent with the previous findings and demonstrate that the proposed model outperforms the standard LSTM model in terms of prediction accuracy.

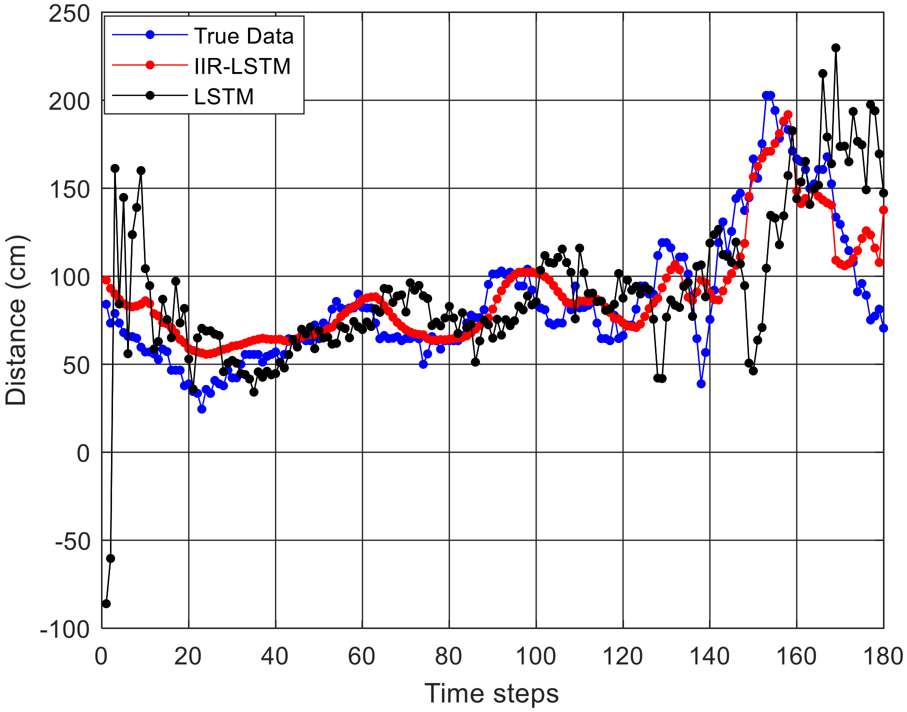


Figure 13. The 10th step predictions of dataset collected at mall Test E.

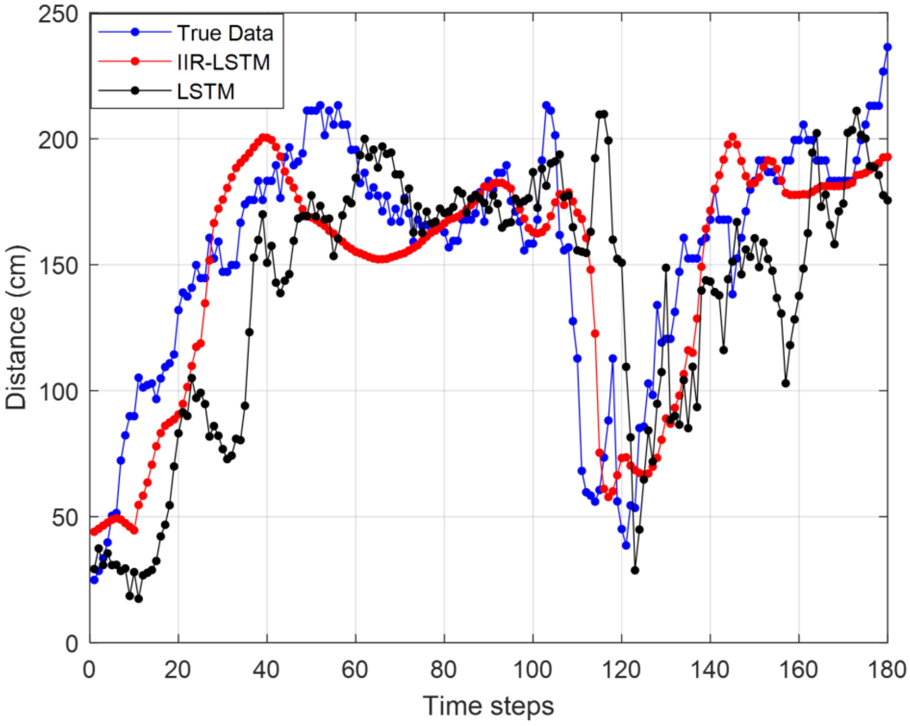


Figure 14. The 10th step predictions of dataset collected at freeway Test F.

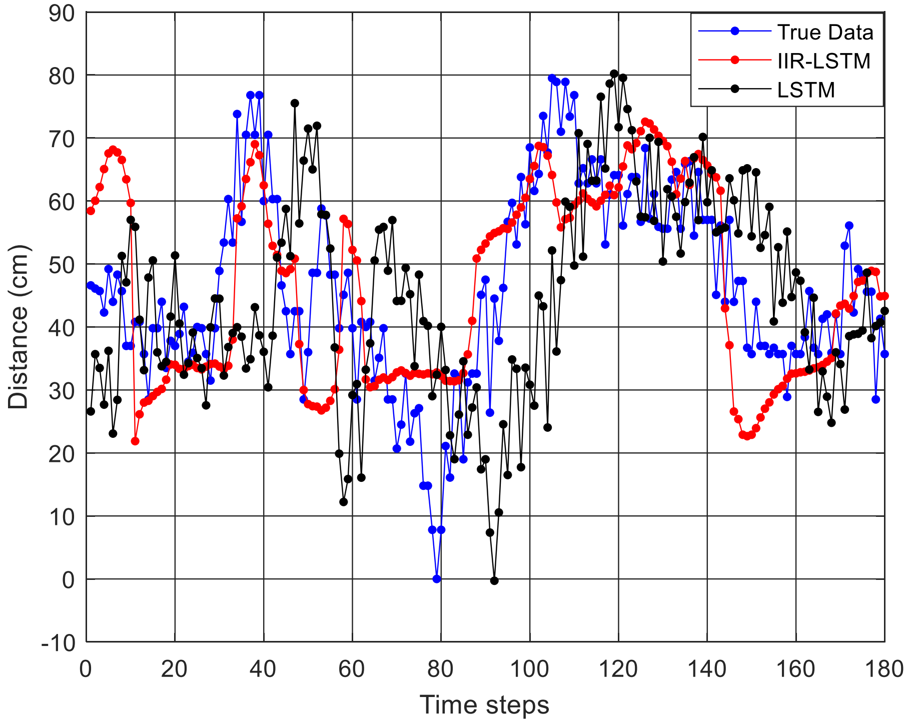


Figure 15. The 10th step predictions of dataset collected at residential area Test G.

Table IV. The lowest, means and variances of the RMSEs of 10 times predictions over dataset of Test E, F and G.

	Test E	Test F	Test G
IIR-LSTM *L*	**17.0**	**29.1**	**11.8**
μ	**19.2**	**30.7**	**13.1**
σ	1.8	**1.3**.	1.0
LSTM *L*	39.7	45.9	19.0
μ	41.2	47.9	19.6
σ	**1.2**	1.6	**0.5**
RMSE decrease %	53.4	35.9	33.2

2.11 Results Analysis

The results from the tests conducted on various prediction models were found to be consistent with each other. However, the results of the IIR-LSTM model exhibited a relatively large error or higher RMSE in the beginning of the forecasting. This phenomenon is mainly caused by the mismatch between the previous cell state and hidden state. For example, as shown in Fig.16, the predictions in the first 50 time steps were highly inconsistent with the true data, but a wrong prediction was still made.

A successful time series prediction using LSTM relies on its memory or the knowledge of past data passing through the cell and hidden states. Incorrect cell and hidden states can lead to poor results. However, with the increase of the prediction steps, more meaningful cell and hidden states are generated and passed on, eventually resulting in correct predictions. This is achieved by the store and retrieve scheme shown in Fig.4 and 5. After initialization, whenever true data is loaded, corresponding previous cell and hidden states are generated from the last true data. The states produced during one multi-step prediction will not influence the next ones.

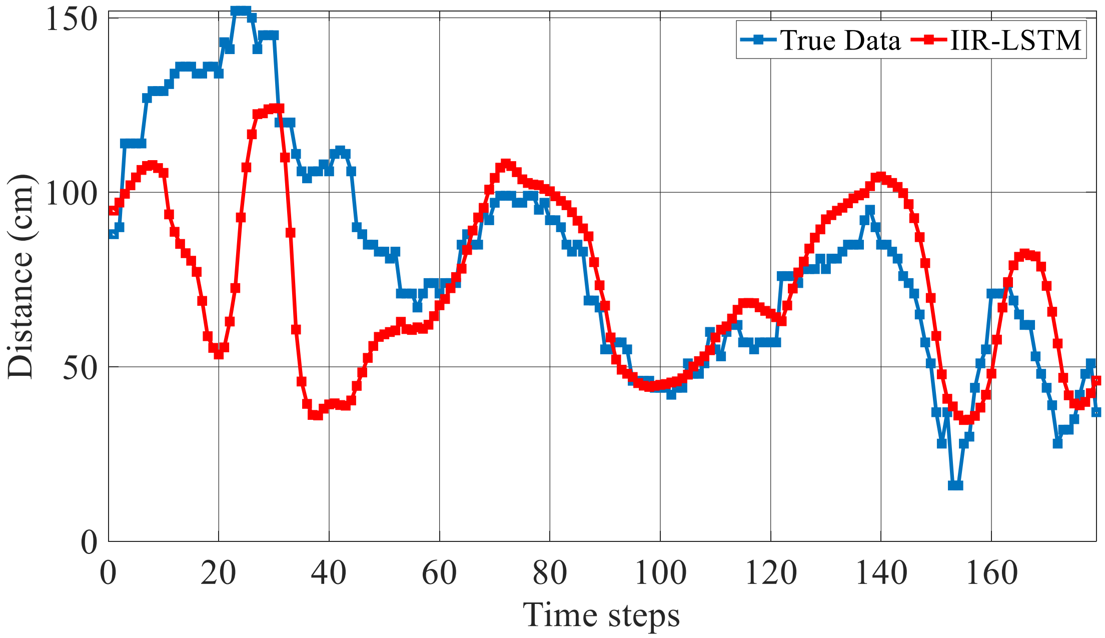


Figure 16. The impact of recurrent cell states and hidden states.

To further investigate the issue of overfitting caused by the training of conventional LSTM models, a low-pass filter was designed and applied to the dataset of Test A to eliminate noise. To determine the most accurate prediction, many experiments were conducted with various cut-off frequencies. The filter chosen was a low-pass Butterworth filter with normalized cut-off frequencies ranging from 0.01 to 0.1. Figure 17 shows that different cut-off frequencies of the filter resulted in variant LSTM prediction accuracies in terms of RMSE in all ten steps. In other words, the accuracy of the prediction is determined by the level of noise that the original data contains. It is critical to select a specific range of cut-off frequencies to achieve a low RMSE when predicting more steps.

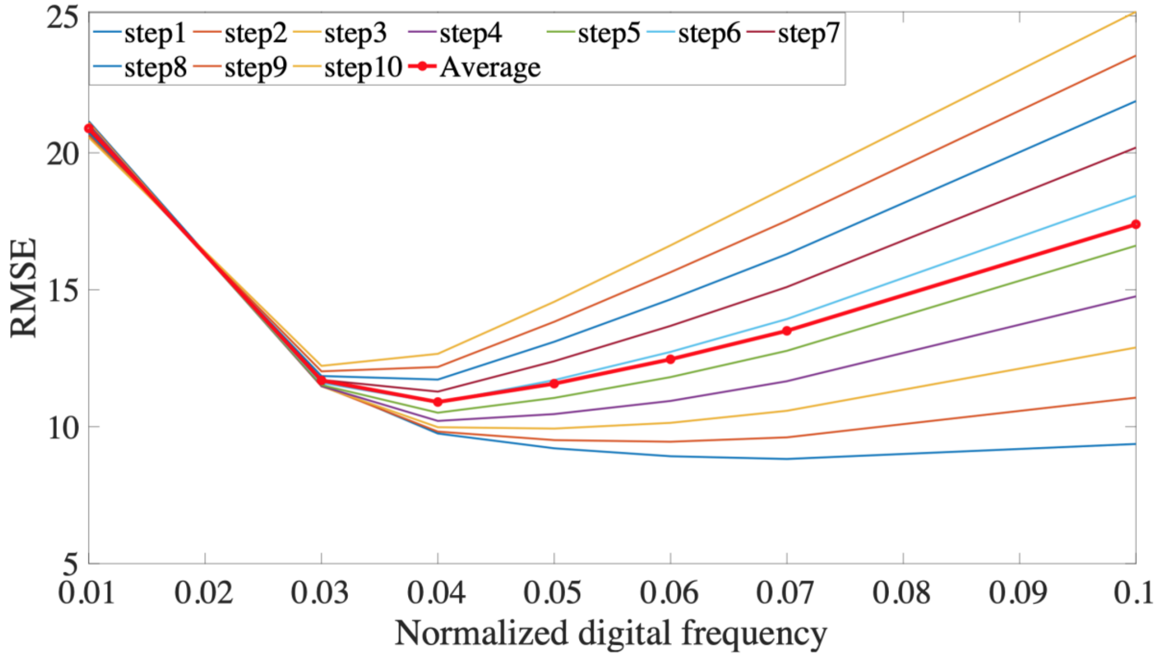


Figure 17. The 10th step and its average prediction RMSE on filtered data with various cut-off frequencies.

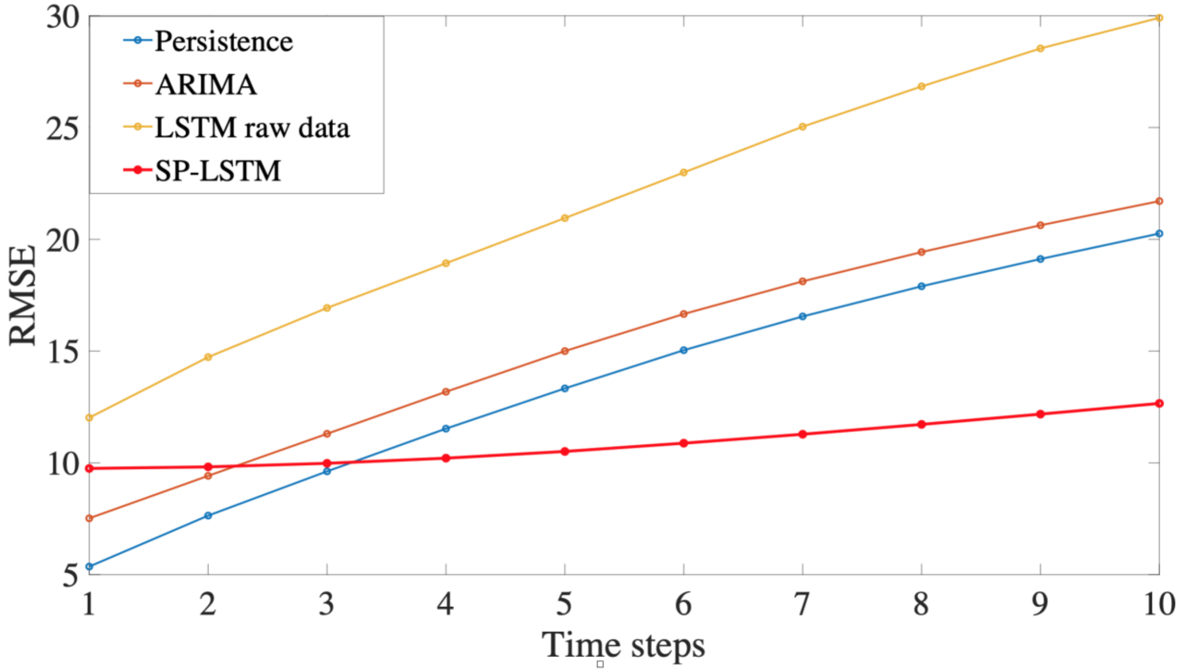


Figure 18. The 10th step prediction RMSEs achieved by the four prediction models.

In Fig.17, it is shown that the normalized frequency of 0.03 results in the lowest RMSE at the 10th step prediction. Therefore, it was chosen as the cut-off frequency to build both a low-pass and a high-pass filter to separate noise from low-frequency data. The outputs of both filters and the original data were then trained by the same LSTM model, respectively. The training loss and validation loss up to 1000 epochs are illustrated in Fig.19 and 20, respectively. By analyzing the relationship between validation loss and training loss, it is evident that only the low-frequency data are likely predictable. Both of the losses decay quickly and stay at low levels, even with occasional peaks caused by the optimization of gradient descent. In practice, a well-trained model should be able to eliminate wrong predictions at the loss peaks. On the other hand, very similar losses are generated from both the original data and noise where the training loss is decreasing and the validation loss is increasing. This is an indication of overfitting, which results in performance degradation on new data, even if one increases training time.

In conclusion, LSTM is highly sensitive to noise, and multipath fading of GPS channels negatively impacts the forecasting outcome. However, by implementing a data smoothing process, the underlying pattern contained in the low-frequency data can still be captured by LSTM. Thus, it is essential to pick the correct cut-off frequencies for filters to optimize prediction results, especially for multi-step prediction. Choosing the correct cut-off frequency depends on two factors, the characteristic of noise and the number of prediction steps. Each dataset has its own unique optimized cut-off frequency, and a set of tryouts is required to find it. Such a requirement could be impractical for some applications, especially for time-varying datasets, like wireless channels. However, the proposed model removes such limitations by integrating the smoothing process into the neural network. The backpropagation automatically adjusts the filters to reach optimal results, making the system applicable to various datasets and different step predictions.

The evaluation on the RMSE at the 10th step on the persistence model, ARIMA, LSTM with raw data, and LSTM with smooth-processed data (SP) are illustrated in Fig.18. The SP data were generated by the low-pass filter mentioned above. Fig.18 shows that LSTM with raw data has the worst prediction due to overfitting. However, the persistence model and ARIMA outperform the LSTM with SP data in short-term prediction, especially the persistence model with the lowest RMSE before the 4th step. It is because RMSE is calculated between predictions and true data, the LSTM with SP data produces smoothed data as output. There would be an obvious difference between the smoothed data and raw data, causing relatively large RMSE in short-term prediction when comparing to the persistence model. The persistence model is trivial. The LSTM with SP, however, significantly outperforms all other models in the long-term prediction. The proposed model targets long-term time series prediction, and LSTM is proven as a suitable framework for our GPS datasets.

Figures 21 and 22 illustrate the comparison of LSTM prediction performances using raw data as input and using SP as data preprocessing. As demonstrated, when raw data is utilized as input, the 10th step prediction exhibits a substantial lag to the true data (exactly 10 steps delay). However, when SP is utilized as data preprocessing, the lag is significantly reduced. This observation suggests that by utilizing SP to eliminate high frequency noise, LSTM can restore its long-term prediction capability. Moreover, an overfitting model magnified the prediction errors in the long-term prediction, resulting in large spikes in the 10th step prediction in Figure 21. This, in turn, caused a large RMSE and degraded the model's performance to a point worse than that of the persistence model.

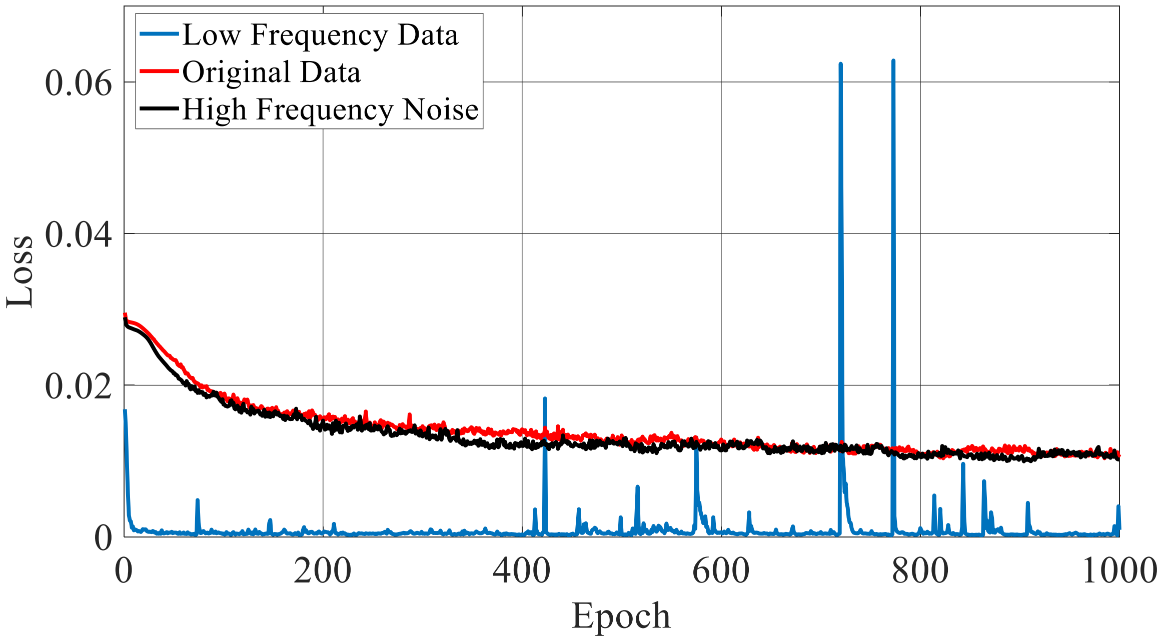


Figure 19. The 10th step prediction RMSEs achieved by the four prediction models.

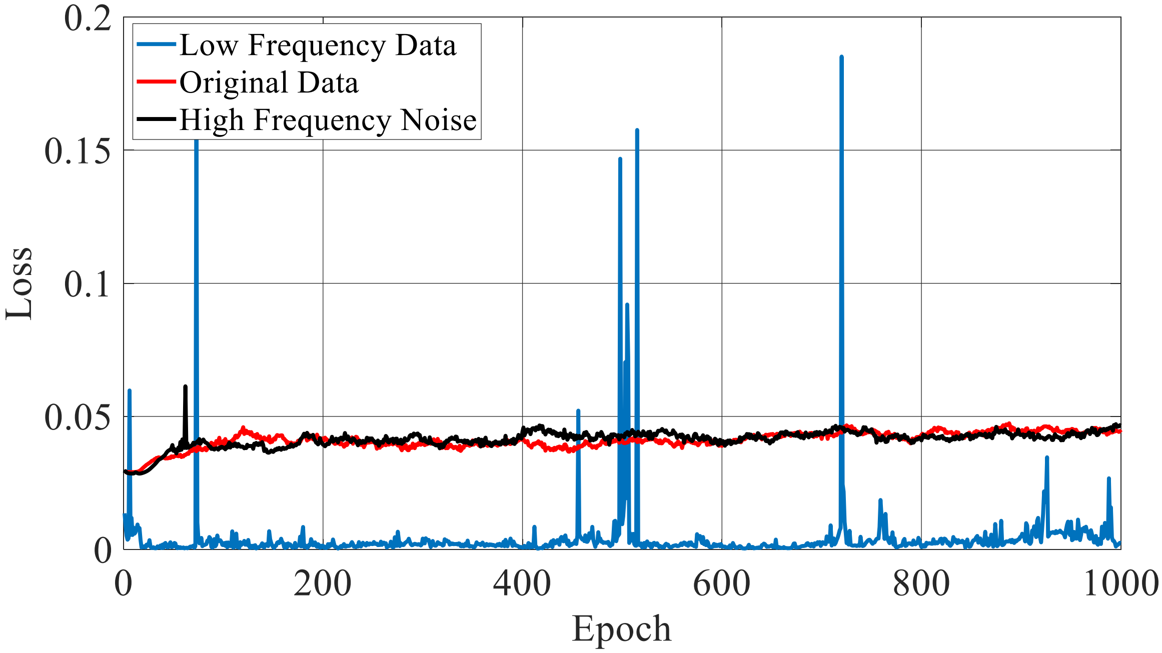


Figure 20. The 10th step prediction RMSEs achieved by the four prediction models.

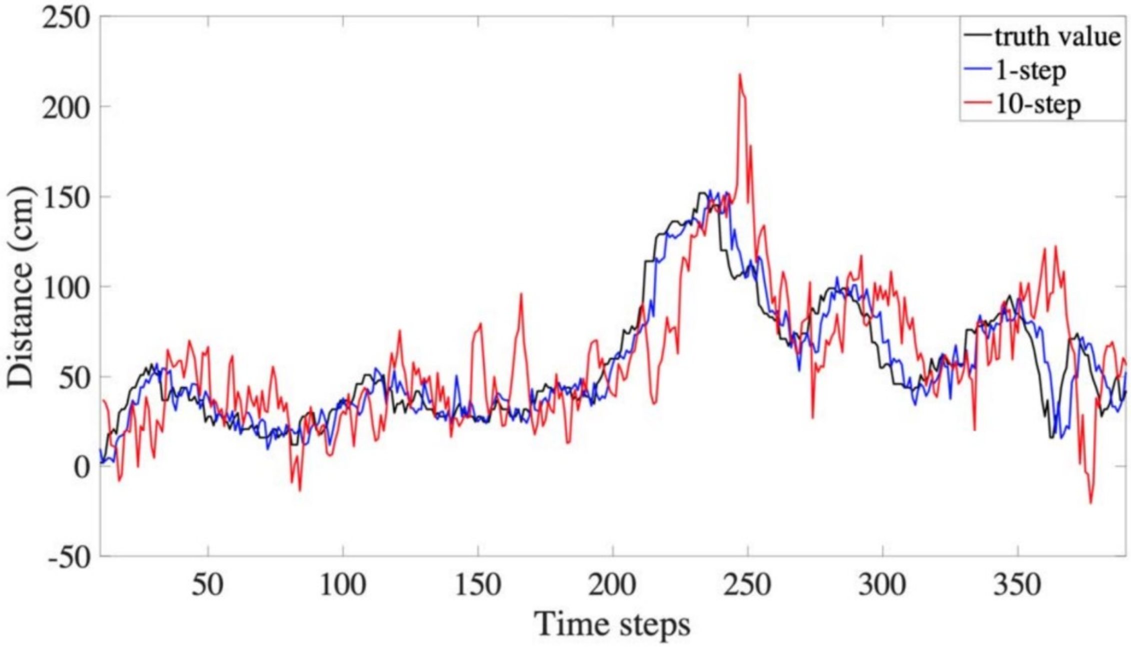


Figure 21. The 1st and 10th step predictions of LSTM model with raw data input.

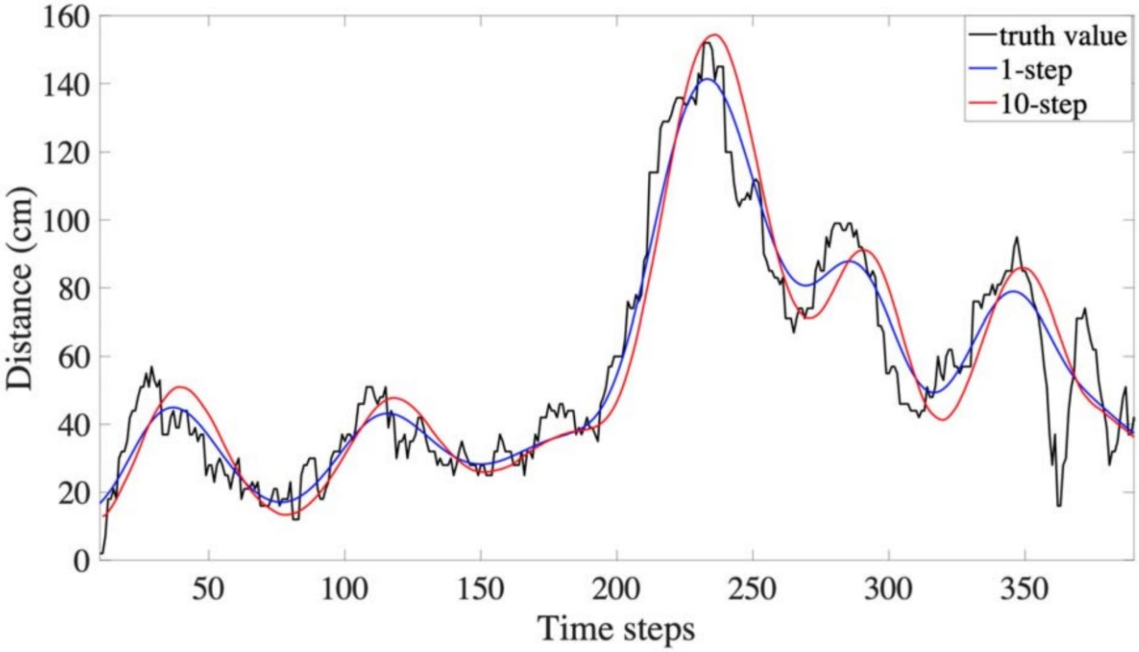


Figure 22. The 1st and 10th step predictions of SP-LSTM.

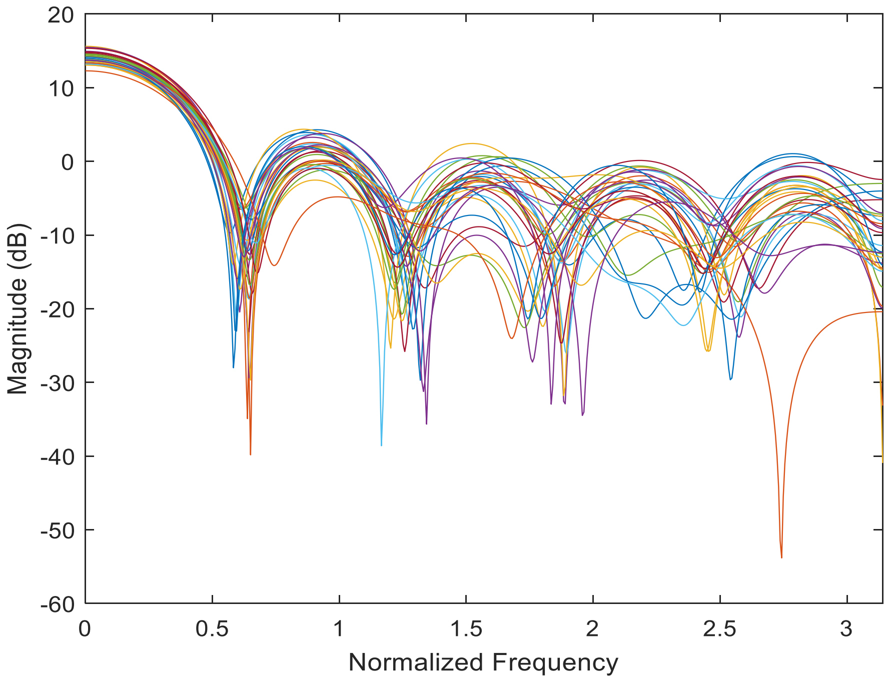


Figure 23. The frequency responses of the IIR filters.

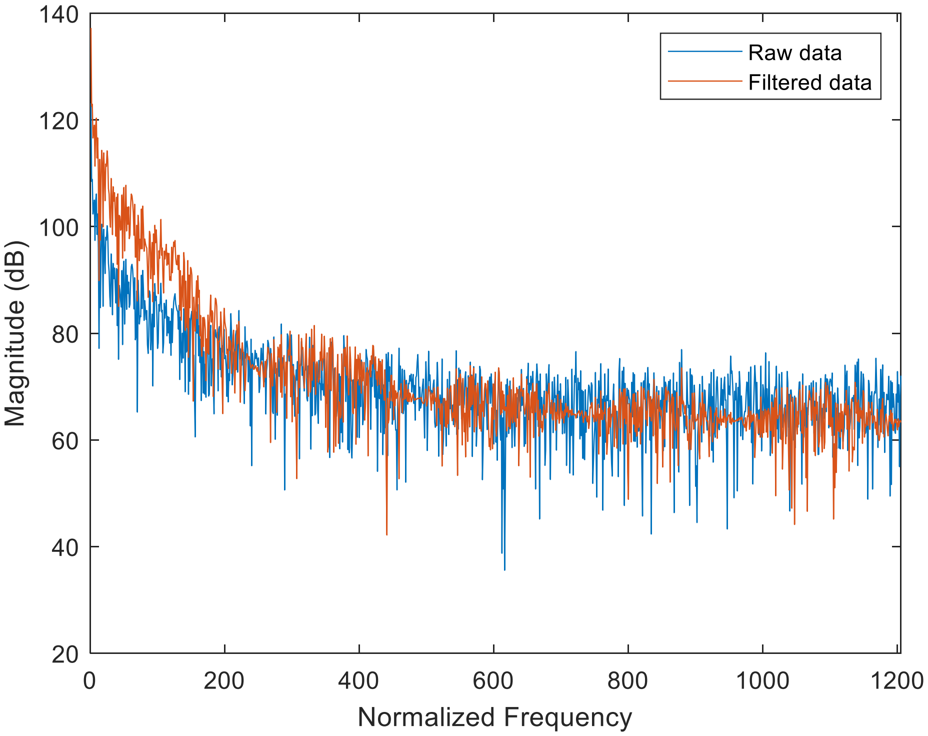


Figure 24. The spectrum of input and output data.

To further investigate the performance of the filters in the IIR-LSTM cells, the frequency responses of the IIR filters were plotted in Fig.23 after a training process. As expected, all filters function as low-pass filters. The responses in the lower frequency domain are more consistent, indicating that information at lower frequencies is more important, while high frequency information is less significant. A filter was created by averaging the frequency responses, and the spectrum of its input and output data is shown in Fig.24. The result demonstrates that the filters pass signals with lower frequencies and attenuate signals with higher frequencies. It is noteworthy that since the filters are integrated into the LSTM cells, the output of each filter is associated with a different weight, which implies that different filters have different impacts on the prediction result. In addition to input gates, the IIR gates offer an additional mechanism for regulating the flow of input data, which is customized for each cell and dedicated to smoothing data. In conjunction with the other LSTM gates, the training of the IIR-LSTM is capable of achieving the minimum loss with less overfitting.

Figure 17 and 18 illustrated that a low-pass filter with an appropriate cut-off frequency can enhance the LSTM prediction accuracy, especially for the long-term prediction. In order to investigate the performance of the IIR units, ablation experiments were conducted and some IIR units were randomly deactivated before training, and predictions were made after training. Fig.25 shows the predictions of one test with hidden layer size of 30 when some IIR units were activated. The experiments were performed 10 times and the average and variance of RMSEs were displayed in Fig. 26. The experiments were then performed with hidden layer sizes of 10 and 100, and their average prediction RMSEs were illustrated in Fig. 26. Test E was selected as the test dataset, and Fig. 26 shows that with a hidden layer size of 30, the highest average RMSE occurred when less than 10% of the IIR units were activated. The RMSE began to drop when more than 30% of IIR units were activated and reached the minimum after 80%. The same downtrend of RMSE also occurred when the hidden layer sizes were set at 10 and 100, but with different starting and ending points. The result of the hidden layer size of 30 was used for further analysis since it was closest to the configuration of the testing model, which has a hidden layer size of 32. The lowest RMSE of test E was found to be 18.3, which was higher than that of the IIR-LSTM model with all IIR units activated (17.5) when the data were preprocessed with low-pass filters with a normalized cut-off frequency of 0.03. Table V shows the comparison of RMSE values of manually selected filters and IIR-LSTM, with the latter having slightly lower RMSE values on average. This ablation experiment indicates that the IIR units indeed have a positive impact on long-term predictions. The proposed model produces competitive results compared to the data preprocessing method, but without the need for complicated filter selection processes.

Table V. The optimal cut-off frequencies, their RMSEs compare with the RMSE results from the LSTM and IIR-LSTM.

Test	*A-D*	*E*	*F*	*G*	Avg
Normalized cut-off frequency results the lowest RMSE	0.03	0.03	0.03	0.04	0.028
RMSE result of the normalized cut-off frequency	**12.3**	18.3	29.3	12.4	18.0
RMSE result of the LSTM (no IIR units applied)	32.7	43.6	45.9	19.0	32.7
RMSE result of the IIR-LSTM (all IIR units activated)	12.5	**17.5**	**29.1**	**11.8**	**17.7**

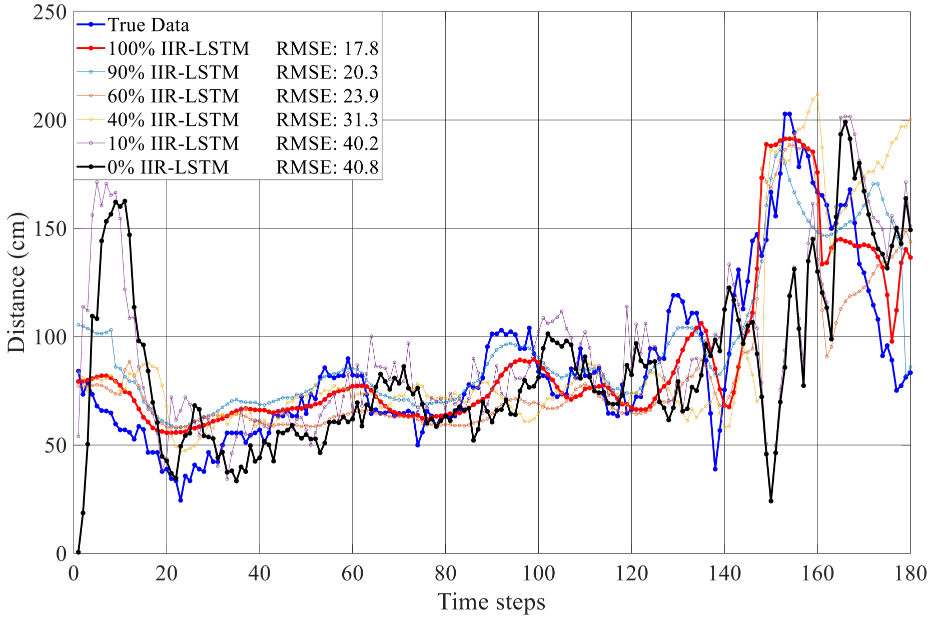


Figure 25. The 10th step predictions of test E when various percentage of the IIR units are activated.

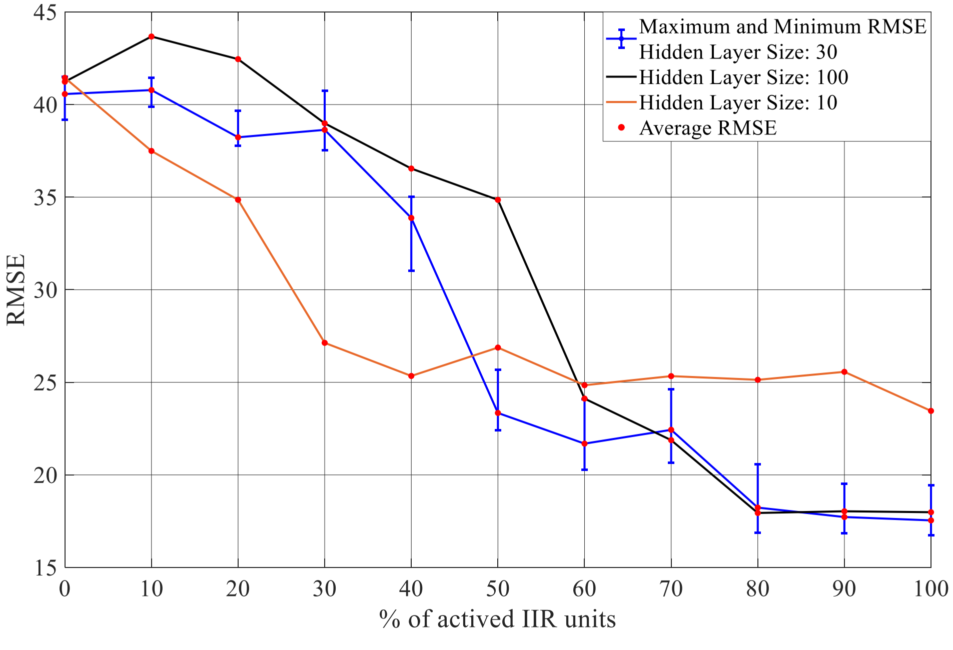


Figure 26. The average RMSEs of the 10th step predictions (test E) when various percentage of IIR units are activated.

Chapter 3 A Novel Noise Insensitive Anomaly Detector for Time Series Based on 1D-CNN

3.1 Related Works of Anomaly Time Series Detection

This section reviews recent work associated with time series patterns and anomaly detections, including both statistical and machine learning models. Statistical models include Autoregressive Moving Average (ARMA), ARIMA, and Autoregressive Moving Average (VARMA). Those models predict future values based on previous data [42]. Anomaly instances are denoted if there were significant discrepancy between observed values and predicted values [43]. Moreover, Cumulative SUM Statistics (CUSUM) was first utilized for anomaly detections in [44]. CUSUM set the boundaries of the maximum and minimum acceptable values. Anomaly instances are marked observed values exceed boundaries. The machine leaning approaches can be classified into unsupervised, supervised and semi-supervised categories. Unsupervised approaches are further divided into discriminative and parametric schemes. Discriminative schemes cluster the sequences based on a defined similarity function. Anomal score is introduced defined as the distance between observed values to the centroid of the closet cluster. Clustering methods include k-Means clustering [45], subsequence time-series clustering (STSC) [46], Expectation–maximization (EM) [47], dynamic clustering [48], single-linkage clustering [49], self-organizing maps [50], density-based spatial clustering of applications with noise (DBSCAN) [51], local outlier factor (LOF) [52], and isolation forest [53]. Parametric schemes construct summary modes based on base data, while anomalous data are excluded. At this moment, an anomaly score is related to the probability of each element of the sequence. The observed sequence is denoted as anomaly if the probability of generating such a sequence by the model were low. Popular models fallen into this category include finite state automata (FSA) [54], Markov models and hidden Markov models (HMMs) [55] [56]. Some supervised approaches have been proposed as well, which can be extended to time series anomaly detection include Elman network [57], motion features with SVMs [58], Naïve Bayes [59], rule-based classifiers [60], Siamese network [61] and Extreme Gradient boosting (XGBoost) [62]. A detailed exposition of those schemes will not be discussed here since they are out of the scope of this paper. In fact, they are supervised approaches, both the normal and anomalous data need to be labeled. On the other hand, Regression models detects anomalies based on the errors of predictions. Finally, a semi-supervised scheme uses only the normal data as training set. After training, the fitted model detects anomalies based on the similarity between training set and test set. Such models include one-class support vector machines (OC-SVM) [63] and Auto-encoder [64].

Recently, neural networks have achieved promising results in the fields of computer vision and natural language processing, outperforming statistical counterparts. There is increasing interest in using them for time series forecasting and analysis. As the fact, they are capable of learning without the knowledge of the data structure, and output beyond the information contained in the training data set. The NN based time series anomaly detection has been well studied. Many NN architectures have been presented, implemented and tested. MLP is a basic NN architecture. MLP can be further extended to anomaly detection by applying sliding window over the input data and then predicting them, whereas the prediction errors are used as anomaly scores to classify the data [65]. RNNis capable of learning and modeling time-varying data series by feeding hidden states into the input of the sequence [66]. LSTM [67] [68] and GRU [69] are both belong to the RNN architectures. Chauhan and Vig used the probability distribution of the prediction errors of

LSTM models to detect anomalous data for healthy electrocardiography (ECG) signals [70]. Malhotra et al. adopted a similar approach but using the maximum likelihood estimation (MLE) [71] to detect the anomalies. Wu et al. [72] used a stacked GRU model to predict online time series data and detect anomalies based on prediction errors. Convolutional neural networks (CNNs) focus more on the local patterns of data. It is mainly used for computer vision due to its ability of internal representation and feature extraction of an image [73]. Deep-learning based anomaly detection approach (DeepAnT) was proposed by Munir et al [74] to detect time series anomalies, where the anomaly score was calculated based on prediction errors.

3.1.1 One-dimensional Convolutional Neural Network

In recent years, Convolutional Neural Networks (CNNs) have emerged as the standard for various Computer Vision and Machine Learning tasks. CNNs are feed-forward Artificial Neural Networks (ANNs) that consist of alternating convolutional and subsampling layers. Deep 2D CNNs with numerous hidden layers and millions of parameters are capable of learning intricate objects and patterns, provided they are trained on a massive visual database with ground-truth labels. However, this may not be feasible in several applications involving 1D signals, particularly when the training data is limited or domain-specific. To address this issue, 1D CNNs have been proposed and have quickly attained state-of-the-art performance in several domains, including personalized biomedical data classification, early diagnosis, structural health monitoring, anomaly detection and identification in power electronics and motor-fault detection. Another significant advantage of 1D CNNs is their ability to enable real-time and low-cost hardware implementation, as they utilize a simple and compact configuration that performs only 1D convolutions. A compact 1D CNN with only 1 to 2 hidden layers and less than 10K parameters becomes prominent in many applications, while a typical 2D CNN easily contains parameters up to 1M-10M [80]. Moreover, in recent studies, it has been demonstrated that compact 1D CNNs exhibit superior performance on applications that suffer from a scarcity of labeled data and high signal variations derived from diverse sources.

1D CNNs consist of multiple convolutional layers that apply a set of learnable filters or kernels to the input signal. These filters are designed to identify important features or patterns in the data. Each filter performs a convolution operation by sliding across the input signal, performing a dot product between the filter weights and the local input values at each position. This produces a feature map that highlights the presence of the filter's specific pattern in the input signal. After the convolution operation, a non-linear activation function is applied to the feature map to introduce non-linearity and improve the model's ability to learn complex patterns. This is followed by a pooling layer, which reduces the dimensionality of the feature map by summarizing or down-sampling the values within each local region. Common pooling operations include max pooling, average pooling, and global pooling. The output of the final convolutional layer is typically flattened into a 1D vector and fed into one or more fully connected (dense) layers, which perform a classification or regression task based on the learned features. The fully connected layers can also be followed by a Softmax activation function to produce probabilities for multi-class classification problems.

3.1.2 Auto-encoder

An auto-encoder is a specialized feedforward neural network that is composed of an encoder, followed by a decoder. The input data are first compressed by the encoder into a lower-dimensional representation, which is then used by the decoder to reconstruct the output. The purpose of the model is to generate output data that is identical to the input data. If the norm of the error between the input and output data exceeds a predefined threshold, the input data sequence is deemed abnormal. Auto-encoder-based anomaly detection is a semi-supervised learning approach, which means that the data no longer have the same structure as those fed into the encoder. Various auto-encoder-based anomaly detection models have been proposed in recent years. For instance, Vartouni et al. developed a stacked auto-encoder for detecting web attacks [81], while Farahnakian and Heikkonen proposed a deep auto-encoder for intrusion detection [82]. Lonescu et al. combined auto-encoders with a supervised classifier for abnormal event detection in video streams. The model first encodes both motion and appearance information using a convolutional auto-encoder. A one-versus-rest abnormal event classifier is then applied to cluster the feature vectors, thus separating each normality cluster from the rest. As the classifier is a supervised learning approach, training such an auto-encoder requires other clusters labeled as dummy anomalies [83].

Auto-encoder-based anomaly detection methods have three main advantages over traditional statistical methods in time series anomaly detection:

1. Nonlinear Relationships: Auto-encoders are neural network models that can capture complex and nonlinear relationships in time series data. This is in contrast to traditional statistical methods, which often assume linear relationships between variables.
2. Feature Learning: Auto-encoders can learn features automatically from the data. This can be useful in cases where there are many variables or the variables are high-dimensional. In contrast, traditional statistical methods require manual feature engineering, which can be time-consuming and error-prone.
3. Data Efficiency: Auto-encoders can be trained using a relatively small amount of labeled data. This is because auto-encoders learn to represent the data in a lower-dimensional space, which can reduce the complexity of the problem and make it easier to learn.

The neural network architectures of an auto-encoder can also be 1D-CNN. In fact, 1D-CNN auto-encoders have been implemented and shown great success in time series classification and anomaly detection tasks [84][85][86]. The proposed model's performance will be evaluated by comparing the results from the 1D-CNN auto-encoder model.

3.2 Model Uncertainty Under Noisy Data

The softmax activation function is commonly used for classification tasks, producing a set of probabilities where the sum equals one. However, the softmax function is unable to identify outliers, and instead outputs one of the predefined classes. Bayesian Neural Networks (BNNs) [42] have been proposed to model prediction uncertainty. Unlike regular neural networks with deterministic weights, the weights of BNNs are random and follow certain distributions, resulting in different predictions for the same input. The prediction uncertainty is estimated by gathering predictions by running the model multiple times over the same data, and measuring the variance of predictions [87] [88]. Evidential deep learning takes a step further by learning the higher order distributions over the predictions. It considers learning as an evidence acquisition process and enables direct estimation of prediction uncertainties [89]. In this paper, a new approach is proposed for efficiently detecting and separating normal data from anomalous data. The distribution of the values output from softmax is utilized to quantify the uncertainty of the prediction. The proposed model is specifically designed for disturbed time series signals, such as wireless signals, which are inherently unstable due to noise, interference, and fading, resulting in distorted signals. Detection and prediction of such signals impose challenges in wireless communications.

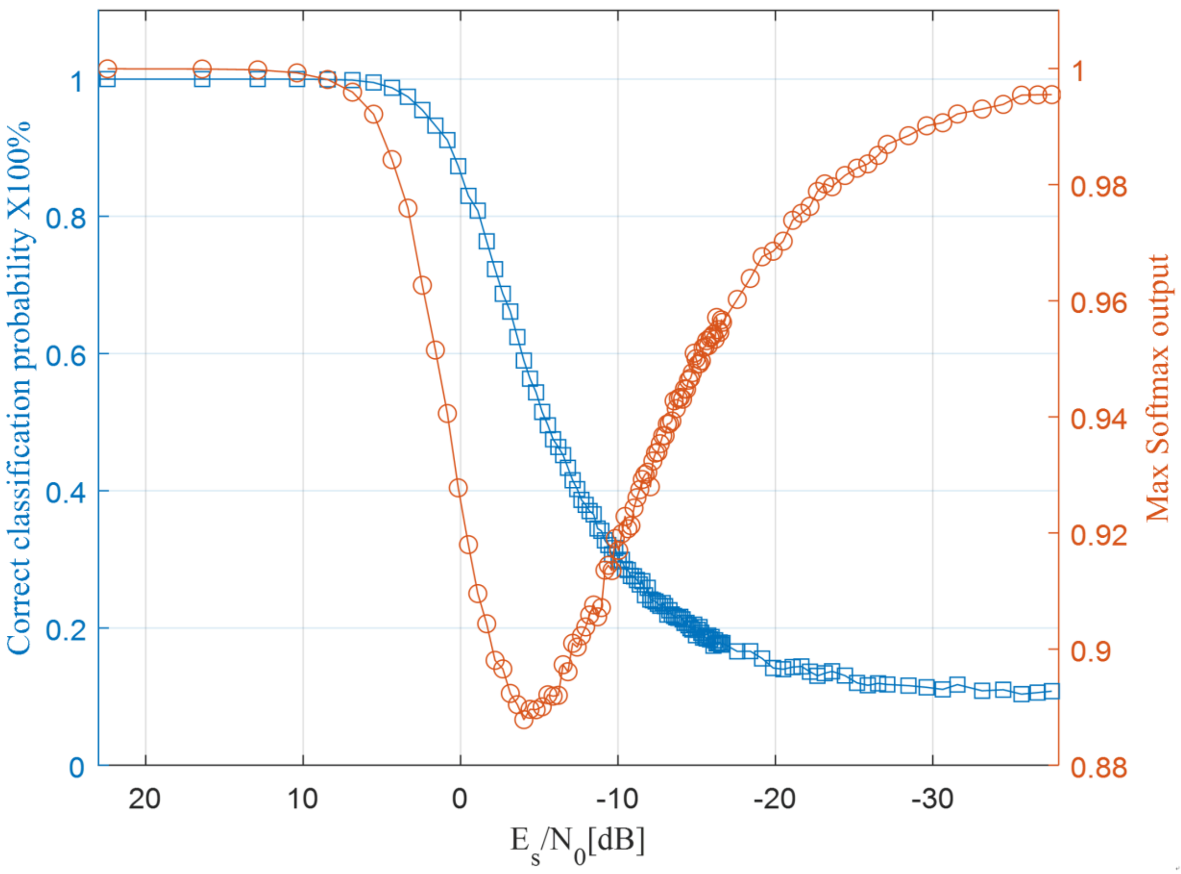


Figure 27. Classification accuracy versus the maximum value of softmax output under various levels of noise.

In order to evaluate the effectiveness of the softmax, a 1D-CNN was employed to cluster ten signals that exhibited individual patterns. Each signal comprised 100 samples that were contaminated with Gaussian noise. Figure 1 depicts the results, which show that the average classification rate decreases significantly when the signal-to-noise ratios (SNRs) of the signals are below 5 dB, and it drops to 10% when the SNRs are -30dB. In other words, at an SNR of -30 dB, the predictions are no better than random guesses, with a prediction rate of 10% across the ten classes. From this, one can conclude that the predictor functions well in cases where the SNRs are greater than -30dB.

3.3 Methodology of the Model

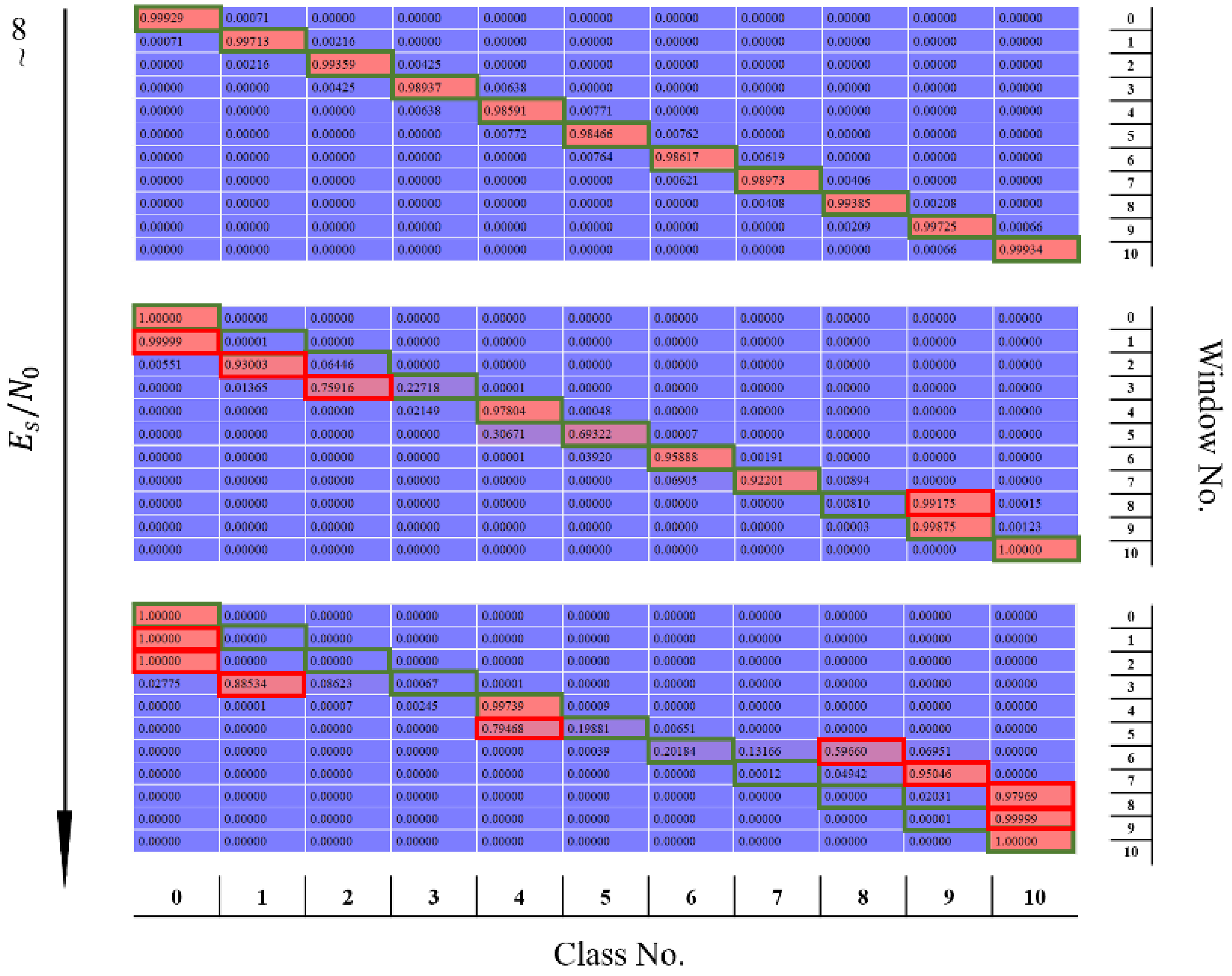
To determine the prediction uncertainties, the position information is utilized instead of the output values of the softmax. This involves applying a sliding window of length "n" to a sequence of length "l". A time series data S_T is then transformed to (3.1) $X \subseteq \mathbb{R}^{n\times(l-n)}$.

$$X(n,l) \subseteq \{\, S_T\} = \{\, x_i, \dots,\ x_{i+n} \mid i \in \{1, \dots, l-n\}\} \tag{3.1}$$

In the meantime, the data is labeled using one-hot encoding. Adjacent categories have their inputs shifted by one step. The target variable Y is represented as an $n \times n$ diagonal matrix (3.2):

$$Y(n,l) = \{d_{j,k}\} \quad \forall j, k \in \{1,2, \dots, n\}, j \neq k \implies d_{j,k} = 0 \tag{3.2}$$

The input data is inputted into a 1D-CNN for training, and the same tests are performed after fitting the model. Figure 28 illustrates the positions of the predicted classes after applying varying levels of noise, where the test patterns are listed in the order shown at the bottom of the figure. The softmax prediction probability is depicted in different colors, ranging from blue to red, while the true class positions are indicated with green boxes, and the incorrect prediction classes are marked with red boxes. The results for three samples are shown under varying noise levels. The top chart displays the class positions with high signal-to-noise ratios (SNRs), resulting in 100% classification accuracy. As the noise level increases, the prediction positions begin to shift from the true classes, as demonstrated in the middle chart of the figure. The sliding window acts as a correlator to explore the similarity between adjacent windows. The shift becomes more pronounced with increasing noise, as illustrated in the bottom chart of the figure. Notably, this position shifting occurs not only in cases with the largest value but also in those with the second and third largest values.


∞
~
E_S/N_0
Window No.
Class No.

	0	1	2	3	4	5	6	7	8	9	10
0	0.99929	0.00071	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00071	0.99713	0.00216	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.00216	0.99359	0.00425	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00425	0.98937	0.00638	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.00638	0.98591	0.00771	0.00000	0.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.00000	0.00772	0.98466	0.00762	0.00000	0.00000	0.00000	0.00000
6	0.00000	0.00000	0.00000	0.00000	0.00000	0.00764	0.98617	0.00619	0.00000	0.00000	0.00000
7	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00621	0.98973	0.00406	0.00000	0.00000
8	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00408	0.99385	0.00208	0.00000
9	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00209	0.99725	0.00066
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00066	0.99934

	0	1	2	3	4	5	6	7	8	9	10
0	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.99999	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00551	0.93003	0.06446	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.01365	0.75916	0.22718	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.02149	0.97804	0.00048	0.00000	0.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.00000	0.30671	0.69322	0.00007	0.00000	0.00000	0.00000	0.00000
6	0.00000	0.00000	0.00000	0.00000	0.00001	0.03920	0.95888	0.00191	0.00000	0.00000	0.00000
7	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.06905	0.92201	0.00894	0.00000	0.00000
8	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00810	0.99175	0.00015
9	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00003	0.99875	0.00123
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

	0	1	2	3	4	5	6	7	8	9	10
0	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.02775	0.88534	0.08623	0.00067	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00001	0.00007	0.00245	0.99739	0.00009	0.00000	0.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.00000	0.79468	0.19881	0.00651	0.00000	0.00000	0.00000	0.00000
6	0.00000	0.00000	0.00000	0.00000	0.00000	0.00039	0.20184	0.13166	0.59660	0.06951	0.00000
7	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00012	0.04942	0.95046	0.00000
8	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.02031	0.97969
9	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.99999
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

Figure 28. Positions of the predicted class with the input data contenting different levels of noise.

Each of the subset of samples is referenced as one window frame, denoted as F, and $F \subseteq \mathbb{R}^{n \times n}$, the collection of the indexes of the maximum value in each window frame can be expressed as (3.3):

$$I^f = \{argmax(\psi(\varphi(X^f)))\} \tag{3.3}$$

Where f is the index of the current frame, φ denotes the 1D-CNN function and $\psi$the softmax function. As discussed before, to improve noise tolerance of the model, the argmax and the jth positions to it are both considered. Here, this window with size of j is referred as the error margin window. The index sets of the j^{th} to the maximum value can be obtained, denoted as I_j . A new set of $Q,$ is therefore constructed by containing all j number of sets,

Let f be the index of the current frame, φ represent the 1D-CNN function, and ψ denote the softmax function. As previously discussed, to enhance the model's noise tolerance, both the argmax and the j^{th} positions are taken into account. In this case, the window with a size of j is referred to as the error margin window. The index sets of the j^{th} to the maximum value can be obtained, and are denoted as I_j. A new set Q is then constructed in (3.4), which contains all j number of sets.

$$Q^f = \{I_i^f \mid i \in \{1, \dots, j\}\} \tag{3.4}$$

Where$Q \subseteq \mathbb{R}^{n \times j}$. One can reshape Q to $Q \subseteq \mathbb{R}^{1 \times k}$ where $k = n \cdot j$. Each of the element in Q is multiplied with a weight w, and the standard deviation (STD) is performed in (3.5), and the set of the STD rolling across over all the window frames can be expressed as (3.6):

$$\sigma^f = \sqrt{\frac{\sum_i^k (w_i \cdot Q_i^f - \bar{Q}^f)^2}{k}} \tag{3.5}$$

$$\sigma = \{\sigma^f \mid f \in \{1, \dots, l - n\}\} \tag{3.6}$$

Last but not least, having an anomaly threshold $\delta \in R$, x_i is marked as abnormal if and only if $\sigma_i > \delta$. An appropriate value for δ is determined based on the training data, we chose $0.2 \cdot (\sigma_{max} + \sigma_{\min})$ as the δ for all the experiments demonstrated in this paper.

The training process can be treated as an unsupervised approach by setting w_i to 1, indicating that outputs within the error margin window are equally weighted. This setup can effectively enhance the model's performance, particularly when the SNR is low. Rather than relying solely on the argmax, the error margin window provides a buffer for position shifting errors caused by noise. However, in cases where the SNR is high, equally weighted output may diminish the performance of the model, since it forces the model to process irrelevant outputs. For example, when the anomaly data can be detected with 100% accuracy using the argmax, adding adjacent output (such as the second to the argmax) to the process is equivalent to adding additional noise, resulting in difficulty in choosing the optimal threshold.

To address this issue, the proposed model incorporates a new STD (standard deviation) layer, which utilizes the backpropagation of the neural network to adjust the weights for outputs within the error margin window based on the training dataset. This layer enables the model to optimize its performance by learning the channel characteristics, such as SNR, as well as other underlying patterns caused by the environment. However, the downside of this method is that it necessitates training datasets. Using the training set, the proposed model aims to minimize the error by utilizing the mean squared error optimization function (3.7), and the w_i can be computed through the backpropagation in (3.8).

$$E = \frac{(\widehat{\sigma} - \sigma)^2}{2} \tag{3.7}$$

$$\frac{dE}{dw_i} = \frac{d}{2dw_i}(\hat{\sigma} - \sigma)^2 = \frac{d\sigma}{dw_i} \cdot (\sigma - \hat{\sigma})$$

$$\frac{d\sigma}{dw_i} = \frac{d}{dw_i}\sqrt{\frac{\sum_i^k (w_i \cdot Q_i - \bar{Q}_\iota)^2}{k}}$$

$$\frac{dE}{dw_i} = \frac{(\sigma - \hat{\sigma})}{k}\sum_i^k x_i(w_i \cdot x_i - \bar{Q}_\iota)\left\{\sqrt{\frac{\sum_i^k (w_i \cdot x_i - \bar{Q}_\iota)^2}{k}} - \hat{\sigma}\right\}\left\{\sqrt{\frac{\sum_i^k (w_i \cdot x_i - \bar{Q}_\iota)^2}{k}}\right\} \quad (3.8)$$

We name the model Argmax_STD, the model's structure is depicted in Fig. 29. Throughout the rest of this paper, data that belong to known patterns are referred to as normal data, whereas data that do not belong to known patterns are referred to as anomaly data. The training process can be split into two phases: pattern classification and pattern detection. In the pattern classification phase, a 1D-CNN is utilized with a filter size of 64 and a rectified linear unit (ReLU) as the activation function. It is further processed by a max-pooling layer, a flatten layer, and a softmax output layer. The targets are one-hot vector encoded. Similar to the Auto-encoder, the pattern is adopted for training in this phase. In the second phase, mixed normal and anomaly data are inputted into the model for classification. The index positions of the softmax output are utilized for further training the model in the second phase. The outputs within the error margin window for each input frame are fed into a dense layer. The sigmoid function is adapted to ensure the output ranges from 0 to 1, the same as the input. A STD layer with one node is utilized to calculate the standard deviation of this frame of data, and its output is compared to a predetermined threshold. If it is smaller than the threshold, the current frame is classified as normal data; otherwise, it is classified as anomaly data.

3.4 Experiments

The training is conducted in a supervised manner. Input data are labeled with 1 if they contain anomaly data and 0 otherwise. These labeled data are referred to as pattern position targets in Fig. 29.The model's performance was evaluated using both simulation data and experimental data, and the results were compared to those obtained from a 1D-CNN Auto-encoder. A convolution layer with the same hyperparameters was utilized as the encoder for comparison purposes. The models' thresholds were carefully adjusted based on the training data, using empirical methods.

For the simulation data, a linear chirp waveform was selected as the base data to construct the normal datasets. With a sliding window, 100 patterns were extracted and labeled based on the base data, creating the pattern bank. Both square waves (shown in Fig. 5) and Gaussian noise (shown in Fig. 6) were selected as the anomaly data. Normal data were randomly chosen from the pattern bank and inserted randomly into the anomaly data. In this manner, the simulation dataset was generated. It is noteworthy that additional Gaussian noise was added to the simulation dataset to differentiate the training data and testing data, hence the normal data are not identical to the data in the pattern bank. Conversely, for the tests with real data, the normal data and the data in the pattern bank belong to different sequences from the start.

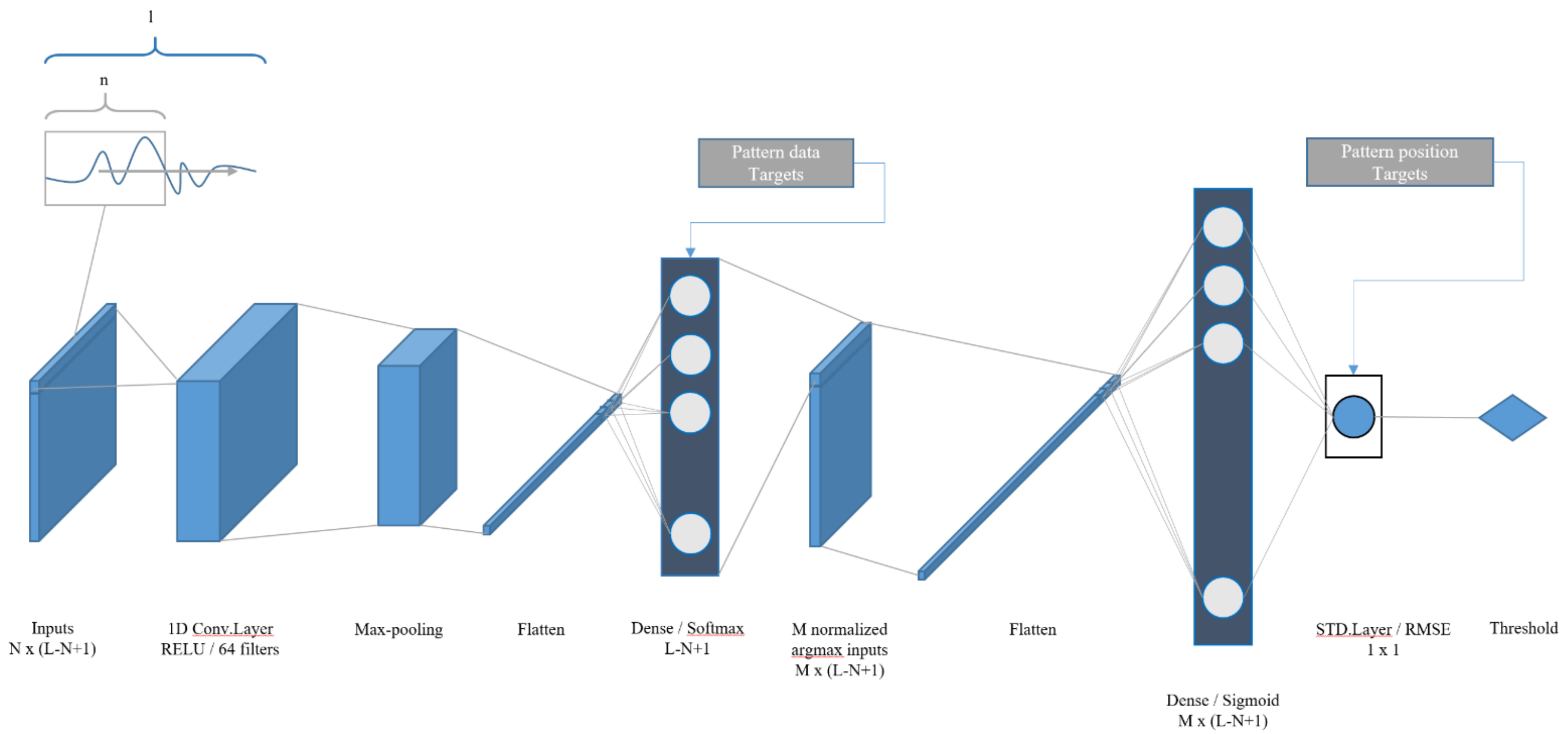


Figure 29. Argmax_STD anomaly detection model structure.

Table VI. Model configuration and hyperparameters of Argmax-STD.

sliding window size, n	20
normal data size, l	100
error margin window size	20
STD frame size	20x20
filter size	64
kernel size	2
pool size	2

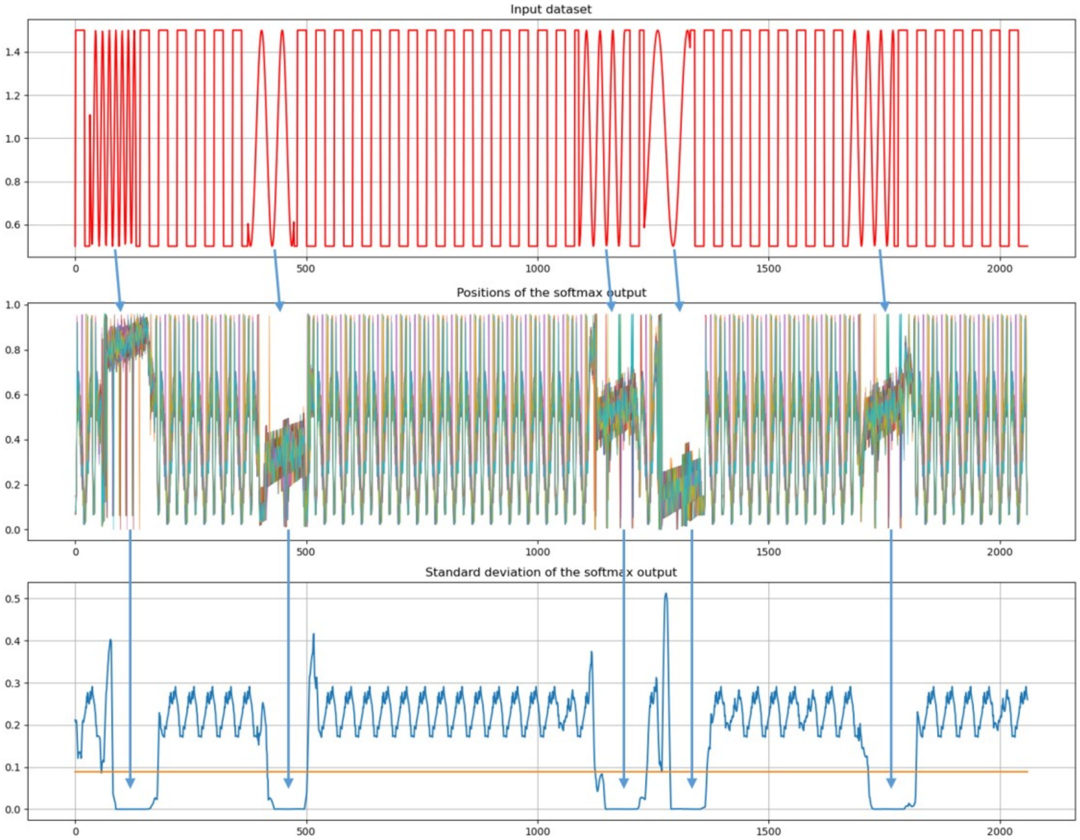


Figure 30. An example of a normal data detection without noise in the input data.

The figure presented in Fig. 30 depicts an illustration of the input dataset containing an anomaly data of square wave. The top panel displays the input dataset, while the middle panel showcases the positions of the softmax outputs within the error margin window, which are also the input for the last STD layer. In this panel, different output positions are individually colored. The bottom panel exhibits the output of the STD layer, where various weights are applied over the data within the error margin window, and the rolling standard deviation is calculated. The predetermined threshold is indicated as the orange straight line, where normal data is denoted if corresponding standard deviation output values are below it. As illustrated in the figure, the proposed model accurately detected all the normal data included in the dataset. Moreover, the proposed model remained effective even when certain levels of noise were present.

3.4.1 Experiment Results with Simulation Data

The Argmax_STD model and the 1D-CNN Auto-encoder model were tested by simulation datasets. Two sets of simulated time series data were generated, wherein the target signal in both datasets belongs to a chirp signal. The first test data includes an unwanted pulse signal, while the second test data comprises Gaussian noise. To increase the difficulty of detection, noise with a SNR of 30dB was added to both datasets. The detection results are shown in Fig. 31 and 32. For better visualization, the grey area is used to display the output of the standard deviation layer. The larger the range size, the higher the output value, and vice versa. The red waveforms represent the input data, while the purple, blue and pulse waves indicate the true indexes of the normal data, the detection indexes of the model and the detection indexes generated by the Auto-encoder. At this moment, the threshold values for both models are the same as those applied to the case without noise. In this example, the proposed model provides higher detection accuracy than Auto-encoder, it reaches 94% and 87% of accuracy for the two simulation datasets, outperforming the 86% and 83% of accuracy from the Auto-encoder adopted.

To assess performance, Gaussian noise was incrementally added to the testing dataset. The detection probabilities of the two models under various SNRs are depicted in Fig. 33. The value of each point is the average of 1000 tests, while the error bars represent the corresponding variances. For every test, the target data were randomly selected and inserted. As shown in the figure, the variances of the results obtained from the Auto-encoder are more stable than those of the proposed approach, especially when the SNR is low. However, across the testing SNR range, the proposed approach outperformed the Auto-encoder in terms of average correct detection rate. The Auto-encoder exhibits the best performance when the SNR is above 30dB and completely fails when the SNR drops below 17dB. On the other hand, the proposed model remains functional even when the SNR falls below 20dB. Both models cannot achieve 100% accuracy because some datasets within the sliding window may contain normal data partially but are still labeled as normal data, such datasets would reduce the correct detection rate. Overall, the proposed model outperforms the Auto-encoder. Even in cases of high SNRs, the proposed model offers a higher correct detection rate by approximately 10%. The detection rates exhibit considerable variability over the testing datasets, but it is still meaningful to conduct a comparison study when the same test dataset is applied to the two models. The number of incorrect detections includes the number of missing detections (M), and wrong detections (W). The number of incorrect detections may exceed the number of true normal data (A), resulting in negative probabilities when the SNR is low, as shown in Fig. 33. Thus the Correct detection probability is defined as (3.9).

$$Correct\ detection\ probability = 1 - \frac{M + W}{A} \tag{3.9}$$

F

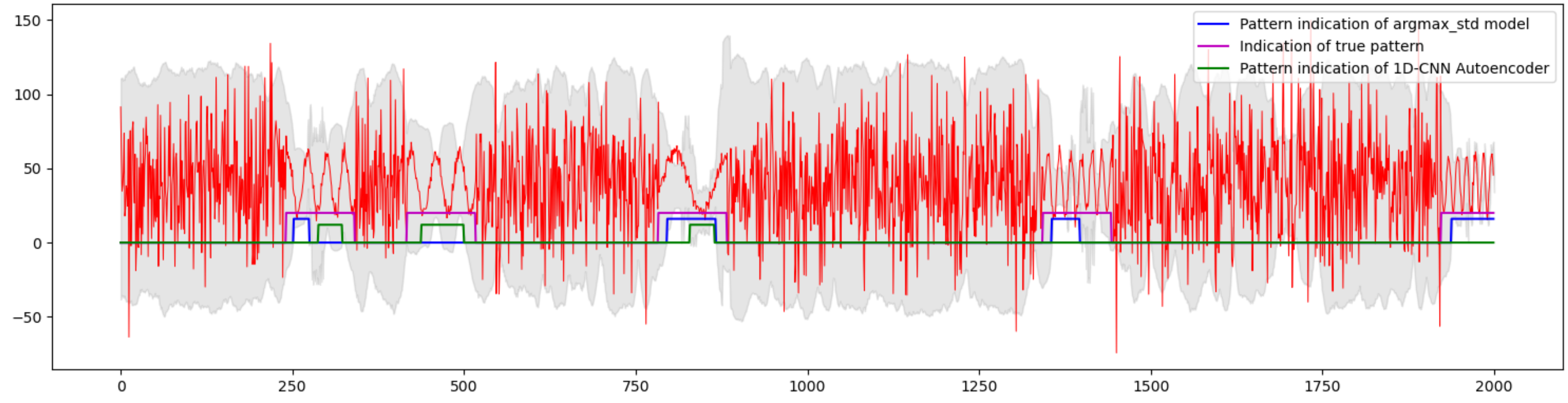


Figure 31. A snapshot of detection result where the anomaly pattern is Gaussian noise.

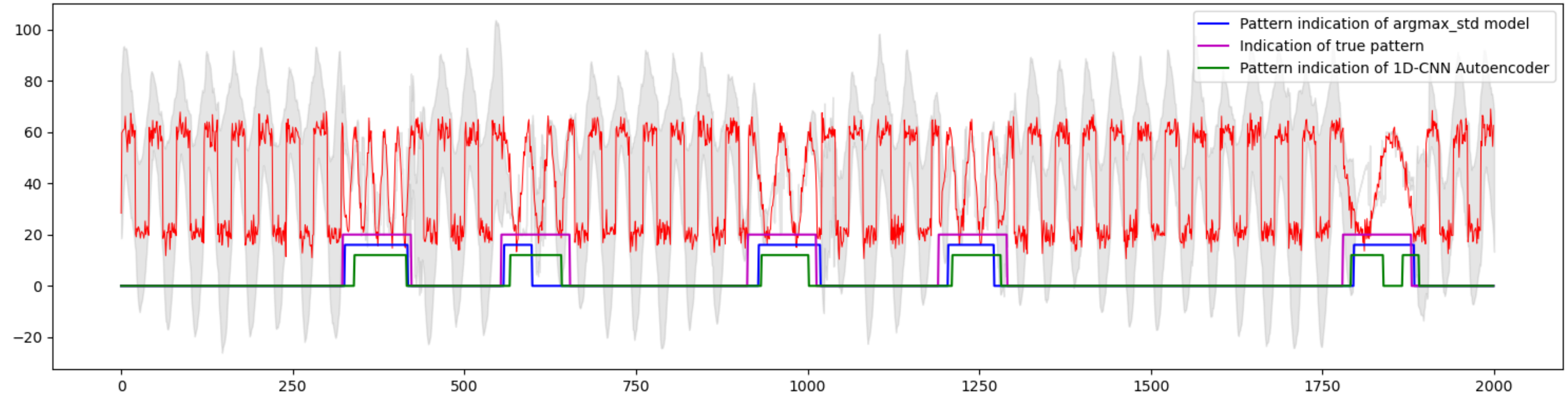


Figure 32. A snapshot of detection result where the anomaly patterns are square waves.

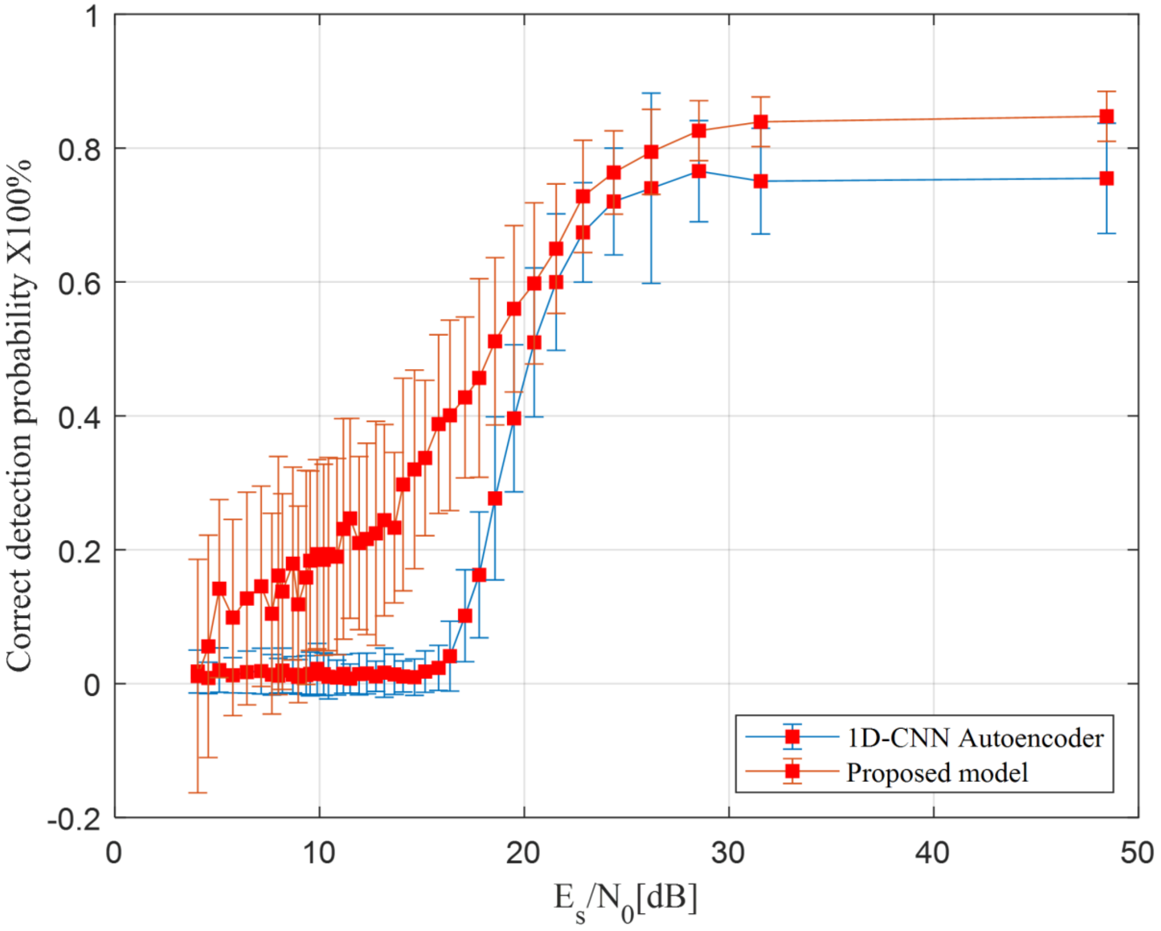


Figure 33. The comparison of the detection accuracies between proposed model and 1D-CNN Auto-encoder under various levels of noise.

3.4.2 Experiment Results with Wireless Sensor Data

The wireless sensor data used for performance evaluation were collected using two LoRa devices. During the experiments, the receiver was situated in a static room at a fixed location, while the transmitter was placed on the tester's wrist to simulate the motions of a smartwatch. LoRaWAN protocol was employed to exchange beacon packets every 50ms for testing purposes. Upon receiving the packets, the receiver conducted Received Signal Strength Indicator (RSSI). Time-domain features were then extracted from the raw data to compress the time series. Three basic human activities were performed in this test setup: slow pace walking, medium pace walking, and fast pace walking. In addition, another dataset collected and published by the University of California Irvine was utilized for the experiments. In this dataset, the RSS data were collected using IRIS nodes embedded with a radio subsystem that implements the IEEE 802.15.4 standard. Unlike our tests, the transmitter note was placed on the tester's right ankle. The packet rate was also set at 20Hz, and the dataset contains activities such as bending, cycling, lying down, sitting, standing, and walking. Cycling activity and walking activity were chosen as normal and anomaly data, respectively, since their data patterns exhibit the highest similarity.

Each figure from Fig. 34 to Fig. 37 presents the results of one of the four experiments, each of which employs a different combination of activities. Each experiment was conducted 200 times, and their overall performances are illustrated in the cumulative distribution function (CDF) of their accuracies in Fig. 38. The average accuracies and accuracy variances of the experiments are listed in Table VII. It is observed that the proposed model achieves a higher average accuracy than the 1D-CNN Auto-encoder, with a minimum margin of 10%. Additionally, the proposed model exhibits more consistent performance than the Auto-encoder, as it attains an average accuracy of 95% in all four experiments while maintaining an accuracy variance of under 1%. Conversely, the performances of the Auto-encoder are less consistent, exhibiting a significant degree of variability depending on the dataset.

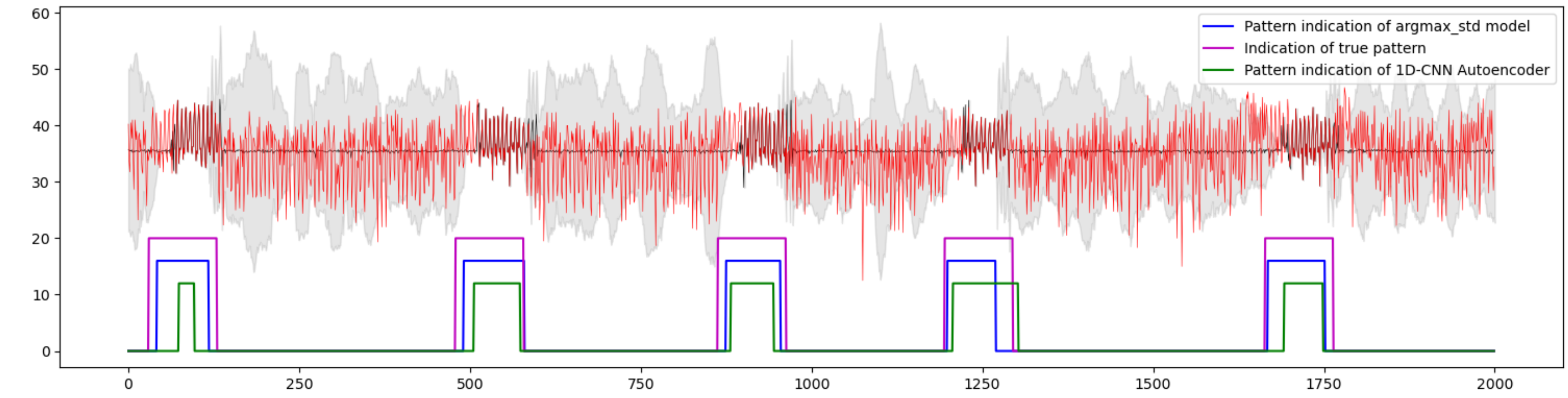


Figure 34. An illustration of a testing result where the normal data is the RSS data of the cycling activity and the anomaly pattern the walking activity.

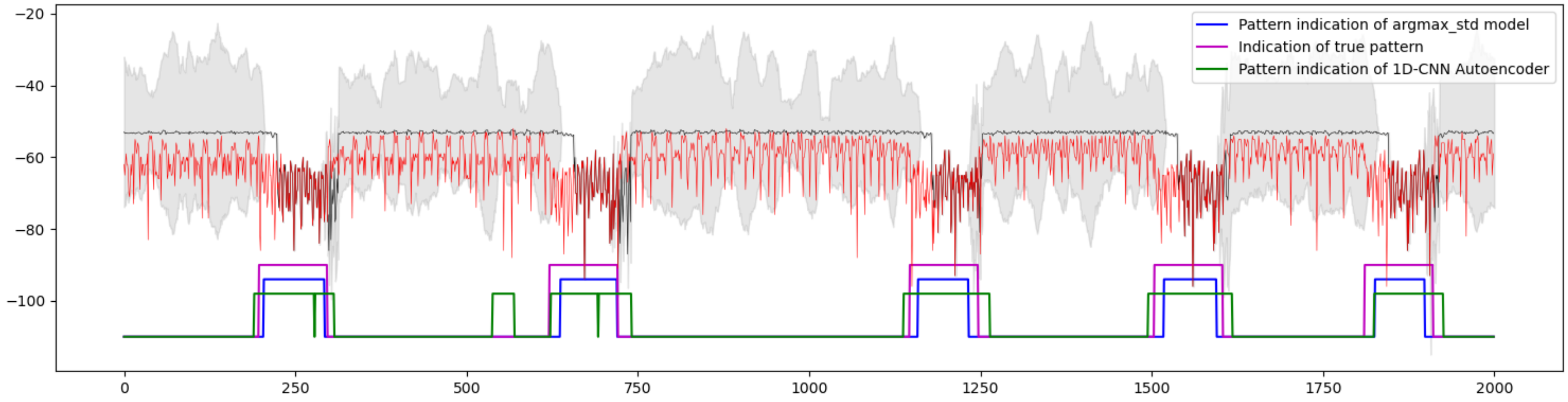


Figure 35. An illustration of a testing result where the normal data is the RSS data of the slow pace walking activity and the anomaly pattern the fast pace walking activity.

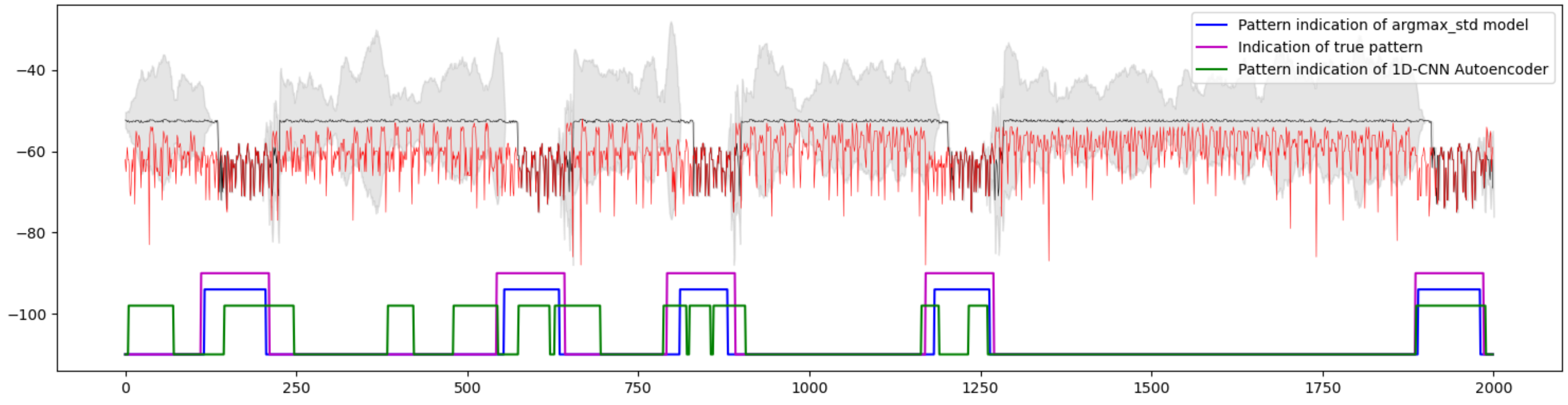


Figure 36. An illustration of a testing result where the normal data is the RSS data of the medium pace walking activity and the anomaly pattern the fast pace walking activity.

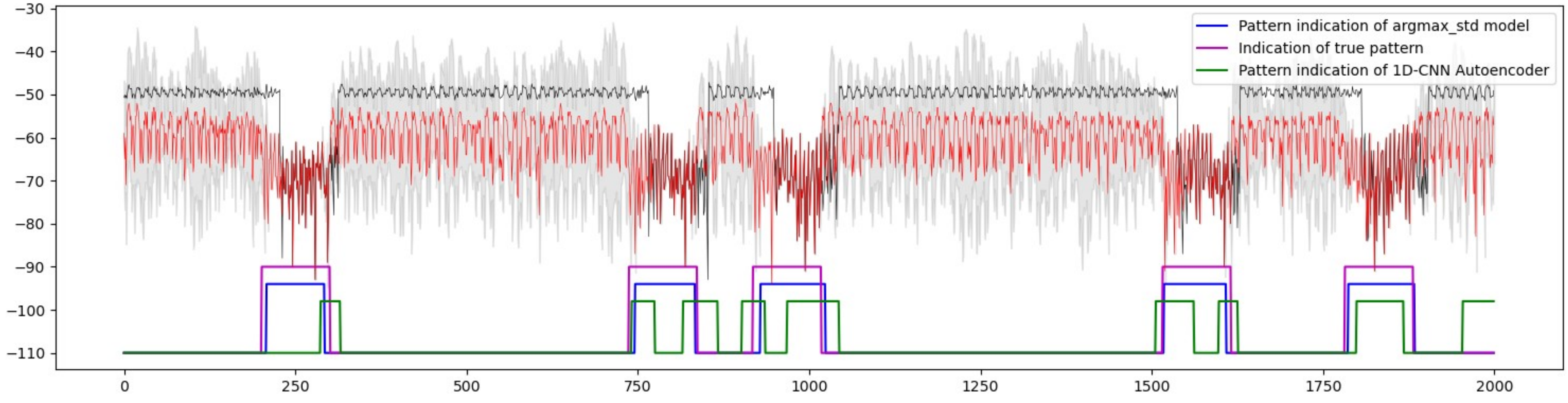
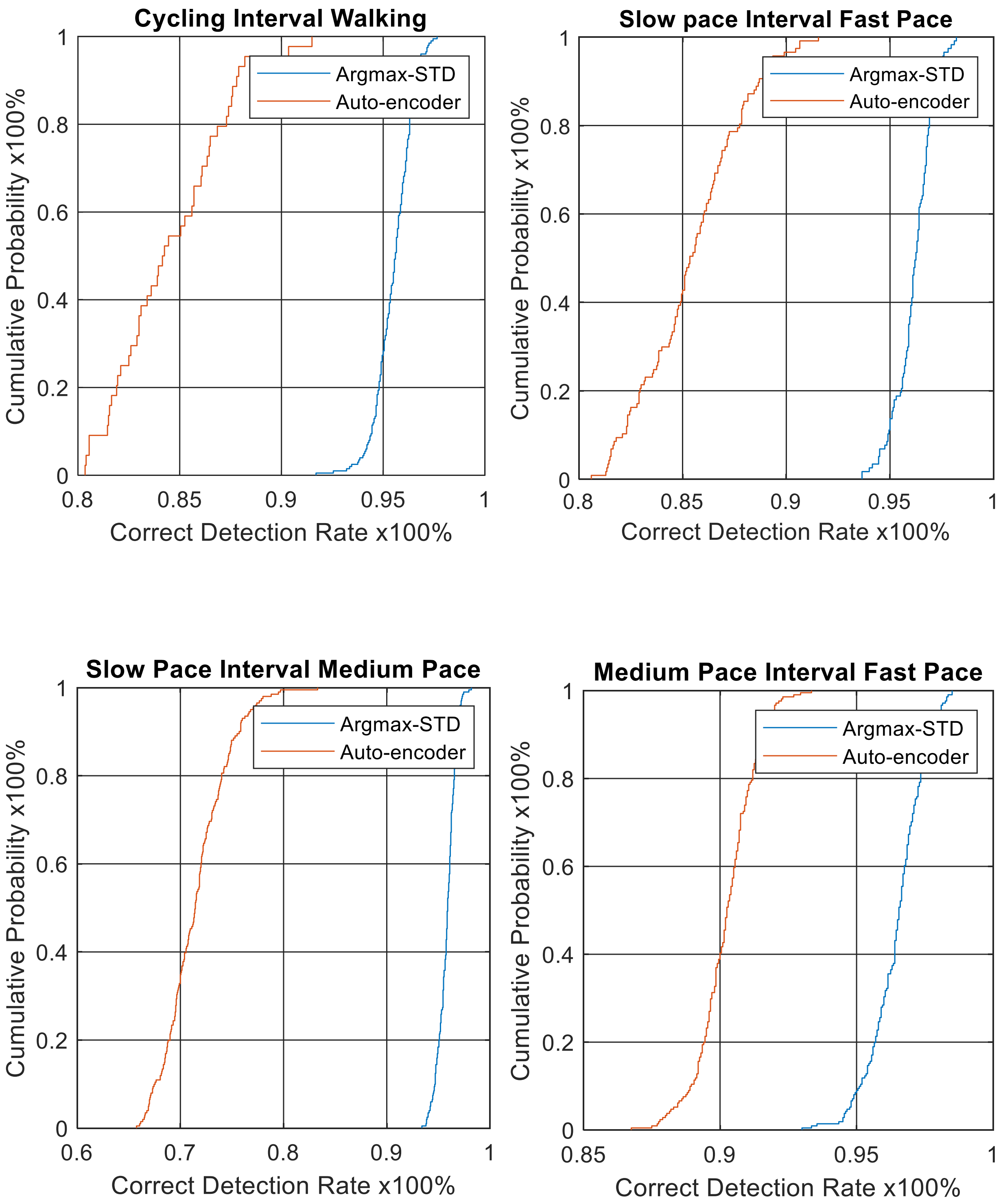


Figure 37. An illustration of a testing result where the normal data is the RSS data of the slow pace walking activity and the anomaly pattern the medium pace walking activity.

Table VII. The average and variance of the detection rate.

Activities	Argmax-STD		Auto-encoder	
	Avg.	Var.	Avg.	Var.
Cycling interval Walking	**95.5**	**0.8**	84.5	7.4
Slow pace interval Fast pace	**96.2**	**0.8**	85.4	5.9
Slow pace interval Medium pace	**95.8**	**0.7**	71.5	9.3
Medium pace interval Fast pace	**96.5**	**1.0**	90.2	1.1

The proposed model outperforms the Auto-encoder in all experiments, with an average improvement of 13% ranging from 83% to 96%. Such a model can be utilized as a novel predictor. For instance, in the case of cycling interval walking, the task is to initially detect cycling activity from walking activity and predict future data. If the cycling activity is not detected, the model performs 1-step prediction using the attention-based LSTM encoder-decoder. If the activity is detected, a 10-step prediction is conducted using the 1D-CNN. The 1D-CNN performs classification on the current input data and outputs the 10th step prediction. The prediction results are represented as black waveforms in the result figures, while the red waveforms denote the true data. As illustrated, the LSTM predictor failed in the 1-step prediction, whereas the 1D-CNN successfully detects normal data and predicts the 10th step of data with a high degree of accuracy. Furthermore, we combine the LSTM with the Auto-encoder and measure the root-mean-square errors (RMSEs) of predictions. As expected, higher accuracy detection leads to a lower RMSE.


Cycling Interval Walking

Slow pace Interval Fast Pace

Slow Pace Interval Medium Pace

Medium Pace Interval Fast Pace

Figure 38. The comparison of the Empirical CDF of the detection accuracies of the proposed model and the Auto-encoder.

Chapter 4 Conclusions

This dissertation involves the design and development of two machine learning-based models for wireless time series applications. One model is designed for long-term predictions, while the other is developed for pattern/anomaly detection. Both models have demonstrated superior performance compared to some state-of-the-art models when applied to the same tasks. For the prediction model, we reconfigure the LSTM cell by integrating a set of IIR gates, the model is able to reduce or eliminate noise and outliers from training dataset, smoothing input data passed to the LSTM gates. A self-adaptive IIR filter is built for each IIR-LSTM cell. Their coefficient weights are constantly updated throughout the training process and eventually reach optimization by implementing gradient descent. A set of GPS distance error data have been collected and are used for evaluating the performance of this model. A comparison is performed between the results of the proposed model and those from conventional neural networks, such as LSTM, GRU, RNN and MLP as well as statistical models, including ARIMA and ETS. The results show that the new model successfully smoothed input data and provided the most accurate results in long-term forecasting. Compared to the conventional models, in average, the proposed model enhance accuracy of the10th step prediction by 34%. In addition, the conventional models often suffer from the noise caused overfitting, result in the lagging between their predictions and true data. On the other hand, the proposed mode shows its capability to overcome or minimize such lagging, owing to the IIR units. The effectiveness of the IIR units is further validated by a set of ablation experiments, the results indicates that the IIR units make a positive impact especially on the long-term predictions. The model is aimed to solve noise caused overfitting problems in wireless channels' time series forecasting.

This model was evoked by the inefficiency and underperformance of the cascade of an IIR and LSTM for long-term prediction applied to our GPS dataset. Our approach is to add an IIR filter as a preprocessing module to each of LSTM cell. Unlike the convention cascade of IIR filter and LSTM, where the coefficients of IIR is predefined based on the statistics of GPS data and cannot be changed during the training phase. The filters in our model are an integral part of the neural network architecture, and their coefficients are dynamically adjusted through the backpropagation process, without prior knowledge of the statistical properties of the input data. Through our experimentation, we have demonstrated the significant impact that proper optimization of the IIR filter can have on the LSTM long-term predictions, and automated optimization is possible with the proposed model. The results based on our GPS data demonstrated that the average performance of our approach outperforms the simple cascaded IIR filters and LSTM solution with the optimized IIR configurations. Furthermore, a retrieve/store mechanism of cell and hidden state were designed for the testing to enhance the long-term prediction accuracy. Our objective redefined LSTM cell with the same input and output interfaces and is compatible with other conventional architectures of neural network. Consequently, this integration can void overfitting caused by noise and outliers of GPS data when adopting conventional LTSM. In addition, the IIR-LSTM cell keeps the same input and output interfaces as a conventional LSTM cell, and can be easily integrated to existing NN.

A new 1D-CNN based pattern/anomaly detection model called Argmax_STD CNN is also developed, which targets on the data detections on the wireless time series data. It utilizes the positional information of the sfotmax output to estimate the confidence of the neural networks.

This novel approached successfully quantified the uncertainty of the CNNs while requiring very low computational resources. We further adopted an error margin window to create a data buffer and thus increase the noise tolerance capability of the model. A standard deviation layer is also created to provide customized weights on the softmax output, optimizing the model performance under different environments. Both simulation and experimental datasets validates that the model outperforms the state-of-art 1D-CNN auto-encoder. Moreover, The Argmax_STD CNN demonstrates its strength in dealing with low SNR data. Simulation results show that when the noise level in the datasets is increased, the 1D-CNN auto-encoder begins to fail completely in detection. However, the Argmax_STD CNN can maintain a correct detection rate of approximately 40%. This suggests that the Argmax_STD CNN is more robust to noise and can perform better than the 1D-CNN auto-encoder in low SNR environments. This is an important advantage in wireless communications, where signals are often distorted by noise, interference, and other factors. The ability to accurately detect and classify signals in such environments can greatly improve the reliability and performance of wireless networks.

Bibliography

[1] K. V. S. Hari, “Channel Models for Wireless Communication Systems”, Wireless Network Design – Optimization Models and Solution Procedures, vol 158, pp 47-64, October 2010.

[2] R. H. Shumway and D. S. Stoffer, “Time Series Analysis and Its Applications”, Springer, pp 8-11, September 2010.

[3] Manaswi and M, Kumar, “RNN and LSTM”, Apress, April 2018.

[4] R. Verma, J. Sharma, and S. Jindal, “Time Series Forecasting Using Machine Learning”, Advances in Computing and Data Sciences, pp.372-381, April 2020.

[5] Y. Liao, J. Moody and L. Wu, “Applications of Artificial Neural Networks to Time Series Prediction”, Handbook of Neural Network Signal Processing, CRC Press, pp 9-4 – 9-5, 2002.

[6] A. Khodabakhsh1, I. Ari1, M. Bakır, and S. M. Alagoz, “Forecasting Multivariate Time-Series Data Using LSTM and Mini-Batches”, Data Science: From Research to Application, vol.45, pp. 121, January 2020.

[7] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, November 1997.

[8] A. Azari1, P. Papapetrou, S. Denic, and G. Peters, “Cellular Traffic Prediction and Classification: A Comparative Evaluation of LSTM and ARIMA”, Discovery Science, pp 129-144, October 2019.

[9] N. T. Co, H. H. Son , N. T. Hoang, T. T. P. Lien, and T. M. Ngoc, “Comparison Between ARIMA and LSTM-RNN for VN-Index Prediction”, Intelligent Human Systems Integration 2020, vol 1131, pp 1107-1112, February 2020.

[10] A. Pinho, R. Costa, H. Silva and P. Furtado, “Comparing Time Series Prediction Approaches for Telecom Analysis”, Theory and Applications of Time Series Analysis, pp 331-345, October 2019.

[11] V.L.Granatstein, “Physical Principles of Wireless Communications, Second Edition”, CRC Press. Taylor & Francis Group, pp 141, February 2012.

[12] R. O.Duda, P. E. Hart and D. G. Stork, “Pattern Classification Second Edition”, New York: Wiley Interscience, 2000.

[13] S. Siami-Namini, N. Tavakoli and A. Siami Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, 2018, pp. 1394-1401, December 2018.

[14] R. Madan and P. S. Mangipudi, "Predicting Computer Network Traffic: A Time Series Forecasting Approach Using DWT, ARIMA and RNN," 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, 2018, pp. 1-5, August 2018.

[15] Z. Zeng and M. Khushi, "Wavelet Denoising and Attention-based RNN- ARIMA Model to Predict Forex Price," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-7, July 2020.

[16] Q. Zhang, F. Li, F. Long and Q. Ling, "Vehicle Emission Forecasting Based on Wavelet Transform and Long Short-Term Memory Network," in IEEE Access, vol. 6, pp. 56984-56994, October 2018.

[17] H. Lu and F. Yang, "A Network Traffic Prediction Model Based on Wavelet Transformation and LSTM Network," 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, pp. 1-4, March 2019.

[18] C. Y. Guo and J. P. Li, "Development and future of wavelet analysis," 2013 10th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, pp. 335-338, December 2013.

[19] S. Liu, V. Elangovan and W. Xiang, "A Vehicular GPS Error Prediction Model Based on Data Smoothing Preprocessed LSTM," 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), Honolulu, HI, USA, pp. 1-5, September 2019.

[20] H. Zhao, X. Zeng and Z. He, "Low-Complexity Nonlinear Adaptive Filter Based on a Pipelined Bilinear Recurrent Neural Network," in IEEE Transactions on Neural Networks, vol. 22, no. 9, pp. 1494-1507, September. 2011.

[21] K. Pachori and A. Mishra, "Design of FIR Digital Filters Using ADALINE Neural Network," 2012 Fourth International Conference on Computational Intelligence and Communication Networks, Mathura, pp. 800-803, November 2012.

[22] N. Allakhverdiyeva, "Application of neural network for digital recursive filter design," 2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT), Baku, pp. 1-4, October 2016.

[23] M. Ohira, A. Yamashita, Z. Ma and X. Wang, "A novel eigenmode-based neural network for fully automated microstrip bandpass filter design," 2017 IEEE MTT-S International Microwave Symposium (IMS), Honololu, HI, pp. 1628-1631, June 2017.

[24] I. Fatih, O. Gurkan and Kuntalp, M, “Importance of data preprocessing for neural networks modeling: The case of estimating the compaction parameters of soils”. Energy Education Science and Technology Part A: Energy Science and Research. 29. 2012.

[25] M. Najim and E. Grivel, “Digital Filters Design for Signal and Image Processing”, ISTE Ltd, pp 137-189, 2006.

[26] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," in IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 279-297, March 1994, doi: 10.1109/72.279191.

[27] P. Campolucci, A. Uncini, F. Piazza and B. D. Rao, "On-line learning algorithms for locally recurrent neural networks," in IEEE Transactions on Neural Networks, vol. 10, no. 2, pp. 253-271, March 1999, doi: 10.1109/72.750549.

[28] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” IEEE Trans. Acoust., Speech, Signal Processing, vol. 37, Mar. 1989.

[29] K. J. Lang, A. H. Waibel, and G. E. Hinton, “A time-delay neural network architecture for isolated word recognition,” Neural Networks, vol. 3, pp. 23–43, 1990.

[30] P. Frasconi, M. Gori, and G. Soda, “Local feedback multilayered networks,” Neural Comput., vol. 4, pp. 120–130, 1992.

[31] P. Frasconi, “Reti ricorrenti ed elaborazione adattativa di sequenze,” Ph.D. dissertation, Dip. di Sistemi e Informatica, Universita di Firenze, Italy, Feb. 1994 (in Italian).

[32] A. D. Back and A. C. Tsoi, “FIR and IIR synapses, a new neural network architecture for time series modeling,” Neural Comput., vol. 3, pp. 375–385, 1991.

[33] E. A. Wan, “Temporal backpropagation for FIR neural networks,” in Proc. Int. Joint Conf. Neural Networks, 1990, vol. 1, pp. 575–580.

[34] N. Benvenuto, F. Piazza, and A. Uncini, “Comparison of four learning algorithms for multilayer perceptron with FIR synapses,” in Proc. IEEE Int. Conf. Neural Networks, 1994.

[35] D. Arman, P. T. Nguyen, R. Faizullin, I. Iswanto, E. F. Armay, “Resolving the Shortest Path Problem using the Haversine Algorithm,” Journal of Critical Reviews, 2020, vol. 7, pp. 62-64.

[36] K. Malek, K. Tashfeen, N. Aboelmagd, E. Ahmed, “GPS Cycle Slip Detection and Correction at Measurement Level,” British Journal of Applied Science & Technology, 2014, pp 4239-4251.

[37] F. M. Salem, “Recurrent Neural Networks From Simple to Gated Architectures,” Springer, 2022, ISBN: 978-3-030-89928-8.

[38] G. Notton, C. Voyant, “Advances in Renewable Energies and Power Technologies”, Elsevier, 2018, pp. 77-114, ISBN 978-0-128-12959-3.

[39] A. Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (end ed.),” O’Reilly Media, 2019. ISBN: 978-1-492-03264-9.

[40] M. Schneider, “Uncertainty management for spatial datain databases: Fuzzy spatial data types”, In Advances in Spatial Databases: 6th International Symposium, SSD’99 Hong Kong, China, July 20—23, 1999 Proceedings 6 (pp. 330-351). Springer Berlin Heidelberg.

[41] P. Ranacher, R. Brunauer, W. Trutschnig, S. Van der Spek and S. Reich, “Why GPS makes distances bigger than they are”. International Journal of Geographical Information Science, 30(2), 2016, pp.316-333.

[42] Erdem, Ergin, and Jing Shi. "ARMA based approaches for forecasting the tuple of wind speed and direction." Applied Energy 88.4 (2011): 1405-1414. networks. In ICML, 2015.

[43] Cook, Andrew A., Göksel Mısırlı, and Zhong Fan. "Anomaly detection for IoT time-series data: A survey." IEEE Internet of Things Journal 7.7 (2019): 6481-6494.

[44] E. S. PAGE, CONTINUOUS INSPECTION SCHEMES, Biometrika, Volume 41, Issue 1-2, June 1954, Pages100–115.

[45] Münz, Gerhard, Sa Li, and Georg Carle. "Traffic anomaly detection using k-means clustering." Gi/itg workshop mmbnet. Vol. 7. No. 9. 2007.

[46] Zolhavarieh, Seyedjamal, Saeed Aghabozorgi, and Ying Wah Teh. "A review of subsequence time series clustering." The Scientific World Journal 2014.

[47] Song, Xiuyao, et al. "Conditional anomaly detection." IEEE Transactions on knowledge and Data Engineering 19.5, 2007

[48] Lee, Seungmin, Gisung Kim, and Sehun Kim. "Self-adaptive and dynamic clustering for online anomaly detection." Expert Systems with Applications 38.12 (2011): 14891-14898. theory of belief functions, pages 73–104. Springer, 2008.

[49] Moshtaghi, Masud, et al. "Clustering ellipses for anomaly detection." Pattern Recognition 44.1 (2011): 55-69. variational inference. ICLR Workshops, 2016.

[50] Ramadas, Manikantan, Shawn Ostermann, and Brett Tjaden. "Detecting anomalous network traffic with self-organizing maps." Recent Advances in Intrusion Detection: 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003. Proceedings 6. Springer Berlin Heidelberg, 2003.

[51] Çelik, Mete, Filiz Dadaşer-Çelik, and Ahmet Şakir Dokuz. "Anomaly detection in temperature data using DBSCAN algorithm." 2011 international symposium on innovations in intelligent systems and applications. IEEE, 2011.

[52] Cheng, Zhangyu, Chengming Zou, and Jianwei Dong. "Outlier detection using isolation forest and local outlier factor." Proceedings of the conference on research in adaptive and convergent systems. 2019.

[53] Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." 2008 eighth ieee international conference on data mining. IEEE, 2008.

[54] Fu, Qiang, et al. "Execution anomaly detection in distributed systems through unstructured log analysis." 2009 ninth IEEE international conference on data mining. IEEE, 2009.

[55] Xie, Yi, and Shun-Zheng Yu. "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors." IEEE/ACM transactions on networking 17.1 (2008): 54-65.

[56] Cho, Sung-Bae, and Hyuk-Jang Park. "Efficient anomaly detection by modeling privilege flows using hidden Markov model." computers & security 22.1 (2003): 45-55.

[57] Han, Sang-Jun, and Sung-Bae Cho. "Evolutionary neural networks for anomaly detection based on the behavior of a program." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 36.3 (2006): 559-570.

[58] Xu, Dan, et al. "Learning deep representations of appearance and motion for anomalous event detection." arXiv preprint arXiv:1510.01553 (2015).

[59] Sebyala, Abdallah Abbey, et al. "Active platform security through intrusion detection using naive bayesian network for anomaly detection." London Communications Symposium. 2002.

[60] Duffield, Nick, et al. "Rule-based anomaly detection on IP flows." IEEE INFOCOM 2009. IEEE, 2009. Wiley, New York, 2000.

[61] Dhaliwal, Sukhpreet Singh, Abdullah-Al Nahid, and Robert Abbas. "Effective intrusion detection system using XGBoost." Information 9.7 (2018): 149.

[62] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In NIPS, 2017.

[63] Amer, Mennatallah, Markus Goldstein, and Slim Abdennadher. "Enhancing one-class support vector machines for unsupervised anomaly detection." Proceedings of the ACM SIGKDD workshop on outlier detection and description. 2013.

[64] Zhou, Chong, and Randy C. Paffenroth. "Anomaly detection with robust deep autoencoders." Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017.

[65] Braei, Mohammad, and Sebastian Wagner. "Anomaly detection in univariate time-series: A survey on the state-of-the-art." arXiv preprint arXiv:2004.00433 (2020).

[66] Su, Ya, et al. "Robust anomaly detection for multivariate time series through stochastic recurrent neural network." Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019.

[67] Elangovan, Vivekanandh, Sheng Liu, and Weidong Xiang. "An Ensembled Approach to Time Series Prediction for Vehicle Communication." 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall). IEEE, 2021.

[68] Elangovan, Vivekanandh, Weidong Xiang, and Sheng Liu. "A Real-Time C-V2X Beamforming Selector Based on Effective Sequence to Sequence Prediction Model using Transitional Matrix Hard Attention." IEEE Access (2023).

[69] Qu, Zhaowei, et al. "A unsupervised learning method of anomaly detection using gru." 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE, 2018.

[70]. S. Chauhan and L. Vig, "Anomaly detection in ECG time signals via deep long short-term memory networks," 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 2015, pp. 1-7, doi: 10.1109/DSAA.2015.7344872.

[71] Malhotra, Pankaj, et al. "Long Short Term Memory Networks for Anomaly Detection in Time Series." ESANN. Vol. 2015. 2015.

[72] Wu, Wentai, et al. "Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality." IEEE Transactions on Knowledge and Data Engineering 34.9 (2020): 4147-4160.

[73] Napoletano, Paolo, Flavio Piccoli, and Raimondo Schettini. "Anomaly detection in nanofibrous materials by CNN-based self-similarity." Sensors 18.1 (2018): 209.

[74] Munir, Mohsin, et al. "DeepAnT: A deep learning approach for unsupervised anomaly detection in time series." Ieee Access 7 (2018): 1991-2005.

[75] Kiranyaz, Serkan, et al. "Convolutional neural networks for patient-specific ECG classification." 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2015.

[76] Li, Dan, et al. "Classification of ECG signals based on 1D convolution neural network." 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom). IEEE, 2017.

[77] Abdeljaber, Osama, et al. "1-D CNNs for structural damage detection: Verification on a structural health monitoring benchmark data." Neurocomputing 275 (2018): 1308-1317.

[78] Eren, Levent, Turker Ince, and Serkan Kiranyaz. "A generic intelligent bearing fault diagnosis system using compact adaptive 1D CNN classifier." Journal of Signal Processing Systems 91 (2019): 179-189.

[79] Azizjon, Meliboev, Alikhanov Jumabek, and Wooseong Kim. "1D CNN based network intrusion detection with normalization on imbalanced data." 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). IEEE, 2020.

[80] Kiranyaz, Serkan, et al. "1D convolutional neural networks and applications: A survey." Mechanical systems and signal processing 151 (2021): 107398.

[81] Vartouni, Ali Moradi, Saeed Sedighian Kashi, and Mohammad Teshnehlab. "An anomaly detection method to detect web attacks using stacked auto-encoder." 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS). IEEE, 2018.

[82] Farahnakian, Fahimeh, and Jukka Heikkonen. "A deep auto-encoder based approach for intrusion detection system." 2018 20th International Conference on Advanced Communication Technology (ICACT). IEEE, 2018.

[83] Georgescu, Mariana Iuliana, et al. "A background-agnostic framework with adversarial training for abnormal event detection in video." IEEE transactions on pattern analysis and machine intelligence 44.9 (2021): 4505-4523.

[84] Chen, Shumei, Jianbo Yu, and Shijin Wang. "One-dimensional convolutional auto-encoder-based feature learning for fault diagnosis of multivariate processes." Journal of Process Control 87 (2020): 54-67.

[85] Li, Fen, et al. "Feature extraction and classification of heart sound using 1D convolutional neural networks." EURASIP Journal on Advances in Signal Processing 2019.1 (2019): 1-11.

[86] Chen, Huan, Yue-Hsien Wang, and Chun-Hung Fan. "A convolutional autoencoder-based approach with batch normalization for energy disaggregation." The Journal of Supercomputing 77 (2021): 2961-2978.

[87] Lampinen, Jouko, and Aki Vehtari. "Bayesian approach for neural networks—review and case studies." Neural networks 14.3 (2001): 257-274.

[88] Pearce, Tim, Felix Leibfried, and Alexandra Brintrup. "Uncertainty in neural networks: Approximately bayesian ensembling." International conference on artificial intelligence and statistics. PMLR, 2020.

[89] Sensoy, Murat, Lance Kaplan, and Melih Kandemir. "Evidential deep learning to quantify classification uncertainty." Advances in neural information processing systems 31 (2018).

[90] Liu, Sheng, Weidong Xiang, and M. Xavier Punithan. "An empirical study on performance of DSRC and LTE-4G for vehicular communications." 2018 IEEE 88th vehicular technology conference (VTC-Fall). IEEE, 2018.