Sun *et al. Journal of Big Data* (2024) 11:83
https://doi.org/10.1186/s40537-024-00940-7

Journal of Big Data

METHODOLOGY Open Access

On data efficiency of univariate time series anomaly detection models

Wu Sun[1], Hui Li[1*], Qingqing Liang[1,2], Xiaofeng Zou[1], Mei Chen[1] and Yanhao Wang[2*]

*Correspondence:
cse.HuiLi@gzu.edu.cn;
yhwang@dase.ecnu.edu.cn

[1] State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang 550025, China
[2] School of Data Science and Engineering, East China Normal University, Shanghai 200062, China

Abstract

In machine learning (ML) problems, it is widely believed that more training samples lead to improved predictive accuracy but incur higher computational costs. Consequently, achieving better *data efficiency*, that is, the trade-off between the size of the training set and the accuracy of the output model, becomes a key problem in ML applications. In this research, we systematically investigate the data efficiency of *Univariate Time Series Anomaly Detection* (UTS-AD) models. We first experimentally examine the performance of nine popular UTS-AD algorithms as a function of the training sample size on several benchmark datasets. Our findings confirm that most algorithms become more accurate when more training samples are used, whereas the marginal gain for adding more samples gradually decreases. Based on the above observations, we propose a novel framework called FastUTS-AD that achieves improved data efficiency and reduced computational overhead compared to existing UTS-AD models with little loss of accuracy. Specifically, FastUTS-AD is compatible with different UTS-AD models, utilizing a sampling- and scaling law-based heuristic method to automatically determine the number of training samples a UTS-AD model needs to achieve predictive performance close to that when all samples in the training set are used. Comprehensive experimental results show that, for the nine popular UTS-AD algorithms tested, FastUTS-AD reduces the number of training samples and the training time by 91.09–91.49% and 93.49–93.82% on average without significant decreases in accuracy.

Keywords: Univariate time-series anomaly detection, Data efficiency, Sampling

Introduction

With the rapid advancement of sensor [2] and Internet of Things (IoT) [14] technologies, large volumes of time-series data are generated at an unprecedented speed. Such big time-series data has been widely used to assist real-time decision making in many areas, including IT operations [64], finance [4], and healthcare [28]. For example, cloud service providers collect a series of key performance indicators (KPIs), such as CPU utilization, memory usage, and network I/O, to analyze and optimize the performance and health of their servers. As another example, wearable sensors continuously monitor heart rates, blood pressure, and other measures of body conditions, which can be analyzed to provide actionable information on the health and well-being of patients.

Among various problems with time series data, *anomaly detection* plays a central role due to its prevalence and importance in industrial applications. Specifically, time-series anomaly detection (TSAD) aims to identify unexpected patterns that do not follow the expected behavior from a series of data points observed over time. Those unexpected patterns, or anomalies, typically signify unusual events, such as attacks in enterprise networks [54], structural defects in jet turbine engineering [58], seizures in brain activities [31], and ecosystem disturbances in earth sciences [16]. Accurate anomaly detection can therefore trigger prompt warnings and troubleshooting, helping to avoid potential losses. As such, to detect different types of anomalies from time-series data in various domains, numerous TSAD algorithms have been proposed over the last decades. For example, [48] reported 158 different methods to detect time series anomalies, ranging from statistical analysis, signal processing, and data mining to deep learning models.

Despite extensive studies on TSAD in the literature, to the best of our knowledge, there have not yet been many explorations on their efficiency, that is, how to build a TSAD model with high accuracy using fewer computational resources. Generally, existing studies follow the common assumption that more training samples (typically *sliding windows* in the time series for TSAD) lead to better predictive performance, known as scaling laws [27]. Meanwhile, building a large-scale machine learning (ML) model for TSAD also incurs high costs, especially when computational resources are limited. According to our experimental results, building a deep learning model, e.g., LSTM-AD [37], on the electrocardiogram (ECG) [40] dataset with more than 200k training samples takes nearly 1 week using an Nvidia RTX A6000 GPU (with batch size 128, subsequence length 64, and number of epochs 50). Furthermore, training a classic ML model such as LOF [11] on the same ECG dataset takes about 11 h using a server with 16 CPU cores at 2.40 GHz. Consequently, they fail to support real-time decision making when time series are generated rapidly. Improving the efficiency of TSAD models becomes imperative for their industrial implementation. Toward this end, we aim to improve the *data efficiency* of the TSAD models, that is, the trade-off between the size of the training set and the accuracy of the output model, as fewer data naturally leads to higher time efficiency and lower computational resource consumption.

In this paper, we systematically investigate the data efficiency of Univariate Time Series Anomaly Detection (UTS-AD) models, for which the data points in the time series consist of only one variable without inter-variable dependencies and correlations. In addition, we focus on the task of detecting subsequence anomalies over sliding windows in time series. We first benchmark nine popular UTS-AD methods in different areas (data mining, classic ML, and deep learning) with different learning paradigms (unsupervised and semi-supervised) on univariate time series randomly sampled from three datasets with various anomaly ratios and average subsequence lengths. Based on the benchmark results (see Fig. 1 for details), we obtain three key observations. First, the accuracy of the output model initially improves when the number of training samples increases but then becomes stable or even decreases. Second, the accuracy of the output model built on only a small fraction of sliding windows can be very close to that of all sliding windows. Third, in most cases, the time to build a model with little loss of accuracy on the "small" data is much less than the time to build a model on the full "big" data. These

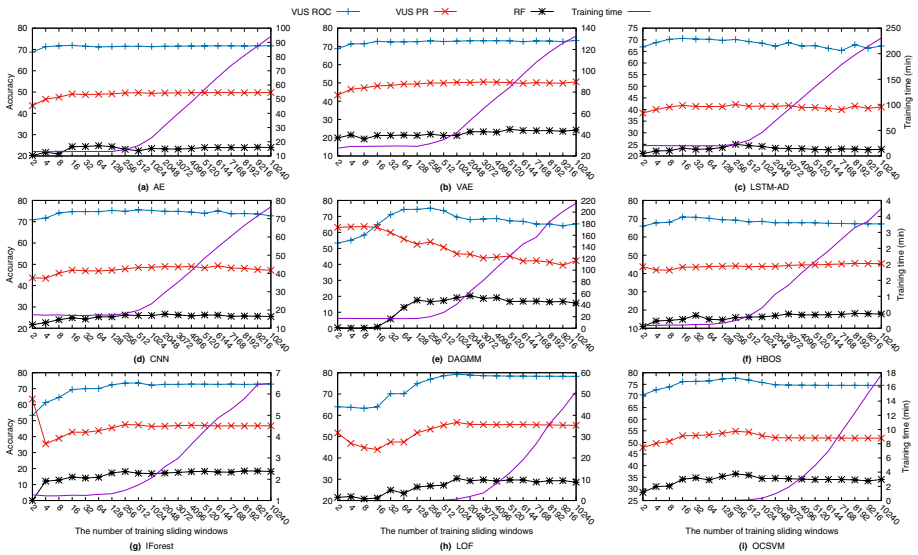

Fig. 1 Accuracy measures (*VUS ROC*, *VUS PR*, and *RF*) and training time (in minutes) of different UTS-AD methods with varying numbers of randomly sampled training sliding windows

findings demonstrate that there is much room to improve the data efficiency of the UTS-AD methods, thus providing strong support for our motivation.

Based on the above experimental observations, we propose a novel and generic framework called FastUTS-AD to improve the data efficiency and reduce the computational overhead of different UTS-AD models with little loss of accuracy. Specifically, instead of feeding all training data to the UTS-AD model at once, FastUTS-AD trains the model in multiple stages, each of which only inputs a small fraction of the training set. Then, inspired by scaling laws [27], FastUTS-AD utilizes a heuristic method, based on the observation that the accuracy of the model remains nearly unchanged when too much data is used, to automatically and adaptively determine the number of sliding windows the UTS-AD model requires. In this way, once the model performance becomes stable, FastUTS-AD will terminate the incremental training procedure and output the model trained only on the sampled data.

Finally, we conduct extensive experiments among eight benchmark datasets and nine popular UTS-AD methods. The results indicate that our FastUTS-AD framework exhibits much higher data efficiency than existing methods. For all the UTS-AD methods that we test, FastUTS-AD reduces the number of training samples and the training time by 91.09–91.49% and 93.49–93.82% on average without significant decreases in different accuracy measures.

The main contributions of this paper are summarized as follows:

- We experimentally explore the relationship between the training sample size, training time, and accuracy of different UTS-AD algorithms on large benchmark datasets, thus finding a chance to improve their data efficiencies.
- We propose a novel FastUTS-AD framework that improves data efficiency and reduces computational overhead for UTS-AD tasks at little expense of accuracy. In the FastUTS-AD framework, we design a multi-step continual training (MCT) strat-

egy to automatically determine the appropriate training data size for a given dataset and a UTS-AD model to achieve the desired performance.
- We perform comprehensive experiments on eight benchmark datasets to confirm the superior performance of FastUTS-AD for the trade-off between data efficiency and model accuracy.

Paper Organization: The rest of this paper is organized as follows. Section Related work reviews the related work. Section Background introduces the basic concepts and formally defines the UTS-AD problem. Section Experimental observations on data efficiency of UTS-AD methods provides our key observations from an experimental study on the relationship between model performance and the number of training samples. Section The FastUTS-AD framework presents the FastUTS-AD framework in detail. Section Experimental evaluation for FastUTS-AD demonstrates the experimental results to evaluate the performance of FastUTS-AD. Section Conclusion concludes this work and indicates possible future directions.

Related work

Univariate time series anomaly detection

Numerous algorithms have been proposed for the problem of univariate time series anomaly detection (UTS-AD). We broadly categorize existing UTS-AD algorithms into three types: unsupervised, semi-supervised, and supervised methods, based on whether and how the training sliding windows are labeled. We refer interested readers to [13, 17, 18, 44, 48, 64] for extensive surveys on UTS-AD methods. Next, we briefly discuss algorithms of each kind separately.

Unsupervised UTS-AD methods

Methods of this type do not require training sliding windows to be labeled and are thus widely applicable in different scenarios. They implicitly assume that anomalous instances of the time series can be distinguished from their normal counterparts since they are scarce and generated from a different distribution. Generally, they consider using different measures to assign an anomaly score to each instance and find those with the highest anomaly scores as anomalies. Histogram-based Outlier Score (HBOS) [22] is a histogram-based anomaly detection algorithm. It models the densities of instance features using histograms with a fixed or dynamic bin width and then computes the anomaly score of each instance by computing how likely the instance is to fall within the histogram bins for each dimension. The Local Outlier Factor (LOF) [11] is a local density-based method that measures the degree to which a data point is isolated by comparing it with its neighbors. LOF finds data points with lower local densities than their neighbors as anomalies. Isolation Forest (IForest) [56] is a tree-based method for anomaly detection. Its basic idea is that anomalous instances are easier to separate from the rest of the samples. As such, it recursively generates random partitions on the samples and organizes them hierarchically in a tree structure, where an instance closer to the root of the tree is assigned a higher anomaly score. The Deep Autoencoding Gaussian Mixture Model (DAGMM) [65] is a deep learning method for anomaly detection based on reconstruction, which assumes that anomalies cannot be effectively reconstructed from

low-dimensional projections. DAGMM utilizes the autoencoder to reconstruct the input data and the Gaussian Mixture Model (GMM) for density estimation.

Semi-supervised UTS-AD methods

Methods of this type assume that only the normal class of time series is labeled and build models to identify normal patterns. Consequently, new instances will be recognized as abnormal if they diverge from the expected patterns. The one-class support vector machine (OC-SVM) [49] is a classic support vector method that identifies the boundary of normal data and regards instances outside the boundary as anomalies. Autoencoder (AE) [47] is a neural network-based method that performs a nonlinear dimensionality reduction to detect subtle anomalies in which the linear principal component analysis fails. AE is to detect anomalies by learning the representation of normal patterns and computing the reconstruction errors as anomaly scores. The variable autoencoder (VAE) [5, 29] is a deep generative Bayesian network, a.k.a. probabilistic encoders and decoders, with the latent variable and the observed variable. VAE computes the reconstruction probability [5] between the input and output as anomaly scores. DeepAnT [42] is a convolutional neural network (CNN) model that uses the concept of forecasting. It uses a CNN to predict the next value of l, where l is the prediction length. Then, the predicted errors between the real values and the predicted values are seen as anomaly scores. LSTM-AD [37] is a forecasting model that trains a Long Short-Term Memory (LSTM) network on non-anomalous data to forecast future values and uses the prediction error as an indicator of anomalies. The main difference between semi-supervised and unsupervised methods is that semi-supervised methods represent normal patterns based on labeled normal instances, whereas unsupervised methods depend only on the data distribution.

Supervised UTS-AD methods

Methods of this type consider that the training sets contain labeled instances of normal and abnormal classes and build predictive models to distinguish differences between the two classes, which are then applied to unseen instances for prediction. Opprentice [33] is an ensemble method in which multiple existing detectors are used to extract anomaly features and a random forest classifier is applied to automatically select the appropriate combinations of detector parameters and thresholds. RobustTAD [21] is a time series anomaly detection framework that combines time series decomposition and CNNs to handle complicated anomaly patterns. TCQSA [34] is a generic time series anomaly detection method consisting of a two-level clustering-based segmentation algorithm and a hybrid attentional LSTM-CNN model. Random Forest (RF) [10] is a tree-based ensemble method that fits several decision trees on different subsamples of the subsequence and uses averaging to improve predictive accuracy and reduce overfitting. It is widely used for UTS-AD, such as [33, 36, 48]. [36] proposed a novel framework that supports anomaly detection in uncertain data streams based on effective period pattern recognition and feature extraction techniques. However, supervised methods are very restricted for UTS-AD because labeled instances are often unavailable [48]. Therefore, most of the recent literature, such as [9, 44, 48, 51], mainly focuses on semi-supervised and unsupervised methods for UTS-AD.

Despite the extensive methods for UTS-AD in the literature, to the best of our knowledge, they have paid little attention to data efficiency. In this work, we propose to improve the data efficiency for two of the three types of UTS-AD methods in a generic framework. We do not consider supervised algorithms because labeled instances are often unavailable in real-life scenarios [48].

Data efficiency of ML models

The *data efficiency* of ML models has been extensively studied in the literature. Existing studies on data efficiency exploit the relationship between accuracy and training data size for many ML models and problems, including CNNs [25], meta-learning [3], graph neural networks (GNNs) [62], and reinforcement learning [61]. Their results mostly indicate that using more training samples can lead to better model performance to some extent. However, excessive training samples bring little gain in accuracy, but, on the contrary, incur unnecessary computational costs. Note that no prior work on data efficiency is specific to UTS-AD models, to the best of our knowledge.

A fundamental approach to improving the data efficiency of ML models is to perform *sampling* on the training set. There are three main types of sampling methods for ML tasks [46]: uniform, importance, and grid. Uniform sampling methods [32], which include each sample in the training set with equal probability, are the simplest and most common method that works on a wide spectrum of ML tasks. Importance sampling methods, often based on the notion of *coresets* [26], extract a small subset of samples based on their importance for the given ML problem. Grid sampling methods [1] are to divide the input space into small groups, extract the representatives from each group, and then weight them by the number of input points in the group. Although these methods have been applied to many ML tasks, as far as we know, we are the first to study the data efficiency of UTS-AD.

Background

This section provides an overview of the fundamental concepts considered in this paper, followed by a formal definition of the problem under investigation.

Univariate time series

Time series data can be divided into univariate time series (UTS) and multivariate time series. In this work, we focus on the UTS data. UTS data refers to a series of observations at a single variable, which are usually collected at regular time intervals, such as 1 min. Formally, a UTS is denoted as $D = D^n = \{d_1, d_2, \ldots, d_n\}$, where n is the length of the UTS and d_i is an observation at timestamp i. Each data point d_i has a label (0 or 1) that indicates whether the data point is an anomaly or not. By default, we use "0" for a normal data point and "1" for an anomaly data point. Therefore, a data point in UTS can be represented as a triple $d_i = (t_i, v_i, l_i)$, where t_i is the timestamp at which d_i is observed, v_i is the observation value, and l_i is the label for d_i. For example,

$$D = \{(12{:}30, 0.5, 0), (12{:}31, 0.1, 0), (12{:}32, 0.9, 1), (12{:}33, 0.2, 0), (12{:}34, 0.1, 0)\} \quad (1)$$

is a UTS with five data points. In this example, the first data point $d_1 = (12{:}30, 0.5, 0)$ is observed at 12:30 with a value of 0.5 and labeled as 0 (i.e., *normal*), and the third point

$d_3 = (12{:}32, 0.9, 1)$ is the only anomaly data point observed. The time interval for this UTS is $t_i - t_{i-1} = 1$.

Sliding window

A sliding window is a consecutive subsequence of length l from a UTS D. We use W_D to denote all sliding windows generated from D with length l and time step t_{step}. Formally, a sliding window w_i is defined as $w_i = \{d_{i-l+1}, d_{i-l+2}, \ldots, d_i\} \in W_{D^n}$ $(l \leq n)$, where l is the length of the sliding window and n is the length of the UTS D. Taking the UTS D in Eq. 1 as an example, the first sliding window contains the first three data points in D when the window size $l = 3$ and the time step $t_{step} = 1$, i.e., $w_3 = \{d_1, d_2, d_3\}$ (w_1, w_2 are omitted because they do not have enough points). The second and third sliding windows are $w_4 = \{d_2, d_3, d_4\}$ and $w_5 = \{d_3, d_4, d_5\}$. Therefore, W_D consists of three sliding windows: w_3, w_4, and w_5. The sliding window is an effective method for processing UTS data and has been widely used in the literature [15, 24, 41, 55, 59, 60]. It captures the correlation between the data point d_i and neighboring data points. In addition, it allows us to divide a UTS into multiple subsequences with fixed length l and time step t_{step} as input for ML algorithms.

Range-based anomaly detection

We need to convert point labels to window labels when using sliding windows for semi-supervised anomaly detection since anomaly windows should be removed. In this work, following the existing literature [15, 41, 55, 59, 60], we consider a sliding window to be abnormal if it contains any anomaly data points. Also taking the UTS D in Eq. 1 as an example, the first window $w_3 = \{d_1, d_2, d_3\}$ is anomalous since d_3 is an anomaly data point. Formally, if $\exists d_j \in w_i$ is an anomaly, then w_i is labeled as an anomalous sliding window.

We consider detecting anomalies based on ranges, that is, anomalies that occur consecutively over a period of time [52]. For example, if the window labels of a UTS are $\{a, b, c, d, e\} = \{1, 0, 1, 1, 0\}$, then the UTS contains two range anomalies instead of three (there are three 1's in the labels, but only two consecutive ranges), where the first range anomaly only contains the first value a, and the second range anomaly consists of the third and fourth values $\{c, d\}$. The normal labels act as a separator for dividing the original continuous labels into multiple sub-intervals. Each sub-interval with continuous anomalies indicates a range anomaly.

Problem formulation

Our goal in this paper is to improve the data efficiency of a specified UTS-AD algorithm. We treat the problem of improving data efficiency as identifying the minimum set of sliding window training $W_D^* \subseteq W_D$, where W_D is the set of all sliding windows generated from D, on which the trained model can achieve a performance close to that trained on W_D. Given a UTS D and an anomaly detection algorithm $\mathcal{A}$, the problem is formalized as follows:

$$W_D^* = \underset{W_D' \subseteq W_D}{\arg\min} |W_D'| \quad \text{s.t.} \quad Acc_{\mathcal{A}(W_D)} - Acc_{\mathcal{A}(W_D')} \leq \theta, \tag{2}$$

where W_D' is a subset of training sliding windows sampled from W_D, $Acc_{\mathcal{A}(W_D)}$ and $Acc_{\mathcal{A}(W_D')}$ are the accuracy measures of $\mathcal{A}$ trained on the original and sampled sets of sliding windows, and $\theta \in (0, 1)$ is the threshold to control the accuracy loss. Note that we

evaluate the accuracy of an algorithm at a UTS level and perform hypothesis testing at a dataset level. Because for a specific UTS D, the number of accuracy values ($Acc_{\mathcal{A}(W_D)}$ and $Acc_{\mathcal{A}(W'_D)}$) is not enough to perform hypothesis testing. For a dataset consisting of m individual UTS $D_1, \ldots, D_m$, we obtain m different (original) models $\mathcal{A}(W_{D_1}), \ldots, \mathcal{A}(W_{D_m})$ by running $\mathcal{A}$ on all windows of each UTS. Meanwhile, by solving Eq. 2, we also obtain m models $\mathcal{A}(W^*_{D_1}), \ldots, \mathcal{A}(W^*_{D_m})$ by running $\mathcal{A}$ on the windows sampled from each UTS. We use Welch's t-test to decide whether the accuracy significantly decreases between $\mathcal{A}(W_{D_1}), \ldots, \mathcal{A}(W_{D_m})$ and $\mathcal{A}(W^*_{D_1}), \ldots, \mathcal{A}(W^*_{D_m})$ since the variances of the accuracy measures are often not available. The null hypothesis is that the accuracy measures $Acc_{\mathcal{A}(W^*_{D_1})}, \ldots, Acc_{\mathcal{A}(W^*_{D_m})}$ on $\mathcal{A}(W^*_{D_1}), \ldots, \mathcal{A}(W^*_{D_m})$ are not less than the accuracy measures $Acc_{\mathcal{A}(W_{D_1})}, \ldots, Acc_{\mathcal{A}(W_{D_m})}$ on $\mathcal{A}(W_{D_1}), \ldots, \mathcal{A}(W_{D_m})$. If the p-value of Welch's t-test is larger than or equal to the significant level (e.g., 0.01), we say that the hypothesis is supported.

Experimental observations on data efficiency of UTS-AD methods

In this section, we will start by examining the UTS-AD approaches that are under experimental evaluation. Subsequently, the accuracy measures, datasets, and fundamental settings of our experimental investigation are presented. Finally, we present the primary observations on the data efficiency of existing UTS-AD approaches.

Review of UTS-AD methods

Local outlier factor (LOF)

LOF [11] is an unsupervised density-based method for anomaly detection. It computes the local density deviation of a given data instance (a sliding window w_i in our context) with respect to its neighbors. The data point will be regarded as an anomaly if its local density is lower than that of its neighbors. On the contrary, the local density of a normal data instance is typically similar to that of its neighbors.

LOF follows the following two steps to detect anomalies: 1) Calculate the Local Reachability Density (LRD) of a data instance x in its k-neighborhoods as Eq. 3.

$$\mathrm{LRD}_k(x) = \frac{|N_k(x)|}{\sum_{o \in N_k(x)} (r_k(x, o))}, \tag{3}$$

where k is the number of neighbors, $N_k(x)$ is the set of k-nearest neighbors of x, and r_k is the reachability distance of the data instance x. The reachability distance is used to reduce statistical fluctuations in the assessment of the anomaly score. 2) Calculate the anomaly score s_{LOF} as the average of the ratio between the LRD of x and those of the k-nearest neighbors of x by Eq. 4.

$$s_{\mathrm{LOF}}(x) = \frac{\sum_{o \in N_k(x)} \mathrm{LRD}_k(o)}{|N_k(x)| \cdot \mathrm{LRD}_k(x)}, \tag{4}$$

where $\mathrm{LRD}_k(\cdot)$ is the local reachability density of a data instance computed by Eq. 3. It is easy to see that the lower $\mathrm{LRD}_k(x)$ is and the higher $\sum_{o \in N_k(x)} \mathrm{LRD}_k(o)$ is, the higher $s_{\mathrm{LOF}}(x)$ is. For LOF and all remaining methods, a higher anomaly score indicates a high probability of being an anomaly.

Histogram-based outlier score (HBOS)

HBOS [22] is an unsupervised histogram-based method for anomaly detection. It creates a histogram for each dimension (feature) and calculates the anomaly score based on the density estimation of each feature on the corresponding histogram bins. Then, the anomaly score is used to determine whether a data instance is an anomaly.

The HBOS anomaly detection procedure is presented as follows: 1) Compute an individual histogram for each of the d dimensions (features), where the height of each single bin represents a density estimation. 2) Normalize each histogram to ensure an equivalent weight of each dimension in the outlier score. 3) Calculate the HBOS score of an instance x using the corresponding height of the bins where the instance is located by Eq. 5.

$$s_{\text{HBOS}}(x) = \sum_{j=0}^{d} \log\left(\frac{1}{Hist_j(x)}\right), \tag{5}$$

where d is the number of features of data x (the window length l in our context). The score $s_{\text{HBOS}}(x)$ is a multiplication of the inverse of the estimated densities, assuming the independence of each feature.

Isolation Forest (IForest)

IForest [35] is an unsupervised tree-based method for anomaly detection. IForest uses the concept of isolation instead of measuring distance or density to detect anomalies and assumes that an anomaly can be isolated in a few steps. A short path indicates that a data instance is easy to isolate because its attribute values are significantly different from other values. IForest builds multiple trees by performing recursive random splits on attribute values. Then, the anomaly score of a given data instance x is defined as the average path length from a particular sample to the root in Eq. 6.

$$s_{\text{IForest}}(x) = 2^{-\frac{E(h(x))}{c(n)}} \tag{6}$$

where $h(x)$ is the length of the path for a data point from its leaf to the tree root, $E(h(x))$ is the average of $h(x)$ from a collection of the build trees, $c(n)$ is the average path length of an unsuccessful search in a binary search tree, and n is the number of the original data instances. According to Eq. 6, the lower the path to the root of x, the higher the anomaly score of x.

One-class support vector machine (OCSVM)

OCSVM [49] is a semi-supervised SVM-based method for anomaly detection. It learns a decision boundary (a.k.a. hyperplane) from the normal data instances. Then, data instances outside the boundary are considered anomalies. The distance of an instance from the boundary is used to compute an anomaly score. The larger the distance (outside the hyperplane), the higher the anomaly score. OCSVM solves the following quadratic problem to learn a decision boundary around normal instances:

$$\begin{array}{ll} \min_{\omega,\rho,\xi_i} & \frac{1}{2}\|\omega\|^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i - \rho \\ \text{subject to} & \omega\Phi(x_i) \geq \rho - \xi_i, \xi_i \geq 0,\ \forall i \in \{1,\ldots,n\} \end{array} \tag{7}$$

where N is the number of training samples, $\Phi(\cdot)$ is a nonlinear feature map that can be computed by evaluating a kernel function $k(\cdot,\cdot)$, ξ is a parameter to avoid overfitting, ω and ρ are the parameters to define the hyperplane, $\nu \in [0,1]$ controls the trade-off between the training classification error and the margin maximization in one class. Once ω and ρ are obtained, the anomaly score is computed by

$$s_{\text{ocsvm}}(x_i) = -\sum_{i=1}^{n} a_i k(x_i, x) - \rho \tag{8}$$

where $a_i, i \in \{1, 2, \ldots, n\}$ is the Lagrange parameter and $k(\cdot)$ is a kernel function. We use the RBF kernel $k_\gamma(x,y) = \exp(-\gamma \|x-y\|^2)$ in our experiments. To ensure that the notion of an anomaly score is consistent, we assign the original decision score a negative sign. The modified $s_{\text{ocsvm}}(x_i)$ represents the signed distance to the separating hyperplane, where a positive value is given for an outlier and a negative value for an inlier.

AutoEncoder (AE)

AE [39] is a semi-supervised reconstruction-based method for anomaly detection. AE tries to learn the normal pattern of the given data to minimize the reconstruction errors for normal instances. Then, new data instances with large reconstruction errors are considered anomalies. AE consists of two components: an encoder network and a decoder network. The encoder $f(\cdot)$ compresses the input data x into a low-dimension representation z, and then the decoder $g(\cdot)$ projects z to the original dimension as the output x', where $x' = g(f(x))$. The objective function of AE is given in Eq. 9.

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \|x_i - x_i'\|^2, \tag{9}$$

where N is the number of training samples and θ is the parameter of AE. In Eq. 9, the mean squared error (MSE) is used to measure the reconstruction error. Other error measures, such as the mean absolute error (MAE), cross entropy, and binary cross entropy, can be used in place of the MSE. We also note that some variants of AE [39] introduce additional regularization terms to Eq. 9 to improve robustness. In this work, we use MSE-based AE without regularization following existing UTS-AD methods [44, 63]. After solving the parameters θ, the anomaly score can be calculated from the reconstruction error in Eq. 10.

$$s_{\text{AE}}(x) = \|x - g(f(x))\|. \tag{10}$$

Variational auto-encoder (VAE)

VAE [5, 29] is also a semi-supervised reconstruction-based method for anomaly detection. Its main component is a deep generative Bayesian network, a.k.a. probabilistic encoders and decoders, with the latent variable z and the observed variable x. The idea of VAE for anomaly detection is similar to that of AE, which first trains a VAE using normal data and then considers a new data instance with a large deviation as an anomaly. VAE consists of a probabilistic encoder network and a decoder network, with the latent variable z and the observed variable x (i.e., data instances). The generative process of

VAE starts with a variable z with a prior distribution $p(z)$. Then, the decoder network g is applied to z and then outputs x' with distribution $p(x'|g(z))$. Variational inference techniques are applied to perform posterior inference of $q(z|x)$. The posterior inference aims to train a separated distribution $q(z|f(x))$ to approximate $q(z|x)$ by the encoder network f. The overall objective of VAE is to maximize the likelihood of the observation x in Eq. 11.

$$P(x) = \int_z p(z)p(x|z)dz, \tag{11}$$

where $p(z)$ is the prior distribution and $p(x|z)$ is the posterior distribution of variable x. Since sampling from z is a stochastic process, the gradients may not be correctly estimated. To overcome this issue, the reparameterization trick is introduced to maximize the evidence lower bound through variational inference for each observation x. Thus, Eq. 11 can be rewritten as Eq. 12.

$$\begin{aligned} \log P(x) &= \int_z q(z|x) \log(\frac{p(z,x)}{q(z|x)})dz + \int_z q(z|x) \log(\frac{q(z|x)}{p(z|x)})dz \\ &= L_b + \mathrm{KL}(q(z|x)\|p(z|x)) \geq L_b, \end{aligned} \tag{12}$$

where $\mathrm{KL}(\cdot)$ is the KL divergence. Since $\mathrm{KL}(\cdot) \geq 0$, we only need to maximize L_b. Thus, VAE uses $q(z|x)$ to approximate $p(z|x)$ by the encoder network f. After p and q are solved, the reconstruction probability [5] is computed as the anomaly score, i.e.,

$$s_{\mathrm{VAE}}(x) = \frac{1}{L}\sum_{l=1}^{L} p\big(x|\mu_{\hat{x}}, \sigma_{\hat{x}}\big), \tag{13}$$

where L is the number of samples from z, $\mu_{\hat{x}}, \sigma_{\hat{x}}$ are the distribution sampled from z.

Deep autoencoding Gaussian mixture model (DAGMM)

DAGMM [65] is an unsupervised method that combines AE with the Gaussian mixture model (GMM) to detect anomalies. DAGMM consists of an AE-based representation network to generate a low-dimensional representation of input data instances and a GMM-based estimation network to compute reconstruction errors and anomaly scores. The objective function of DAGMM is given in Eq. 14.

$$J(\theta_e, \theta_d, \theta_m) = \frac{1}{N}\sum_{i=1}^{N} L\big(x_i, x_i'\big) + \frac{\lambda_1}{N}\sum_{i=1}^{N} E(z_i) + \lambda_2 P(\hat{\Sigma}), \tag{14}$$

where $L(x_i, x_i')$ is the reconstruction error, $E(z_i)$ is the probabilities of observed input samples, N is the number of input instances, $P(\hat{\Sigma})$ is a penalty term to tackle the singularity of GMM, λ_1 and λ_2 are hyper-parameters. Once the unknown parameters are solved, the anomaly score is given by

$$s_{\mathrm{dagmm}}(x) = -\log\Big(\sum_{k=1}^{K} \hat{\phi}_k \frac{\exp(-\frac{1}{2}\big(\varphi(x) - \hat{\mu}_k\big)^T \hat{\Sigma}_k^{-1}\big(\varphi(x) - \hat{\mu}_k\big))}{\sqrt{\big|2\pi\,\hat{\Sigma}_k\big|}}\Big), \tag{15}$$

where $\hat{\phi}_k, \hat{\mu}_k, \hat{\Sigma}_k$ are the mixture probability, mean, and covariance for the k-th component in GMM, and $\varphi(x)$ is the features (combining z_c and z_r) generated by the compression network.

LSTM-AD

LSTM-AD [37] is an LSTM-based semi-supervised model for anomaly detection. LSTM-AD trains an LSTM network on normal data instances to forecast future values and uses the prediction error as an indicator of anomalies. For a new data instance, a higher prediction error implies a more likely anomaly. LSTM-AD detects anomalies in the following steps. First, it uses a stacked LSTM network trained on normal data to predict the next l value, where l is the prediction length. The prediction errors are then fed to a multivariate Gaussian distribution to compute the anomaly score. In this work, we use prediction errors as anomaly scores. The anomaly score is given by

$$s_{\text{LSTM-AD}}(x) = \sqrt{\big(\varphi(x) - g(x)\big)^2}, \tag{16}$$

where $\varphi(x)$ is the last value of the input data instance x (x is a sliding window in our context) and $g(\cdot)$ is the value predicted by LSTM-AD.

DeepAnT

DeepAnT [42] is a semi-supervised CNN-based model for anomaly detection. It uses a convolutional neural network (CNN) to predict the next l value, where l is the prediction length. The objective function of DeepAnT is given by

$$J(\theta) = \frac{1}{N}\sum_{u=1}^{N} |f(x_i) - f(x_i')|, \tag{17}$$

where $f(x_i)$ is the actual value of the sliding window and $f(x_i')$ is the predicted value of the sliding window associated with x_i. Then, the predicted value is fed to an anomaly detector, which uses the Euclidean distance to compute the anomaly score:

$$s_{\text{DeepAnT}}(x) = \sqrt{\big(\varphi(x) - g(x)\big)^2}, \tag{18}$$

where $\varphi(x)$ is the last value of the input data instance x (x is a sliding window in our context) and $g(\cdot)$ is the value predicted by DeepAnT.

Accuracy measures

In the UTS-AD task, the accuracy metrics in most existing studies [8, 21, 43, 44, 48, 57] are computed by the anomaly score. Following the existing literature, we calculate a score for each corresponding sliding window w_i, where a higher score means a high likelihood that the window is an anomaly. Furthermore, we evaluate the accuracy metrics of a UTS method based on the predicted window labels and their corresponding real window labels. Based on the anomaly score, several accuracy measures have been proposed to quantitatively assess the effectiveness of the UTS-AD methods. Subsequently, we review these measures and present some of their limitations.

Precision, recall, and F-score

Let P and N be the number of actual positive and negative windows, and TP, FP, TN, and FN be the number of true positive, false positive, true negative, and false negative classification results. The precision and recall of a classification method are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{19}$$

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{20}$$

However, one limitation of these two measures is that high precision may cause low recall and vice versa. To overcome the shortcomings of precision and recall, the F-score is proposed. The F-Score is calculated as the harmonic mean of precision and recall:

$$\text{F-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{21}$$

Range-precision, range-recall, and range-F-Score (RF)

The original precision, recall, and F-Score are designed primarily for point anomalies. However, time series anomalies are range-based, which means that they occur over a period of time [52]. In other words, the three measures suffer from the inability to represent range-based time series anomalies. To alleviate the shortcomings of traditional measures, [52] expanded the well-known definitions of precision and recall to measure ranges instead of points. They proposed two measures, Range-Precision and Range-Recall, accordingly. Note that although the point labels have been converted to window labels in our study, the converted window labels are also a continuous sequence. The Range-Recall is defined as

$$\begin{aligned}
&\text{Range-Recall}(R, P) = \frac{\sum_{i=1}^{N_r} \text{Recall}_r(R_i, P)}{N_r}, \\
&\text{Recall}_r(R_i, P) = \alpha \cdot \text{ER}(R_i, P) + (1 - \alpha) \cdot \text{OR}(R_i, P), \\
&\text{ER}(R_i, P) = \begin{cases} 1 \text{ if } \sum_{j=1}^{N_p} |R_i \cap P_j| \geq 1 \\ 0 \text{ otherwise} \end{cases}, \\
&\text{OR}(R_i, P) = \text{CF}(R_i, P) \cdot \sum_{j=1}^{N_p} \omega(R_i, R_i \cap P_j, \delta).
\end{aligned} \tag{22}$$

Here, R is a set of real anomaly ranges $R = \{R_1, \ldots, R_{N_r}\}$ and P is a set of predicted anomaly ranges $P = \{P_1, \ldots, P_{N_p}\}$. In our context, R_i and P_i are subsequences of the real window labels and the predicted window labels, respectively. N_r and N_p are the total numbers of the real and predicted anomaly ranges. $\text{ER}(\cdot, \cdot)$ indicates whether a true anomaly range is detected; $\text{OR}(\cdot, \cdot)$ evaluates the overlap between the real and predicted anomaly ranges in three aspects: (1) ω measures how many points in R_i are detected; (2) δ measures the position bias between the real and predicted anomaly ranges; and (3) $\text{CF}(\cdot, \cdot)$ measures the number of fragmented regions corresponding to a given anomaly range R_i. A typical choice for $\text{CF}(R_i, P)$ is $\frac{1}{x}$, where x is the number of distinct overlapped

ranges. Finally, the value of α controls the trade-off between $\mathrm{ER}(\cdot,\cdot)$ and $\mathrm{OR}(\cdot,\cdot)$. Similarly, Range-Precision is defined as

$$\begin{aligned}\text{Range-Precision}(R,P) &= \frac{\sum_{i=1}^{N_p}\text{Precision}_r(R,P_i)}{N_r},\\ \text{Precision}_r(R,P_i) &= \text{CF}(P_i,R)\cdot\sum_{j=1}^{N_r}\omega(P_i,P_i\cap R_j,\delta).\end{aligned} \tag{23}$$

Finally, Range-F-Score (RF) is defined as

$$\text{RF} = \frac{2\cdot\text{Range-Precision}\cdot\text{Range-Recall}}{\text{Range-Precision}+\text{Range-Recall}}. \tag{24}$$

AUC ROC

The above measures depend on a predefined threshold in the anomaly score to decide whether a window is anomalous. Typically, the threshold θ is set to $\mu+3\sigma$ according to [7], where μ is the mean and σ is the standard deviation. Given a threshold θ and an anomaly score s_i of the sliding window w_i, one can decide whether a sliding window w_i is an anomaly or not:

$$w_i \text{ is } \begin{cases}\text{abnormal} & \text{if } s_i \geq \theta\\ \text{normal} & \text{otherwise}\end{cases} \tag{25}$$

However, threshold-based measures are sensitive to the threshold value. To remove the effect of the threshold, the Area Under the Receiver Operating Characteristics Curve (AUC ROC) [20] is introduced. AUC ROC is defined as the area under the curve that represents the relationship between the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis as we vary the anomaly score threshold. The area under the curve is calculated by the trapezoidal rule. Let $\Theta=\{\theta_0,\theta_1,\ldots,\theta_N\},\theta_i<\theta_j,i<j;\theta_i\in[0,1]$ be a set of thresholds. Then, AUC ROC is defined as

$$\begin{aligned}\text{AUC ROC} &= \frac{1}{2}\sum_{k=1}^{N}\triangle_{TPR}^{k}\cdot\triangle_{FPR}^{k}\\ \text{where } &\begin{cases}\triangle_{FPR}^{k} = FPR(\theta_k)-FPR(\theta_{k-1})\\ \triangle_{TPR}^{k} = TPR(\theta_{k-1})+TPR(\theta_k).\end{cases}\end{aligned} \tag{26}$$

AUC PR

Since the AUC ROC may be excessively optimistic when applied to unbalanced samples, another AUC-based metric, the Area Under the Precision-Recall Curve (AUC PR) [19], is introduced. The AUC PR is defined as the area under the curve corresponding to the recall on the x-axis and precision on the y-axis when we vary the

anomaly score threshold. The area under the curve is calculated by the trapezoidal rule. Let $\Theta = \{\theta_0, \theta_1, \ldots, \theta_N\}, \theta_i < \theta_j, i < j; \theta_i \in [0,1]$ be a set of thresholds, AUC PR is defined as

$$\begin{aligned} &\text{AUC PR} = \frac{1}{2}\sum_{k=1}^{N} \Delta^k_{\text{Precision}} \cdot \Delta^k_{\text{Recall}} \\ &\text{where} \begin{cases} \Delta^k_{\text{Recall}} = \text{Recall}(\theta_k) - \text{Recall}(\theta_{k-1}) \\ \Delta^k_{\text{Precision}} = \text{Precision}(\theta_{k-1}) + \text{Precision}(\theta_k). \end{cases} \end{aligned} \tag{27}$$

VUS ROC and VUS PR

As mentioned above, range-based measures are better than point-based measures in the UTS-AD task. However, AUC ROC and AUC PR [43] are point-based metrics. To address this issue, [43] proposed the Volume Under the Surface (VUS) for the ROC and PR curves: AUC ROC and AUC PR. AUC-based measures are robust to lag, noise, anomaly cardinality ratio, and high separability between accurate and inaccurate AD methods [43]. Let $\Theta = \{\theta_0, \theta_1, \ldots, \theta_N\}, \theta_i < \theta_j, i < j; \theta_i \in [0,1]$ be a set of thresholds, $\mathcal{L} = \{\ell_0, \ell_1, \ldots, \ell_l\}, \ell \in [0,1]; \ell_i < \ell_j, i < j$ be the buffer length introduced based on the idea that there should be a transition region between the normal and abnormal subsequences. The ℓ is used to accommodate the false tolerance of the labeling in the ground truth. Then, the VUS ROC is defined as

$$\begin{aligned} &\text{VUS ROC} = \frac{1}{4}\sum_{w=1}^{L}\sum_{k=1}^{N} \Delta^{(k,w)} \cdot \Delta^w \\ &\text{where} \begin{cases} \Delta^{(k,w)} = \Delta^k_{\text{TPR}_{\ell_w}} \cdot \Delta^k_{\text{FPR}_{\ell_w}} + \Delta^k_{\text{TPR}_{\ell_{w-1}}} \cdot \Delta^k_{\text{FPR}_{\ell_{w-1}}} \\ \Delta^k_{\text{FPR}_{\ell_w}} = \text{FPR}_{\ell_w}(\theta_k) - \text{FPR}_{\ell_w}(\theta_{k-1}) \\ \Delta^k_{\text{TPR}_{\ell_w}} = \text{TPR}_{\ell_w}(\theta_{k-1}) + \text{TPR}_{\ell_w}(\theta) \\ \Delta^w = |\ell_w - \ell_{w-1}|. \end{cases} \end{aligned} \tag{28}$$

Here, TPR_ℓ and FPR_ℓ are range-based versions of TPR and FPR. Similarly, VUS PR is defined as

$$\begin{aligned} &\text{VUS PR} = \frac{1}{4}\sum_{w=1}^{L}\sum_{k=1}^{N} \Delta^{(k,w)} \cdot \Delta^w \\ &\text{where} \begin{cases} \Delta^{(k,w)} = \Delta^k_{Pr_{\ell_w}} \cdot \Delta^k_{Re_{\ell_w}} + \Delta^k_{Pr_{\ell_{w-1}}} \cdot \Delta^k_{Re_{\ell_{w-1}}} \\ \Delta^k_{Re_{\ell_w}} = \text{Recall}_{\ell_w}(\theta_k) - \text{Recall}_{\ell_w}(\theta_{k-1}) \\ \Delta^k_{Pr_{\ell_w}} = \text{Precision}_{\ell_w}(\theta_{k-1}) + \text{Precision}_{\ell_w}(\theta) \\ \Delta^w = |\ell_w - \ell_{w-1}|. \end{cases} \end{aligned} \tag{29}$$

Here, Recall_ℓ and Precision_ℓ are the range-based measures corresponding to traditional recall and precision in the AUC PR.

Key observations

To evaluate the data efficiency of different UTS-AD methods, we first observe the relationship between the number of sliding windows sampled n_{sw} for training and

Table 1 Summary of UTS-AD methods in the evaluation

Learning paradigm	UTS-AD method	Description
Semi-supervised	AE [47]	Autoencoder-based algorithm
Semi-supervised	VAE [5, 29]	Variational autoencoder-based algorithm
Semi-supervised	CNN [42]	Convolutional neural network-based algorithm
Semi-supervised	LSTM-AD [37]	Long short-term memory network-based algorithm
Semi-supervised	OC-SVM [49]	One-class support vector machine-based algorithm
Unsupervised	DAGMM [65]	Deep autoencoding Gaussian mixture algorithm
Unsupervised	HBOS [22]	Histogram-based algorithm
Unsupervised	LOF [11]	Local density-based algorithm
Unsupervised	IForest [56]	Isolation forest algorithm

Table 2 Summary of benchmark datasets in our experiments

Dataset	Anomaly rate (%)	Average length	Description
SVDB [23]	11.33	230,400	Electrocardiogram recordings
DAP [6]	10.37	20,309	Acceleration sensors on Parkinson's disease patients
ECG [40]	8.25	229,900	Standard electrocardiogram dataset
OPP [45]	4.03	34,042	Motion sensors for human activity recognition
IOPS [12]	3.57	96,474	Performance indicators of web servers
SMD [50]	3.52	25,365	Server Machine Dateset
YAHOO [30]	0.45	1543	Real and synthetic data from Yahoo! production systems
MGAB [53]	0.20	100,000	Mackey-Glass time series with non-trivial anomalies

For each dataset, we randomly selected 15 UTS for evaluation. For MGAB, we picked 10 UTS since it only has 10 UTS

their performance (accuracy and training time). We evaluate all UTS-AD methods in section Review of UTS-AD methods, as summarized in Table 1. We evaluate these methods on 45 randomly selected time series from three benchmark datasets, namely YAHOO, SMD, and DAP in Table 2, which span two different domains: YAHOO and SMD include KPIs of computer servers, while DAP consists of healthcare data. The selected datasets also vary in length and anomaly rates, with YAHOO having an average length of 1.54k and an anomaly rate of 0.45%, SMD having an average length of 25.37k and an anomaly rate of 3.52%, and DAP having an average length of 20.3k and an anomaly rate of 10.37%. A more detailed description of the datasets is provided in section Experimental setup. The accuracy and training time of each method on all sliding windows are shown in Table 3 and these results by varying the number of sliding windows sampled are shown in Fig. 1. Based on the results, we draw three main observations.

Table 3 Accuracy and training time (in hours) of UTS-AD methods on the original datasets

Method	RF	VUS PR	VUS ROC	Training time (hour)
LSTM-AD	18.55	31.37	64.98	158.94
OCSVM	27.49	40.57	69.65	132.38
LOF	28.18	43.54	73.40	98.68
DAGMM	6.49	42.02	60.05	74.00
VAE	19.59	40.28	69.62	36.49
CNN	20.23	35.61	69.01	28.31
AE	21.10	40.09	69.24	23.88
IForest	17.61	39.87	70.94	3.52
HBOS	21.33	38.92	67.50	1.32

Observation 1: As the amount of data increases, most accuracy metrics initially improve and then tend to stabilize or decrease. In other words, a method generally becomes more accurate when more training windows are fed, but the marginal utility of the extra windows gradually diminishes. Taking the AE model as an example, the VUS ROC shows a growing trend with increasing $n_{sw} \in [2, 16]$, where n_{sw} is the number of sliding windows used to train the model. However, the VUS ROC does not increase when $n_{sw} \in [16, 10240]$. A counterintuitive fact is that only 16 sliding windows can train an AE model with good accuracy. This can be explained by the UTS in Fig. 2: two sliding windows ($n_{sw} = 2$) can be used to train a good semi-supervised model because they have represented typical regular patterns of the UTS, which the AE model needs to learn the normal pattern of the UTS.

Observation 2: The accuracy of the model obtained with a small number of sliding windows is essentially equal to the accuracy obtained with all sliding windows. Taking AE as an example, the VUS ROC value has reached 71.90 when $n_{sw} = 16$, which is higher than the VUS ROC value of 69.62, as shown in Table 3. In other words, one can train a model with a small number of sliding windows instead of all. This finding can be verified across all nine methods with different n_{sw}, as shown in Fig. 1.

Observation 3: In most cases, the time to train a model on a small number of sliding windows is much shorter than the time to train it on full sliding windows. Also, taking AE as an example, the training time on a few sliding windows is much less than the full training time of 23.88 h (as shown in Table 3). Combining the above observations, we can reduce the number of training data instances and time and improve the data efficiency if we develop a strategy to find the smallest n_{sw} (denoted by n^*_{sw}) for each algorithm and UTS. Here, we use W^*_D to denote the smallest training windows by the strategy and $\text{len}(W^*_D) = n^*_{sw}$, where $\text{len}(W^*_D)$ is the number of sliding windows in W^*_D.

Remark: The above three observations clearly imply great opportunities to improve the data efficiency of UTS-AD methods. However, there are four challenges to address for such improvements. First, *the accuracy increase is not always continuing*, that is, the accuracy can decrease with increasing n_{sw}. Second, *different accuracy measures can show different trends*, that is, there exist various accuracy measures (e.g., VUS ROC, VUS PR, and RF in our evaluation) and they can show different trends when the number of sliding windows for training increases. Third, *anomaly detection methods and datasets vary*

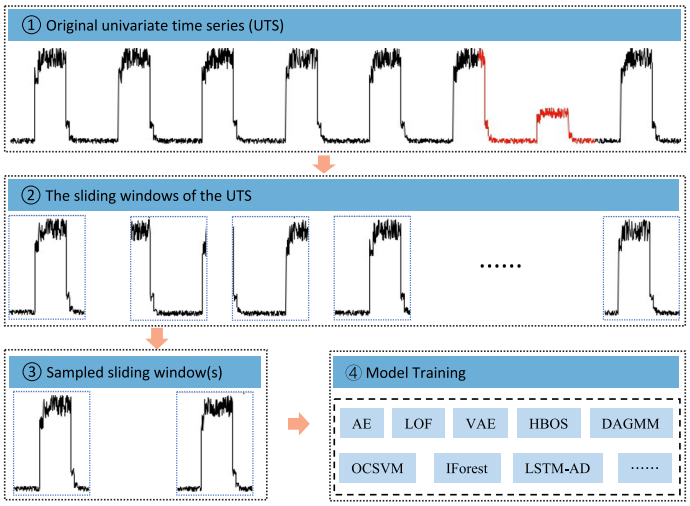

Fig. 2 Illustration of the training process for semi-supervised and unsupervised UTS-AD methods on sampled UTS data (red: abnormal, black: normal). In this case, two sliding windows ($n_{sw} = 2$) may be used to train a good semi-supervised model because they have represented typical regular patterns of the UTS

with each other, that is, different combinations of algorithms and datasets have different appropriate n_{sw}^*. Fourth, *most methods have an extremely long training time*, as shown in Table 3. In the following, we will elaborate on how to design a generic framework to improve data efficiency and address the above challenges.

The FastUTS-AD framework

Overview of FastUTS-AD

To solve the problem of finding the smallest training sliding windows W_D^*, we propose the FastUTS-AD framework. The basic idea of FastUTS-AD involves integrating sampling techniques with scaling laws. This integration can be succinctly described as a *multi-step continual training* (MCT) technique that trains the model $\mathcal{A}$ iteratively in many steps, as opposed to a single training process with full training sliding windows W_D. In each step, we gradually increase the number of sliding windows n_{sw} sampled from the original sliding windows for training. We observe an increase in model accuracy Acc_{inc} (such as VUS ROC, VUS PR, and RF). If Acc_{inc} is too small (i.e., $Acc_{inc} < \alpha$ for some prespecified small α), we will stop the training process and return the best algorithm $\mathcal{A}^*$ and the smallest training windows W_D^* found so far. Otherwise, we increase the number of training windows n_{sw} and then retrain the model $\mathcal{A}$. In the MCT process, the value of α reflects the sensitivity of FastUTS-AD: A smaller α means that FastUTS-AD will train the model with more training windows, although the increase in accuracy has been small. Figure 3 illustrates the architecture of the FastUTS-AD framework. In subsequent subsections, we will describe each component of FastUTS-AD in more detail.

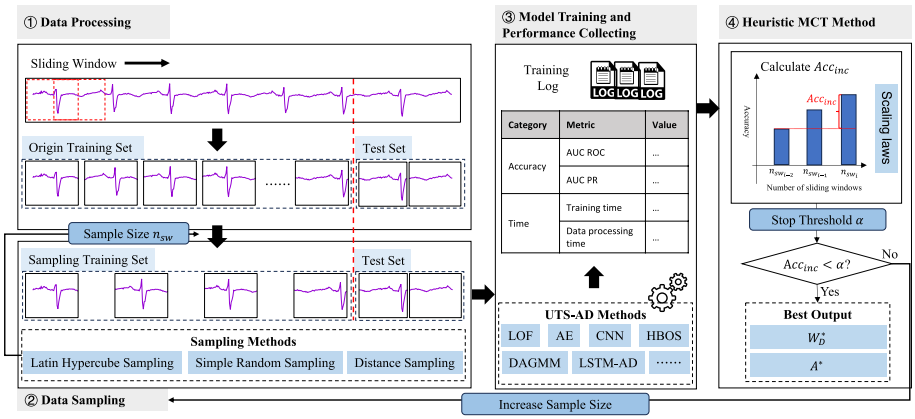

Fig. 3 The architecture of FastUTS-AD

Input of FastUTS-AD

The input of FastUTS-AD consists of three parts: (1) a UTS, (2) an anomaly detection algorithm, and (3) the configuration settings of FastUTS-AD. We consider that the UTS contains labels for the accuracy calculation to help decide whether to stop training progress. The algorithm can be any UTS-AD algorithm that works on sliding windows. The configuration consists of the sampling method used, the sampling gap s_{gap}, and the stop threshold α. Here, the sampling gap s_{gap} indicates how many training windows will increase in the next iteration. In practice, we typically set s_{gap} to a constant, e.g., 256. The stop threshold α controls when FastUTS-AD terminates the training process and is usually set to 0.1%.

Data processing

The first step of the FastUTS-AD framework is *data processing*, which consists of data imputation, data standardization, conversion to sliding windows, and data splitting. First, data imputation fills in the missing value by the mean of the original UTS. Second, the UTS is standardized by mean and standard deviation. Formally, $v_i^* = \frac{v_i - \mu}{\sigma}$, where v_i is the original value, σ is the standard deviation, and v_i^* is the standardized value. Third, the standardized UTS is converted to sliding windows with a window size of 64 and a time step of 1. We note that the window size and time step are flexible in FastUTS-AD and can be changed to other values. Finally, we use a five-fold cross-validation to split the dataset into training, test, and validation sets in order to fairly evaluate each algorithm's accuracy.

Data sampling

The second step of the FastUTS-AD framework is *data sampling*, which draws a subset of sliding windows from the original training sliding windows. We consider the following three sampling methods for FastUTS-AD, namely *simple random sampling*, *Latin hypercube sampling*, and *distance sampling*.

Simple Random Sampling: Random sampling is a basic method that randomly and uniformly samples sliding windows with a given size n_{sw} from W_D without replacement. It does not assume data distribution and reduces sampling bias [38].

Latin Hypercube Sampling (LHS): Latin hypercube sampling divides the UTS into n_{sw} equally spaced intervals and randomly draws a sample sliding window from each interval to achieve a uniform distribution over time.

Distance Sampling: Distance sampling refers to sample data based on their distances, which is specific to time series data over sliding windows. Its basic idea is similar to that of stratified sampling: it calculates the distances of instances (i.e., sliding windows) and then divides them into multiple groups of equal distance intervals accordingly. Finally, we randomly draw samples of equal size from each group. The distance calculation takes into account three aspects of each sliding window w_i: the 95% maximum value, the 5% minimum value, and the Euclidean distance from the zero vector. Formally,

$$Dist(w_i) = max_{95\%}(w_i)^2 + min_{5\%}(w_i)^2 + \sum_{i=1}^{L} w_i^2, \tag{30}$$

where L is the length of the sliding window w_i.

Model training and performance collecting

The third step of the FastUTS-AD framework is to perform the training procedure of the UTS-AD algorithm with the sampled windows, instead of the original sliding windows. Then, the full test set is used to evaluate the accuracy of the algorithm. Notice that the accuracy is averaged in five folds. Regarding the evaluation of algorithm accuracy, we adopt all 13 accuracy measures as used by [44]. Furthermore, we also collect measures of time efficiency, including training time and data processing time. The training time only contains the time to perform the training procedure, where the test time is excluded. Data processing time consists mainly of sampling time and distance computation time. Data sampling time refers to the time to sample the subset of the original training sliding windows, and distance computation time refers to the time to calculate the distances of all sliding windows using Eq. 30.

Heuristic MCT method

The fourth step of the FastUTS-AD framework is to run the heuristic multi-step continual training (MCT) strategy, which considers the accuracy measures in the most recent three iterations, i.e., those from iteration $i-2$ to i in the i-th iteration, to determine whether to increase the number of training sliding windows in the next iteration or terminate the training procedure. MCT proceeds in the following subroutines. First, it calculates the increase in accuracy Acc_{inc} in the i-th iteration by comparing the accuracy measures Acc_{i-1} and Acc_{i-2} in the previous two iterations with Acc_i. Formally,

$$Acc_{inc} = \max(\{Acc_i, Acc_{i-1}\}) - Acc_{i-2}. \tag{31}$$

In Eq. 31, we consider the accuracy measures in the previous two iterations for two reasons. On the one hand, MCT provides a relatively loose condition for incremental training. On the other hand, MCT increases the robustness of FastUTS-AD. According to our observations in the experiments, the accuracy of a UTS-AD algorithm might not always improve when more training samples are used. For example, the increment Acc_{inc} is small between the i-th iteration and the $(i-1)$-th iteration, but it becomes

large enough between the i-th iteration and the $(i-2)$-th iteration. Then, MCT decides whether the training process is continued. If $Acc_{inc} < \alpha$, we break the training loops and return the best $\mathcal{A}^*$ and W_D^* found so far. If $Acc_{inc} \geq \alpha$, we increase the number of training windows from n_{sw_i} to $n_{sw_{i+1}} = n_{sw_i} + s_{gap}$ and continue to train the algorithm with $n_{sw_{i+1}}$ samples in the next iteration.

Experimental evaluation for FastUTS-AD

In this section, we evaluate the performance of FastUTS-AD on the eight benchmark datasets in Table 2, as well as compare it with the nine widely recognized UTS-AD methods in Table 1. The experimental setup is briefly described at the beginning of the experiment.

Experimental setup

Datasets: To systematically evaluate FastUTS-AD, we used the eight datasets in Table 2. These datasets span three different domains: computer systems, healthcare, and social. In addition, they also vary in length and distribution (anomaly rates). For each dataset, we randomly selected 15 UTS as representatives. In total, we selected 115 UTS to evacuate FastUTS-AD (note that MGAB contains only 10 UTS). Every point in a UTS is associated with a label of either "normal" (0) or "abnormal" (1). Table 2 summarizes the relevant characteristics of the datasets, including their size, length, and distribution. Here, the length indicates the average size of every UTS in the corresponding dataset. We run each anomaly detection method on each UTS separately. We briefly describe the eight datasets in the following.

- SVDB (MIT-BIH Supraventricular Arrhythmia Database) [23] includes 78 half-hour ECG recordings chosen to supplement the examples of supraventricular arrhythmias in the MIT-BIH Arrhythmia Database.
- DAP (Daphnet) [6] contains the annotated readings of three acceleration sensors at the hip and leg of Parkinson's disease patients that experience freezing of gait (FoG) during walking tasks.
- ECG [40] is a standard electrocardiogram dataset, and the anomalies represent ventricular premature contractions. The long series (MBA_ECG14046) of length $\sim 10^7$ was divided into 47 series by identifying the periodicity of the signal according to TSB-UAD.
- OPP (OPPORTUNITY) [45] is a dataset for human activity recognition from wearable, object, and ambient sensors, which is devised to benchmark human activity recognition algorithms (classification, automatic data segmentation, sensor fusion, feature extraction, etc.).
- IOPS [12] is a public dataset consisting of 27 key performance indicators (KPIs) for artificial intelligence-based IT operations (AIOps), which was collected from five large internet companies, including Sougo, eBay, Baidu, Tencent and Alibaba.
- SMD (Server Machine Dataset) [50] is a 5-week-long dataset collected from a large Internet company, which contains 3 groups of entities from 28 different machines.

- YAHOO [30] is a dataset published by Yahoo! Labs consisting of real and synthetic time series based on the real production traffic to some of the Yahoo! production systems.
- MGAB (Mackey-Glass anomaly benchmark) [53] is composed of Mackey-Glass time series with non-trivial anomalies. Mackey-Glass time series exhibit chaotic behavior that is difficult for humans to distinguish.

Algorithms: To evaluate the robustness and effectiveness of FastUTS-AD, we used the nine popular anomaly detection algorithms as described in Table 1.

Hyperparameters: For each algorithm, most of the hyperparameters are kept default configurations as [44, 48]. The following hyperparameters are changed for better performance. The length of the sliding window is set to 64, and the time step is set to 1 when we convert the UTS to sliding windows. For deep learning methods, the batch size is set to 128 and the epoch is set to 50. Other parameters we have not mentioned here can be found in [44, 48].

Performance Metrics: In this work, we consider three accuracy metrics: VUS ROC, VUS PR, and range-base F1 score (RF). According to [43], threshold-independent metrics are more suitable for time series. Therefore, we selected two threshold-independent metrics: VUS ROC and VUS PR. VUS ROC measures the overall performance of a classification algorithm at different classification thresholds, and VUS PR measures the precision and recall at different thresholds. To fairly assess the accuracy of the model, we also selected a threshold RF metric, and the threshold θ is set to $\mu + 3\sigma$ according to [7], where μ is the mean and σ is the standard deviation. Furthermore, we collected metrics for time efficiency, including algorithm training time and data processing time, as in section Model training and performance collecting.

Environment and Implementation: We conducted the experiments on two servers with the following hardware configuration. For non-deep learning methods, we train them on a server with an Intel (R) Xeon (R) CPU E5–2630 v3 (2 sockets, 72 cores) at 2.40GHz and 256GB of memory. We use 64 cores in parallel when training the algorithms. For deep learning algorithms, we train them on a server with eight Nvidia A6000 GPUs. Each GPU runs six jobs in parallel when training those algorithms. We implemented our algorithms using Python 3.8, scikit-learn 1.2.0, TensorFlow 2.9.1, and PyTorch 1.10.0.

Overall results

We demonstrate the performance of FastUTS-AD in three aspects: (1) accuracy; (2) data efficiency; and (3) time efficiency. We compare the accuracy measures (VUS ROC, VUS PR, and RF) between the original model (Acc_{ori}) and FastUTS-AD (Acc_{fast}). We also used statistical hypothesis testing to verify whether the accuracy measures of each method in each dataset have decreased significantly. In detail, we use Welch's t-test to test Acc_{fast} and Acc_{ori} for each method on each dataset since we do not know the variance of the accuracy measures. The null hypothesis is that Acc_{fast} is not less than Acc_{ori}. Both Acc_{ori} and Acc_{fast} are the averaged values among 5-folds. Data efficiency shows the smallest training sliding windows n^*_{sw} (%) found by FastUTS-AD. Time efficiency shows how much training time FastUTS-AD can reduce. The default parameters of FastUTS-AD are set as follows. The sampling method is *simple random sampling*. The sampling

Table 4 Accuracy and data efficiency of FastUTS-AD for different measures and AD methods, where Acc_{ori} is the model accuracy trained by the full dataset, Acc_{fast} is the model accuracy trained by FastUTS-AD, n^*_{sw} is the number of sliding windows sampled by FastUTS-AD (%)

Measure	Method	*p*-value	n^*_{sw} (%)	Acc_{ori}	Acc_{fast}
	AE	0.59	8.45 ± 5.34	20.36 ± 10.10	21.07 ± 11.40
	CNN	0.99	7.51 ± 5.38	19.57 ± 6.39	23.38 ± 6.76
	DAGMM	1.00	7.79 ± 5.29	6.35 ± 4.83	20.90 ± 8.32
	HBOS	0.16	9.76 ± 4.75	20.47 ± 12.52	18.09 ± 10.62
RF	IForest	0.88	8.77 ± 5.38	16.94 ± 9.90	19.26 ± 9.94
	LOF	0.71	9.45 ± 6.71	29.60 ± 13.94	28.79 ± 11.45
	LSTM-AD	1.00	8.06 ± 4.25	17.95 ± 7.11	22.69 ± 6.88
	OCSVM	0.97	8.65 ± 4.46	26.54 ± 12.96	32.35 ± 14.22
	VAE	0.81	8.11 ± 5.06	18.86 ± 10.23	21.23 ± 11.46
	AE	0.55	9.22 ± 5.33	38.71 ± 15.71	39.14 ± 15.67
	CNN	0.91	10.25 ± 5.96	34.47 ± 13.04	38.50 ± 13.47
	DAGMM	0.62	6.45 ± 4.19	40.77 ± 9.89	41.86 ± 13.90
	HBOS	0.62	7.90 ± 5.16	37.51 ± 14.37	38.57 ± 14.59
VUS PR	IForest	0.63	10.37 ± 4.70	38.47 ± 15.05	39.56 ± 15.78
	LOF	1.00	12.76 ± 8.53	43.71 ± 17.62	50.64 ± 15.94
	LSTM-AD	0.95	7.54 ± 4.38	30.34 ± 12.52	34.88 ± 14.05
	OCSVM	0.87	7.62 ± 4.36	39.16 ± 15.40	42.95 ± 15.97
	VAE	0.59	8.08 ± 4.93	38.91 ± 14.32	39.70 ± 15.01
	AE	0.63	8.57 ± 4.99	68.79 ± 10.63	69.54 ± 10.16
	CNN	1.00	10.18 ± 5.89	68.68 ± 7.86	73.61 ± 6.61
	DAGMM	1.00	7.11 ± 4.98	60.07 ± 7.07	75.32 ± 7.99
	HBOS	0.88	7.99 ± 5.15	67.05 ± 11.01	69.56 ± 10.16
VUS ROC	IForest	0.67	9.97 ± 5.33	70.43 ± 9.84	71.33 ± 9.98
	LOF	1.00	12.02 ± 7.47	73.75 ± 11.72	78.91 ± 9.64
	LSTM-AD	1.00	7.93 ± 5.17	64.78 ± 7.86	70.68 ± 7.21
	OCSVM	0.96	6.89 ± 4.45	69.16 ± 10.03	72.82 ± 10.11
	VAE	0.67	7.63 ± 4.83	69.21 ± 9.58	70.17 ± 9.52

gap s_{gap} is set to 256 (as described in Section Overall results). Consequently, the training sample sizes n_{sw} are $\{256, 512, 768, \ldots, N\}$ over iterations, where N is the number of sliding windows in a given UTS. The stop condition α is fixed to 0.1%. The significance level is set to 0.01. The hyperparameters of the models are discussed in section Experimental setup. The results for the accuracy and data efficiency of different algorithms are shown in Table 4, and their time efficiency is shown in Table 5.

Accuracy: In general, the accuracy of the model returned by FastUTS-AD does not decrease significantly compared to the original model. As shown in Table 4, the minimum p-values for RF, VUS ROC, and VUS PR are 0.16, 0.55, and 0.63, respectively. According to the null hypothesis (Acc_{fast} is not less than Acc_{ori}) and $\forall$p-value > 0.01, we do not have enough evidence to reject the null hypothesis. Therefore, we accept the null hypothesis: Acc_{fast} is not less than Acc_{ori} at the significant level of 0.01. The main reason why the accuracy of the model does not decrease is that a small number of sliding windows is enough to train a good model.

Furthermore, the average accuracy found by FastUTS-AD is greater than the original accuracy of 6.55–17.58% for different accuracy metrics. The FastUTS-AD model

Table 5 Time efficiency of FastUTS-AD for different measures and AD methods, where T_{ori} is the training time on original sliding windows, T_{fast} is the training time of FastUTS-AD, T_{dp} is the data processing time, and T_{redu} is the time reduced by FastUTS-AD, all in hours

Measure	Method	T_{ori}	T_{fast}	T_{dp}	T_{redu} (%)
	AE	26.69 ± 0.00	2.51 ± 0.00	0.01 ± 0.00	90.57
	CNN	31.85 ± 0.00	2.73 ± 0.00	0.01 ± 0.00	91.39
	DAGMM	81.89 ± 0.00	5.65 ± 0.00	0.01 ± 0.00	93.08
	HBOS	1.46 ± 0.00	0.09 ± 0.00	0.02 ± 0.00	91.86
RF	IForest	3.81 ± 0.00	0.35 ± 0.00	0.02 ± 0.00	90.16
	LOF	106.58 ± 0.01	0.13 ± 0.00	0.03 ± 0.00	99.84
	LSTM-AD	177.43 ± 0.02	6.20 ± 0.00	0.01 ± 0.00	96.50
	OCSVM	139.48 ± 0.01	0.04 ± 0.00	0.02 ± 0.00	99.95
	VAE	40.71 ± 0.00	3.66 ± 0.00	0.01 ± 0.00	91.00
	AE	26.69 ± 0.00	2.67 ± 0.00	0.01 ± 0.00	89.96
	CNN	31.85 ± 0.00	2.96 ± 0.00	0.01 ± 0.00	90.68
	DAGMM	81.89 ± 0.00	4.15 ± 0.00	0.01 ± 0.00	94.92
	HBOS	1.46 ± 0.00	0.12 ± 0.00	0.03 ± 0.00	90.27
VUS PR	IForest	3.81 ± 0.00	0.37 ± 0.00	0.03 ± 0.00	89.64
	LOF	106.58 ± 0.01	0.40 ± 0.00	0.05 ± 0.00	99.59
	LSTM-AD	177.43 ± 0.02	7.11 ± 0.00	0.01 ± 0.00	95.98
	OCSVM	139.48 ± 0.01	0.05 ± 0.00	0.02 ± 0.00	99.95
	VAE	40.71 ± 0.00	3.90 ± 0.00	0.01 ± 0.00	90.39
	AE	26.69 ± 0.00	2.82 ± 0.00	0.01 ± 0.00	89.39
	CNN	31.85 ± 0.00	2.62 ± 0.00	0.01 ± 0.00	91.73
	DAGMM	81.89 ± 0.00	4.68 ± 0.00	0.01 ± 0.00	94.28
	HBOS	1.46 ± 0.00	0.09 ± 0.00	0.02 ± 0.00	92.20
VUS ROC	IForest	3.81 ± 0.00	0.37 ± 0.00	0.03 ± 0.00	89.57
	LOF	106.58 ± 0.01	0.51 ± 0.00	0.05 ± 0.00	99.47
	LSTM-AD	177.43 ± 0.02	6.85 ± 0.00	0.01 ± 0.00	96.13
	OCSVM	139.48 ± 0.01	0.05 ± 0.00	0.02 ± 0.00	99.95
	VAE	40.71 ± 0.00	3.80 ± 0.00	0.01 ± 0.00	90.64

shows an increase in accuracy compared to the original accuracy for the three accuracy metrics. Specifically, for the RF, Acc_{fast} is 23.08 on average, which is 17.58% more than the original Acc_{ori} of 19.63. Similarly, for the VUS PR, Acc_{fast} is 40.64, which is 6.92% higher than Acc_{ori} of 38.01. Lastly, for the VUS ROC, Acc_{fast} is 72.44, which is 6.55% higher than Acc_{ori} of 67.99. It should be noted that Acc_{fast} is higher than Acc_{ori} for all accuracy metrics. The main reason is that more training windows will cause the model to overfit the training set and thus perform poorly on the test set. To explain the relationship between the number of training windows and accuracy, we explore the relationship between the training loss and the test loss on the number of training windows since a lower test loss usually means better accuracy. We randomly sample a UTS from the IOPS dataset and plot the relationship between training/test loss and the percentage of training windows, as shown in Fig. 4. The test loss at 30% of the original windows is the lowest, instead of 100% of the windows. Therefore, we believe that 30% of the training sliding windows can achieve better accuracy than the original sliding windows. In addition, 80% to 100% of the original windows will have a lower training loss but a higher test loss, a.k.a. overfitting.

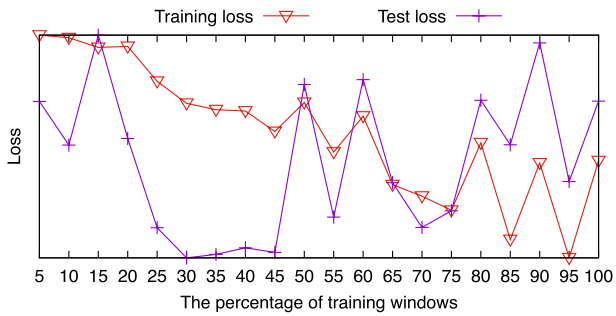

Fig. 4 The relationship between the training/test loss and the percentage of sliding windows for training

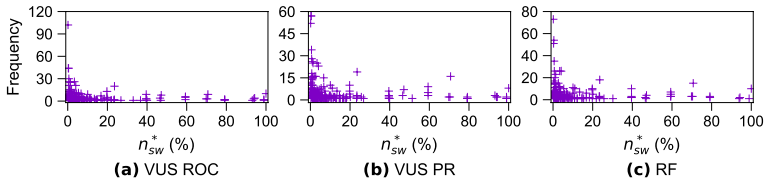

Fig. 5 Occurrences of different n^*_{sw} (%) for three accuracy measures

Data Efficiency: According to the previous section, FastUTS-AD can achieve good accuracy on average. Here, we show that FastUTS-AD only needs a small number of sliding windows (8.51–8.91% for different accuracy metrics) to train a model with the promised accuracy. In detail, with respect to the accuracy metric RF, FastUTS-AD only needs 8.51% on average of the original training sliding windows to train a model, that of 8.91% for VUS PR, and that of 8.70% for VUS ROC. In summary, training a model without significantly losing model accuracy does not need to use all the sliding windows. Instead, a small number of training sliding windows are enough (8.51–8.91%). To clear the distribution of n^*_{sw}, we counted the number of times on different n^*_{sw} on the selected 9 algorithms and 115 UTS. We show the results in Fig. 5. According to Fig. 5, 85% of the models only need less than 30% of the original windows. A small number of models need more than 30% of the original windows. These models are trained on the YAHOO dataset. Since the average length of YAHOO is small, this ratio becomes higher. See more information in section Effect of data size.

Time Efficiency: According to the above two sections, FastUTS-AD has been shown to achieve better model accuracy and reduce the model training sliding windows. Because the sliding windows in model training are reduced, the training time is naturally reduced. As shown in Table 5, we reduced model training from 67.77 h to less than 2.5 h. In summary, FastUTS-AD can reduce the model training time by about 93.49–93.82% without significantly losing algorithm accuracy within the significant level of 0.01. Here, we have gained some new insights about UTS-AD tasks: (1) More training data do not necessarily lead to better model performance when the training data is huge; (2) Scaling laws may work in classical algorithms.

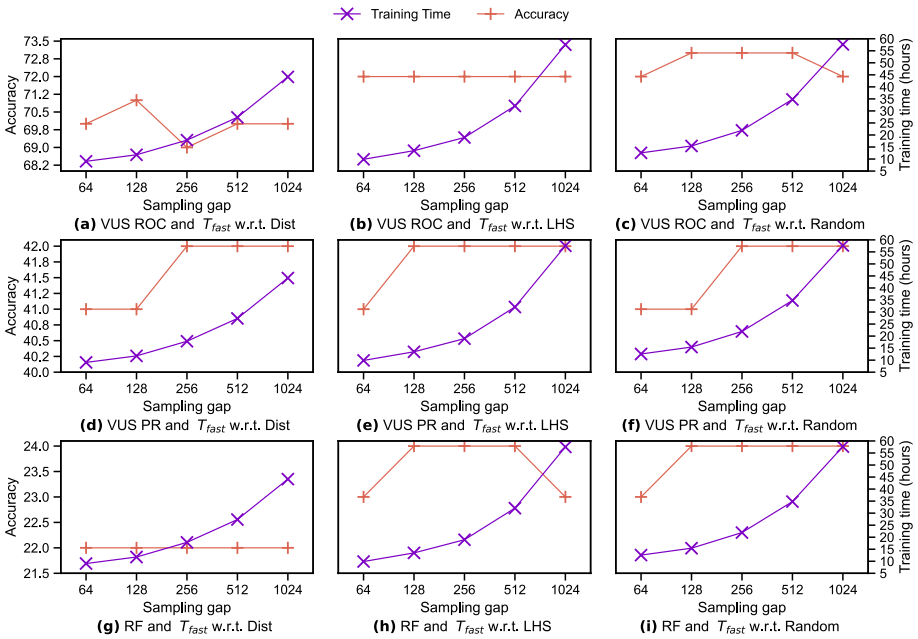

Fig. 6 Effect of sampling methods and gaps on model accuracy and training time (in hours)

Effects of sampling methods and gaps

In this section, we show the effect of sampling methods (Random, LHS, and Dist) and sampling gaps (64, 128, 256, 512, and 1024) on model accuracy (VUS ROC, VUS PR, and RF) and training time (hour). These performance metrics are the averaged value over five folds. The results are shown in Fig. 6. According to Fig. 6, the Dist is shown to have poor accuracy on both three accuracy metrics (e.g., VUS ROC). Instead, the random method can achieve the best accuracy on all three measures with a sample gap of 256. It achieves the best accuracy on all three accuracy metrics, and the training time is minimal. The Dist sampling method achieved the worst accuracy because it changed the distribution (anomaly rate) of the anomaly sliding windows. To verify the distribution of the sampling window, we draw 8.5% sliding windows (achieved the best result in section Experimental observations on data efficiency of UTS-AD methods) from the 115 W_D generated from the selected 115 UTS. We then calculated the anomaly rate of the sampled sliding windows. We find that the anomaly rate of the Dist method is 21.84%, which is significantly different from the original anomaly rate of 10.44%. That is, the distribution of the sliding windows has changed by the Dist method. Therefore, Dist performs worse than LHS and random. Instead, the sampling distribution of LHS and random is the same as the original distribution (LHS is 10.44%, random is 10.46%). This could explain why the effect of LHS is often similar to that of random in Fig. 6.

Effect of stop threshold

This subsection evaluates the influence of different stop thresholds α in FastUTS-AD. As shown in Table 6, it can be seen that all accuracy measures decrease with increasing α. Similarly, the computation time T_{fast} of FastUTS decreases. This means that the

Table 6 Average accuracy measures and training time (hours) of FastUTS-AD at different α values

α	VUS PR		VUS ROC		RF	
	Acc_{fast}	T_{fast}	Acc_{fast}	T_{fast}	Acc_{fast}	T_{fast}
0.001	40.64 ± 14.93	2.41	72.44 ± 9.04	2.42	23.09 ± 10.12	2.37
0.01	40.38 ± 14.79	1.99	72.27 ± 9.07	1.97	22.81 ± 10.05	1.95
0.1	39.58 ± 14.55	1.37	71.70 ± 9.10	1.39	22.34 ± 9.87	1.40
0.5	39.36 ± 14.53	1.29	71.46 ± 9.25	1.29	22.21 ± 9.84	1.29

Table 7 Mean, median, and standard deviation of W_D^* (%) on each dataset

Dataset	Mean	Median	Std.
DAP	5.98	3.36	5.56
ECG	0.46	0.42	0.36
IOPS	3.52	0.90	6.37
MGAB	1.26	0.96	1.12
OPP	3.13	2.51	2.59
SMD	3.86	3.35	2.61
SVDB	0.45	0.42	0.37
YAHOO	50.99	47.15	26.95

smaller the value of α is, the better FastUTS-AD performs. However, the model training time used by FastUTS-AD will be large. The reason is that with a larger α, FastUTS-AD becomes stricter, that is, requiring a larger Acc_{inc}. And it is easy to stop training after the first three steps. That is, it can stop before finding the optimal sliding windows for training W_D^*. Hence, it exhibits worse accuracy. The value of α depends on whether the user is concerned with time or accuracy. If the user cares about time, one can choose a larger α such as 0.1. If the user cares about accuracy, one can choose a smaller α such as 0.001.

Effect of data size

In this section, we explore how the size of the original training data affects the minimal training sliding windows W_D^* FastUTS-AD found. We computed the mean, median, and standard deviation of W_D^* on all algorithms and UTS corresponding to the specified dataset. The results of W_D^* for the different datasets are shown in Table 7. As shown in Table 7, YAHOO has the highest mean, median, and standard deviation, as the original data size is small (1.54k data points). Therefore, W_D^* will be large. For large datasets ($\geq 230k$) such as SVDB, FastUTS-AD only needs a small proportion (0.45%) of the original sliding windows to train a model without nearly losing the accuracy of the model. Therefore, W_D^* will be small.

Conclusion

In this paper, we introduce FastUTS-AD, a new framework that improves the data efficiency of UTS-AD methods. In detail, FastUTS-AD reduces the number of training sliding windows and the training time of existing UTS-AD methods without significantly reducing their accuracy. It features a highly adaptable and expandable architecture with different sampling methods and a heuristic MCT method to decide the smallest

number of training windows a UTS-AD method requires. We evaluate FastUTS-AD on nine UTS-AD algorithms on eight benchmark datasets. The experiments show that FastUTS-AD reduces the training data and the training time by about 91.09–91.49% and 93.49–93.82% without significantly losing model accuracy at a significant level of 0.01. In general, FastUTS-AD provides significant insights on improving the data efficiency of UTS-AD methods, which can facilitate the deployment of UTS-AD models in real-world industrial scenarios.

There are still several problems that we have not addressed in this work. First, we have not yet considered anomaly detection on multivariate time series. Second, we focus on subsequence-based anomaly detection but ignore point-based anomaly detection. Finally, FastUTS-AD can only work well on large datasets. We will continue our efforts in future work to overcome these limitations.

Abbreviations

TSAD	Time series anomaly detection
UTS-AD	Univariate time series anomaly detection
FastUTS-AD	The framework we propose in this research
UTS	Univariate time series
MTS	Multivariate time series
D	A univariate time series
l	The size (length) of a sliding window on a UTS
t_{step}	The time step for generating sliding windows
W	The sliding windows generated from a UTS D
w_i	A sliding window in $W, w_i \in W$
W_D^*	The smallest training sliding windows found by FastUTS-AD
$\mathcal{A}$	An algorithm for UTS-AD
n_{sw}	The number of sampled sliding windows
n_{sw}^*	The minimal number of sampled sliding windows
Acc_{inc}	The increased accuracy
MCT	Multi-step continual training strategy
Acc_{ori}	The model accuracy trained by the original sliding windows
Acc_{fast}	The model accuracy trained by FastUTS-AD
T_{ori}	The training time on the original sliding windows
T_{fast}	The training time by the FastUTS-AD
T_{dp}	The time for data processing
T_{redu}	The time reduction by the FastUTS-AD
α	The stop threshold of the MCT
s_{gap}	The sampling gap of the MCT

Author contributions

Conceptualization, HL and WS; investigation, WS; methodology, WS; administration, HL; software, WS; supervision, HL, QL, MC, XZ, and YW; validation, HL, YW, and QL; visualization, WS and QL; analysis, WS and QL; writing—original draft, WS; writing-review and editing, HL and YW. All authors have read and agreed to the published version of the manuscript.

Funding

The National Natural Science Foundation of China (No. 61562010), National Key Research and Development Program of China (No. 2023YFC3341205), and the Research Projects of the Science and Technology Plan of Guizhou Province (Nos. [2021] 449, [2021] 261, [2023] 010, [2023] 276, and [2023] 338) funded this work.

Availability of data and materials

In this work, we used eight publicly available datasets and nine popular algorithms to validate our proposed framework. The datasets are available at https://www.thedatum.org/datasets/TSB-UAD-Public.zip. The algorithms used in this study are modified from public algorithms, which are available at https://github.com/TheDatumOrg/TSB-UAD and https://github.com/timeeval/timeeval-algorithms.

Code availability

The source code will be available upon publication.

Declarations

Ethics approval and consent to participate

This article does not contain any studies with human participants or animals performed by any authors.

Consent for publication
I hereby give my consent to publish the research article. I confirm that all authors have agreed to the publication. I understand that the article will be made publicly available.

Competing interests
The authors declare that they have no conflict of interest.

Received: 23 November 2023 Accepted: 31 May 2024
Published online: 11 June 2024

References

1. Agarwal PK, Har-Peled S, Varadarajan KR. Geometric approximation via coresets. Comb Comput Geom. 2005;52(1):1–30.
2. Akyildiz IF, Su W, Sankarasubramaniam Y, et al. A survey on sensor networks. IEEE Commun Mag. 2002;40(8):102–14. https://doi.org/10.1109/MCOM.2002.1024422.
3. Al-Shedivat M, Li L, Xing EP, et al (2021) On data efficiency of meta-learning. In: Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, pp 1369–1377, http://proceedings.mlr.press/v130/al-shedivat21a.html
4. Amihud Y. Illiquidity and stock returns: cross-section and time-series effects. J Financial Markets. 2002;5(1):31–56. https://doi.org/10.1016/S1386-4181(01)00024-6.
5. An J, Cho S. Variational autoencoder based anomaly detection using reconstruction probability. Special Lecture IE. 2015;2(1):1–18.
6. Bachlin M, Plotnik M, Roggen D, et al. Wearable assistant for parkinson's disease patients with the freezing of gait symptom. IEEE Trans Inf Technol Biomed. 2009;14(2):436–46. https://doi.org/10.1109/TITB.2009.2036165.
7. Barnett V, Lewis T. Outliers in statistical data. New York: Wiley; 1994.
8. Boniol P, Linardi M, Roncallo F, et al. Unsupervised and scalable subsequence anomaly detection in large data series. VLDB J. 2021;30(6):909–31. https://doi.org/10.1007/s00778-021-00655-8.
9. Boniol P, Paparrizos J, Kang Y, et al. Theseus: navigating the labyrinth of time-series anomaly detection. Proc VLDB Endow 2022;15(12):3702–05.
10. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32. https://doi.org/10.1023/A:1010933404324.
11. Breunig MM, Kriegel HP, Ng RT, et al. LOF: Identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp 93–104, 2000; https://doi.org/10.1145/342009.335388
12. Challenge A (2018) Kpi anomaly detection competition. https://competition.aiops-challenge.com/home/competition/1484452272200032281. Accessed 7 Nov 2023
13. Chandola V, Banerjee A, Kumar V. Anomaly detection: a survey. ACM Comput Surv. 2009;41(3):1–58. https://doi.org/10.1145/1541880.1541882
14. Chatterjee A, Ahmed BS. IoT anomaly detection methods and applications: a survey. Internet of Things. 2022;19:100568. https://doi.org/10.1016/j.iot.2022.100568.
15. Chen W, Xu H, Li Z, et al. Unsupervised anomaly detection for intricate kpis via adversarial training of VAE. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications; 2019, p. 1891–1899
16. Cheng H, Tan PN, Potter C, et al. Detection and characterization of anomalies in multivariate time series. In: Proceedings of the 2009 SIAM International Conference on Data Mining (SDM), pp 413–424; 2009. https://doi.org/10.1137/1.9781611972795.36
17. Cook AA, Misirli G, Fan Z. Anomaly detection for iot time-series data: A survey. IEEE Internet Things J. 2020;7(7):6481–94. https://doi.org/10.1109/JIOT.2019.2958185.
18. Darban ZZ, Webb GI, Pan S, et al (2022) Deep learning for time series anomaly detection: a survey. CoRR abs/2211.05244. https://doi.org/10.48550/arXiv.2211.05244
19. Davis J, Goadrich M. The relationship between precision-recall and ROC curves. In: Proceedings of the Twenty-Third International Conference on Machine Learning, 2006, pp 233–240, https://doi.org/10.1145/1143844.1143874
20. Fawcett T. An introduction to ROC analysis. Pattern Recogn Lett. 2006;27(8):861–74. https://doi.org/10.1016/J.PATREC.2005.10.010.
21. Gao J, Song X, Wen Q, et al. Robusttad: robust time series anomaly detection via decomposition and convolutional neural networks. CoRR abs/2002.09545. 2020; https://doi.org/10.48550/arXiv.2002.09545.
22. Goldstein M, Dengel A. Histogram-based outlier score (HBOS): a fast unsupervised anomaly detection algorithm. In: Poster and Demo Track of the 35th German Conference on Artificial Intelligence (KI-2012), 2012, pp 59–63.
23. Greenwald SD, Patil RS, Mark RG. Improved detection and classification of arrhythmias in noise-corrupted electrocardiograms using contextual information. In: [1990] Proceedings Computers in Cardiology; 1990, p. 461–464. https://doi.org/10.1109/CIC.1990.144257.
24. de Haan P, Löwe S. Contrastive predictive coding for anomaly detection. CoRR abs/2107.07820. 2021. https://arxiv.org/abs/2107.07820.
25. Hlynsson HD, Escalante-B. AN, Wiskott L. Measuring the data efficiency of deep learning methods. In: Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods (ICPRAM) - Volume 1, 2019, pp 691–698, https://doi.org/10.5220/0007456306910698
26. Jubran I, Maalouf A, Feldman D. Overview of accurate coresets. WIREs Data Mining and Knowl Discov. 2021;11(6):e1429. https://doi.org/10.1002/widm.1429.
27. Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models. 2020; CoRR abs/2001.08361. https://doi.org/10.48550/arXiv.2001.08361

28. Kaushik S, Choudhury A, Sheron PK, et al. AI in healthcare: Time-series forecasting using statistical, neural, and ensemble architectures. Front Big Data. 2020;3:4. https://doi.org/10.3389/fdata.2020.00004.
29. Kingma DP, Welling M. Auto-encoding variational bayes. In: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings, arXiv:1312.6114
30. Laptev N, Amizadeh S, Billawala Y (2015) S5-a labeled anomaly detection dataset, version 1.0 (16m). 2015. https://webscope.sandbox.yahoo.com/catalog.php
31. Lehnertz K, Elger CE. Can epileptic seizures be predicted? evidence from nonlinear time series analysis of brain electrical activity. Phys Rev Lett. 1998;80(22):5019. https://doi.org/10.1103/PhysRevLett.80.5019.
32. Li Y, Long PM, Srinivasan A. Improved bounds on the sample complexity of learning. J Comput Syst Sci. 2001;62(3):516–27. https://doi.org/10.1006/JCSS.2000.1741.
33. Liu D, Zhao Y, Xu H, et al. Opprentice: towards practical and automatic anomaly detection through machine learning. In: Proceedings of the 2015 ACM Internet Measurement Conference, 2015, pp 211–224, https://doi.org/10.1145/2815675.2815679
34. Liu F, Zhou X, Cao J, et al. Anomaly detection in quasi-periodic time series based on automatic data segmentation and attentional LSTM-CNN. IEEE Trans Knowl Data Eng. 2022;34(6):2626–40. https://doi.org/10.1109/TKDE.2020.3014806.
35. Liu FT, Ting KM, Zhou Z. Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp 413–422, https://doi.org/10.1109/ICDM.2008.17
36. Ma J, Sun L, Wang H, et al. Supervised anomaly detection in uncertain pseudo-periodic data streams. ACM Trans Internet Technol (TOIT). 2016;16(1):1–20.
37. Malhotra P, Vig L, Shroff G, et al. Long short term memory networks for anomaly detection in time series. In: 23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22–24, 2015, pp 89–94, https://www.esann.org/sites/default/files/proceedings/legacy/es2015-56.pdf
38. Mayeza CA, Munyeka W. The socialization of first entering students: an exploratory study at south african university. Int J Educ Excell. 2021;7(1):99–115.
39. Michelucci U (2022) An introduction to autoencoders. arXiv preprint arXiv:2201.03898
40. Moody GB, Mark RG. The impact of the MIT-BIH arrhythmia database. IEEE Eng Med Biol Mag. 2001;20(3):45–50. https://doi.org/10.1109/51.932724.
41. Moon J, Yu J, Sohn K. An ensemble approach to anomaly detection using high- and low-variance principal components. Comput Electr Eng. 2022;99: 107773.
42. Munir M, Siddiqui SA, Dengel A, et al. DeepAnT: a deep learning approach for unsupervised anomaly detection in time series. IEEE Access. 2019;7:1991–2005.
43. Paparrizos J, Boniol P, Palpanas T, et al. Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. Proc VLDB Endow. 2022;15(11):2774–87.
44. Paparrizos J, Kang Y, Boniol P, et al. TSB-UAD: An end-to-end benchmark suite for univariate time-series anomaly detection. Proc VLDB Endow. 2022;15(8):1697–711. https://doi.org/10.14778/3529337.3529354.
45. Roggen D, Calatroni A, Rossi M, et al. Collecting complex activity datasets in highly rich networked sensor environments. In: Seventh International Conference on Networked Sensing Systems (INSS), 2010, pp 233–240, https://doi.org/10.1109/INSS.2010.5573462
46. Ros F, Guillaume S. Sampling techniques for supervised or unsupervised tasks. Springer. 2020. https://doi.org/10.1007/978-3-030-29349-9.
47. Sakurada M, Yairi T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, 2014, pp. 4–11.
48. Schmidl S, Wenig P, Papenbrock T. Anomaly detection in time series: a comprehensive evaluation. Proc VLDB Endow. 2022;15(9):1779–97. https://doi.org/10.14778/3538598.3538602.
49. Schölkopf B, Williamson RC, Smola A, et al. Support vector method for novelty detection. Adv Neural Inf Process Syst. 1999;12:582–588.
50. Su Y, Zhao Y, Niu C, et al. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2828–2837.
51. Sylligardos E, Boniol P, Paparrizos J, et al. Choose wisely: an extensive evaluation of model selection for anomaly detection in time series. Proc VLDB Endow. 2023;16(11):3418–32.
52. Tatbul N, Lee TJ, Zdonik S, et al. Precision and recall for time series. Adv Neural Inf Process Syst. 2018:31.
53. Thill M, Konen W, B¨ack T (2020) Time series encodings with temporal convolu- tional networks. In: Bioinspired optimization methods and their applications - 9th international conference, BIOMA 2020, Brussels, Belgium, November 19–20, 2020, Proceedings, pp 161–173.
54. Van NT, Thinh TN, et al (2017) An anomaly-based network intrusion detection system using deep learning. In: 2017 international conference on system science and engineering (ICSSE), IEEE, pp 210–214
55. Wagner D, Michels T, Schulz FCF, et al. Timesead: benchmarking deep multivariate time-series anomaly detection. Trans Mach Learn Res. 2023. https://openreview.net/forum?id=iMmsCl0JsS.
56. Wang R, Nie F, Wang Z, et al. Multiple features and isolation forest-based fast anomaly detector for hyperspectral imagery. IEEE Tran Geosci Remote Sens. 2020;58(9):6664–76. https://doi.org/10.1109/TGRS.2020.2978491.
57. Wang R, Liu C, Mou X, et al. Deep contrastive one-class time series anomaly detection. In: Proceedings of the 2023 SIAM International Conference on Data Mining (SDM), pp 694–702, 2023;https://doi.org/10.1137/1.9781611977653.ch78.
58. Woike M, Abdul-Aziz A, Clem M. Structural health monitoring on turbine engines using microwave blade tip clearance sensors. In: Smart Sensor Phenomena, Technology, Networks, and Systems Integration 2014. SPIE, p 90620L, 2014; https://doi.org/10.1117/12.2044967.
59. Xu H, Chen W, Zhao N, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, 2018, pp. 187–196

60. Yao Y, Ma J, Ye Y. Regularizing autoencoders with wavelet transform for sequence anomaly detection. Pattern Recognit. 2023;134: 109084.
61. Yuan Y, Yu ZL, Gu Z, et al. A novel multi-step q-learning method to improve data efficiency for deep reinforcement learning. Knowl Based Syst. 2019;175:107–17. https://doi.org/10.1016/j.knosys.2019.03.018.
62. Zhang W, Yang Z, Wang Y, et al. Grain: Improving data efficiency of graph neural networks via diversified influence maximization. Proc VLDB Endow. 2021;14(11):2473–82. https://doi.org/10.14778/3476249.3476295.
63. Zhao Y, Nasrullah Z, Li Z. Pyod: a python toolbox for scalable outlier detection. J Mach Learn Res. 2019;20(96):1–7.
64. Zhong Z, Fan Q, Zhang J, et al (2023) A survey of time series anomaly detection methods in the AIOps domain. CoRR abs/2308.00393. https://doi.org/10.48550/arXiv.2308.00393
65. Zong B, Song Q, Min MR, et al. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In: Conference Track Proceedings of the 6th International Conference on Learning Representations (ICLR), 2018. https://openreview.net/forum?id=BJJLHbb0-

Publisher's Note