



An XAI-based meta-parameter tuning for time-series forecasting

Hiroyuki Nakagawa^{1,2} · Shimon Sumita² · Ryuichi Iida¹ · Tatsuhiko Tsuchiya²

Accepted: 14 February 2025 / Published online: 11 March 2025
© The Author(s) 2025

Abstract

How can we efficiently determine meta-parameter values for deep learning-based time-series forecasting given a time-series dataset? This paper introduces *Xtune*, an efficient and novel meta-parameter tuning method for deep learning-based time-series forecasting, leveraging explainable AI techniques. In particular, this study focuses on optimizing the window size for time-series forecasting. *Xtune* determines the optimal meta-parameter value for these methods and can also be applied to tune the window size for anomaly detection methods that utilize deep learning-based time-series forecasting. Extensive experiments on real-world datasets and forecasting methods demonstrate that *Xtune* efficiently identifies the optimal meta-parameter value and consistently outperforms the existing methods in terms of execution speed.

Keywords Parameter optimization · XAI · Deep learning · Time-series forecasting · Anomaly detection

Mathematics Subject Classification (2010) 68T20

1 Introduction

How can we efficiently and effectively determine the optimal meta-parameter value for a time-series forecasting method that utilizes deep learning models like LSTM and GRU, without the need to run the method with a large number of meta-parameters? This study focuses on meta-parameter tuning for deep learning-based time-series forecasting. Existing methods for meta-parameter tuning include grid search and random search. However, these methods explore a predefined set of values for each meta-parameter. They systematically evaluate the model's performance considering all possible combinations of these values to

✉ Hiroyuki Nakagawa
h-nakagawa@okayama-u.ac.jp

Shimon Sumita
s-sumita@ist.osaka-u.ac.jp

Tatsuhiko Tsuchiya
t-tutiya@ist.osaka-u.ac.jp

¹ Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530, Japan

² Osaka University, 1-5 Yamadaoka, Suita 565-0871, Osaka, Japan

find the best meta-parameter configuration and can be computationally expensive owing to the exhaustive search over the entire parameter grid.

The goal of this study is to determine the optimal meta-parameter effectively and efficiently. In particular, we focus on tuning the window size of time-series forecasting that uses deep learning techniques. We present *Xtune*, an efficient and novel meta-parameter tuning method for time-series forecasting using deep learning. *Xtune* utilizes explainable AI (XAI), which renders black-box AI models more transparent and interpretable to humans, to grasp the internal state of the deep-learning based forecasting model. We perform our experiments on real time-series datasets and demonstrate the effectiveness, efficiency, and applicability of our method. Our method is an improvement over that reported in [1].

The contributions of this paper are as follows:

- We propose an effective meta-parameter tuning method, named *Xtune*, which leverages explainable AI (XAI) techniques to determine the meta-parameter value for time-series forecasting using deep learning techniques.
- We implemented the proposed method, *Xtune*, as a meta-parameter tuning tool and released it as open-source software.
- We provide empirical evidence demonstrating that *Xtune* efficiently identifies the optimal window size for time-series forecasting utilizing deep learning techniques.

This paper is a revised and extended version of our previous conference paper [1]. The revisions include algorithm acceleration, tool implementation, an enhanced problem definition, evaluation with additional experiments, and an expanded discussion.

The remainder of this paper is organized as follows: Section 2 explains the theoretical background of this study including deep learning-based time-series forecasting and explainable AI (XAI). Section 3 explains the goal of our study. Section 4 explains our effective and novel meta-parameter tuning method. Section 5 explains experiments conducted in this study and results, and Section 6 discuss the performance and applicability of *Xtune* based on the experimental results. Finally, Section 7 concludes this paper and outlines our method.

2 Background

2.1 Deep learning-based time-series forecasting

Deep learning-based time series forecasting involves utilizing neural network architectures to predict future values or patterns in a time-dependent dataset. Several state-of-the-art methods have been developed in this field. Long Short-Term Memory (LSTM) networks [2] belong to the recurrent neural network (RNN) framework that excel in capturing long-term dependencies in sequential data. They have been widely used for time series forecasting. LSTMs incorporate memory cells and gates that allow them to selectively remember or forget information over time. This capability makes them effective at capturing complex temporal patterns and handling sequences of varying lengths. Gated Recurrent Units (GRU) [3] also belong to the recurrent neural network framework similar to LSTM networks. They have a simplified architecture with fewer gates, making them computationally more efficient than LSTMs. GRUs can effectively capture temporal dependencies and have been successfully applied to time series forecasting tasks. Transformer models [4], initially introduced for natural language processing tasks, have also yielded promising results in time series forecasting. These models leverage self-attention mechanisms to capture dependencies between different time steps in the input sequence. By considering relevant time steps, transformer models can

effectively model long-range dependencies and capture complex patterns in time series data. Temporal Convolutional Networks (TCN) [5] are convolutional neural networks designed for sequential data processing. TCNs use dilated convolutions, which allow the receptive field to grow exponentially with depth, enabling them to capture long-term dependencies. TCNs have demonstrated strong performance in time series forecasting tasks, especially when dealing with high-resolution or irregularly sampled data. WaveNet [6] is a deep generative model for time series data developed by DeepMind. It uses dilated causal convolution and autoregressive modeling to generate new samples from the time series distribution. Although primarily designed for audio generation, WaveNet has also been adapted for time series forecasting tasks. Variational Autoencoders (VAE) [7] combine deep learning with probabilistic modeling. They can capture latent representations of time series data and generate new samples from the learned distribution. VAEs have been employed for time series forecasting by encoding the input sequence into a lower-dimensional latent space and then decoding it to generate future predictions.

These state-of-the-art methods demonstrate the effectiveness of deep learning approaches in time series forecasting. They leverage the functionalities of neural networks to learn complex temporal dependencies and capture patterns in the data. The choice of the specific method depends on various factors, such as the nature of the time series data, length of the sequence, presence of irregularities, and computational resources available. Researchers are continuously exploring and refining these methods to achieve even better performance and address specific challenges in time series forecasting.

2.2 Sliding time window approach

A fixed-length sliding window approach is a common technique used in time-series analysis and forecasting to process sequential data. It involves dividing the time-series data into fixed-size windows or segments and analyzing each window individually. The window slides or moves across the time series with a predefined step size, allowing for sequential forecasting and capturing temporal patterns.

The fixed-length sliding window approach works as follows (Figure 1): First, the window size is determined, specifying the number of consecutive time steps included in each window. For example, if the window size is set to 10, each window will contain 10 consecutive data points. Next, the step size or stride is defined, indicating the number of time steps the window moves forward after each analysis, determining the overlap or gap between successive windows. For instance, with a step size of 1, the window moves by one time step at a time. The initial window is then placed at the beginning of the time series, and the data points within the window are extracted for analysis or forecasting. The desired analysis or forecasting task is performed on the data within the current window, which can involve applying statistical methods, machine learning models, or other relevant techniques. After conducting the analysis or forecasting within the current window, the window is moved forward by the defined step size. The next window starts at the next time step, and the process repeats. This sliding window movement continues until the end of the time series is reached, covering either the entire time series data or only a portion of it, depending on the step and window sizes.

Using a fixed-length sliding window approach, we can effectively capture local patterns, trends, and seasonality within the time series data. This method enables localized analysis and modeling while accounting for the temporal nature of the data. It is widely applied in tasks such as feature extraction, anomaly detection, trend analysis, and short-term forecasting.

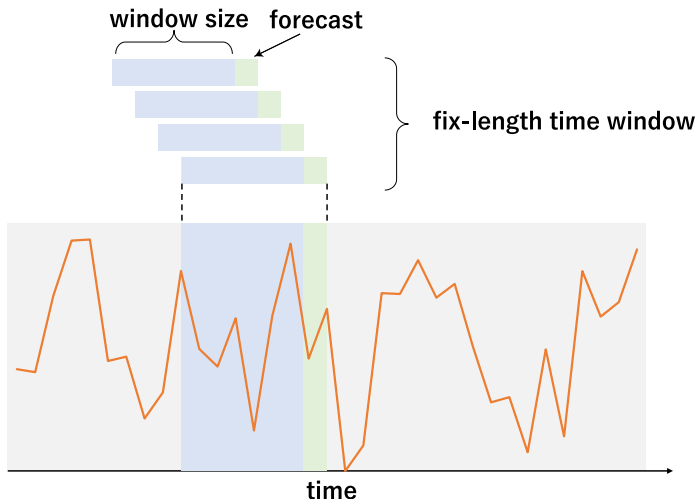


Fig. 1 Supervised learning for time-series forecasting using sliding window approach

Determining the optimal window size, however, is not a straightforward task. Larger window sizes provide more historical data as input, increasing the likelihood of generating accurate forecasts by capturing temporal trends. However, as the window size increases, the amount of input data grows, leading to a higher number of parameters in the model. This, in turn, results in longer training times. Consequently, selecting an optimal window size becomes a critical consideration.

2.3 SALAD

Self-adaptive lightweight anomaly detection (SALAD) [8] is an efficient method for real-time anomaly detection in time-series data with periodic patterns. Many real-world time-series datasets, including transit ridership, network traffic, energy usage, and human gait, are generated in real time and exhibit periodic patterns. Various anomaly detection methods based on statistics and machine learning have been proposed and extensively studied across different domains. However, machine learning-based methods typically require labeled datasets, offline training, and substantial computational resources. On the other hand, statistical methods rely on having prior knowledge of the target time-series for building statistical models. Consequently, neither of these approaches is well-suited for real-time anomaly detection. In contrast, SALAD addresses these limitations and serves as a real-time time-series anomaly detection method. It outperforms five state-of-the-art anomaly detection methods, namely AnomalyDetectionTs (ADT), AnomalyDetectionVec (ADV) [9], GrammerViz3.0 [10], ReRe [11], and RePAD [12], demonstrating superior performance in various scenarios. SALAD utilizes LSTM to construct a predictive model for anomaly detection. Generally, LSTM networks possess complex structures, and incorporating them into systems can result in high computational costs. To mitigate this issue, SALAD adopts a simplified LSTM architecture consisting of a single hidden layer with 10 nodes, aiming to minimize computational effort. However, this straightforward LSTM structure may not effectively capture the intricate patterns present in time-series datasets that exhibit complex behaviors. As an alternative to training a basic LSTM directly on the time-series data to predict future data points and detect anomalies, SALAD converts the time-series data in real-time into a sequence of aver-

age absolute relative error (AARE) values. AARE is a metric used to assess the prediction accuracy of prediction methods. In this approach, a sliding window technique is utilized to train a simplified LSTM using past data. The resulting AARE series data tends to exhibit smoother patterns compared to the original time series, making it conducive for both learning and prediction tasks. The AARE value is calculated using Equation 1, where v_y represents the measured value, $\hat{v}_y$ denotes the predicted value, and b corresponds to the number of past data points utilized in the training process. A lower AARE value indicates a closer alignment between the prediction and measured value.

$$AARE_t = \frac{1}{b} \cdot \sum_{y=t-b-1}^t \frac{|v_y - \hat{v}_y|}{v_y}, t \geq 2b - 1 \quad (1)$$

After SALAD has obtained an adequate number of AARE values, it proceeds with training a basic LSTM model using recent AARE values to predict future AARE values. Subsequently, this trained LSTM model is utilized for anomaly detection. Meanwhile, SALAD continuously updates the detection threshold based on all previously derived AARE values, employing the 3-sigma rule. The calculation of the detection threshold involves Equations 2, 3, and 4. In the SALAD approach, if the discrepancy between the predicted and actual AARE values surpasses the dynamically updated detection threshold, it indicates an abnormal data point. Conversely, if the difference is below the detection threshold, the data point is classified as normal. The detection threshold is continuously adjusted to adapt to changing environmental conditions.

$$thd_{AARE} = \mu_{AARE} + 3 \cdot \sigma \quad (2)$$

$$\mu_{AARE} = \frac{1}{t - b - 1} \cdot \sum_{x=2b-1}^t AARE_x \quad (3)$$

$$\sigma = \sqrt{\frac{\sum_{x=2b-1}^t (AARE_x - \mu_{AARE})^2}{t - b - 1}} \quad (4)$$

The features of SALAD are summarized as follows :

- **Unsupervised:** SALAD does not rely on offline learning, and it does not require experts to build forecasting models in advance or define detection thresholds.
- **Flexible:** SALAD automatically re-trains the LSTM model and updates the detection threshold in real-time as the pattern of the time-series data changes.
- **Lightweight and cost-effective:** By transforming the intricate original time-series data into a simplified series of AARE values, SALAD avoids the need for LSTM models with complex structures, thereby reducing computational complexity.

2.4 XAI

XAI, or explainable artificial intelligence [13], focuses on developing methods and techniques to make AI models more interpretable and understandable to humans. State-of-the-art methods in XAI aim to provide insights into how AI models arrive at their decisions and explanations that can be comprehended and trusted by users. Shapley Additive Explanations (SHAP) [14] is a unified framework for explaining the output of any machine learning model. It is based on cooperative game theory and assigns each feature in a given instance a Shapley value, which represents the contribution of that feature towards the model's prediction.

SHAP values provide a global view of feature importance and can be used to generate intuitive explanations for individual predictions. Gradient-weighted Class Activation Mapping (Grad-CAM) [15] is an XAI technique primarily used in computer vision tasks. It generates visual explanations by highlighting the regions in an input image that contribute the most to the model's prediction. By examining these heatmaps, users can understand which image regions the model focused on when making a decision. Layer-wise Relevance Propagation (LRP) [16] is an XAI method that assigns relevance scores to the input features of a deep neural network. It aims to distribute the prediction's relevance over the input space, providing an explanation for each feature's contribution. LRP can be used to understand how different input features influence the model's decision-making process. Anchor-based methods [17] aim to find simple and interpretable "anchors" or rules that sufficiently explain the model behavior. These anchors are defined as human-understandable conditions on the input data that consistently lead to a particular prediction. By identifying such anchors, users can gain insights into the decision boundaries of the model and understand the model behavior in specific contexts. Counterfactual explanations [18] provide hypothetical scenarios or inputs that, if changed, would lead to a different model prediction. These explanations highlight the features or attributes that have the most significant impact on the model's decision. Counterfactual explanations can help understand why a model made a particular prediction and provide insights into the decision-making process of the model.

These are just a few examples of state-of-the-art XAI methods. The field is continuously evolving, with researchers developing new techniques and approaches to improve interpretability and explainability of AI models. The ultimate goal is to provide users with transparent and trustworthy AI systems that can be easily understood and elaborated on.

3 Problem definition

In this study, we tackle a problem, namely meta-parameter tuning for deep learning-based time-series forecasting. We begin by defining a few key concepts.

Definition 1 (One-dimensional time-series dataset X) *Let $X = \{x_1, \dots, x_n\}$ be a one-dimensional time-series dataset, where x_t is a value at time-tick t .*

Definition 2 (Deep learning-based time-series forecasting method) *Let TF denote a deep learning-based time-series forecasting method. TF trains a forecasting model using the fixed-length sliding window approach.*

We formulate forecasting as a supervised deep learning task. Given a one-dimensional time-series containing N data points $x_{t-N+1}, \dots, x_{t-1}, x_t$, for 1-step ahead prediction, the input X of the supervised deep learning model is $x_{t-N+1}, \dots, x_{t-1}$, and output Y is x_t .

Our goal is to determine an optimal meta-parameter value both effectively and efficiently. Forecasting methods tuned with the optimal meta-parameter value exhibit high prediction performance without overfitting, while also reducing model training time. We can use hyper-parameter optimization (or tuning) methods [19, 20] for this purpose. Grid search and random search explore a range of parameter values, selecting the one that yields the highest prediction accuracy. Bayesian optimization algorithms, particularly effective for black-box optimization problems, offer an alternative approach. For example, Bayesian Optimization with Gaussian Processes (BO-GP) [21] constructs a probabilistic surrogate of the objective function from accumulated observations and identifies the next evaluation point by optimizing the surrogate

function. Similarly, the Tree-structured Parzen Estimator (TPE) [19] uses a tree-structured estimator to enhance search efficiency. However, these approaches can be computationally intensive. Furthermore, while RMSE is a common metric for evaluating forecast performance, identifying truly effective meta-parameter values remains challenging due to overfitting. Forecasting models with poor generalization performance may still achieve high forecasting accuracy, making it difficult to assess the genuine quality of meta-parameter choices.

In this study, we focus on optimizing the window size of the sliding window approach as a meta-parameter. We propose a novel meta-parameter tuning method that addresses the limitations of conventional approaches. Specifically, the problem that we want to solve is as stated follows:

Definition 3 (Meta-parameter tuning for deep learning-based time-series forecasting) *Given a one-dimensional time-series dataset X and deep learning-based time-series forecasting method TF , find the optimal window size w_{opt} .*

How is the window size related to the performance of the forecasting method as a meta-parameter? Larger window sizes provide more historical data as input and are more likely to lead to good forecasts capturing temporal trends. However, the number of input data increases with the number of parameters in the model, which in turn increases the training time. Therefore, selecting the optimal window size is an important issue.

4 Xtune

This section describes our proposed method and its implementation, named *Xtune*¹. First, consider how to reduce the extensive computational time required by conventional meta-parameter tuning methods. The conventional approach involves running the prediction method with various meta-parameter values and selecting the one that yields the highest accuracy as the optimal meta-parameter value based on the results. This process has a time complexity of $O(mn)$, where m is the time required for one run and n is the number of meta-parameters to be explored.

To significantly reduce the computation time, one approach is to find an evaluation metric that can estimate prediction performance to some extent before executing the prediction method. One effective way to achieve this is by considering the internal state of the deep learning prediction model that serves as the foundation of the forecasting method. The internal state of the forecasting model refers to the specific components or elements that are considered crucial when calculating the forecasted value given a particular input. As the input time-series data includes more past data, deep learning models are believed to have a diminishing impact on the forecasting process. Identifying these significant components can potentially reduce the number of meta-parameters that need to be explored. In this context, XAI proves to be effective in understanding the internal dynamics of the forecasting model. XAI techniques enable a better understanding of the model's internal workings, allowing us to identify the important parts and gain insights into how the model generates forecasts. To determine the contribution of each input value to the prediction, XAI techniques can be applied to a deep learning-based prediction model constructed using a subset of the full dataset. In this study, we utilized SHAP, which offers specialized calculation methods for deep learning models.

SHAP [14], which stands for SHapley Additive exPlanations, is a method in explainable artificial intelligence (XAI) domain that aims to provide interpretability and understanding

¹ <https://github.com/h-nak/Xtune/>

of the predictions made by complex machine learning models. It is based on game theory and the concept of Shapley values, which were introduced by Lloyd Shapley in the 1950s. In machine learning, SHAP provides a way to explain the contribution of each feature or input variable to the output or prediction of a model. This approach quantifies the importance of each feature based on a Shapley value, which represents the average marginal contribution of that feature across all possible combinations of features, as shown in Figure 2.

The high-level overview of SHAP is as follows: SHAP utilizes the concept of Shapley values from cooperative game theory, where players form coalitions and receive payoffs based on their contributions. Shapley values quantify the contribution of each player by considering all possible coalitions and their marginal contributions. SHAP focuses on providing explanations at the individual prediction level, aiming to explain why a specific prediction was made by the model for a particular instance. It assigns an importance value to each feature of the prediction by considering different combinations of features and calculating their contributions to the prediction. By averaging these contributions across all possible feature combinations, SHAP determines the importance of each feature. To efficiently compute Shapley values, SHAP introduces the Shapley kernel, which approximates the Shapley values by taking samples from the joint distribution of the input features. The kernel assigns weights to the samples based on their similarity to the instance being explained. Finally, SHAP provides an additive feature attribution property, meaning that the sum of the feature importances explains the difference between the model's output for a specific instance and a baseline prediction, such as an average prediction or background data. SHAP provides insights into how individual features influence the model's predictions. It helps us identify the influential factors, detect biases, understand model behavior, and build trust in AI systems.

Figure 3 illustrates the outcomes obtained by applying SHAP to an actual prediction model. Specifically, Figure 3 displays the contribution of each input value to the forecast with different window sizes, namely, 10, 50, 100, and 150. As the window size increases, it becomes apparent that certain data points on the left side have no contribution to the forecast. Around 30 data points before the forecast point, it can be observed that data points have a significant influence, while those beyond that do not affect the model's forecast. This suggests that there is no need to search for window sizes where non-contributing data points appear. Moreover, the fact that even some past data contributes to the forecast indicates that the model effectively captures the major trends in the data. Therefore, if we can determine the optimal window size where the model no longer contributes to the forecast, the model can be constructed to exhibit good forecasting performance without overfitting. Building upon these assumptions, we propose a novel meta-parameter tuning method, namely, *Xtune*, that utilizes the contribution of input values to the prediction calculated by SHAP.

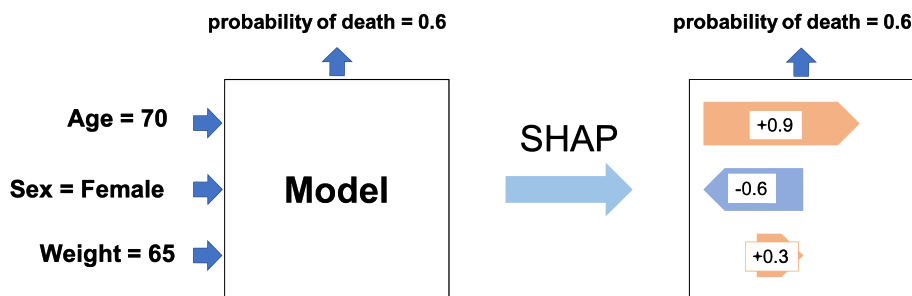


Fig. 2 Using SHAP to make a black-box AI model interpretable to humans

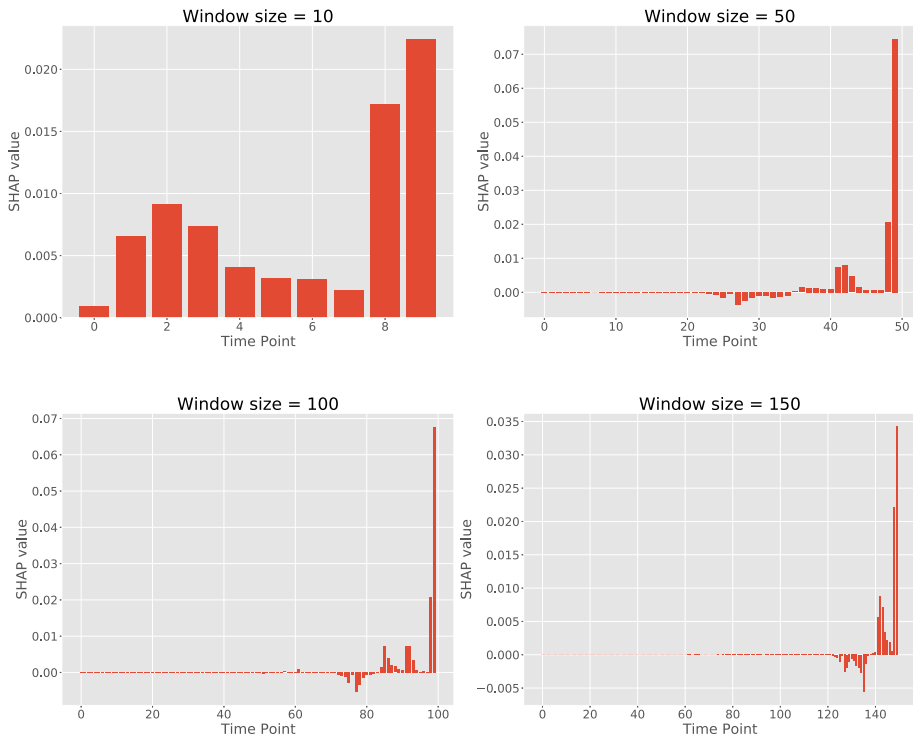


Fig. 3 SHAP values for window sizes of 10, 50, 100, and 150

Algorithm 1 presents the overall procedure of *Xtune* (Figure 4). Given a one-dimensional time-series dataset X , a deep learning-based time-series forecasting method TF , and training window size w_{train} , *Xtune* firstly extracts n samples from X . It then trains a forecasting model

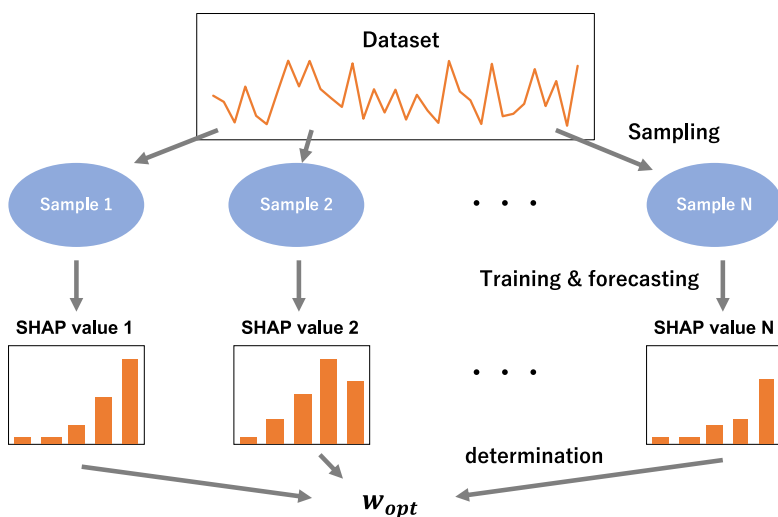


Fig. 4 *Xtune* uses SHAP to determine the optimal window size (w_{opt})

Algorithm 1 *Xtune*

Input: X : one-dimensional time-series dataset, TF : deep learning-based time-series forecasting method,
 w_{train} : training window size;
Output: w_{opt} : optimal window size;
1: $L \leftarrow \emptyset$; /* L : candidate list */
2: Extract sample S_i from X for $i = 1, \dots, n$;
3: **for each** sample S_i **do**
4: TF trains a forecasting model with w_{train}
5: Compute SHAP values V_{shap}
6: Compute the candidate window size w_i /* Equation 5 */
7: Add w_i into L
8: **end for**
9: $w_{opt} \leftarrow \text{determine}(L)$
10: **return** w_{opt}

with w_{train} using each sample S_i and computes SHAP values V_{shap} of the prediction. Using the SHAP values, it determines the point, after which the model does not contribute to the prediction. The candidate window size w_i is computed as follows:

$$\frac{\sum_{j=1}^{w_i} |V_{shap}[w_{train} - j]|}{\sum_{j=1}^{w_{train}} |V_{shap}[j]|} \geq \theta_{thd} \quad (5)$$

The left side of Equation 5 represents what percentage of the total contribution is provided from w_{train} to w_i . The smallest integer value by which it exceeds a certain threshold θ_{thd} is the candidate window size. w_i is added into the candidates list L . The optimal window size is determined from the candidates in the list L . If the values of all elements fall within a narrow range, the mean of these values may be selected as the window size. If the candidates are more dispersed, the value with the highest frequency should be adopted.

5 Experimental evaluation

In this section, we demonstrate the effectiveness of *Xtune* on real datasets. To ensure the repeatability of our results, we used publicly available datasets. Our experiments were conducted on a PC equipped with an Intel Core i9-14900K CPU and 64.0 GB of memory, running Windows 11 Pro. We performed our experiments on the following real-world datasets. We normalized the values contained in each dataset.

- **Case 1 (New York City Taxi Demand)²:** This dataset was obtained from the Numanta Anomaly Benchmark, a novel benchmark for evaluating algorithms for anomaly detection in streaming. It consists of 10,320 data points, which represent the number of cab passengers in New York City from July 2014 to January 2015, aggregated every 30 minutes. It has been used in several studies to evaluate anomaly detection accuracy [23]. This dataset includes 22 anomalies (Figure 5). These anomalies are defined in the following literature [8]. Most of the anomaly days are holidays, anniversaries, and Sundays with special events. Table 1 lists the dates and contents that are defined as anomalies.

² Numerical data is available from <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>.

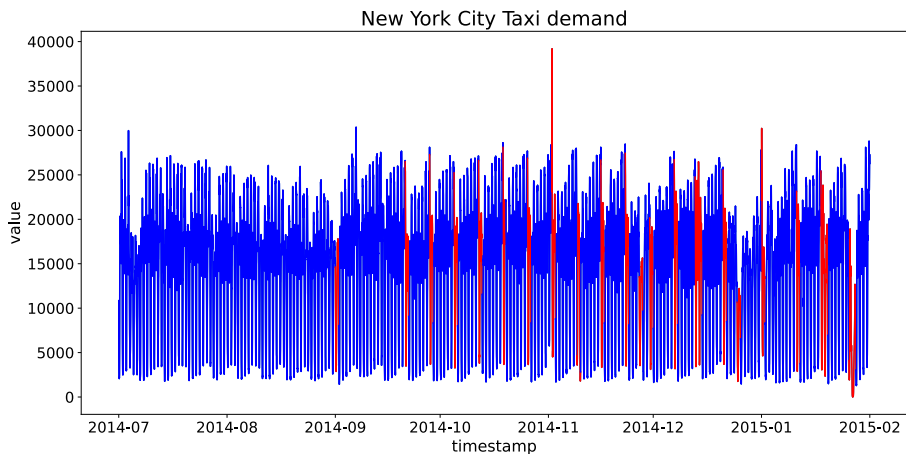


Fig. 5 New York City Taxi Demand dataset with anomalies highlighted in red

- **Case 2 (Wikipedia Page View)**³: This dataset consists of the volume of searches for peyton manning from December 2007 to January 2016, aggregated every day. It consists of 2,905 data points.

5.1 Experiment 1: Performance

We evaluated and compared the following five meta-parameter tuning methods in Experiment 1:

- *Xtune*: This is the proposed meta-parameter tuning algorithm introduced in this paper. The parameters in Algorithm 1 were set as follows: n (number of samples)=3, $w_{train} = 120$, and $\theta_{thd} = 0.98$.
- *XtuneBASE*: In this paper, we refer to the previous tuning method proposed in [1] as *XtuneBASE*. The primary difference between *Xtune* and *XtuneBASE* lies in their approach to determining the optimal window size. *XtuneBASE* starts from the smallest window size and identifies the point where the input portion no longer contributes to forecasting as the optimal window size.
- *Grid search*: Grid search is a hyperparameter tuning method that systematically evaluates a predefined set of values for each parameter. It exhaustively evaluates the model's performance across all possible combinations of these values to find the best parameter configuration. Although effective, it can be computationally expensive due to its exhaustive search over the entire parameter grid. In this study, grid search was applied to determine the window size as a meta-parameter tuning method.
- *Random search*: Random search generates a grid of potential perparameter values and randomly selects combinations to train and evaluate the model. In this experiment, we used optuna [24], a hyperparameter optimization framework, to perform random search. The number of trials was set to 30.
- *TPE (Tree-structured Parzen Estimator)*: TPE [19] is a Bayesian optimization algorithm commonly used for hyperparameter tuning in machine learning. It is particularly suitable

³ <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

Table 1 Dates and events classified as anomalies. Holidays and events, such as the NYC Marathon, are marked as anomalies, consistent with [22]

Year/Month	Day	Holiday/Event
2014/9	1	Labor Day
	21	Climate March
	28	Sunday event
2014/10	5	Sunday event
	12	New York Comic Con
	19,26	Sunday event
2014/11	2	NYC Marathon
	9,16	Sunday event
	23	Football match
	27	Thanksgiving
	30	Sunday event
2014/12	7	Sunday event
	13-14	Millions March NYC
	21	Sunday event
	25	Christmas
2015/1	1	New Year's Day
	11,18	Sunday event
	19	MLK day
	26-27	Blizzard

for black-box optimization problems. In this experiment, we used the TPE functionality provided by optuna, with the number of trials also set to 30.

We used the two dataset, referred to as Cases 1 and 2, and applied the five meta-parameter tuning methods to the time-series forecasting tasks using an LSTM forecasting model. Since *Xtune* produced similar window sizes across three trials in both cases, the average of the results was used as the final window size. Figure 6 illustrates a contribution of each input value to the forecast when the window size w_{train} is set to 120 for both Cases 1 and 2.

Table 2 compares *Xtune* with its competitors in terms of the estimated optimal window size and the computation time required to determine the optimal parameter value in Cases 1 and 2. Additionally, Figure 7 plots the RMSE (root mean square error) values for forecasting models with different window sizes.

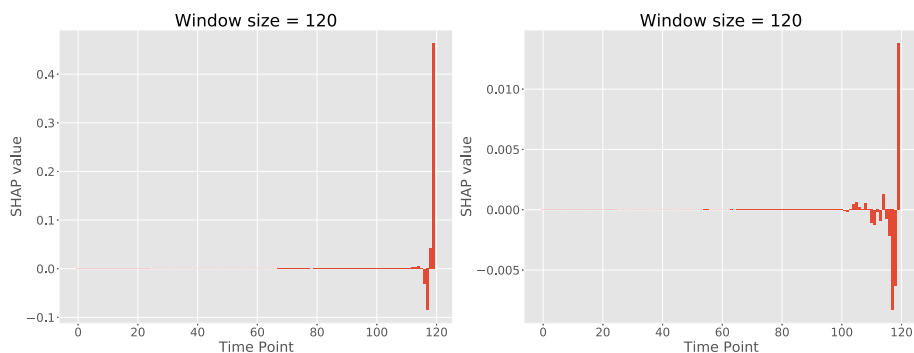


Fig. 6 SHAP values of a trained model with a window size of 120 ($w_{train} = 120$). (Left) Case 1 and (Right) Case 2. The plots suggest optimal window sizes of approximately 8 and 26, respectively.=

Table 2 Optimal window sizes and wall-clock times for various parameter tuning methods

Case Method	New York City Taxi Demand		Wikipedia Page View	
	Window size	Time [s]	Window size	Time [s]
<i>Xtune</i>	8	19.46	26	20.03
<i>XtuneBASE</i>	18	75.38	16	104.87
Grid search	60	882.45	11	981.50
Random search	13	425.52	11	265.26
TPE	12	197.66	14	158.41

5.2 Experiment 2: Applicability

In order to validate the applicability of *Xtune*, we conducted experiments where we applied it to the meta-parameter tuning of SALAD [8], an anomaly detection method that utilizes deep learning-based time series forecasting. This experiment was performed using the dataset from Case 1. Table 3 lists the results, which indicate that both *XtuneBASE* and *Xtune* determine the optimal window size to be nearly 30. Additionally, the findings reveal that *Xtune* achieves a computation speed that is approximately five times faster than that of *XtuneBASE*.

Next, we performed an experiment where SALAD was executed with a specific window size and four metrics, recall, precision, F-measure, and run time, were measured. Subsequently, SALAD was executed with various window sizes to assess whether the window size

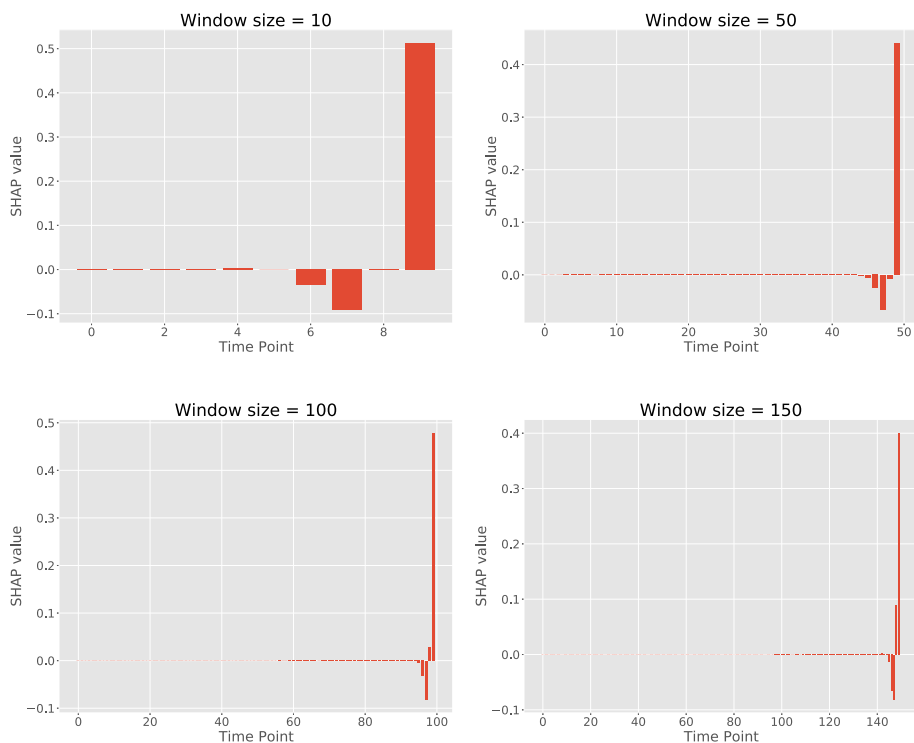
**Fig. 7** RMSE values for different window sizes in Experiment 1. (Left: Case 1, Right: Case 2)

Table 3 Optimal window size and wall-clock time in Experiment 2 (SALAD)

Method	Optimal window size	Wall-clock time[s]
<i>Xtune</i>	29	63.382
<i>XtuneBASE</i>	30	368.195

calculated by the proposed method was indeed optimal. For this experiment, SALAD was run with window sizes of 6, 24, 36, 48, 72, and 96. The corresponding SHAP values, anomaly detection accuracy, and execution times for each window size are presented in Figure 8 and Table 4. Figure 8 displays the SHAP values for the model's prediction when trained with different window sizes. These SHAP values indicate the contribution of each input value to the prediction at time $t = 601$, using data from $t = (600 - \text{window size} + 1)$ to $t = 600$ as input values. With a window size of 6, both the most recent data point ($t = 595$) and that closest to the forecast ($t = 600$) exhibit similar contributions. However, as the window size increases, the historical data ceases to contribute to the forecast from a window size of 30 onwards.

Table 4 summarizes the anomaly detection accuracy and execution times for various window sizes. The results demonstrate an increase in execution time as the window size increases. Additionally, the F-measure accuracy improves until a window size of 30, beyond which it remains stable at approximately 0.75. Based on these results, it can be concluded that the window size determined by *Xtune* is indeed the optimal window size for SALAD. The optimal window size not only leads to high identification accuracy but also ensures efficient execution without significant time delays. This experiment showed that *Xtune* is also effective in anomaly detection methods that utilize deep learning-based time series prediction.

6 Discussion

We now discuss the performance and applicability of *Xtune*.

Performance The results from Experiment 1 demonstrate that *Xtune* successfully identifies the best or equivalent window size for time-series forecasting. Similarly, Experiment 2 shows that the window size determined by *Xtune* is indeed optimal for SALAD. Tables 2 and 3 provide a comparative analysis of *Xtune*'s meta-parameter tuning performance. These results show that *Xtune* is significantly outperforms conventional meta-parameter tuning methods, including grid search and random search, and our previous approach, *XtuneBASE*, in terms of computation time required to determine the best parameter value. Experiment 2 further

Table 4 Results (SALAD)

Window size	Recall	Precision	F-measure	Run time(s)
6	0.381	0.800	0.515	417.4
24	0.809	0.654	0.723	999.1
30	0.857	0.667	0.750	1037.4
36	0.952	0.606	0.740	1238.2
48	0.857	0.667	0.750	1709.7
72	0.905	0.678	0.775	2223.7
96	0.952	0.625	0.755	2446.8

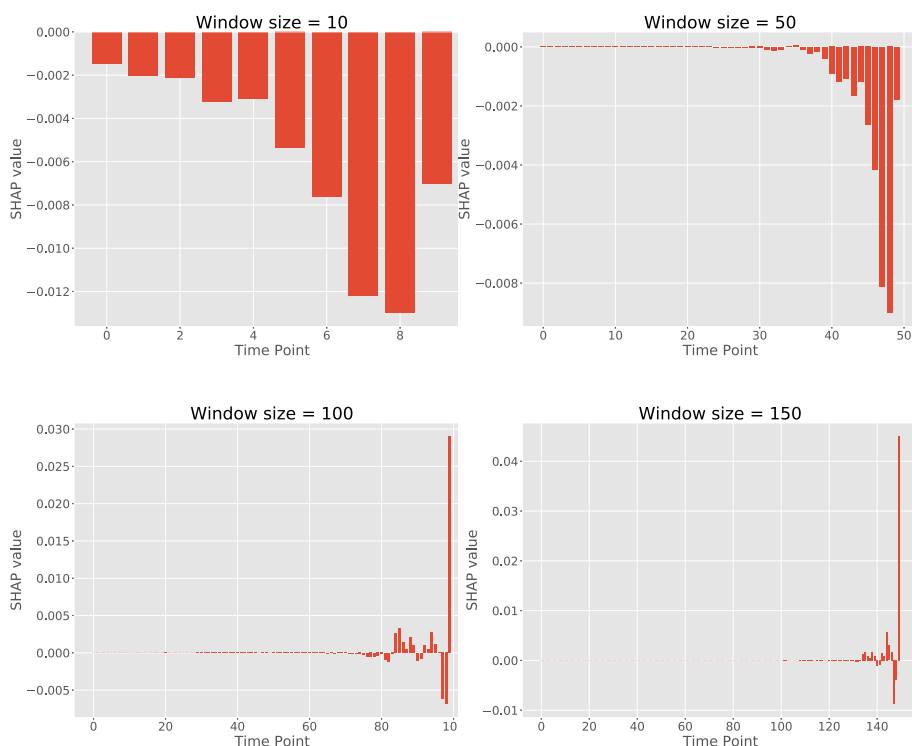


Fig. 8 SHAP values for window sizes of 6, 24, 36, 48, 72, and 96

highlights that the optimal window size identified by *Xtune* not only leads to high identification accuracy but also ensures efficient execution without significant time delays. Figure 9 illustrates SHAP values for various window size in Case 1. These plots indicate that the window size obtained by *Xtune*, 8, is small yet sufficient for accurate forecasting. In general, smaller window size allow for faster model training and quicker forecasting.

Despite its demonstrated efficiency, *Xtune*'s current algorithm has room for improvement, particularly in refining the process for determining the optimal window size (w_{opt}) after several trials. While experiments show that the window size can be determined quickly,

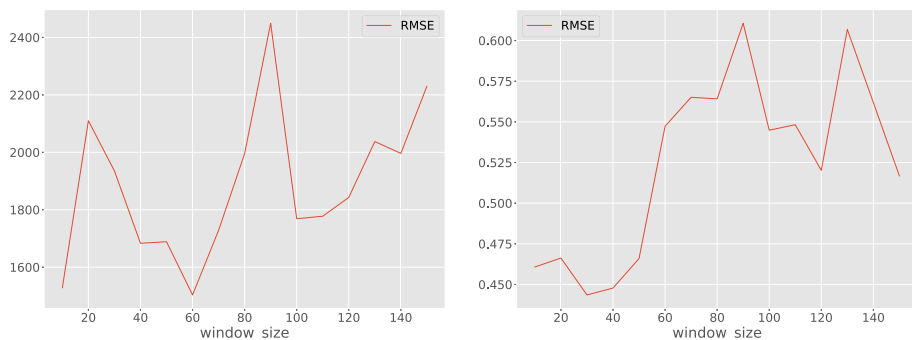


Fig. 9 SHAP values for window sizes of 10, 50, 100, and 150 in Case 1, supporting the claim that a window size approximately 8 is sufficient

increasing the number of trials could further enhance the reliability of the value determination process.

Applicability *Xtune* can be seamlessly integrated with various deep learning-based time-series forecasting models, as demonstrated with the LSTM forecasting model in Experiment 1 and SALAD in Experiment 2. Hyperparameter tuning tools such as KerasTuner [25] focus on identifying optimal hyperparameter values for machine learning models. Most of these search processes employ black-box optimization methods like GP-EI [26] and TPE [27], which are Bayesian optimization algorithms, or CMA-ES [28], which generates solutions from multivariate normal distributions. Other methods, such as HyperBand [29], use multi-fidelity optimization by monitoring the intermediate progress of machine learning training.

While these tuners aim to find appropriate values for the hyperparameters that constitute machine learning models for specific learning objectives, *Xtune* specifically targets the optimal window size, making it highly complementary to existing tuners. This flexibility ensures that *Xtune* can be applied to a wide range of forecasting and anomaly detection tasks.

7 Conclusions

In this study, we addressed the problem of meta-parameter tuning for time-series forecasting using deep learning. Our proposed meta-parameter tuning method, *Xtune*, utilizes SHAP, an explainable AI (XAI) technique, to determine the optimal or equivalent meta-parameter values for deep learning-based time-series forecasting methods. Experimental results demonstrate that *Xtune* effectively identifies the optimal window size significantly faster than existing meta-parameter tuning methods. Additionally, the application of *Xtune* to SALAD further highlights its effectiveness in tuning meta-parameters for anomaly detection methods based on time series prediction.

Future work will primarily focus on the determination of the optimal window size (w_{opt}). Currently, *Xtune* delegates the final decision regarding parameter values to users. We aim to develop an automated decision-making mechanism through exhaustive experimentation and refinement.

Acknowledgements This study was supported by the JSPS Grants-in-Aid for Scientific Research (No.20H04167).

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Hiroyuki Nakagawa, Shimon Sumita, Ryuichi Iida, and Tatsuhiro Tsuchiya. The draft of the manuscript was written by Hiroyuki Nakagawa and Shimon Sumita, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open Access funding provided by Okayama University. This work was supported by the JSPS Grants-in-Aid for Scientific Research (No.20H04167).

Declarations

Competing interests The authors declare no competing interests.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Code availability Not applicable.

Disclosure of potential conflicts of interest Not applicable.

Research involving Human Participants and/or Animals Not applicable.

Informed consent Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Sumita, S., Nakagawa, H., Tsuchiya, T.: Optimal parameter selection using explainable ai for time-series anomaly detection. In: PRIMA 2022: Principles and Practice of Multi-Agent Systems: 24th International Conference, Valencia, Spain, November 16–18, 2022, Proceedings, pp. 281–296. Springer, Berlin, Heidelberg (2022)
2. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**, 1735–80 (1997)
3. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 Workshop on Deep Learning, December 2014 (2014)
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
5. Bai, S., Kolter, J., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. (2018)
6. Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. In: Arxiv (2016)
7. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes (2022)
8. Lee, M., Lin, J., Gran, E.: Salad: Self-adaptive lightweight anomaly detection for real-time recurrent time series. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 344–349. IEEE Computer Society, Los Alamitos, CA, USA (2021)
9. Hochenbaum, J., Vallis, O., Kejariwal, A.: Automatic anomaly detection in the cloud via statistical learning. (2017)
10. Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A.P., Chen, C., Frankenstein, S.: Grammarviz 3.0: Interactive discovery of variable-length time series patterns. *ACM Trans. Knowl. Discov. Data.* **12**(1) (2018)
11. Lee, M., Lin, J., Gan, E.: Rere: A lightweight real-time ready-to-go anomaly detection approach for time series. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 322–327. IEEE Computer Society, Los Alamitos, CA, USA (2020)
12. Lee, M.-C., Lin, J.-C., Gran, E.: RePAD: Real-Time Proactive Anomaly Detection for Time Series, pp. 1291–1302 (2020)
13. Gunning, D., Aha, D.: Darpa's explainable artificial intelligence (xai) program. *AI Mag.* **40**(2), 44–58 (2019)
14. Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17, pp. 4768–4777. Curran Associates Inc., Red Hook, NY, USA (2017)
15. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 618–626 (2017)
16. Binder, A., Montavon, G., Lapuschkin, S., Müller, K.-R., Samek, W.: Layer-wise relevance propagation for neural networks with local renormalization layers. In: International Conference on Artificial Neural Networks (2016)
17. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. **32** (2018)

18. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harvard journal of law & technology*. **31**, 841–887 (2018)
19. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems. NIPS'11*, pp. 2546–2554. Curran Associates Inc., Red Hook, NY, USA (2011)
20. Olof, S.S.: A comparative study of black-box optimization algorithms for tuning of hyper-parameters in deep neural networks. (2018). <https://api.semanticscholar.org/CorpusID:197859832>
21. Brochu, E., Cora, V.M., Freitas, N.: A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning (2010). <https://arxiv.org/abs/1012.2599>
22. Zhang, V.: A Tour of AI Technologies in Time Series Prediction (2019). <https://www.soa.org/4a21e0/globalassets/assets/files/resources/research-report/2019/tour-ai-technologies.pdf>
23. Vasheghani Farahani, I., Chien, A., King, R.E., Kay, M.G., Klensz, B.: Time series anomaly detection from a markov chain perspective. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1000–1007 (2019)
24. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019)
25. Keras: KerasTuner. https://keras.io/keras_tuner/
26. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*. **25** (2012)
27. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*. **24** (2011)
28. Loshchilov, I., Hutter, F.: CMA-ES for Hyperparameter Optimization of Deep Neural Networks (2016). <https://arxiv.org/abs/1604.07269>
29. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **18**(185), 1–52 (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.