Fast, Parameter-free Time Series Anomaly Detection

Kristiyan Blagov [1], Carlos Enrique Muñiz-Cuza [2], and Matthias Boehm [2]

Abstract: Time series anomaly detection is a common problem across many domains. Despite the existence of numerous algorithms leveraging deep learning, classical machine learning, and data mining techniques, no dominating approach has emerged. A common challenge is extensive parameter tuning and the high computational costs associated with many existing methods. To address this problem, we propose a parameter-free anomaly detection algorithm, STAN (**s**ummary s**ta**tistics e**n**semble). STAN applies a set of summary statistics over sliding windows and compares the results to the normal behavior learned during training. STAN's flexibility allows for integrating different statistical aggregates, which effectively handle diverse types of anomalies. Our evaluation shows that STAN achieves a detection accuracy of 60.4%, close to the widely used MERLIN algorithm (63.6%) while reducing execution time by more than an order of magnitude compared to all baselines.

Keywords: Anomaly Detection, Summary Statistics Ensemble, MERLIN, Machine Learning, Deep Learning

1 Introduction

Time series are extensively used in various fields, including medicine, science, and finance [WK23]. Due to their critical importance, time series anomaly detection has attracted substantial research interest [Ye16, SWP22, RDN23, Na20, TCJ22, DC22, Gu16, Ba19]. Despite advances in deep learning and the development of numerous methods utilizing machine learning and data mining techniques, no single approach consistently outperforms others [SWP22]. Recent studies [SWP22, RDN23, Li24] suggest that different methods are better suited for detecting specific types of anomalies. Notably, a 2021 anomaly detection competition [Ke21a] revealed that many top-performing teams employed relatively simple techniques based on time series discords [KLF05].

Accuracy and Parametrization: In contrast to other fields, where deep learning models often outperform simpler techniques, anomaly detection shows different results [SWP22]. These results are supported by a recent benchmark comparing three classical machine learning algorithms and three deep learning algorithms for anomaly detection [RDN23]. The findings indicate that MDI [Ba19] and MERLIN [Na20] are among the most effective algorithms. This insight is valuable as MDI and MERLIN require only the specification of upper and lower bounds for subsequence lengths, significantly reducing the number of parameters and hyperparameters that need to be tuned compared to deep learning models.

[1] TU Berlin, kristiyan.blagov@campus.tu-berlin.de, https://orcid.org/0009-0008-1966-0793
[2] TU Berlin & BIFOLD, muniz.cuza@tu-berlin.de, https://orcid.org/0000-0003-4286-3448; matthias.boehm@tu-berlin.de, https://orcid.org/0000-0003-1344-3663

However, even these two single parameters can significantly affect both the accuracy and execution time of the algorithms.

The Execution Time Challenge: When including the training process, most supervised and semi-supervised algorithms are among the slowest, with execution times of approximately 255 ms per data point [SWP22]. In contrast, for unsupervised algorithms such as MDI and MERLIN, which have a time complexity of $O(n^2)$ over the time series size n, the results are mixed [RDN23]. Especially as the subsequence length increases, both algorithms experience significant slowdowns, with execution times extending to several minutes [RDN23].

Contributions: The high parameter sensitivity and slow execution times of existing anomaly detection algorithms have motivated the development of a more efficient, parameter-free approach. In this paper, we introduce STAN (**s**ummary s**ta**tistics e**n**semble), a fast, parameter-free algorithm for time series anomaly detection. STAN operates in two main phases: a training phase that computes summary statistics over sliding windows to capture normal (non-anomalous) patterns and an evaluation phase that compares these statistics against new patterns to identify anomalies. The window size is essential for the method's accuracy and is automatically determined using the autocorrelation function (ACF). This eliminates the need for manual tuning. Our key contributions are:

- *Background:* We provide a description of various anomaly types in Section 2.2 and an updated review of anomaly detection methods for time series in Section 2.3.
- *Methodology:* We introduce a novel framework for anomaly detection based on a summary statistics ensemble, offering an efficient and parameter-free approach by automatically computing the window size. Section 2.1 introduces the necessary notation and definitions, while Section 3.1 outlines STAN's overall design.
- *Implementation:* We holistically describe STAN, providing a proof-of-concept implementation using eight different summary statistics, including low-order (e.g., mean, standard deviation), high-order (e.g., skewness, kurtosis), and custom functions (e.g., turning points, point anomaly). Details of the implementation and the ACF-based window size computation are provided in Section 3.
- *Experiments:* We conduct multiple experiments on the UCR dataset [Ke21a] comparing our STAN and relevant baseline methods. The results show that STAN achieves a detection accuracy of 60.4%, close to MERLIN's results (63.6%), while improving execution time by more than an order of magnitude compared to all baselines. We provide all the details in Section 4.

2 Background and Related Work

This section provides the essential background for time series anomaly detection and the concepts underlying STAN. We review related work, discuss various anomaly types, and introduce the autocorrelation function, which is crucial for our method.

2.1 Definitions and Notation

A univariate *time series* $T = (t_1, \ldots, t_n)$ consists of n real-numbered data points, each indexed chronologically and separated by equal time intervals [Ye16]. A key aspect of analyzing time series data involves extracting contiguous subsets of length κ from the time series, also known as subsequences. Given a time series T, a *time series subsequence* starting at index i is defined as $T_i^\kappa = (t_i, t_{i+1} \ldots, t_{i+\kappa-1})$. Subsequence extraction allows for local anomaly detection, as anomalies often manifest within specific data segments rather than across the entire sequence.

Summary Statistics: We define a summary statistic as a function that computes a single value s over a subsequence T_i^κ such that $f(T_i^\kappa) = s_i$. This statistical function can represent any commonly used low- or higher-order statistic of a time series, such as the mean, variance, or skewness. Alternatively, the statistics can be a custom function that takes a subsequence as input and returns a single summary statistic. For example, $f(T_i^\kappa) = \max(t_i, t_{i+1} \ldots, t_{i+\kappa-1})$, which calculates the maximum value in the subsequence. Subsequence comparison can be achieved by comparing their respective summary statistics. Furthermore, we can compute an ensemble of summary statistics over sliding windows across a time series T, providing an alternative representation of T.

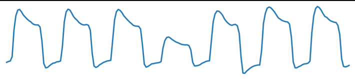
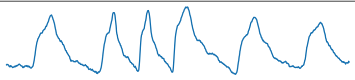
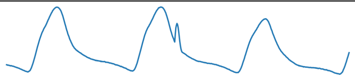
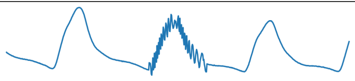
Summary Statistics Ensemble: We refer to a summary statistics ensemble as a collection of multiple summary statistics computed over the same subsequence or time window of T. By aggregating different statistics, the ensemble offers a richer, multi-dimensional view of the time series. We define the ensemble over a non-overlapping sliding window operator as:

$$\mathbf{E}(T) = \begin{bmatrix} f_1(T_1^\kappa) & f_2(T_1^\kappa) & \ldots & f_m(T_1^\kappa) \\ f_1(T_\kappa^\kappa) & f_2(T_\kappa^\kappa) & \ldots & f_m(T_\kappa^\kappa) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(T_{n'}^\kappa) & f_2(T_{n'}^\kappa) & \ldots & f_m(T_{n'}^\kappa) \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & \ldots & s_{1m} \\ s_{\kappa 1} & s_{\kappa 2} & \ldots & s_{\kappa m} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n'1} & s_{n'2} & \ldots & s_{n'm} \end{bmatrix} \tag{1}$$

where T_i^κ represent the i-th subsequence of length κ, $f_j(T_i^\kappa) = s_{ij}$, represents the j-th summary statistic computed over the subsequence T_i^κ, m is the total number of summary statistics, $n' = \lfloor \frac{n}{\kappa} \rfloor$ is the total number of non-overlapping windows and $\mathbf{E}(T) \in \mathcal{R}^{n' \times m}$. We can apply simple matrix operations on $\mathbf{E}(T)$ to extract information about the statistics or the subsequences. For example, $\mathbf{E}[:, j]$ returns all the values of the j-th statistic across all subsequences, while $\mathbf{E}[i, :]$ returns all the statistics for the subsequence T_i^κ. Additionally, to simplify the notation, we use the name of the summary statistic as an index for the ensemble matrix. For instance, $\mathbf{E}[mean]$ refers to retrieving all the values of the *mean* statistic across all subsequences.

Autocorrelation Function: The autocorrelation function (ACF) of a time series is a fundamental statistical tool that quantifies the correlation between observations in a time series at different time lags [NB00, Pa17, A 08, Wa05]. The ACF measures the Pearson

Tab. 1: Frequent Anomaly Types in Time Series [RDN23]

Anomaly Type	**Example**
Amplitude Change: The signal amplitude was modified.	
Frequency Change: The periodic length of a subsequence was shortened or elongated.	
Local Peak: An unusual peak, lower than the global maximum, was added to a subsequence.	
Noise: Noise was included in a subsequence.	

correlation between the time series and a lagged version of itself, computed for each lag from 1 to a user-defined maximum K. Given a time series T, the ACF at lag τ is calculated as follows:

$$\rho_T(\tau) = \frac{1}{(n-\tau)\sigma^2} \sum_{t=1}^{n-\tau} (t_t - \mu)(t_{t+\tau} - \mu) \tag{2}$$

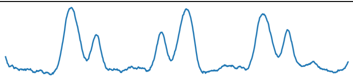
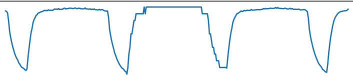
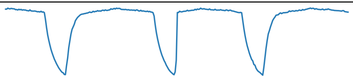
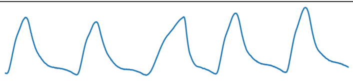
where μ and σ are the mean and standard deviation of T. Note that the ACF for all lags can be efficiently computed in $O(n \log n)$ using the Discrete Fourier Transform [CLW69]. The ACF is essential for identifying patterns such as periodicity and seasonality in the time series. The efficient computation and ability to capture temporal dynamics make the ACF a valuable tool for determining an appropriate value for κ (the length of subsequences) since a subsequence length that aligns with the periodic structure of the data will likely capture the temporal dependencies [ESL22]. The peaks in the ACF can guide the selection of κ, as they indicate the lags where the time series exhibits significant autocorrelation. Section 3.1 describes STAN's use of the ACF.

2.2 Time Series Anomaly Types

Identifying types of time series anomalies is complex, as they vary across different applications. However, recent analyses of the UCR dataset [Ke21a, WK23], a widely-used benchmark dataset for anomaly detection, identified 17 distinct anomaly types [RDN23]. We extend this classification by introducing a new category to classify subsequences inverted along the horizontal axis.

Basic Anomaly Characteristic: Table 1 illustrates that the anomalies disrupt the periodicity and statistical properties of the time series within the cycles where they occur. This

Tab. 2: Complex Anomaly Types in Time Series [RDN23].

Anomaly Type	**Example**
Reversed Vertically: A subsequence was inverted along the vertical axis.	
Sampling Rate: The sampling frequency of a subsequence was changed.	
Steep Increase: A smooth increase was sharpened.	
Time Warping: The peak of a subsequence was shifted, while the periodic length was kept the same.	

observation forms the foundation of our method STAN: detecting anomalies involves comparing statistical properties across different periodic cycles in the time series. In other words, significant deviations in summary statistics between regular, anomaly-free cycles and those affected by anomalies serve as clear indicators of their presence. Based on this idea, STAN proposes a unifying framework that leverages the ACF to extract the periodic cycle and summary statistics to detect deviations from normal behavior.

Complex Anomaly Characteristic: While many anomalies in Table 1 can be detected using simple, low-order statistics, some anomaly types exhibit more subtle deviations from regular cyclical patterns. These anomalies require more sophisticated or custom statistics for effective detection. Table 2 shows an example of anomalies like *Reversed Vertically* or *Time Warping*, which do not disrupt basic summary statistics but instead alter temporal or structural relationships. STAN's framework supports the integration of advanced statistics, enabling the detection of these more complex anomalies.

Custom Summary Statistics: Anomalies such as those in Table 2 require more advanced summary statistics for detection. In response, we designed STAN to be flexible, allowing any summary statistic to be incorporated without altering the core algorithm. In Section 3.1, we detail how we achieve this modularity, and in Section 4.1, we present the full set of statistics used in our experiments. Below, we introduce additional anomaly types, offering a complete overview of the challenges in developing custom statistics for detection:

- **Local Drop**: An unusual drop was added to a subsequence.
- **Flat**: An unusual flat subsequence was included.
- **Missing Drop**: An expected drop in a subsequence was removed.
- **Missing Peak**: An expected peak in a subsequence was removed.
- **Outlier**: The anomaly is a global maximum or minimum.

- **Reversed Horizontally**: A subsequence was inverted along the horizontal axis.
- **Signal Shift**: A subsequence was moved up or down.
- **Smoothed Increase**: A sharp increase was smoothed.
- **Time Shift**: The interval between two peaks was extended.
- **Unusual Pattern**: A part of the data was replaced with an unusual sequence.

2.3 Related Work

Time series anomaly detection has been a long-standing area of research, with approaches ranging from traditional statistical methods to more recent machine learning and deep learning techniques [BPP23, Li24]. Over time, researchers have developed various methods, making anomaly detection applicable to a broad range of domains. However, accurate anomaly detection at a low computation overhead and minimum parameter tuning has remained the core challenge of this research area.

Matrix Profile: The introduction of the Matrix Profile marked a significant advancement in anomaly detection and time series analysis in general. STAMP (Scalable Time Series Anytime Matrix Profile) [Ye16] efficiently computes the nearest neighbor subsequences within a time series, identifying the subsequence that is most dissimilar to all others, also referred to as discord. Matrix Profile has been praised for its simplicity, effectiveness, and extensive body of work expanding its applicability [Zh16, Me21, De22, De23]. However, its primary limitation lies in its sensitivity to the subsequence length. Setting this parameter requires extensive trial-and-error or domain knowledge to select an optimal value, which in turn can affect its usability in automated applications.

MERLIN: MERLIN [Na20] addresses Matrix Profiles' limitation by detecting discords across a range of subsequence lengths. By automating the search over subsequence lengths between specified lower and upper bounds, MERLIN removes the need to predefine a specific subsequence length, making it more adaptable. MERLIN's minimal requirement for hyperparameter tuning (only needing bounds for subsequence length) makes it especially appealing for a broad range of applications with minimal user intervention. However, the need to set the subsequence bounds and their long execution time affects their application in scenarios where scalability is critical.

Deep Learning for Anomaly Detection: In contrast, deep learning approaches such as Autoencoders (AE) [RHW86], Transformer-based models like TranAD [TCJ22], and Graph Augmented Normalizing Flows (GANF) [DC22] offer powerful solutions for handling complex, high-dimensional time series data. However, these models often come with significant drawbacks, including high computational costs and the need for extensive hyperparameter tuning. The reliance on large amounts of labeled training data, which may not be available in all anomaly detection applications, is another significant challenge. While deep learning models can capture complex non-linear patterns, the results from benchmark studies (e.g., Schmidl et al. [SWP22]) suggest that they do not consistently outperform simpler, less computationally expensive models, mainly when applied to univariate or low-dimensional multivariate time series.

Benchmark Studies and Contests: Many benchmark studies have provided critical insights into the relative performance of different anomaly detection methods. Schmidl et al. [SWP22] conducted a comprehensive benchmark study comparing 71 anomaly detection methods from diverse categories: classical machine learning, deep learning, and signal processing. They evaluated the performance of these algorithms on 976 uni- and multivariate time series across three different metrics. The study revealed that despite their high computational costs and extensive hyperparameter tuning, deep learning methods did not consistently outperform simpler, classical algorithms. This finding highlights the importance of context-specific solutions and challenges the assumption that deep learning is always superior. Rewicki et al. [RDN23] further reinforced these findings by comparing three classical machine learning methods and three deep learning models on the UCR Anomaly Archive [Ke21a]. Their results revealed that classical methods, such as MDI (Maximally Divergent Intervals) [Ba19] and MERLIN, outperformed deep learning models in accuracy and simplicity. In a more extensive and recent experimental analysis, Liu and Paparrizos [LP24] demonstrate again that simpler architectures and statistical methods frequently outperform advanced neural network architectures. In 2021, the Hexagon ML/UCR Time Series Anomaly Detection contest, held at SIGKDD, provided further insights into the practical challenges of time series anomaly detection [Ke21b, WK23, Ke21a]. Teams competed to detect anomalies in the UCR dataset [WK23, Ke21a] employing various strategies, including ensembling techniques and discord-based methods like STAMP and MERLIN. However, while ensemble methods enhance robustness, balancing and weighing different models is difficult. On the other hand, discords-based algorithms, used by many top-performing teams, were also complex to apply due to the subsequence length selection process.

STAN Positioning: The challenges outlined above underscore the need for a method that can accurately detect anomalies in time series data while minimizing the overhead of parameter tuning and computational time. STAN offers such a solution by combining a summary statistical ensemble, which can be efficiently computed, with the ACF to determine the optimal subsequence length automatically. This approach reduces the need for parameter tuning with a very efficient execution time without sacrificing detection accuracy.

3 Anomaly Detection Framework

STAN involves two main phases: (1) a **training phase** where a reference summary statistics ensemble matrix $\mathbf{E}(T_{train})$ is constructed from the training data, and (2) an **evaluation phase** where a new summary statistics ensemble matrix $\mathbf{E}(T_{test})$ is generated from the test data and compared against $\mathbf{E}(T_{train})$. STAN then identifies anomalies by detecting deviations that exceed the bounds of the expected values in the training ensemble matrix. The subsequence

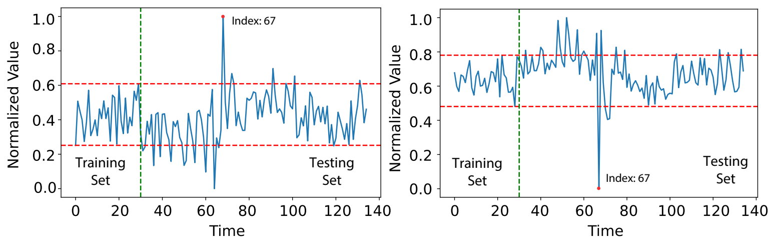


Fig. 1: Anomaly Detection Using *min* and *max* Statistics. The Green Vertical Line Separates the Training from the Testing. The Red Horizontal Lines Show the Bounds Found in $\mathbf{E}(T_{train})$.

with the largest deviation is flagged as a potential anomaly. Figure 1 illustrates an example where the minimum ($\mathbf{E}(T_{test})[min]$) and maximum ($\mathbf{E}(T_{test})[max]$) summary statistics are plotted, with STAN successfully detecting anomalies at index 67 by identifying values outside the normal range defined by the training data. In the remainder of this section, we detail the process of constructing $\mathbf{E}(T_{train})$ and explain how $\mathbf{E}(T_{train})$ is applied to detect anomalies in the testing set. Additionally, we discuss several important considerations in the design of STAN to ensure its effectiveness and flexibility.

3.1 Overall Method Design

Algorithm 1 presents the pseudocode for STAN. Given the input training and testing time series, STAN begins by computing the subsequence size (κ) using the training data (ComputeWindowSize, line 4). Next, STAN detrends the data by using first-order differencing and normalizes the time series. Our method then constructs the summary statistics ensemble matrices $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ by calculating a set of predefined summary statistics such as minimum, maximum, and mean, over sliding windows of length κ for both the training and testing sets. After constructing the ensemble matrices, STAN compares the summary statistics between the training and testing sets, identifying deviations that exceed the maximum and minimum values observed in the training set. STAN then selects the subsequence with the largest deviation as the anomaly and returns the corresponding index.

Computing the Window Size: A key contribution of this paper is the use of the ACF to set the window size. The intuition behind this approach is that the period length, or seasonal cycle of the time series, contains critical information that can effectively compare subsequences of the same length. Disruptions in the regular patterns within a seasonal period often indicate anomalies in the time series. Based on this assumption, we employ a simple yet effective method to estimate the seasonal period using the ACF [ESL22]. We outline the steps as follows:

1. Compute the ACF of the time series for a range of lags. Refer to Section 2 for an overview of the ACF.

Algorithm 1 STAN

```
Input T_train = [t_1, ..., t_n], T_test = [t_1, ..., t_z] : Training and testing time series
Output Index ∈ {1, ..., z}: Index of the anomaly
1: Deviations ← []                    // Initialize list to store deviations for each summary statistic
2: Indices ← []                       // Initialize list to store indices of the largest deviations
3: Statistics ← [min, max, mean, ···]               // Initialize the list of summary statistics
4: $\kappa$ ← ComputeWindowSize($T_{train}$)                // Compute the subsequence size using ACF
5: $T_{train}$ ← DeTrend($T_{train}$)                              // Detrend the training data
6: $T_{test}$ ← DeTrend($T_{test}$)                                 // Detrend the testing data
7: $T_{train}$ ← Normalize($T_{train}$)              // Normalize the training data in range [0,1]
8: $T_{test}$ ← Normalize($T_{test}$)                 // Normalize the testing data in range [0,1]
9: $\mathbf{E}_{train}$ ← Empty($\lfloor \frac{n}{\kappa} \rfloor$, Size(Statistics))  // Initialize the ensemble matrix for training data
10: $\mathbf{E}_{test}$ ← Empty($\lfloor \frac{z}{\kappa} \rfloor$, Size(Statistics))  // Initialize the ensemble matrix for testing data
11: for all w ∈ SlidingWindows($T_{train}$, $\kappa$) do
12:     for all ss ∈ Statistics do
13:         $\mathbf{E}_{train}[w, ss]$ ← ss($w$) // Compute and store the statistic for each window in training data
14: for all w ∈ SlidingWindows($T_{test}$, $\kappa$) do
15:     for all ss ∈ Statistics do
16:         $\mathbf{E}_{test}[w, ss]$ ← ss($w$)  // Compute and store the statistic for each window in testing data
17: for all ss ∈ Statistics do
18:     dv, idx ← ComputeLargestDeviation($\mathbf{E}_{train}$[ss], $\mathbf{E}_{test}$[ss])  // Find largest deviation
19:     Deviations ← Append(Deviations, dv)                   // Store the deviation in the list
20:     Indices ← Append(Indices, idx)           // Store the corresponding index of the deviation
21: max_dv ← ArgMax(Deviations)              // Identify the statistic with the largest deviation
22: index ← Indices[max_dv]        // Retrieve the index corresponding to the maximum deviation
23: return index · $\kappa$ + $\frac{\kappa}{2}$   // Return the final detected anomaly index, adjusted by the window size
```

2. Identify the ACF's significant peaks, local maxima, and minima. The highest local maximum that occurs after the first local minimum is selected as the dominant periodic length of the time series.

3. Set this peak lag as the window size κ.

Figure 2 shows an example of a time series sampled every 15 minutes and a daily seasonal period. Thus, the peak in ACF is found at lag 96, as indicated by the red dot. Although this method is not the only one that can be used to compute the window size, empirical results show it is the most effective. Section 4 provides evidence supporting this claim.

Computing the Largest Deviation: Once both summary statistic ensemble matrices $\mathbf{E}_{\text{train}}$ and $\mathbf{E}_{\text{test}}$ are computed, STAN finds the anomaly index by comparing the deviations between the values observed during training and those during evaluation. Specifically, as shown in line 18 of Algorithm 1, STAN calls the function ComputeHighestDeviation for each summary statistic ss. This function first checks if all values in the test set fall within the range defined by the minimum and maximum values in the training set. If the values fall within this range, ComputeHighestDeviation returns a deviation and index of -1, indicating no anomaly is detected. Suppose the test values exceed the training range. In that case, the function computes two differences: the deviation of the maximum test value

Algorithm 2 ComputeLargestDeviation

```
    Input $\mathbf{E}_{train}[\text{ss}]$: All training subsequences for summary statistics ss
          $\mathbf{E}_{test}[\text{ss}]$ : All testing subsequences for summary statistics ss
    Output deviation $\in [0, 1]$ : Distance to the closest training data point
           index $\in [1, \ldots, |\mathbf{E}_{test}[\text{ss}]|]$ : Index of the anomaly
 1: if (Max($\mathbf{E}_{test}[\text{ss}]$) $\leq$ Max($\mathbf{E}_{train}[\text{ss}]$)) and (Min($\mathbf{E}_{test}[\text{ss}]$) $\geq$ Min($\mathbf{E}_{train}[\text{ss}]$)) then
 2:     deviation $\leftarrow -1$
 3:     index $\leftarrow -1$
 4:     return deviation, index
 5: max_test_mean_diff $\leftarrow$ Max($\mathbf{E}_{test}[\text{ss}]$) $-$ Mean($\mathbf{E}_{train}[\text{ss}]$)
 6: min_test_mean_diff $\leftarrow$ Mean($\mathbf{E}_{train}[\text{ss}]$) $-$ Min($\mathbf{E}_{test}[\text{ss}]$)
 7: if max_test_mean_diff > min_test_mean_diff then
 8:     deviation $\leftarrow$ Max($\mathbf{E}_{test}[\text{ss}]$) - Max($\mathbf{E}_{train}[\text{ss}]$)
 9:     index $\leftarrow$ ArgMax($\mathbf{E}_{test}[\text{ss}]$)
10: else
11:     deviation $\leftarrow$ Min($\mathbf{E}_{test}[\text{ss}]$) - Min($\mathbf{E}_{train}[\text{ss}]$)
12:     index $\leftarrow$ ArgMin($\mathbf{E}_{test}[\text{ss}]$)
13: return deviation, index
```

from the mean of the training values and the deviation of the minimum test value from the mean of the training values. ComputeHighestDeviation then selects the larger deviations to identify the most significant anomaly. The function returns the maximum or minimum deviation and the corresponding index in the test set depending on the most significant deviation. This process allows STAN to pinpoint the subsequence in the test data that deviates most from the regular behavior during training. Algorithm 2 shows a pseudocode of the function.

Returning a Valid Index: To identify a valid anomaly index in the original time series, STAN calculates $index \cdot \kappa + \frac{\kappa}{2}$ in Line 23, where *index* is the outlier index from $\mathbf{E}_{\text{test}}$. This *index* is first multiplied by the window size κ to map it back to the time series' original scale, pointing to the start of the anomalous subsequence. Adding $\frac{\kappa}{2}$ shifts the position to the middle of the subsequence, providing a more precise location for the detected anomaly and reducing potential errors in its identification.

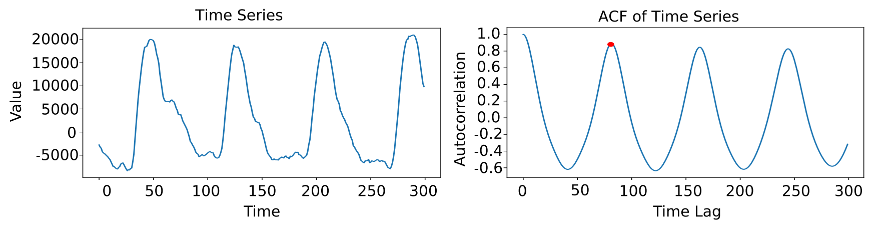


Fig. 2: Example of Seasonal Period Detected Using the ACF.

3.2 Summary Statistics

The choice of summary statistics in STAN significantly influences the accuracy and computational efficiency of anomaly detection. STAN is agnostic to the number and type of summary statistics used, allowing flexibility in capturing different characteristics of the time series. Among the basic statistics to use with STAN, we include the *mean*, *maximum*, *minimum*, and *standard deviation*. The *mean* provides information about the central tendency of each subsequence, helping in the detection of the following anomaly types: **Local Drop**, **Flat**, **Amplituded Change**, and basic **Outliers**. On the other hand, *minimum* and *maximum* values help identify spikes or dips, for example, **Steep Increase** or **Missing Peak**. Finally, the standard deviation and variance measure the spread of values, which can help identify changes in volatility inserted by **Unusual Patterns**. These basic statistics are computationally inexpensive, which makes them suitable for real-time or large-scale applications. However, their simplicity may limit their ability to detect more complex anomalies, such as **Time Warping**, **Reversed Vertically** or **Reversed Horizontally**. The reader can revisit Section 2 for background on anomaly types.

High-Order and Custom Summary Statistics: In addition to basic statistics, STAN incorporates higher-order statistics such as *skewness* and *kurtosis* to capture more complex patterns in the data. For example, *skewness* measures the asymmetry of the distribution within a subsequence, helping identify anomalies where the data exhibits an unexpected bias toward higher or lower values. On the other hand, *kurtosis* can help detect anomalies that involve outliers. Custom summary statistics can also be incorporated. For example, we propose a function that counts the number of *turning points*. This function counts the local minima and maxima in a subsequence, which helps detect changes in trends within a subsequence. While high-order and custom functions like these can improve accuracy, they introduce additional computational costs, especially if they involve complex calculations.

Time Complexity: STAN's execution time increases linearly with the number of summary statistics, allowing the inclusion of multiple statistics without significantly impacting performance. However, complex statistics can significantly affect execution time. Therefore, we recommend using summary statistics of linear or sublinear complexity to keep the execution time of STAN manageable. Specifically, given $m = |ss\_s|$, representing the number of summary statistics, if all statistics have linear complexity, then STAN's overall complexity is $O(n \cdot \max(\log n, m))$, where the $\log n$ factor comes from the ACF computation with complexity $O(n \log n)$. However, if any summary statistic has a higher complexity, such as $O(n^p)$ for $p > 1$, the overall complexity is dominated by this function rendering a $O(n^p)$ complexity. Thus, selecting efficient summary statistics is key to maintaining STAN's performance.

4 Experiments

Our experiments study STAN's accuracy and efficiency versus relevant anomaly detection baselines. As part of this analysis, we evaluate the accuracy of STAN in detecting specific anomaly types compared to several baselines. Additionally, we study the contribution of each summary statistic to the overall accuracy and their contribution to detecting specific anomaly types. Finally, we evaluate the accuracy of STAN under two other strategies to find the window size parameter and compare it with our strategy using the ACF.

4.1 Experimental Setup

This section provides information about the hardware and software specifications, the dataset, the metrics, and the selected baselines for conducting our experiment.

Hardware and Software Specifications: The experiments ran on a machine with an Intel Core i5-1035G1 @ 1.00-3.60 GHz CPU, 8 GB RAM, and 1 TB SSD. Furthermore, we use Python 3.11.6, MATLAB R2023b Update 4 (23.2.0.2428915), MERLIN 3.1 [Na20].

UCR Anomaly Archive: We used the UCR Anomaly Archive (UCR) benchmark dataset [WK23, Ke21a]. The UCR dataset consists of 250 univariate time series from different scientific areas. UCR is designed to be more robust than existing benchmarks by including manually verified anomalies and avoiding common flaws like mislabeled events. This benchmark dataset contains synthetic and real-world time series with diverse patterns and anomaly types, ensuring a comprehensive evaluation of anomaly detection methods. In Section 2.2, we introduce the anomaly types that can be found in this dataset. A common characteristic across the time series is that each one contains exactly one anomaly, representing a single data point or a subsequence from the time series. Each dataset is split into training and test data, where the training data is considered non-anomalous. Table 3 shows a summary description of this dataset.

Anomaly Detection Criteria and UCR-Score: The UCR dataset's scoring system evaluates anomaly detection algorithms by assigning a binary score to each detected anomaly, which is then aggregated into a final percentage score [Ke21a]. The scoring system uses a flexible error margin based on the length of the ground truth anomaly ($\mathcal{L}$) to accommodate different types of anomalies and detection methods. A data point t_i returned by an algorithm is

Tab. 3: Summary Statistics of the UCR Dataset.

Statistic	**Mean**	**Std. Dev.**	**Min**	**Max**
Time Series Length	77,415	120,080	6,684	900,000
Time Series Length (Train)	21,209	32,416	1,000	250,000
Time Series Length (Test)	56,205	92,099	3,302	707,630
Anomaly Length ($\mathcal{L}$)	197	237	1	1,701

considered a correct detection if it lies within a specified range around the ground truth anomaly's *begin* and *end* positions. For example, in time series T_j, if an algorithm returns the point t_i^j and the ground truth anomaly spans from $begin_j$ to end_j, the score for that time series is defined as:

$$\text{Score}(j) = \begin{cases} 1 & \text{if } (begin_j - \max(\mathcal{L}_j, 100)) \leq t_i^j \leq (end_j + \max(\mathcal{L}_j, 100)) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The margin of error is set to the maximum between the anomaly length $\mathcal{L}_j$ or 100 data points, allowing the score to handle both short and long anomalies. If an algorithm detects a whole subsequence as an anomaly, the suggested approach is to return its central data point [Ke21a]. The final UCR-Score is calculated as the percentage of correctly detected anomalies across the entire dataset as:

$$\text{UCR-Score} = \frac{100}{250} \sum_{j=1}^{250} \text{Score}(j) \tag{4}$$

Anomaly Detection Baselines: To perform our comparison, we partially reuse the baselines and results from a previous benchmark study in the UCR dataset [RDN23]. In this study, the authors selected three deep-learning and three classical machine-learning methods for unsupervised anomaly detection. Besides these baselines, we also evaluate a matrix profile algorithm. Below, we introduce our baselines:

- **Robust Random Cut Forest (RRCF) [Gu16]:** RRCF is an extension of the Random Cut Forest algorithm, designed to detect anomalies in streaming data. It works by constructing an ensemble of random trees (forests) using random data splits. Anomalies are identified based on the structure of these trees; data points that result in unusual splits (cuts) are flagged as anomalies.
- **Maximally Divergent Intervals (MDI) [Ba19]:** MDI identifies intervals with significant deviations by computing the Kullback-Leibler (KL) [KL51] divergence between the statistical distribution of candidate intervals and a baseline distribution representing expected behavior. Intervals with the highest KL divergence scores are flagged as anomalies.
- **MERLIN [Na20]:** An extension of the Matrix Profile, MERLIN searches for anomalies over a range of subsequence lengths, removing the need to predefine a fixed window size. MERLIN constructs an ensemble of Matrix Profiles for different subsequence lengths and detects anomalies by finding the subsequences with the highest distances to their nearest neighbors.
- **Autoencoder (AE) [RHW86]:** An autoencoder is a neural network model designed to learn a compressed representation of the input data. The network is trained on normal (anomaly-free) data for anomaly detection. During testing, anomalies are identified by measuring the reconstruction error; if the error exceeds a predefined threshold, the input is considered an anomaly.

- **Graph Augmented Normalizing Flows (GANF) [DC22]:** GANF extends normalizing flows by incorporating graph structures into the model, allowing GANF to learn complex data distributions.
- **Transformer Network for Anomaly Detection (TranAD) [TCJ22]:** TranAD leverages the Transformer neural network architecture, known for its ability to capture long-term dependencies in sequential data. The model is trained to predict the next data point in a sequence, and anomalies are detected based on the prediction error.
- **Matrix Profile (STUMPY) [La19]:** Matrix Profile is a technique for finding patterns, motifs, and anomalies in time series data by computing the distances between all subsequences in the series. STUMPY is a highly efficient implementation of the Matrix Profile that allows for fast and scalable analysis. Anomalies are identified as subsequences with the most significant distance to their nearest neighbor.

MERLIN Configuration: MERLIN requires two input parameters: a lower bound **l** and an upper bound **u** for the subsequence lengths. We evaluate two configurations of MERLIN based on different values of **l** and **u**:

- **MERLIN (ACF):** In this variant, the lower and upper bounds are set using our method for determining the window size κ with the ACF. Specifically, we define $\mathbf{l} = \kappa - 5$ and $\mathbf{u} = \kappa + 5$, providing a narrow range around the detected window size. This limited range helps keep the execution time manageable.
- **MERLIN (fixed):** This configuration uses fixed values for the lower and upper bounds, following the reference benchmark study [RDN23]. Here, we set $\mathbf{l} = 75$ and $\mathbf{u} = 125$. This range was selected in an unsupervised manner to allow the detection of a wide range of anomalies.

STUMPY Configuration: STUMPY requires an input parameter: the subsequence size. Like MERLIN, we evaluate two different configurations of STUMPY based on different strategies to set ss.

- **STUMPY (ACF):** We automatically compute the subsequence length using our ACF-based method.
- **STUMPY (fixed):** The subsequence length is fixed to 100, which is the middle point of the lower and upper bounds used in the MERLIN (fixed) variant.

Other Baselines Configurations: For the remaining baseline methods, we utilize the accuracy results reported in the reference benchmark study [RDN23]. For detailed information on these methods' configuration and parameter settings, we refer the reader to the original paper and the available artifacts at https://gitlab.com/dlr-dw/is-it-worth-it-benchmark.

STAN Configuration: STAN provides flexibility in choosing summary statistics. However, we recommend a default configuration that covers eight summary statistics: four low-order, two high-order, and two custom implementations. The low-order statistics include *mean*, *standard deviation*, *minimum*, and *maximum*. For high-order statistics, we use *kurtosis* and *skewness*. Lastly, the custom summary statistics are *turning points* and *point anomaly*. The *point anomaly* statistic computes the standard deviation over every three consecutive data points, aiming to enhance the detection of single-point outliers that aggregates might obscure over longer subsequences. The *turning points* statistic counts the number of local minima and maxima in a subsequence, which is particularly useful for identifying changes in trends and potential time-warping anomalies. While STAN is not limited to these eight summary statistics, our experimental results show that this selection is already sufficient to achieve comparable accuracy and a significant execution time improvement. We make our STAN code available at https://git.tu-berlin.de/kristiyanblagov/btw2025.

4.2 Anomaly Detection Accuracy

In this set of experiments, we evaluate the anomaly detection accuracy of the UCR dataset. Figure 3 presents the UCR-Score for STAN and the baseline methods. STAN achieves the second-highest score at 60.4%, correctly identifying 151 out of 250 anomalies. At the same time, MERLIN (fixed) performs best with a score of 63.6%. The choice of subsequence length plays a critical role in these results. STUMPY benefits from the ACF-based method, while MERLIN performs better with a fixed length. This difference suggests that a broader range of subsequence lengths, like the 50 lengths used by MERLIN's fixed strategy, captures more anomalies. However, expanding the ACF-based range would increase computational costs, especially when the suggested ACF-based length is large. For example, the optimal subsequence length suggested by ACF can be 1,440 for a time series sampled every minute and with a daily seasonality. The accuracy difference between STUMPY and MERLIN also emphasizes the importance of the subsequence length parameter. Although MERLIN essentially extends STUMPY's operations across multiple lengths, the results highlight how the choice of subsequence length directly impacts anomaly detection accuracy.

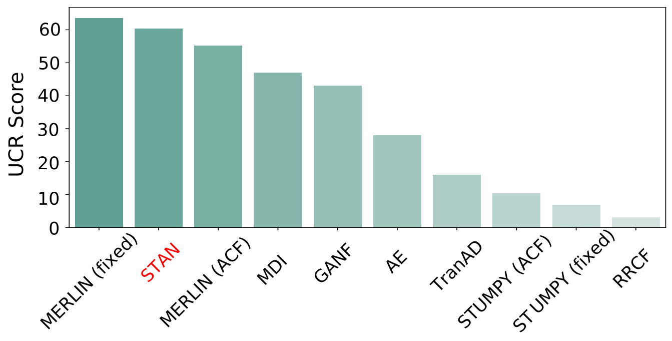


Fig. 3: UCR Score for our STAN and Baseline Methods.

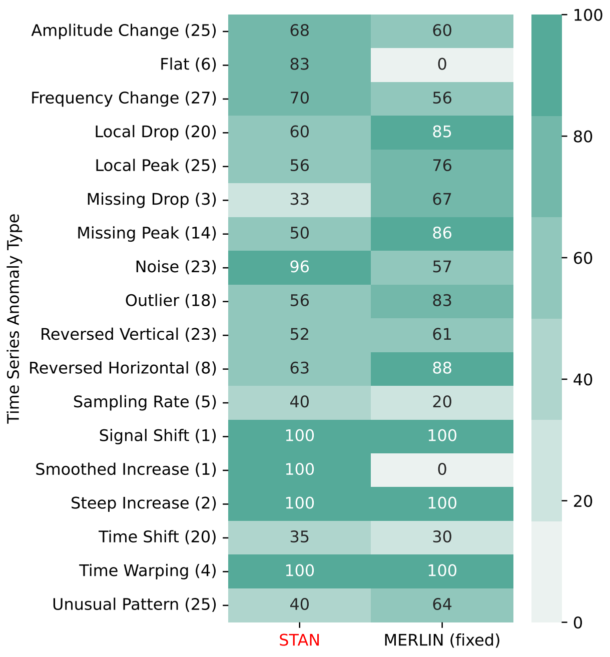


Fig. 4: Percentage of Correctly Detected Anomaly per Type. In Brackets is the Number of Anomalies.

On the other hand, the accuracy difference between STUMPY and MERLIN underscores the importance of the subsequence size parameter. The core operations are identical since MERLIN applies STUMPY efficiently across a range of subsequence lengths. However, the results vary significantly, highlighting the impact that the choice of subsequence size can have on anomaly detection accuracy.

Accuracy per Anomaly Type: Figure 4 presents the accuracy of our STAN and MERLIN (fixed) across different anomaly types. The figure shows significant variations across different time series anomalies. STAN substantially outperforms MERLIN (fixed) in detecting complex anomalies such as *Noise* (96% vs. 57%), *Frequency Change* (70% vs. 56%), and *Flat* (83% vs. 0%). Additionally, STAN shows high accuracy in identifying anomalies like *Signal Shift*, *Steep Increase*, and *Time Warping*, achieving perfect detection (100%), although the number of instances is quite limited. Another key strength of STAN is its ability to detect anomalies in more diverse categories, such as *Sampling Rate* and *Smoothed Increase*, where MERLIN either struggles or fails to detect anomalies altogether. These results suggest that the combination of summary statistics in STAN, especially those designed to capture complex patterns like *turning points*, provides a better detection strategy.

Limitations: MERLIN outperforms STAN in detecting *Outlier* anomalies (83% vs. 56%) and has better results in detecting *Reversed Vertical* (61% vs. 52%) and *Reversed Horizontal* (88% vs. 63%) anomalies. These results indicate that while STAN's summary statistics are

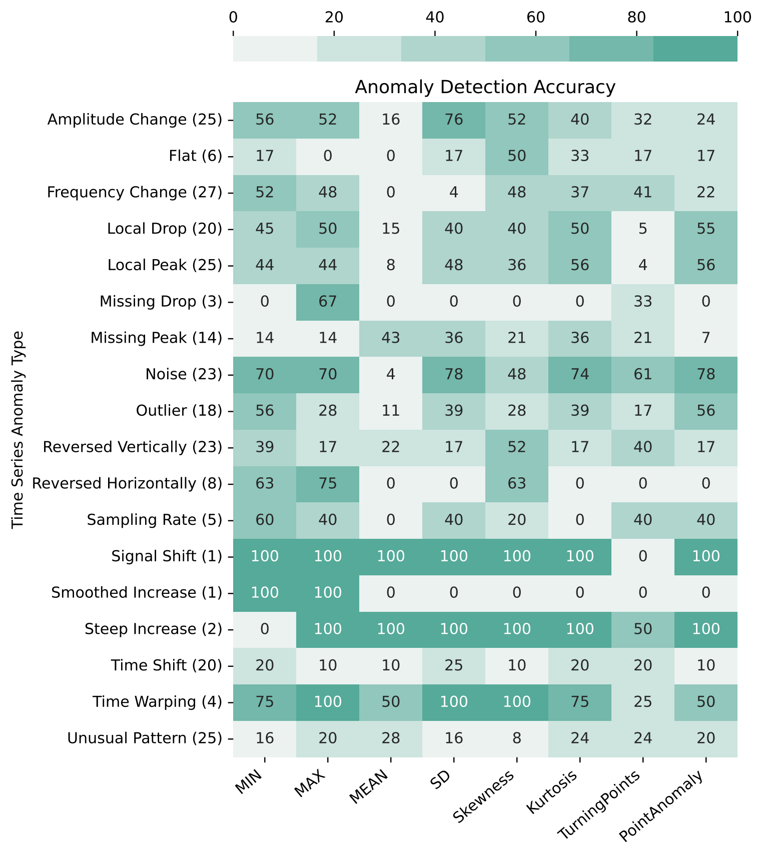


Fig. 5: Percentage of Correctly Detected Anomalies per Summary Statistics.

effective for a broad range of anomalies, additional statistics may be needed to enhance its accuracy for these specific types. In future work, we will explore complex statistics tailored specifically to improve the detection of these anomaly types.

Contribution per Summary Statistic: We also analyze the individual contribution of each summary statistic to anomaly detection accuracy. Figure 5 displays the accuracy obtained for each summary statistic, broken down by anomaly type. Below, we rank the summary statistics based on their overall performance and describe their specific contributions:

1. **Standard Deviation (SD):** With 108 out of 250 correctly detected anomalies (43.2%), SD is the best-performing stand-alone statistic. SD demonstrates consistent performance across various anomaly types, particularly excelling in detecting *Amplitude Change* (76%), *Noise* (78%) while showing relative success detecting *Outlier* (39%).
2. **Minimum (Min)**: Detecting 107 out of 250 anomalies (42.8%), Min shows good

performance in detecting *Outlier* (56%) and *Sampling Rate* (60%) anomalies. However, Min struggles with certain anomaly types like *Flat* (17%), indicating its strength lies in capturing pronounced value drops.

3. **Maximum (Max):** Detecting of 99 out of 250 (39.6%). Max is effective for *Local Drop* (50%) and *Reversed Horizontal* anomalies (75%), for which it achieves the best among all statistics.
4. **Kurtosis (KU):** Detecting of 95 out of 250 (38%), KU shows good performance in detecting *Local Peak* and *Noise* anomalies with 56% and 74%, respectively.
5. **Skewness (SK):** Detecting 94 out of 250 anomalies (37.6%), SK is among the best detecting *Reversed Horizontally* (63%) anomalies. At the same time, most other statistics fail to detect at least one of these anomalies.
6. **Point Anomaly (PA):** Detecting 84 out of 250 anomalies (33.6%), this custom function is, as expected, strong in identifying *Noise* (78%) and *Outlier* (56%) anomalies. By focusing on local variance over small segments, PA successfully detects subtle changes that aggregates miss over long subsequence lengths.
7. **Turning Points (TP):** With 58 out of 250 anomalies correctly detected (23.2%), TP is effective in detecting *Missing Drop* and *Frequency Change* anomalies. In the case of *Missing Drop*, only Max and TP are capable of detecting anomalies.
8. **Mean:** With 46 out of 250 anomalies (18.4%), Mean shows limited effectiveness overall, with modest accuracy in detecting most of the anomaly types. Nonetheless, Mean is the best detecting *Missing Peak* (43%). Thus, despite its lower performance, it can still improve accuracy with very little computational overhead.

Accumulated Impact: In the next set of experiments, we evaluate the incremental contribution of summary statistics to STAN's performance. Starting with SD (the best-performing statistic individually), we progressively added more statistics, ending with Mean (the least effective individually). The results, shown in Figure 6, demonstrate that as additional statistics are incorporated, STAN's performance steadily improves, ultimately achieving a UCR Score of 60.4. These results confirm that while statistics like SD and Max are highly effective at detecting a broad range of anomalies, others such as PA, TP, and Mean play a complementary role by targeting specific anomaly types, making them valuable additions to the ensemble. For future work, we plan to explore strategies to further enhance STAN's performance by combining the strengths of each statistic through a weighting scheme informed by preprocessing and feature extraction of the time series.

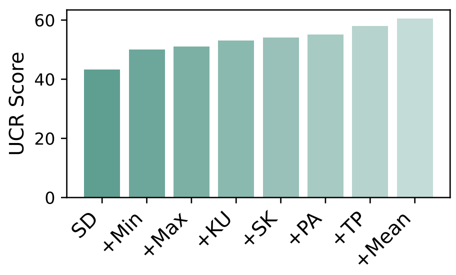
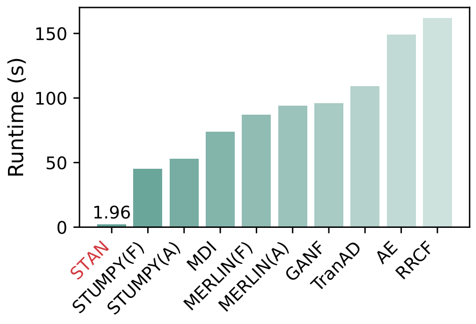
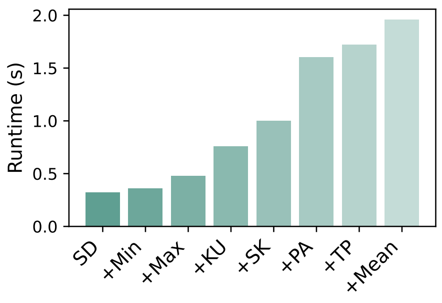


Fig. 6: Accumulated Accuracy per Statistics.



(a) STAN vs Baselines



(b) Accumulated Runtime per Statistic

Fig. 7: STAN's and Baselines Runtime (a) and Accumulated Runtime per Summary Statistics (b).

4.3 Execution Time Evaluation

Figure 7 presents two key aspects of STAN's runtime performance. Specifically, Figure 7(a) shows the mean runtime (in seconds) of STAN compared to baseline methods, while Figure 7(b) illustrates STAN's cumulative runtime as additional summary statistics are incrementally incorporated. The reported runtimes include both the training and evaluation phases for STAN and all baselines. We use (F) and (A) to refer to the fixed- and ACF-based strategies for computing the window size, respectively. Overall, STAN achieves the fastest execution time, requiring approximately 2 seconds per time series, compared to STUMPY, which takes over an order of magnitude longer (close to 45 seconds). MERLIN and MDI, the baselines closest to STAN in terms of accuracy, take between 74 and 87 seconds per time series, making STAN at least 40x faster. On the other hand, Figure 7(b) shows a linear increase in runtime as summary statistics are added. A slight increase of 0.5 seconds is observed when PA, one of our custom-designed summary statistics, is added. This sudden increase is due to PA's use of sliding windows of size one, which introduces additional computations compared to the other statistics. These results, combined with STAN's high detection accuracy, make a strong case for the proposed approach and highlight its potential as a foundation for future methods employing similar strategies.

4.4 Window Size Evaluation

In this set of experiments, we evaluate the performance of STAN using different strategies to compute the window size parameter. Specifically, we compare our ACF-based method (referred to as ACF for simplicity) with two alternative methods: the Fast Fourier Transform (FFT) and the Multi-Window-Finder (MWF) [ESL22, IK21]. The FFT method identifies the dominant frequency in the time series by applying the Fourier transform and sets the window size based on the most dominant Fourier coefficient. In contrast, the MWF method selects the window size by comparing the variance of moving averages across a range of candidates, choosing the one with the smallest variance as the most suitable. These methods were selected based on their strong performance in a previous benchmark study [ESL22].

Results Analysis: Figure 8 presents the results of STAN using the window sizes derived from these three methods on the UCR dataset. As shown, the ACF method achieves the highest accuracy (60.4%), consistent with the results reported in previous sections. When using the MWF method, STAN correctly detects 137 out of 250 anomalies (54.8%), outperforming most baseline methods. In comparison, the FFT-based approach identifies 120 out of 250 anomalies (48%), resulting in approximately a 12% decrease in accuracy for STAN.

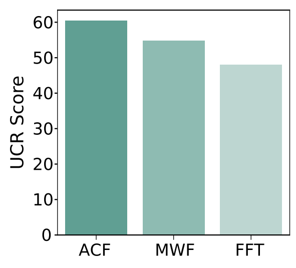


Fig. 8: Different Window Size Selection Methods Accuracy.

5 Conclusions and Future Work

We introduced STAN, a fast and parameter-free algorithm for time series anomaly detection. STAN leverages an ensemble of summary statistics to capture the diverse patterns of non-anomalous time series during the training phase. These summary statistics are then used in the evaluation phase to identify divergences from regular patterns, thereby detecting anomalies. Together with an ACF-based window size computation, our method is accurate and eliminates the need for user intervention and hyperparameter tuning. Based on our experimental results, we draw the following conclusions: 1) STAN achieves high detection accuracy, surpassing more complex anomaly detection methods such as MDI and Matrix Profile. 2) STAN is more efficient than the baselines, with an execution time of at least an order of magnitude faster. 3) Combining different, low- and high-order summary statistics positively impacts detection accuracy. 4) Our ACF-based strategy for automatically selecting the window size parameter improves detection accuracy and frees users from manually setting this parameter. Together, these results strongly support using summary statistics ensembles in anomaly detection, which reduces the need for extensive parameter tuning and computational costs. For future work, we plan to explore the addition of more summary statistics and develop strategies for weighting their contribution to the ensemble. We will also investigate runtime improvements by leveraging parallelization to achieve high-performance anomaly detection. Finally, we will test STAN's performance in recently published benchmark datasets [LP24].

Acknowledgments

We gratefully acknowledge funding from the German Federal Ministry of Education and Research (under research grant BIFOLD24B).

Bibliography

[A 08] A Dictionary of Statistics. Oxford University Press, 2008.

[Ba19] Barz, Björn; Rodner, Erik; Garcia, Yanira Guanche; Denzler, Joachim: Detecting Regions of Maximal Divergence for Spatio-Temporal Anomaly Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(5):1088–1101, 2019.

[BPP23] Boniol, Paul; Paparrizos, John; Palpanas, Themis: New Trends in Time Series Anomaly Detection. In: EDBT. OpenProceedings.org, pp. 847–850, 2023.

[CLW69] Cooley, James W.; Lewis, Peter A. W.; Welch, Peter D.: The Fast Fourier Transform and Its Applications. IEEE Transactions on Education, 12(1):27–34, 1969.

[DC22] Dai, Enyan; Chen, Jie: Graph-Augmented Normalizing Flows for Anomaly Detection of Multiple Time Series. In: International Conference on Learning Representations. 2022.

[De22] Der, Audrey; Yeh, Chin-Chia Michael; Wu, Renjie; Wang, Junpeng; Zheng, Yan; Zhuang, Zhongfang; Wang, Liang; Zhang, Wei; Keogh, Eamonn J.: Matrix Profile XXVII: A Novel Distance Measure for Comparing Long Time Series. In: ICKG. IEEE, pp. 40–47, 2022.

[De23] Der, Audrey; Yeh, Chin-Chia Michael; Zheng, Yan; Wang, Junpeng; Chen, Huiyuan; Zhuang, Zhongfang; Wang, Liang; Zhang, Wei; Keogh, Eamonn J.: Time Series Synthesis Using the Matrix Profile for Anonymization. In: BigData. IEEE, pp. 1908–1911, 2023.

[ESL22] Ermshaus, Arik; Schäfer, Patrick; Leser, Ulf: Window Size Selection In Unsupervised Time Series Analytics: A Review and Benchmark. In: 7th Workshop on Advanced Analytics and Learning on Temporal Data. 2022.

[Gu16] Guha, Sudipto; Mishra, Nina; Roy, Gourav; Schrijvers, Okke: Robust Random Cut Forest Based Anomaly Detection on Streams. In: ICML. volume 48. PMLR, pp. 2712–2721, 20–22 Jun 2016.

[IK21] Imani, Shima; Keogh, Eamonn: Multi-window-finder: Domain agnostic window size for time series data. Proceedings of the MileTS, 21, 2021.

[Ke21a] Keogh, E.; Dutta Roy, T.; Naik, U.; Agrawal, A.: , Multi-dataset Time-Series Anomaly Detection Competition, SIGKDD 2021. https://compete.hexagon-ml.com/practice/competition/39/, 2021.

[Ke21b] Keogh, Eamonn et al.: Time-Series Anomaly Detection Datasets. SIGKDD, 2021.

[KL51] Kullback, S.; Leibler, R. A.: On Information and Sufficiency. The Annals of Mathematical Statistics, 22(1):79 – 86, 1951.

[KLF05] Keogh, Eamonn J.; Lin, Jessica; Fu, Ada Wai-Chee: HOT SAX: efficiently finding the most unusual time series subsequence. ICDM, pp. 8 pp.–, 2005.

[La19] Law, Sean M.: STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining. The Journal of Open Source Software, 4(39):1504, 2019.

[Li24] Liu, Qinghua; Boniol, Paul; Palpanas, Themis; Paparrizos, John: Time-Series Anomaly Detection: Overview and New Trends. Proc. VLDB Endow., 17(12):4229–4232, 2024.

[LP24] Liu, Qinghua; Paparrizos, John: The Elephant in the Room: Towards A Reliable Time-Series Anomaly Detection Benchmark. In: NeuroIPS/Benchmark Track. 2024.

[Me21] Mercer, Ryan; Alaee, Sara; Abdoli, Alireza; Singh, Shailendra; Murillo, Amy C.; Keogh, Eamonn J.: Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification. In: ICDM. IEEE, pp. 1240–1245, 2021.

[Na20] Nakamura, Takaaki; Imamura, Makoto; Mercer, Ryan; Keogh, Eamonn: MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives (Long version for website). In: ICDM. pp. 1190–1195, 2020.

[NB00] Nounou, Mohamed N.; Bakshi, Bhavik R.: Data Handling in Science and Technology. Elsevier, 2000.

[Pa17] Park, Kun Il: Fundamentals of Probability and Stochastic Processes with Applications to Communications. Springer International Publishing, November 2017.

[RDN23] Rewicki, Ferdinand; Denzler, Joachim; Niebling, Julia: Is It Worth It? Comparing Six Deep and Classical Methods for Unsupervised Anomaly Detection in Time Series. Applied Science, 13:1778, 2023.

[RHW86] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J.: Learning internal representations by error propagation. pp. 318–362, 1986.

[SWP22] Schmidl, Sebastian; Wenig, Phillip; Papenbrock, Thorsten: Anomaly detection in time series: a comprehensive evaluation. Proc. VLDB Endow., 15(9):1779–1797, may 2022.

[TCJ22] Tuli, Shreshth; Casale, Giuliano; Jennings, Nicholas R.: TranAD: deep transformer networks for anomaly detection in multivariate time series data. Proc. VLDB Endow., 15(6):1201–1214, feb 2022.

[Wa05] Wasserman, Larry: All of Statistics: A Concise Course in Statistical Inference. Springer Texts in Statistics. Springer, 2005.

[WK23] Wu, Renjie; Keogh, Eamonn: Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. TKDE, 35(3):2421–2429, 2023.

[Ye16] Yeh, Chin-Chia Michael; Zhu, Yan; Ulanova, Liudmila; Begum, Nurjahan; Ding, Yifei; Dau, Hoang Anh; Silva, Diego Furtado; Mueen, Abdullah; Keogh, Eamonn: Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In: ICDM. pp. 1317–1322, 2016.

[Zh16] Zhu, Yan; Zimmerman, Zachary; Senobari, Nader Shakibay; Yeh, Chin-Chia Michael; Funning, Gareth J.; Mueen, Abdullah; Brisk, Philip; Keogh, Eamonn J.: Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In: ICDM. IEEE, pp. 739–748, 2016.