Politecnico di Torino

ScuDo

Scuola di Dottorato ~ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Artificial Intelligence (38th)

On Mining Time Series Data with Random Forest Models:
Perspectives from Classification, Anomaly Detection, and Distance Measures

Alberto Azzari

* * * * *

Supervisors:
Prof. Pietro Sala, Supervisor
Prof. Manuele Bicego, Co-supervisor
Prof. Carlo Combi, Co-supervisor

University of Verona and Politecnico of Torino
March 09, 2026

I hereby declare that the contents and organization of this dissertation constitute my own original work and do not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Alberto Azzari,
Verona, March 09, 2026

Summary

Time series data are ubiquitous across scientific, industrial, and medical domains, capturing the temporal dynamics of complex systems. Extracting meaningful patterns from these sequences is essential for tasks such as classification, anomaly detection, and forecasting. However, time series analysis often presents unique challenges, such as temporal dependencies, variable lengths, and noise, that require the development of specialized methods. A promising research direction is to extend well-established approaches originally designed for vectorial data to the temporal domain. For instance, techniques like Random Forests have achieved remarkable success in tabular data analysis due to their robustness, interpretability, and ability to handle nonlinear relationships. Yet, their adaptation and systematic exploration in the context of time series data remain relatively underexplored. In this thesis, we advance the state of the art of random forests applied to the time series domain along several directions:

First, we develop robust classification frameworks for intraoperative motor evoked potentials (MEPs), time series signals recorded during neurosurgery. Accurate classification of MEPs is critical for monitoring the integrity of motor pathways and guiding surgical decisions. The proposed models leverage Random Forests and forest-based approaches, which can be effectively trained even with limited datasets, providing reliable predictions in high-stakes medical contexts.

Second, we introduce isolation-based models for the early detection of anomalies in MEPs, extending the Isolation Forest paradigm to identify abnormal patterns before they fully manifest. Isolation Forests are among the most widely used state-of-the-art methods for anomaly detection in vectorial data. However, an effective extension capable of handling time series data is missing. The proposed methods address this gap by enabling timely interventions in clinical settings, helping neurosurgeons receive early warnings.

Third, we propose TSRF-Dist, a novel forest-based distance measure that combines an unsupervised tree-based model with the power of Random Forest–derived distances. Recently, distances computed from Random Forests have been developed for vectorial data and demonstrated strong potential in capturing complex relationships between samples. Motivated by these advances, this thesis presents TSRF-Dist, which transposes this knowledge to the time series domain.

Finally, we present tsdistances, a high-performance Python library with a Rust backend that provides efficient computation of elastic distances for time series datasets on both CPU and GPU architectures. While numerous implementations of time series distance measures exist, many remain limited in terms of scalability, extensibility, or computational efficiency. To overcome these limitations, tsdistances integrates several state-of-the-art optimizations for dynamic programming matrix computation, offering a framework that accelerates experimentation and promotes the adoption of time series distances by making their use feasible even for large-scale datasets.

Through these contributions, this thesis provides tools that are accurate, efficient, and accessible to researchers, bridging methodological innovation with practical applications in time series analysis. The main contributions are detailed in the subsequent chapters, including experimental evaluations, comparisons with state-of-the-art approaches, and discussion of their relevance in both medical and general time series contexts.

Contents

List of Tables

List of Figures

1 Introduction

Humans have an extraordinary ability to recognize patterns and detect anomalies in complex sequences of events. For example, a doctor can quickly notice an irregular heartbeat on an ECG, or a driver can anticipate changes in traffic flow. These tasks, which seem almost effortless to humans, are far from trivial for machines. Unlike humans, machines cannot rely on intuition; they must explicitly learn underlying patterns from data. This necessity has driven decades of research in time series analysis, a field concerned with sequences of observations collected over time that plays a fundamental role in both scientific and industrial domains.

Time series are everywhere. They appear in finance [6, 7], where stock prices fluctuate minute by minute; in medicine [8, 9], where biosignals track patient health; and in engineering [10, 11], where sensors monitor machinery or traffic patterns. Each sequence carries essential information about the system it represents, but this information is often embedded in complex temporal dependencies and subtle variations. Extracting meaningful patterns from such sequences requires carefully designed algorithms and models capable of capturing both local and global behaviors.

To analyze time series, researchers have developed a variety of techniques. Distance-based methods remain central [12], providing a way to measure similarity or dissimilarity between sequences. Naive techniques, such as Euclidean distance, allow simple point-to-point comparisons, while more sophisticated methods, including Dynamic Time Warping (DTW) and its variants [13–15], align sequences in time to account for temporal distortions. Other approaches, like edit-based distances [16] and shape-based measures [17], handle more complex transformations, accommodating insertions, deletions, or variations in the overall shape of the signal. Complementing these, feature-based methods extract meaningful characteristics from sequences [1, 18], which can then be processed with standard machine learning techniques. These include interval-based [19, 20], shapelet-based [21, 22], dictionary-based [23, 24], and convolutional approaches [25]. Interval-based methods extract features from fixed or sliding segments of the time series, capturing local statistics or patterns. Shapelet-based methods identify discriminative sub-sequences that are most informative for classification. Dictionary-based approaches transform sequences into symbolic representations and analyze the frequency of recurring patterns. Convolutional approaches apply learned filters to detect local motifs or temporal structures. Another category consists of deep learning-based techniques, which automatically learn hierarchical representations from raw data. Models that combine different strategies fall under hybrid methods, leveraging the strengths of multiple approaches to improve performance.

Among data mining techniques, Random Forests (RFs) have proven particularly effective for vectorial representations [26]. Building on decision trees and ensemble strategies such as bagging and boosting, RFs provide robustness, interpretability, and scalability. Several widely used extensions of RFs have been developed for

anomaly detection, including Isolation Forest [27, 28]. They have also been applied to clustering [29, 30] and for deriving distances between objects [31, 32]. Over time, these models have been adapted specifically for time series, leading to forest-based models that can capture both the sequential structure of data and the underlying patterns needed for accurate predictions.

While time series analysis has a rich methodological landscape, many applications remain unexplored, particularly in clinical contexts. The first line of work in this thesis focused on the classification of intraoperative motor evoked potentials (MEPs), a medical scenario that had not been studied in depth. We systematically investigated different representations of the data and combinations of models to identify effective strategies. The primary motivation was that the clinical task of classifying MEPs by muscle of origin had not been addressed, despite its potential for supporting decision-making during a pre-surgical checklist. Beyond classification, very few methods existed for anomaly detection of time series, and none addressed scenarios where the anomaly affects the entire sequence rather than individual points, highlighting a clear gap in the literature. To address the clinical problem of detecting anomalous MEPs during neurosurgery, the first extension of the Isolation Forest for time series anomaly detection was developed. Previous studies have shown that distances derived from Random Forests, such as RatioRF or the Zhu distance [31, 32], are effective for vectorial data, even in the absence of labels. Since distances are critical for tasks such as classification, clustering, and retrieval in time series, it was natural to investigate whether similar approaches could be adapted for sequences using models specifically tailored to time series data. This led to the development of the first Random Forest-based distance for time series. Finally, in the era of big data, efficient computation of time series distances is increasingly important. Accurate distance measures are essential for large-scale applications, but naive implementations are often prohibitively slow. This motivated the development of optimized algorithms, including dynamic programming enhancements such as wavefront computation, as well as GPU-accelerated implementations, to ensure that the proposed methods are not only theoretically sound but also practically usable at scale. These solutions were integrated into tsdistances, a high-performance Python library that implements all of these algorithms to provide truly scalable results.

1.1 Contributions

The research presented in this thesis unfolds along four complementary directions.

1. The first contribution concerns the classification of intraoperative MEPs. We constructed a dedicated dataset from intraoperative recordings and systematically evaluated different combinations of classifiers and data representations. The results, compared against expert assessments, demonstrated that machine learning methods can reliably distinguish MEPs by muscle of origin, paving the way for clinically meaningful automation.

2. The second contribution introduces a new anomaly detection framework for MEPs. We proposed the Canonical Isolation Tree and Canonical Isolation Forest (CISOF), which formalize and generalize the isolation principle and extend it to early detection. Experiments on intraoperative data confirmed the potential of CISOF and its variants for both anomaly detection and early anomaly detection.
3. The third contribution is methodological and more general in scope: the proposal of a novel distance measure for time series. Building on the random forest framework, we introduced TSRF-Dist, a distance derived from Extremely Randomized Canonical Interval Forests. Evaluations on datasets from the UCR Time Series Archive showed that TSRF-Dist achieves competitive accuracy and speed compared to state-of-the-art distances.
4. The fourth contribution addresses the practical problem of scalability and usability by providing efficient implementations of time series distances. To bridge this gap, we developed tsdistances, a high-performance Python library with Rust backend implementing both classical and novel distances, optimized for CPU and GPU computation.

While this thesis is partially motivated by a concrete medical application, its primary contribution is methodological. Intraoperative MEPs are used as a challenging and clinically relevant case study that guides and validates the research, whereas the proposed models, distance measures, and computational tools are designed to be general and applicable to time series analysis beyond this specific domain.

1.2 Organization of thesis

The thesis is organized into seven chapters, including the present one.

Section 2 provides the necessary background, starting with an introduction to time series and a motivating real-world example. It then reviews distance-based methods such as Dynamic Time Warping, edit-based and shape-based measures, as well as alternative approaches (feature-based, interval-based, shapelets, dictionary-based, convolutional, and deep learning methods). The chapter concludes with ensemble learning, decision trees, and random forests in the context of time series classification.

Section 3 introduces motor evoked potentials (MEPs) and their clinical relevance. It then presents a study on the classification of intraoperative MEPs, describing the medical background, dataset creation, and model training. The results are compared both with human expert classification and with alternative machine learning classifiers.

Section 4 introduces the Canonical Isolation Tree and Forest (CISOF) for detecting and anticipating pathological MEPs. Variants of CISOF are evaluated experimentally and compared with existing anomaly detection methods.

Section 5 develops TSRF-Dist, a novel time series distance derived from Extremely Randomized Canonical Interval Forests, and evaluates it on UCR benchmark datasets against state-of-the-art alternatives.

Section 6 presents tsdistances, a high-performance Python library for time series distances, including optimized CPU and GPU implementations for elastic measures.

Finally, Section 7 summarizes the main findings, highlights methodological and applied contributions, and outlines future research directions. An appendix provides additional material on TSRF-Dist.

1.3 Publications

Several parts of this thesis have been submitted to, or published in, conference proceedings and international journals. Specifically, the work in Section 3 was first presented as an oral contribution at the European Association of Neurosurgical Societies (EANS) conference [33], and was later published in Computers in Biology and Medicine [34]. The contributions of Section 4 led to a submission to the Journal of Healthcare Informatics Research as well as a patent application, which has already passed the university's validation committee. The ideas presented in Section 5 appeared in Data Mining and Knowledge Discovery [2]. Finally, Section 6 has been published in ACM Transactions on Mathematical Software.

In addition, a study on the stability of the isolation forest, published in the proceedings of the IAPR Joint International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR) [35], has been further developed in collaboration with Manuele Bicego, Thomas G. Dietterich, Si Liu, and Antonella Mensi into a journal article entitled "On the Stability of the Isolation Forest Results", which will be submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE).

2 Background

This chapter provides an overview of the main concepts and techniques relevant to this thesis. We begin by introducing the formal definition of time series and illustrating their role through real-world examples Section 2.1. We then discuss distance-based methods (Section 2.2), which form some of the most fundamental tools for comparing and mining time series. Next, we explore other methodological categories (Section 2.3), including feature-based, interval-based, shapelet-based, dictionary-based, convolution-based, and deep learning approaches, as well as hybrid models that combine multiple paradigms. The structure of this chapter largely follows the recent comprehensive survey by Middlehurst et al. [3]. Figures adapted from previously published work are included with the authors' permission, with the source indicated in the captions; all other figures are original content created specifically for this thesis.

Finally, we turn our attention to random forest models for time series classification (Section 2.4). After reviewing the building blocks of ensemble learning, such as bagging, boosting, and decision trees, we present the random forest algorithm and discuss extensions tailored to sequential data. This background provides the foundation for the forest-based approaches developed later in the thesis.

2.1 Time Series

A time series is a sequence of observations recorded at successive points in time. These observations may be gathered at regular intervals (e.g., daily, monthly, yearly) or continuously over time. Time series data are studied in several domains [36–39].

Formally, a **discrete-time series** is a sequence of real-valued random variables:

$$x = \{x_1, x_2, ..., x_N\} \tag{1.}$$

where each x_i represents the value of the series at time i, and N is the total number of observations [40]. For example, the number of publications per year in Figure 2 is a discrete-time series, where each observation corresponds to the number of publications in a specific year.

A **continuous-time series** is defined as:

$$x = X(i), \quad \forall i \in [0, N] \tag{2.}$$

where $X(i)$ is a real-valued function defined for all time points i in the interval $[0, N]$. The process is observed continuously over time [40]. In practice, continuous signals, such as electrical recordings or temperature measurements, are typically discretized through sampling to facilitate analysis.

In this thesis we will focus on discrete time series, as they are the most common form of time series data encountered in practice. Building on this definition, we now turn to the field of time series data mining, which provides the methodological foundation for analyzing such sequences.

Time series data mining is a subfield of data mining that focuses on the analysis of temporally ordered data. It encompasses a wide range of tasks, including indexing, clustering, classification, prediction, summarization, anomaly detection, and segmentation [41]. The major tasks addressed by the time series data mining community include:

- Indexing: Efficiently storing time series data in databases to enable fast retrieval and analysis [42].
- Clustering: Grouping similar time series based on their patterns or features [43].
- Classification: Assigning time series to predefined categories based on their characteristics [3].
- Prediction: Forecasting future values based on historical data [44].
- Summarization: Reducing the dimensionality of time series while preserving essential information [45].
- Anomaly detection: Identifying unusual patterns or outliers in time series data [46].
- Segmentation: Dividing time series into meaningful segments or subsequences for further analysis [47].

The primary goal is to extract meaningful patterns and insights from time series data to support better decision-making and a deeper understanding of the underlying processes.

This thesis focuses on three key tasks relevant to the study: classification, clustering, and anomaly detection (summarised in Table 1). To illustrate these tasks in a time series context, consider the following examples:

- In a classification scenario (a supervised learning task), a model could be trained to assign incoming MEP signals to their corresponding muscles based on patterns learned from labeled MEP data [34].
- In a clustering scenario (an unsupervised learning task), a collection of unlabeled time series from different sources may be analyzed to identify groups exhibiting similar patterns, such as those from the UCR Archive [3], and automatically group them using a time series distance and an agglomerative clustering technique. This approach can uncover meaningful clusters that correspond to different classes based on the similarity of their temporal patterns [2].
- In an anomaly detection context (an unsupervised learning task), an algorithm could continuously monitor credit card transactions and raise an alert if a purchase deviates significantly from a customer's usual spending behavior, indicating a potential case of fraud. Such deviations represent abnormal patterns that are not consistent with the expected profile.

Task	Labeled Data?	Goal	Example
Classification	Yes (supervised)	Assign to predefined categories	Classify an ECG series as normal vs. arrhythmia
Clustering	No (unsupervised)	Group similar series by similarity	Group unlabeled sensor signals by activity type
Anomaly Detection	No (often semi- or unsupervised)	Identify unusual or deviant series (outliers)	Detect an unexpected spike in network traffic as a security alert

Table 1: This table provides a summary of the key differences between classification, clustering, and anomaly detection in the time series domain.

2.1.1 A Real-World Example

To illustrate the concept, consider the number of total citations collected by papers with 'Time Series' in their title, from 1976 to 2024. This kind of data can be expressed as a discrete time series indexed by year, where each observation represents the number of citations in that year:

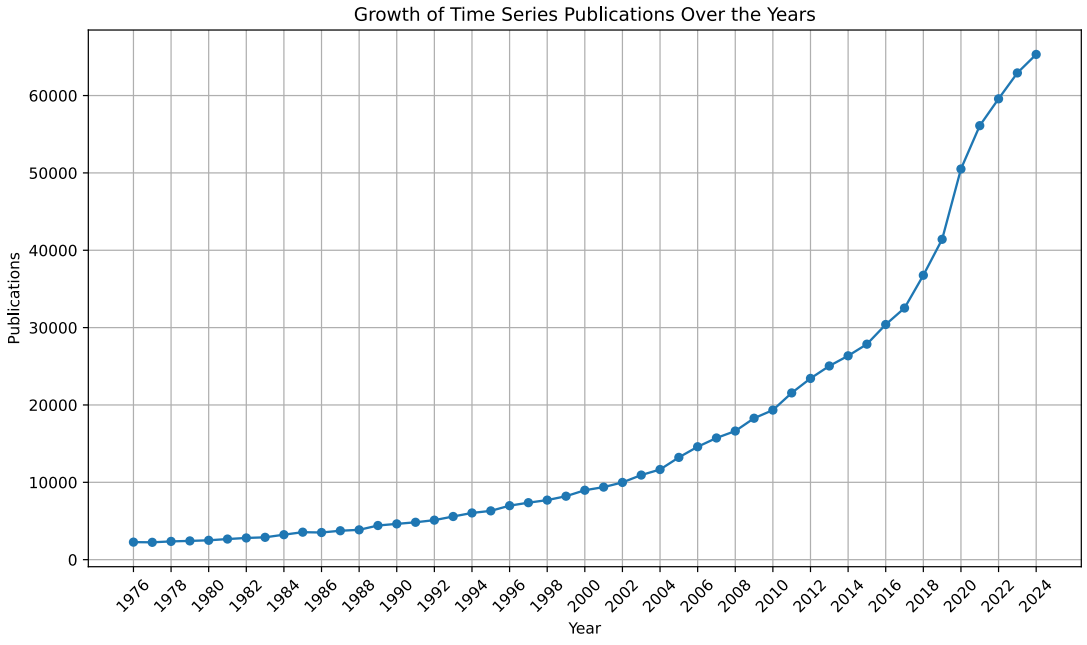

Figure 2: Number of publications per year for papers that include 'Time Series' in the title, retrieved using dimension.ai.

This type of data is useful for understanding the growing interest in the field of time series analysis. The plot in Figure 2 shows a clear upward trend, indicating that time series has become an increasingly important topic in research.

2.2 Techniques for Time Series Mining: Distances and Distance-Based Methods

A central aspect of time series mining is the ability to assess similarity between sequences. Many core tasks, such as classification, clustering, summarization, and anomaly detection, either explicitly or implicitly rely on distance measures to compare time series. Because of this, understanding distance-based approaches is essential, and time series similarity will be examined in detail.

Distance-based methods represent some of the most fundamental and widely applied techniques in time series analysis. According to the taxonomy proposed in Abanda et al. [12], these approaches can be broadly categorized into three subgroups:

- **Direct Use of Distances**: The most straightforward application of time series distances is to use them directly in learning or analysis tasks. For example, a k-nearest neighbor (k-NN) approach compares time series instances using the chosen distance measure, relying on the distances themselves as the primary mechanism for inference. Beyond classification, distances can also drive clustering [48], retrieval [49], anomaly detection [50], and other tasks where similarity between time series is central. A more detailed review of distance measures is provided in the following sections.
- **Distance-Based Feature Transformation**: Distances can also be used to embed time series into a vector space. This is done by transforming each time series into a set of features derived from distances to other series or to specific local patterns. These include:
  - Global distance embeddings: Each time series is embedded into a vector space where its coordinates are given by distances to a predefined subset of reference series. This allows the use of standard machine learning algorithms on the resulting embedding while preserving global dissimilarities among the series (also known as dissimilarity-based representation).
  - Local pattern-based distances: These methods identify and compare characteristic subsequences (patterns, motifs, or shapelets) within the time series. The similarity between two series is then defined in terms of how well their local patterns match, making them robust to distortions, misalignments, or irrelevant regions.
  - Embedding into a latent space using distance-preserving mappings: Distance information between time series is used to learn a new feature space (e.g., via multidimensional scaling, t-SNE, or auto encoders) where distances in the latent

space approximate the original time series distances. This provides a compact, often lower-dimensional representation that facilitates visualization, clustering, or downstream predictive tasks.

- **Distance Kernels**: Distances can also be used to construct kernel matrices, which serve as input to kernel-based learning algorithms such as Support Vector Machines (SVMs) [51] or kernel Principal Component Analysis (kPCA) [52]. The key idea is to transform a distance measure $d(x, y)$ into a similarity measure $k(x, y)$ that captures the relationship between pairs of time series. Depending on the construction, these kernels can be:
  - Indefinite: These are not guaranteed to be positive semi-definite (PSD). While they can still be used in some learning algorithms, indefinite kernels may lead to numerical instabilities or violate theoretical guarantees of methods that assume PSD kernels. Examples include certain transformations of Dynamic Time Warping or other elastic distances.
  - Positive semi-definite (PSD): These satisfy the mathematical condition that any kernel matrix K derived from a set of time series is PSD, meaning all eigenvalues are non-negative. PSD kernels guarantee convergence and correctness in standard kernel-based algorithms. They can be obtained, for instance, by applying specific transformations to distances (e.g., Gaussian RBF kernel) or by designing kernels directly from time series features.

These three categories illustrate the diverse ways in which distances can be leveraged for time series mining, ranging from direct similarity-based inference to more elaborate transformations and kernel methods. Regardless of the strategy, the effectiveness of these approaches ultimately depends on the choice of the underlying distance measure. A wide variety of distances have been proposed in the literature, each with different assumptions, invariances, and computational trade-offs.

In the next section, we provide a detailed review of these distance measures, starting with the simplest point-wise comparison using Euclidean distance, followed by Dynamic Time Warping (DTW) and its variants (DDTW, WDTW, WDDTW, ADTW). We then discuss more specialized approaches, including Move-Split-Merge (MSM), Time Warp Edit (TWE), shape-based distances, and finally MPDist, which provides a recent and flexible similarity measure suitable for variable-length time series.

2.2.1 Euclidean Distance

Consider the task of measuring the distance between two univariate time series of equal length, denoted as $x = \{x_1, x_2, ..., x_N\}$ and $y = \{y_1, y_2, ..., y_N\}$. The Euclidean distance, denoted as d_{ed}, corresponds to the L2 norm (the square root of the sum of squared differences) between the two series:

$$d_{ed}(x, y) = \sqrt{\sum_{i=0}^{N} (x_i - y_i)^2}. \tag{3.}$$

The Euclidean distance d_{ed} is one of the most fundamental distance functions, generally used to compare vectors in a multidimensional space. When time series are of equal length, they can be directly treated as vectors, making Euclidean distance a natural and widely adopted baseline measure in time series analysis. However, it does not account for temporal distortions such as local shifts or varying speeds within the sequences, and is therefore limited in capturing structural similarities. For this reason, elastic distance measures, described in the following sections, have been shown to substantially improve performance in tasks such as k-nearest neighbor classification compared to d_{ed} [53].

2.2.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is the most extensively studied and widely applied elastic distance measure [54]. Unlike Euclidean distance, DTW can handle time series of different lengths, since it allows flexible alignment between their indices. From this point forward, we will consider distance measures that can be applied to sequences of unequal length; however, for the sake of clarity and consistency in notation, we will assume throughout the explanations that the two time series under comparison have the same length. It addresses distortions along the time axis by realigning (or warping) the time indices of the series to achieve the best possible match.

Let $M(x, y)$, denote the $N \times N$ point-wise distance matrix between the series x and y, where each element is defined as $M_{i,j} = \left(x_i - y_j\right)^2$. A warping path $P = [(e_1, f_1), (e_2, f_2), ..., (e_s, f_s)]$ is a sequence of index pairs that describes a traversal through the matrix M (see Figure 4).

A valid warping path must start at position $(1, 1)$ and end at (N, N), while satisfying the monotonicity and continuity constraints: $0 \leq e_{i+1} - e_i \leq 1$ and $0 \leq f_{i+1} - f_i \leq 1$ for all $1 < i < N$. The DTW distance is defined as the minimal cumulative distance across all possible warping paths through M. The total distance along a given path P of length s is computed as:

$$D_{P(x,y,M)} = \sum_{i=1}^{s} M_{e_i, f_i}. \tag{4.}$$

If P represents the set of all admissible warping paths, the optimal DTW path P^* is the one that minimizes this total distance. Consequently, the DTW distance between the series is given by:

$$d_{dtw}(x, y) = D_{P^*}(x, y, M). \tag{5.}$$

The optimal warping path P^* can be computed exactly using a dynamic programming approach, as detailed in Algorithm 1. However, this process can be computationally expensive. To mitigate this, it is common practice to restrict the

allowable warping using predefined constraints. The two most widely used bounding strategies are illustrated in Figure 3: the Sakoe-Chiba band [55] and the Itakura parallelogram [56]. In Figure 3, each cell represents an element of the matrix M, and the darker cells indicate the regions where warping is permitted. The Sakoe-Chiba band enforces a fixed-width window along the diagonal of M, while the Itakura parallelogram allows more flexibility in the center of the series and less at the beginning and end. The dynamic programming procedure presented in Algorithm 1 assumes the use of a Sakoe-Chiba band.

DTW(x, y, w, M):

1 **let** C = zeros$(N+1, N+1)$ // matrix initialised to zero
2 **for** $i \leftarrow 1$ **to** N:
3 **for** $j \leftarrow 1$ **to** N:
4 **if** $|i-j| < w \cdot N$:
5 $C[i,j] = M[i,j] + \min(C[i-1, j-1], C[i-1, j], C[i, j-1])$
6 **return** $C[N, N]$

Algorithm 1: x and y represents time series of length N, w the window proportion and M the point-wise distance matrix.

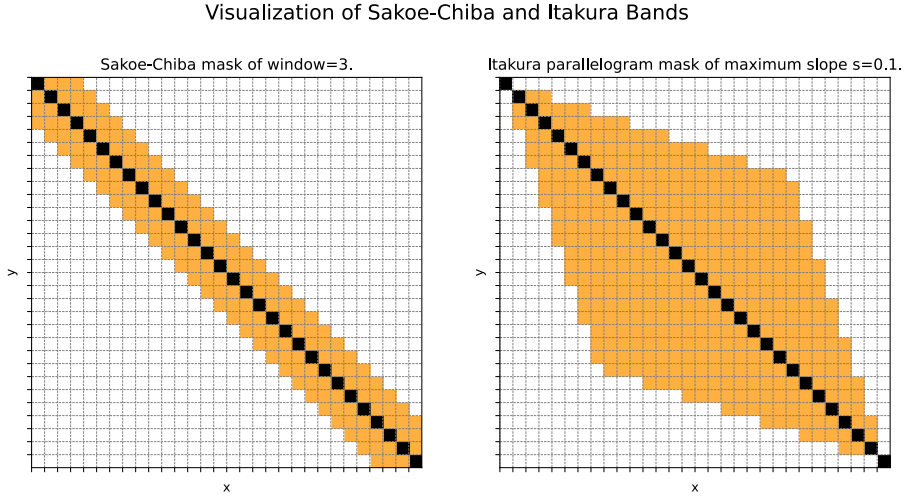

Figure 3: Sakoe-Chiba (left) and Itakura bands (right), which are used to constrain the warping path.

The DTW distance with Sakoe-Chiba window w can be expressed as:

$$d_{dtw}(x, y) = D_{P(x,y,M)} = DTW(x, y, w, M). \quad 6.$$

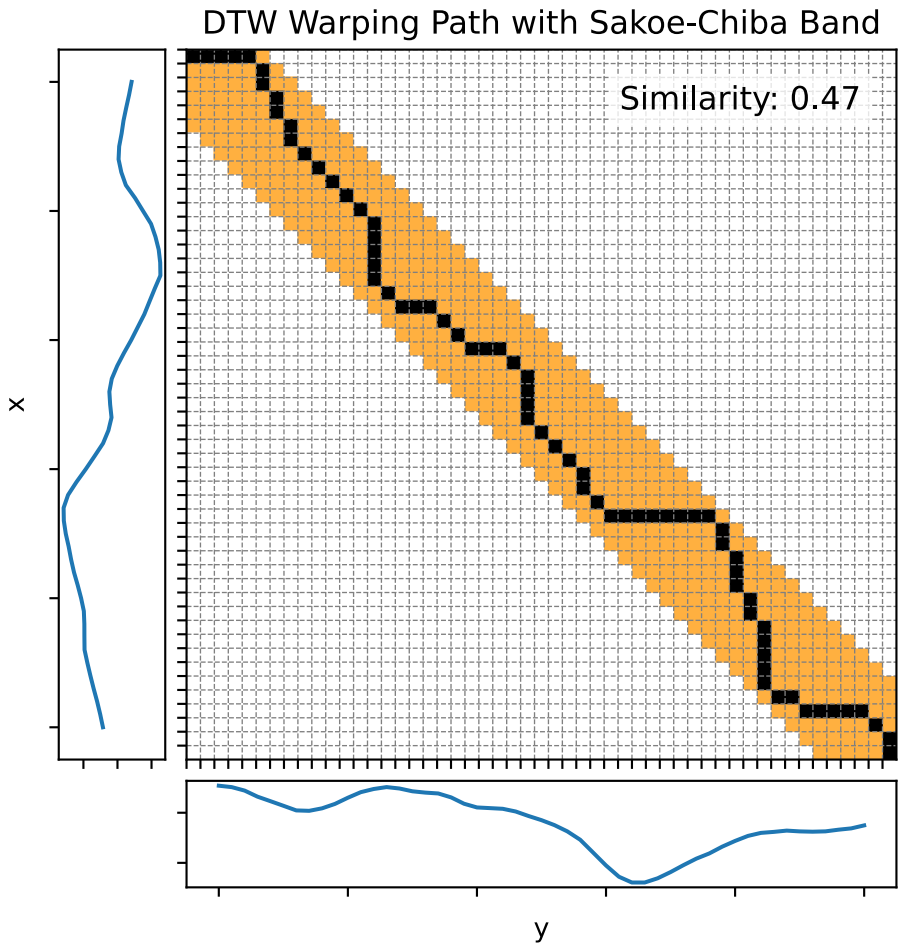

Figure 4: Dynamic time warping (DTW) path with Sakoe-Chiba band. The DTW path is shown as a black line in the matrix, where orange indicates allowed warping and white indicates disallowed warping.

2.2.3 Derivative Dynamic Time Warping

Keogh et al. introduced a variation of DTW known as Derivative Dynamic Time Warping (DDTW) [14]. This method begins by transforming the original time series into a derivative series.

For a given time series x, the derivative series is defined as $x' = \{x'_2, x'_3, ..., x'_{N-1}\}$, where each element x'_i represents the average of the local slopes before and after point i. Specifically, the derivative at position i is calculated as:

$$x'_i = \left(\frac{(x_i - x_{i-1}) + \frac{x_{i+1} - x_{i-1}}{2}}{2} \right), \tag{7.}$$

for all $1 < i < N$.

The key motivation behind DDTW is to compensate for DTW's sensitivity to absolute values. Standard DTW aligns points directly based on their amplitudes, which can lead to undesirable matches when two time series have similar overall shapes but differ in scale or baseline offset. By instead aligning the derivatives (i.e., local slopes), DDTW emphasizes the underlying trends and structural similarities of

the series rather than their raw values. This makes it particularly effective when the shape of the time series carries more information than its absolute magnitudes.

The DDTW distance is then simply the DTW distance computed between the derivative series of x and y:

$$d_{ddtw}(x, y) = d_{dtw}(x', y'). \tag{8.}$$

2.2.4 Weighted Dynamic Time Warping

Jeong et al. proposed a weighted variant of DTW, referred to as Weighted Dynamic Time Warping (WDTW) [57]. This approach introduces a multiplicative weight penalty that depends on the warping distance between points in the warping path. Unlike hard constraints imposed by warping windows, WDTW offers a smooth continuous penalty that discourages excessive warping. When constructing the distance matrix M, a weight penalty $w(|i-j|)$ is applied based on the absolute index difference, such that:

$$M^w_{i,j} = w(|i-j|) \cdot \left(x_i - y_j\right)^2. \tag{9.}$$

The motivation behind WDTW is to address DTW's tendency toward over-warping, where the algorithm may create unrealistic alignments by excessively stretching one sequence to match another. While classical warping window constraints (such as the Sakoe–Chiba band) enforce a hard limit on how much warping is allowed, WDTW instead applies a soft, gradually increasing penalty as the temporal displacement between aligned points grows. This encourages alignments that are both accurate and physically plausible, while still retaining flexibility compared to rigid windowing.

WDTW uses a logistic weighting function, where the penalty for a warping distance of i is defined as:

$$w(i) = \frac{w_{max}}{1 + e^{-g \cdot (i - \frac{N}{2})}}. \tag{10.}$$

where w_{max} is the maximum weight (typically set to 1), N is the length of the time series, and g is a parameter that controls the severity of the penalty for large warping distances. A higher value of g results in stronger penalties for deviations from the diagonal.

It is important to note that WDTW does not benefit from the computational advantages of constrained warping windows, as it does not restrict the search space. The WDTW distance is then defined as:

$$d_{wdtw}(x, y) = D_{P^*}(x, y, M^w) = DTW(x, y, M^w) \tag{11.}$$

.

Additionally, the derivative-weighted distance (WDDTW) can be defined by applying WDTW to the derivative series:

$$d_{wddtw}(x, y) = d_{wdtw}(x', y') \tag{12.}$$

.

2.2.5 Amercing Dynamic Time Warping (ADTW)

A very recent extension of DTW has been developed by Herrmann et al. [15]. Amercing Dynamic Time Warping (ADTW) introduces a warping penalty into the DTW formulation to explicitly control the amount of warping. The warping penalty is a constant value $\omega > 0$ that is added to every non-diagonal step in the warping path, thereby discouraging excessive warping.

The formulation of ADTW modifies the standard DTW (line 5 in Algorithm 1) to include the penalty for vertical and horizontal steps:

$$C[i, j] = M[i, j] + \min(C[i-1, j-1], C[i-1, j] + \omega, C[i, j-1] + \omega) \tag{13.}$$

This modification introduces a tunable bias against warping by favoring diagonal steps, thereby controlling the flexibility of alignment. For large values of ω, the optimal path approaches the diagonal (i.e., behaves more like Euclidean distance), whereas smaller values allow more warping.

2.2.6 Longest Common Subsequence

DTW is typically described as a method for finding an alignment between two time series by warping points onto each other to construct a path. However, an alternative perspective interprets this process as the construction of a common series between the two input sequences. In this view, warping operations can be understood as inserting gaps into one series or deleting elements from another. This interpretation originates from sequence alignment techniques used for discrete data, such as strings or DNA sequences.

The Longest Common Subsequence (LCSS) distance for time series is adapted from algorithms that identify the longest common subsequence between two discrete sequences through edit operations. For example, consider the discrete sequences *abaacb* and *bcacab*; their LCSS is *baab*. Unlike DTW, the LCSS does not produce a continuous path from (1, 1) to (N, N). Instead, it defines a sequence of edit operations—each with an associated cost—that transform one series into the common subsequence. For instance, editing *abaacb* into *baab* requires two deletions.

Within the DTW dynamic programming framework, the three possible steps in line 5 of Algorithm 1 can be reinterpreted: moving from $C_{i-1,j}$ corresponds to a deletion in series y, moving from $C_{i,j-1}$ represents a deletion in series x, and moving from $C_{i-1,j-1}$ is considered a match. From this perspective, the warping operations are essentially gap insertions or deletions in either series.

In the case of discrete sequences, a match in the common subsequence requires identical elements (e.g., matching letters). However, exact matches are unlikely when dealing with real-valued time series. To solve this, the discrete LCSS algorithm can be extended to continuous-valued series by introducing a distance threshold ε,

which defines the maximum permissible difference between two values for them to be considered a match.

The length of the LCSS between two time series x and y can be computed using Algorithm 2. When two elements are considered similar according to the threshold (line 4), the length of the current LCSS is incremented by one. Otherwise, the previously computed LCSS length is propagated forward.

```
LCSS(x, y, ε):
1  let L = zeros(N + 1, N + 1)                    // matrix initialised to zero
2  for i ← 1 to N:
3      for j ← 1 to N:
4          if |x_i − y_j| < ε:
5              L[i, j] = L[i − 1, j − 1] + 1
6          else:
7              L[i, j] = max(L[i − 1, j], L[i, j − 1])
8  return L[N, N]
```

Algorithm 2: x and y represents time series of length N, ε the equality threshold.

The LCSS distance between x and y is

$$d_{LCSS}(x, y) = 1 - \frac{LCSS(x, y)}{N}. \tag{14.}$$

Figure 5 shows the alignment cost between our two example series. Unlike DTW, which forces a complete alignment path from start to end, LCSS connects only matching points. As a result, the warping path can contain gaps where no match occurs. These gaps correspond to positions where the condition $|x_i - b_j| < \varepsilon$ is not satisfied.

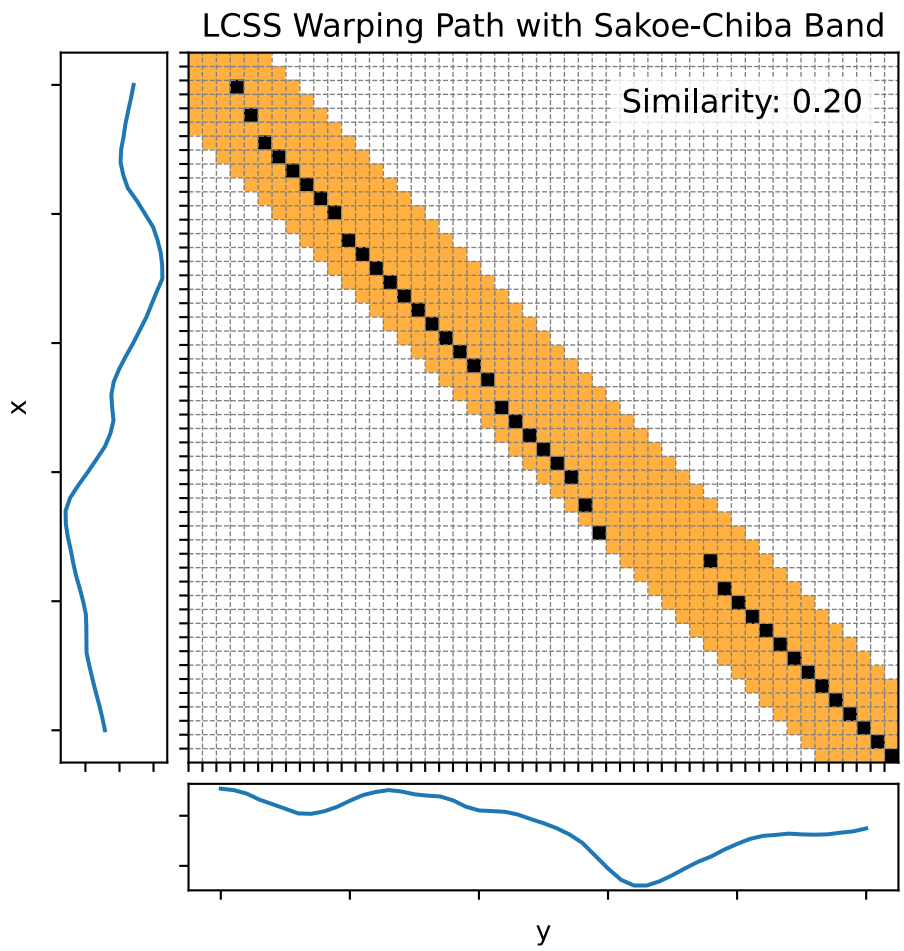

Figure 5: Longest common subsequence (LCSS) path with Sakoe-Chiba band. The LCSS path is shown as a black line in the matrix, where orange indicates allowed warping and white indicates disallowed warping. The time series x and y are plotted next to the matrix, with x on the left and y above. The ε is set to 0.1. The window size for the band is set to 5.

2.2.7 Edit Distance on Real Sequences (EDR)

The LCSS approach was further adapted by Chen et al., who introduced the Edit Distance on Real sequences (EDR) [58]. Similar to LCSS, EDR employs a distance threshold to determine when two elements from different time series are considered a match. However, unlike LCSS which focuses on counting matches to find the longest subsequence, EDR applies a constant penalty for non-matching elements by introducing deletions or gaps during the alignment process.

```
EDR(x, y, ε):
let E = zeros(N + 1, N + 1)                    // matrix initialised to zero
for i ← 1 to N:
    for j ← 1 to N:
        if |x_i − y_j| < ε:
            c = 0
        else:
            c = 1
        match = E[i − 1, j − 1] + c
        insert = E[i − 1, j] + 1
        delete = E[i, j − 1] + 1
        E[i, j] = min(match, insert, delete)
        return E[N, N]
```

Algorithm 3: x and y represents time series of length N, ε the equality threshold.

Given a distance threshold epsilon, the point-wise EDR distance between elements of series x and y is computed as described in Algorithm 3. The total EDR distance between the two series is then defined as:

$$d_{EDR}(x, y) = EDR(x, y, \varepsilon) \tag{15.}$$

Throughout the alignment process, elastic distance measures generally operate by selecting one of three possible costs: diagonal, horizontal, or vertical steps. When viewed as forming a subsequence from series a, these steps can be interpreted as: a match (diagonal), a deletion (horizontal), or an insertion (vertical). Although an insertion in series a can equivalently be described as a deletion from series b, the match/delete/insert terminology is retained for consistency and clarity.

It is important to note that EDR does not satisfy the triangle inequality because it relaxes the equality condition by allowing two elements to be considered equal if their distance is less than or equal to epsilon. Conceptually, EDR closely resembles LCSS, but instead of counting the number of matches, EDR directly computes a distance measure. This eliminates the need for the subtraction step present in Equation 14.

2.2.8 Edit Distance with Real Penalty (ERP)

An alternative to EDR was proposed by Chen and Ng [16], who introduced Edit Distance with Real Penalty (ERP). Unlike LCSS, which identifies subsequences that may contain gaps between elements (for example, as illustrated by the gap in Figure 5), ERP explicitly assigns a penalty for such gaps. This penalty is calculated based on the distance to a constant reference value g.

ERP uses the point-wise distance function $d(x, y) = \sqrt{(x - y)^2}$, which differs from the squared distance $d(x, y) = (x - y)^2$ typically used in the distance matrix M for

DTW. The cost matrix E in ERP is conceptually closer to the DTW formulation than to LCSS: it models a path alignment based on edit operations rather than warping (see Figure 6).

The ERP distance between two time series is computed according to the procedure described in Algorithm 4. The boundary conditions of the cost matrix E are initialized to a large constant value (lines 5 and 7). At each step, the alignment cost is determined by one of three possible operations: the cost of matching $E_{i-1,j-1} + d(x_i, y_j)$ where $d(x_i, y_j)$ is the distance between the two points, or the cost of inserting or deleting an element on either axis, which is computed as $E_{i-1,j} + d(x_i, g)$ or $E_{i,j-1} + d(g, y_j)$.

The ERP distance between series x and y is then defined as:

$$d_{ERP}(x, y) = ERP(x, y, g, d) \tag{16.}$$

Unlike EDR, ERP satisfies the triangle inequality and constitutes a proper metric.

```
ERP(x, y, g, d):
 1  let E = zeros(N + 1, N + 1)                    // matrix initialised to zero
 2  for i ← 1 to N:
 3      for j ← 1 to N:
 4          if i = 0:
 5              E[i, j] = Σ_{k=1}^{N} d(y_k, g)
 6          else if j = 0:
 7              E[i, j] = Σ_{k=1}^{N} d(x_k, g)
 8          else:
 9              match = E[i − 1, j − 1] +
                d(x_i, y_j)
10              insert = E[i − 1, j] + d(x_i, g)
                delete = E[i, j − 1] + d(g, y_j)
11              E[i, j] =
                min(match, insert, delete)
12      return E[N, N]
```

Algorithm 4: x and y represents time series of length N, g the penalty value and d the point-wise distance function.

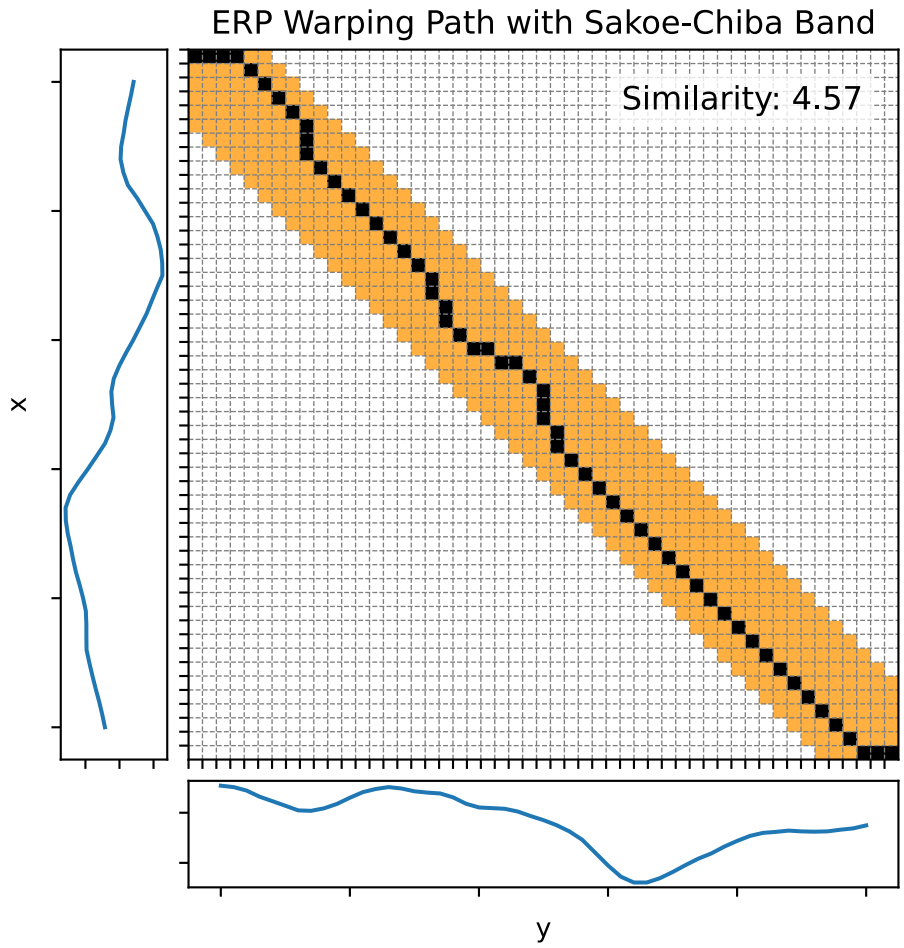

Figure 6: Edit distance with real penalty (ERP) path with Sakoe-Chiba band. The ERP path is shown as a black line in the matrix, where orange indicates allowed warping and white indicates disallowed warping. The time series x and y are plotted next to the matrix, with x on the left and y above. The g is set to 0.1. The window size for the band is set to 5.

2.2.9 Move-Split-Merge (MSM)

Move-Split-Merge (MSM) is a distance measure conceptually related to ERP but designed to address its limitations [59]. The primary motivation behind MSM is that, in ERP, the insertion and deletion costs are based on the distance to a fixed constant g, which leads to a preference for inserting or deleting values that are close to g.

The core difference between MSM and ERP lies in how insertion and deletion operations are handled, as shown in Algorithm 5, particularly in lines 10 and 11. In MSM, the move operation employs the absolute difference instead of the squared distance used for matches in ERP. Furthermore, the insertion cost in ERP, represented by $d(x_i, y_j)$, is replaced by the split operation $C(x_i, x_{i-1}, y_j, c)$, where C is the cost function defined as:

$$cost(x, y, z, c) = \begin{cases} c \text{ if } y \leq x \leq z \vee y \geq x \geq z \\ c + \min(|x - y|, |x - z|) \text{ otherwise} \end{cases} \quad 17.$$

When the value being inserted y_j lies between the two adjacent values in series x—specifically x_i and x_{i-1}—the cost is a constant c. If y_j falls outside this range, the cost is c plus the smallest deviation from either x_i or x_{i-1} to y_j.

Similarly, the deletion cost in ERP, typically computed as $d\big(g, x_j\big)$, is replaced by the merge operation $C\big(y_j, y_{j-1}, x_i\big)$ in MSM, which applies the same logic to the second series.

This design ensures that the cost of splitting and merging elements in MSM depends on the relationship between the element being inserted or deleted and its neighboring values, rather than uniformly applying the same penalty as in ERP. The MSM distance between two series x and y is defined as:

$$d_MSM(x, y) = MSM(x, y, c) \tag{18.}$$

An example of MSM alignment is illustrated in Figure 7. Importantly, MSM satisfies the triangle inequality hence is a proper metric.

MSM(x, y, c, d):

```
1                                                  // matrix initialised to zero
2  let D = zeros(N + 1, N + 1)
3  D[1, 1] = d(a_1, b_1)
4  for i ← 2 to N:
5      D[i, 1] = D[i − 1, 1] + cost(x_i, x_{i−1}, y_1, c)
6  for i ← 2 to N:
7      D[1, i] = D[1, i − 1] + cost(y_i, x_1, y_{i−1}, c)
8  for i ← 2 to N:
9      for j ← 2 to N:
10         match = D[i − 1, j − 1] + d(x_i, y_j)
11         insert = D[i − 1, j] +
           cost(x_i, x_{i−1}, y_j, c)
12         delete = D[i, j − 1] +
           cost(y_j, y_{j−1}, x_i, c)
13         D[i, j] = min(match, insert, delete)
14 return D[N, N]
```

Algorithm 5: x and y represents time series of length N, c the minimum cost and d the point-wise distance function.

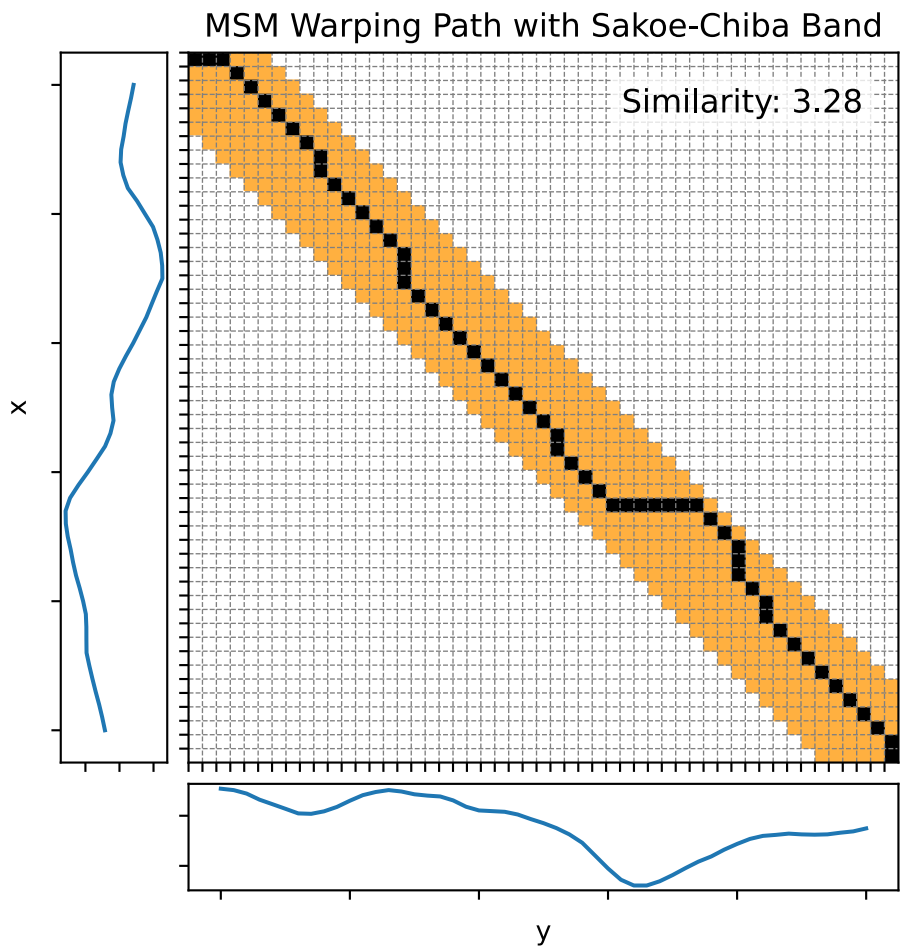

Figure 7: Move-Split-Merge (MSM) path with Sakoe-Chiba band. The MSM path is shown as a black line in the matrix, where orange indicates allowed warping and white indicates disallowed warping. The time series x and y are plotted next to the matrix, with x on the left and y above. The c is set to 0.1. The window size for the band is set to 5.

2.2.10 Time Warp Edit (TWE)

The Time Warp Edit (TWE) distance, introduced by Marteau [60], is an elastic distance measure that integrates features from both warping-based and edit-based approaches. The TWE algorithm is described in Algorithm 6. A key characteristic of TWE is its ability to balance alignment flexibility with a warping penalty, known as stiffness, which is controlled by the parameter ν.

The stiffness parameter imposes a multiplicative penalty on the distance between matched points, similar to the weighting mechanism used in WDTW. When $\nu = 0$, no warping penalty is applied, allowing completely flexible alignment. However, the stiffness penalty is applied only when considering the match operation (line 11 in Algorithm 6). For deletion and insertion operations (lines 12 and 13), TWE applies both a constant stiffness penalty (ν) and an additional edit penalty (λ), reflecting the cost of warping consecutive points within the same series.

An example of a TWE alignment is provided in Figure 8. This distance measure offers a controlled trade-off between the flexibility of warping and the structural penalties of editing, making it a versatile choice for time series comparison.

TWE(x, y, ν, λ, d):

let D = zeros$(N+1, N+1)$ // matrix initialised to zero
for $i \leftarrow 1$ **to** N:
    $D[1,0] = +\inf$
    $D[0,1] = +\inf$
for $i \leftarrow 1$ **to** N:
    for $j \leftarrow 1$ **to** N:
        $\text{match} = D[i-1, j-1] + d(x_i, y_j) + 2\nu(|i-j|)$
        $\text{insert} = D[i, j-1] + d(y_j, y_{j-1}) + \lambda + \nu$
        $\text{delete} = D[i-1, j] + d(x_i, x_{i-1}) + \lambda + \nu$
        $D[i,j] = \min(\text{match}, \text{insert}, \text{delete})$
return $D[N, N]$

Algorithm 6: x and y represents time series of length N, ν the warping penalty factor, λ the edit cost, and d the point-wise distance function

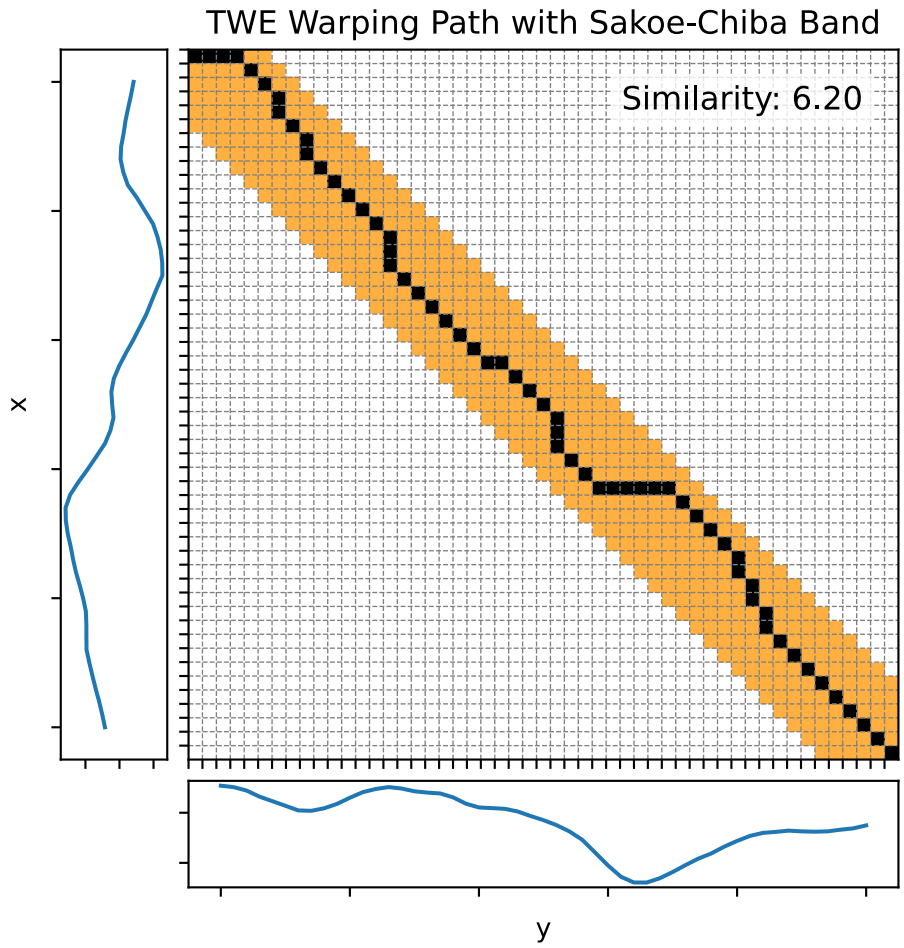

Figure 8: Time Warping Edit Distance (TWE) path with Sakoe-Chiba band. The TWE path is shown as a black line in the matrix, where orange indicates allowed warping and white indicates disallowed warping. The time series x and y are plotted next to the matrix, with x on the left and y above. The ν is set to 0.01 and λ is set to 0.1. The window size for the band is set to 5.

2.2.11 Shape-based Distance (SBD)

Shape-Based Distance (SBD), introduced by Paparrizos et al. [17], is a similarity measure based on normalized cross-correlation (CC) that is invariant to both phase (shifting) and amplitude (scaling). Unlike DTW variants that seek optimal alignments through warping, SBD aligns time series by circularly shifting one series relative to the other and measuring their similarity using the normalized cross-correlation.

For two time series x and y, the first step in computing SBD is z-normalization of each series:

$$x' = \frac{x - \mu_x}{\sigma_x}, \quad y' = \frac{y - \mu_y}{\sigma_y} \tag{19.}$$

where μ and σ are the mean and standard deviation, respectively. This ensures that the comparison is invariant to amplitude differences.

Next, the cross-correlation $CC_{w(x',y')}$ is computed at different circular shifts w, and normalized by the geometric mean of the autocorrelations of the two sequences. The maximum of this normalized value over all possible shifts gives the similarity:

$$SBD(x,y) = 1 - max_w \left(\frac{CC_{w(x,y)}}{\sqrt{(x \cdot x) * (y \cdot y)}} \right) \quad 20.$$

Here, $CC_{w(x',y')}$ denotes the cross-correlation between x' and y' at shift w, and the denominator normalizes the result to lie in the range $[0, 2]$. The subtraction from 1 ensures that SBD behaves as a distance, where smaller values indicate greater similarity.

SBD is particularly useful in time series clustering and classification tasks where global shape similarity is more important than precise temporal alignment. It is also computationally efficient due to the use of the Fast Fourier Transform (FFT) for computing cross-correlation.

2.2.12 Matrix Profile Distance (MPdist)

Matrix Profile Distance (MPdist), is a robust distance measure for time series that compares the similarity of subsequences rather than the global shape or alignment of entire series [61].

The core idea of MPdist is to quantify similarity between two time series by comparing all subsequences of a fixed length L. Let x and y be two time series of length N. First, their All-Subsequences Sets X and Y are extracted using a sliding window of size L:

$$X = \{X_{1:L}, X_{2:L}, ..., X_{T-L+1:L}\}, Y = \{Y_{1:L}, Y_{2:L}, ..., Y_{T-L+1:L}\}. \quad 21.$$

Each subsequence in X is matched to its nearest neighbor in Y using Euclidean distance, and vice versa. This forms the Join Matrix Profile (P_{XY}), a vector of distances for all such best matches in both directions.

Rather than using the minimum or maximum distance, MPdist selects the k^{th} smallest value in this vector to report as the final distance. The default choice is $k = 5\%$ of the length of P_{XY}:

$$MPdist(X,Y) = P_{XY}^k \quad 22.$$

where P_{XY}^k is the k^{th} smallest value in the sorted Join Matrix Profile.

2.3 Techniques for Time Series Mining: Other Methods

A wide range of data mining techniques have been developed and applied to time series data. These techniques aim to extract meaningful patterns, features, or representations for tasks such as classification, clustering, anomaly detection, and forecasting. According to the taxonomy proposed by Middlehurst et al. the main families of approaches include feature-based, interval-based, shapelet-based, dictionary-based, convolution-based, deep learning-based, and hybrid methods [3].

2.3.1 Feature-Based Methods

Feature-based methods approach time series analysis by transforming each sequence into a vector of descriptive attributes or features. By representing time series in this vectorial form, standard data mining and machine learning algorithms can be applied directly, whether for classification, clustering, regression, or other tasks. The general workflow consists of a feature extraction step, where relevant characteristics of the series are computed, followed by the application of vector-based learning or analysis methods, as illustrated in Figure 9. In this way, feature-based approaches aim to solve the data mining problem in a transformed space, leveraging the computational and algorithmic tools available for traditional vector data.

In this thesis, we will introduce only the subset of features that are directly used in our experiments. It should be noted, however, that a wide variety of additional features exist in the literature, covering statistical, spectral, and temporal characteristics of time series.

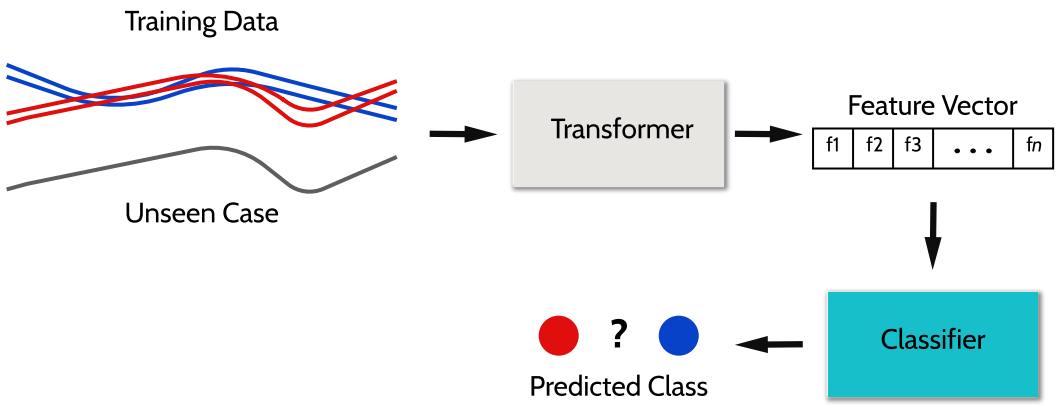

Figure 9: Visualization of a data mining pipeline involving feature extraction followed by classification. Image taken with permission of Matthew Middlehurst from [3].

Canonical Time Series Characteristics (CATCH22)

The CATCH22 framework derives a compact set of 22 features from the highly comparative time series analysis (hctsa) toolbox, which originally contained over 7700 features [1, 62]. The features included in hctsa are drawn from a wide range of algorithmic concepts. They encompass basic statistical properties of time series (e.g., location, spread, Gaussianity, and outlier characteristics), linear correlation measures (e.g., autocorrelation, power spectral features), stationarity assessments (e.g., StatAv, sliding-window metrics, prediction errors), and entropy or complexity measures (e.g., auto mutual information, Approximate Entropy, Lempel-Ziv complexity). The set also incorporates methods from nonlinear and physical time series analysis (e.g., correlation dimension, Lyapunov exponent estimates, surrogate data techniques), as well as parameters, fits, and predictive metrics from linear and nonlinear models (e.g., ARMA, Gaussian Process, and GARCH models). Additional

features include techniques based on wavelets, network-derived properties of time series, and other specialized descriptors.

The selection process was designed to identify features that are robust, interpretable, and informative across a wide range of time series datasets. It involves several key steps:

- Filtering features sensitive to mean and variance: Features that were overly influenced by simple scaling or location shifts were removed to ensure robustness across differently normalized series.
- Removing unreliable features: Any feature that failed to compute on more than 20% of the UCR datasets was discarded, ensuring that the remaining features are broadly applicable.
- Ranking by predictive power: The remaining features were evaluated for their ability to discriminate between different classes in benchmark datasets, and ranked according to predictive performance.
- Eliminating redundancy via hierarchical clustering: To reduce overlap and correlation between features, hierarchical clustering was applied. Features that were highly correlated with others were grouped together to identify clusters of similar descriptors.
- Selecting one representative feature per cluster: From each cluster, a single feature was chosen to represent the group, optimizing for a balance between classification accuracy, computational efficiency, and interpretability. This step ensured that the final set was both compact and diverse, capturing complementary aspects of time series behavior without redundancy.

The final CATCH22 feature set spans a broad range of descriptors, including statistical measures (e.g., variance, autocorrelation), correlation-based features, and entropy or complexity measures. A summary of these features is provided in Table 2.

Category	**Description**
Distribution	• Mode of z-scored distribution (5-bin histogram) • Mode of z-scored distribution (10-bin histogram)
Simple temporal statistics	• Longest period of consecutive values above the mean • Time intervals between successive extreme events above the mean • Time intervals between successive extreme events below the mean
Linear autocorrelation	• First $1/e$ crossing of autocorrelation function • First minimum of autocorrelation function

Category	Description
	• Total power in lowest fifth of frequencies in the Fourier power spectrum • Centroid of the Fourier power spectrum • Mean error from a rolling 3-sample mean forecasting
Nonlinear autocorrelation	• Time-reversibility statistic, $\langle (x_{t+1} - x_t)^3 \rangle_t$ • Automutual information, $m = 2$, $\tau = 5$ • First minimum of the automutual information function
Successive differences	• Proportion of successive differences exceeding 0.04σ [63] • Longest period of successive incremental decreases • Shannon entropy of two successive letters in equiprobable 3-letter symbolization • Change in correlation length after iterative differencing • Exponential fit to successive distances in 2-d embedding space
Fluctuation Analysis	• Proportion of slower timescale fluctuations that scales with DFA (Detrended Fluctuational Analysis) (50% sampling) • Proportion of slower timescale fluctuations that scales with linearly rescaled range fits
Others	• Trace of covariance of transition matrix between symbols in 3-letter alphabet • Periodicity measure, as defined in [64]

Table 2: Simplified descriptions of the CATCH22 features grouped by category, based on Lubba et al. [1] and following the table format proposed in Azzari et al [2].

In this thesis, we employed CATCH22 because it offers an effective balance between predictive performance and computational efficiency. The set is small enough to be computed efficiently, while still capturing diverse and informative aspects of time series. Moreover, these features have been successfully integrated with specialized ensemble methods, such as Canonical Interval Forest (explained in detail in Section), which are time series-specific random forests, making them particularly well-suited for our experimental setting.

TSFresh

TSFresh is a comprehensive feature extraction toolkit that computes nearly 800 features from time series [18]. These features cover a broad spectrum of properties, including statistical moments, autocorrelations, frequency-domain characteristics, entropy measures, and other specialized descriptors that capture temporal patterns, trends, and seasonality.

To identify the most relevant features for a given predictive task, TSFresh incorporates the FRESH algorithm for feature selection, which operates as follows:

- Multiple hypothesis testing: Each feature is independently tested for statistical relevance with respect to the target variable, using tests such as Fisher's exact test [65], the Kolmogorov-Smirnov test [66], and the Kendall rank correlation test [67]. This ensures that features are not selected purely by chance and that their association with the target is statistically significant.
- False discovery rate control: The Benjamini-Yekutieli procedure [68] is applied to adjust for multiple comparisons, controlling the expected proportion of false positives among the selected features. This step improves the robustness and reliability of the selection process.

By combining extensive feature extraction with a statistically rigorous selection mechanism, TSFresh enables automated, high-quality features extraction.

Compared to CATCH22, TSFresh emphasizes feature comprehensiveness, generating hundreds of descriptors from diverse domains and then filtering them statistically for relevance. CATCH22, in contrast, provides a fixed, curated set of 22 features that were pre-selected for general usefulness, interpretability, and efficiency. While TSFresh is well suited for exploratory studies where discovering novel predictive features is important, CATCH22 is advantageous in scenarios where computational efficiency and robustness across datasets are key considerations.

2.3.2 Interval-Based Methods

Interval-based classifiers aim to capture phase-dependent patterns by extracting fixed-length intervals from time series and computing summary statistics over them. First introduced by Deng et al. [19], these approaches reduce the influence of noise by focusing on localized regions of the series, rather than the whole signal. Typically, intervals are randomly selected and shared across all series in the dataset. Most modern methods employ ensembles of decision trees, where diversity is injected by randomizing interval selection for each base learner.

Figure 10 illustrates a scenario where interval-based features may outperform global features, particularly in the presence of localized, class-discriminative patterns.

In the next section, a detailed explanation of interval-based models will be provided, specifically on forest-based approaches.

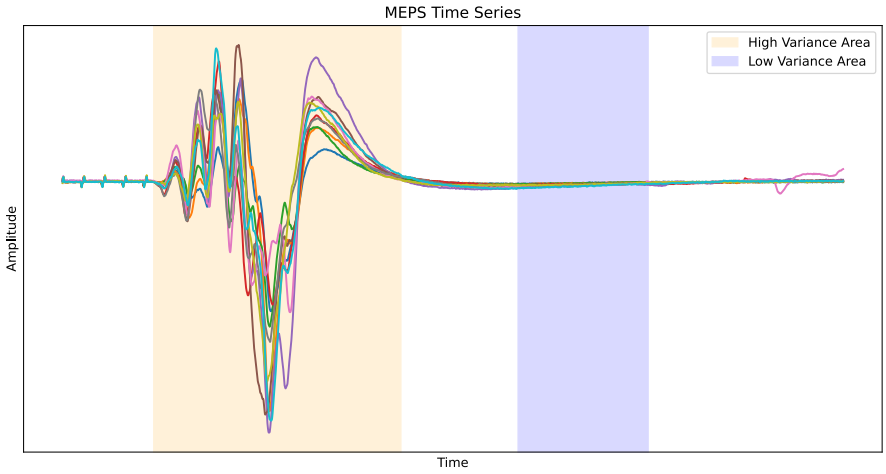

Figure 10: An example of a problem where interval-based approaches may be superior.

2.3.3 Shapelet-Based Methods

Shapelet-based approaches identify discriminative subsequences, called shapelets, within time series. A shapelet is a short, characteristic pattern that captures a key local feature of the series and is often phase-independent. These shapelets can serve as interpretable features for a variety of tasks, including classification, clustering, or anomaly detection, as they highlight portions of the series that are most informative for distinguishing patterns or detecting deviations. Given a candidate shapelet, it is slid across a time series and the z-normalized Euclidean distance is computed at each offset. The shapelet-to-series distance, $sDist$, is defined as the minimum of these distances. This process is visualized in Figure 11, where a shapelet S is matched across series A and its closest alignment is recorded.

Shapelets were introduced by Ye and Keogh [69], initially embedded in decision trees, where the best shapelet is chosen at each node to maximize information gain.

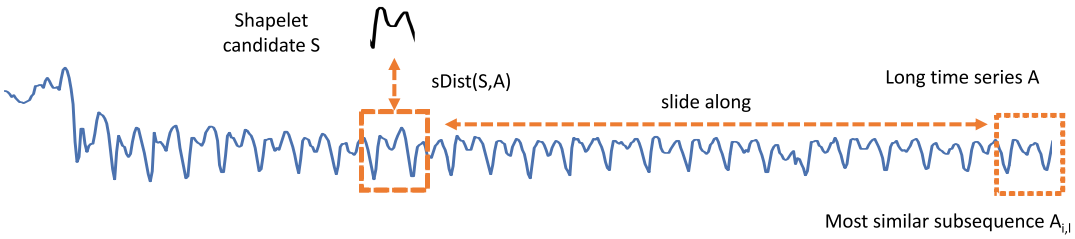

Figure 11: Visualisation of the shapelet distance operation $sDist()$ between a shapelet S and a series A, which finds the closest distances to the shapelet from all possible subseries of the same length. Image taken with permission of Matthew Middlehurst from [3].

Shapelet Transform Classifier (STC)

The Shapelet Transform Classifier (STC) introduced a two-phase pipeline: shapelet discovery followed by classification [22]. Initially, shapelets are extracted via full

enumeration from all training series and filtered by information gain. Each time series is then transformed into a vector of $sDist()$ values to the selected shapelets. A classifier is then trained on this transformed dataset.

The original version of STC used HESCA (later renamed CAWPE [70]), a weighted heterogeneous ensemble of eight classifiers, but suffered from scalability limitations. To address this, later improvements [71, 72] included:

- Randomized shapelet sampling (defaulting to 10,000 candidates), replacing full enumeration.
- Binary labeling of shapelets using one-hot encoding for class origin, accelerating split evaluation via early abandon.
- Replacement of HESCA with a single Rotation Forest classifier [73], resulting in a more efficient pipeline.

Generalised Random Shapelet Forest (RSF)

RSF enhances computational efficiency through ensembling and randomization [74]. Each tree in the forest selects r random shapelets of variable lengths, evaluates them using $sDist()$ and information gain, and chooses the best for splitting. Trees are grown recursively and predictions are made by majority vote. RSF combines the accuracy of shapelets with the scalability of tree ensembles.

Multiple Representation Sequence Learner (MrSEQL)

MrSEQL avoids explicit distance computation by discretising time series subsequences into symbolic words using Symbol Aggregation Approximation (SAX, time-domain [21]) and Symbol Fourier Approximation(SFA [23], frequency domain). Discriminative words are selected using the Sequence Learner (SEQL) [75], and a logistic regression model is trained on their presence/absence [76].

MrSQM [77] extends this with two feature selection strategies:

- Supervised selection via chi-squared tests.
- Unsupervised sampling to avoid correlated or redundant substrings.

A trie structure is used to rank frequent substrings. Window sizes are selected on an exponential scale, and the number of representations (SAX or SFA) is adjusted based on series length.

Random Dilated Shapelet Transform (RDST)

RDST is a scalable shapelet algorithm inspired by convolutional models [78]. It randomly samples thousands of shapelets from the training data and applies dilated matching, where shapelets are compared to spaced time points. This reduces sensitivity to local alignment.

For each shapelet, RDST extracts three features:

- The $sDist()$ value.

- The location of the minimum distance.
- A binary frequency feature indicating how often the shapelet matches below a threshold.

The transformed series is then used to train a linear Ridge classifier, yielding a final feature vector of size $3k$ for k shapelets.

2.3.4 Dictionary-Based Methods

Dictionary-based classifiers represent time series by histograms of symbolic patterns derived from local subseries. Like shapelet-based methods, dictionary approaches are phase-independent, they extract subseries without regard to temporal alignment. However, instead of computing distances between subseries and prototypes, each subseries is discretised into a symbolic word. These word frequencies form the feature space for classification, aligning the method conceptually with bag-of-words models from natural language processing.

The typical dictionary-based pipeline consists of four main steps:

1. Extract sliding or fixed windows (subseries) from the time series.
2. Transform each window into a discrete symbolic word over a fixed alphabet.
3. Represent the series as a histogram of word frequencies.

The key differences between dictionary-based methods lie in the technique used to discretise real-valued windows into symbolic words.

Symbolic Fourier Approximation (SFA)

The foundation for many dictionary-based approaches is Symbolic Fourier Approximation (SFA) [23], which works as follows:

- Each window is z-normalised to remove amplitude bias and standardise scale.
- A truncated Discrete Fourier Transform (DFT) is applied to the window. This reduces dimensionality by retaining only the first l Fourier coefficients, acting as a low-pass filter to suppress noise.
- The distributions of real and imaginary DFT components are binned using Multiple Coefficient Binning (MCB), which can apply either equi-depth or equi-width discretisation.
- Each coefficient is then mapped to a symbol, resulting in a word of length l drawn from a fixed-size alphabet.

Figure 12 shows the SFA pipeline: a raw time series is windowed, transformed via DFT, binned into symbols, and encoded into a word (e.g., $DAAC$).

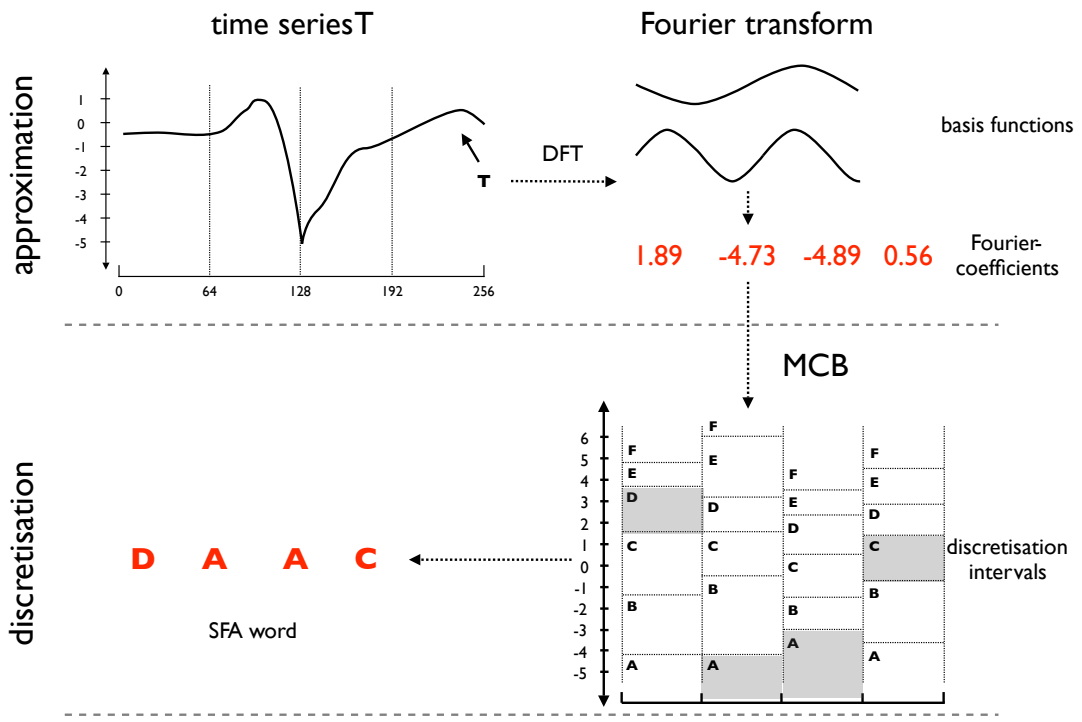

Figure 12: The Symbolic Fourier Approximation (SFA) (adapted from [4]): A time series (top left) is transformed by a truncated Fourier transform (top right) and discretised into the word $DAAC$ (bottom left) using coefficient-wise data-adaptive binning (bottom right). Image taken with permission of Matthew Middlehurst from [3].

Bag-of-SFA-Symbols (BOSS)

The Bag-of-SFA-Symbols (BOSS) classifier applies the SFA transformation to each window in a time series, generating a set of symbolic words [79]. A histogram is constructed over the full set of generated words, resulting in a sparse feature vector.

BOSS employs a non-symmetric distance measure between word histograms (word counts), designed to ignore words that occur only in the reference series and not in the query. This leads to a superior performance compared to standard Euclidean distances.

The full BOSS classifier is an ensemble. It explores a range of window sizes, word lengths, and alphabet sizes, retaining all configurations whose accuracy lies within 92% of the best on a hold-out validation set. Final predictions are made by majority voting across the ensemble.

Temporal Dictionary Ensemble (TDE)

The Temporal Dictionary Ensemble (TDE) builds upon BOSS and incorporates enhancements from multiple dictionary-based models, including WEASEL, cBOSS, and SpatialBOSS [24]. It uses SFA to transform series into word histograms, but introduces several refinements:

- From WEASEL, TDE adopts supervised breakpoint selection for discretisation and includes bi-gram counts of adjacent words to capture local sequential structure.

- From SpatialBOSS, it inherits spatial pyramids: word histograms are computed over disjoint subseries (segments) and concatenated to preserve spatial locality.
- Unlike earlier methods, bi-grams in TDE are computed only over the full series, while unigram counts are computed for each spatial segment.

2.3.5 Convolution-Based Methods

Convolution-based classifiers construct features from time series by applying learned or randomly generated convolutional kernels. Each kernel acts like a sliding filter over the input, and the resulting activation map captures local patterns. Pooling operations (e.g., max, average) are applied to these activation maps to derive fixed-size feature vectors for classification.

Conceptually, convolutional kernels resemble shapelets: both identify discriminative patterns in local windows. Grabocka et al. [80] first noted that shapelet matching could be realized through a convolution followed by a min-pooling operation over windowed Euclidean distances. However, while shapelets are extracted from the training data, convolution kernels are drawn from the continuous space of real-valued parameters.

The general pipeline for convolution-based time series classification (TSC) is depicted in Figure 13: convolutional operations create activation maps, which are pooled into features, concatenated, and passed to a classifier such as Ridge regression.

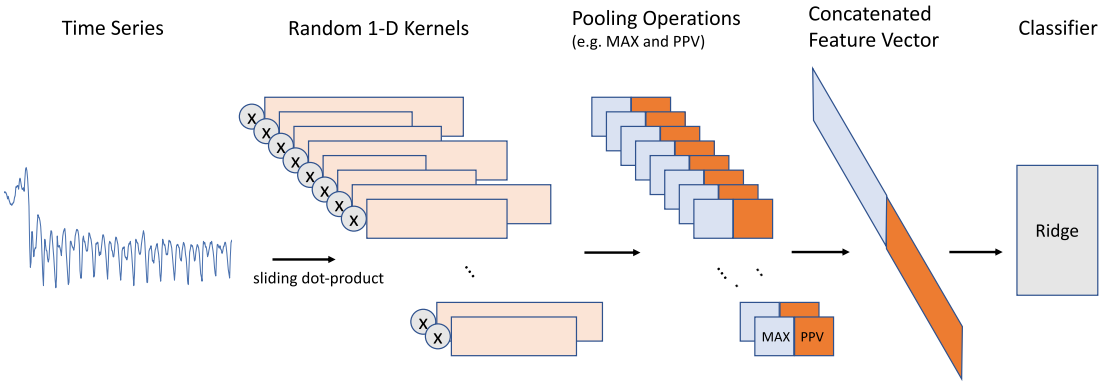

Figure 13: Pipeline of convolution-based classifiers, such as ROCKET, involves applying convolutional kernels to generate activation maps, which are then pooled and passed to a Ridge classifier for final prediction. Image taken with permission of Matthew Middlehurst from [3].

Random Convolutional Kernel Transform (ROCKET)

ROCKET is one of the most influential convolution-based methods [25]. It uses thousands of randomly parameterised convolutional kernels to transform time series into feature vectors. For each kernel, two features are extracted from its activation map:

- The maximum value (max pooling).
- The proportion of positive values (PPV) after convolution.

For k kernels, the resulting feature vector has $2k$ dimensions. Each kernel is defined by:

1. Length l: randomly selected from three different values.
2. Weights w: drawn from a normal distribution.
3. Bias b: added post-convolution.
4. Dilation d: governs spacing between weights, enabling multi-scale feature detection (sampled from an exponential distribution).
5. Padding p: typically zero-padding to preserve the time series length.

The final feature matrix is used to train a Ridge regression classifier. On larger datasets, Logistic Regression is recommended for scalability. Conceptually, ROCKET resembles a single-layer convolutional neural network (CNN) with fixed kernels and a softmax output layer, but without backpropagation.

2.3.6 Deep Learning-Based Methods

Deep learning has emerged as the most active subfield in time series classification. Fawaz et al., in their work "Finding AlexNet for Time Series Classification" [81], explicitly drew analogies to the impact of CNNs in vision, arguing that time series data shared structural properties with images. Earlier surveys (e.g., Fawaz et al.[82]) identified ResNet [83] as the most accurate deep learner at the time. This was soon followed by InceptionTime [81], which incorporated architectural principles from Inception networks in vision. In practice, InceptionTime performs comparably to top-performing hybrid models.

Since then, the number of deep learning classification models has exploded. A recent review by Foumani et al. [84] cites over 240 papers, with the vast majority published in just the last years.

Despite their strong predictive performance, deep learning approaches have several limitations. They typically require large amounts of labeled data for training, which may not be available in specialized domains such as clinical time series. Moreover, many applications involve unlabeled data in clustering scenarios or unsupervised anomaly detection settings, where deep models are harder to apply directly. Their complex architectures also make interpretability more challenging, making it difficult to understand which patterns drive predictions.

More recently, the community has begun exploring foundational models for time series, inspired by the success of large pre-trained models in NLP and vision. These approaches leverage self-supervised pretraining on large collections of time series, allowing them to generalize well even in low-data regimes. Notable examples include Moirai [85], which uses a masked reconstruction objective: parts of the input time series are deliberately hidden during training, and the model learns to predict the missing values. This forces it to capture meaningful temporal patterns and dependencies, both locally and across the full sequence. Chronos [86], in contrast, constructs hierarchical temporal embeddings, representing the time series

at multiple levels of granularity, from short-term fluctuations to long-term trends, so the model can reason about patterns occurring at different timescales simultaneously. These models offer a promising path to bridge the gap between high predictive performance and practical constraints such as limited labels or interpretability requirements. However, in the context of this thesis, such pre-trained foundation models are not utilized. The primary reason is that our application domain, intraoperative motor evoked potentials (MEPs) from neurosurgical procedures, involves time series data with limited volumes and unique characteristics that could differ substantially from the general-purpose time series on which foundation models are typically pre-trained. Moreover, the interpretability requirements and the need for computationally efficient real-time decision-making in clinical settings make the ensemble-based approaches developed in this thesis more suitable than complex pre-trained deep learning models.

For these reasons, we also consider forest-based models, discussed in Section 2.4, which provide an alternative approach. By combining strong predictive performance with interpretable decision structures, forest-based models offer practical advantages in domains where labeled data is scarce or explainability is crucial.

Residual Network (ResNet)

The Residual Network (ResNet) [83] is a deep learning architecture that has been effectively adapted for time series analysis. It consists of three residual blocks, each containing two key components: (a) three convolutional layers that extract features from the input, each followed by batch normalization and a ReLU activation function, and (b) a shortcut connection that directly propagates information from earlier to later layers, as illustrated in Figure 14.

The shortcut connections help alleviate the vanishing gradient problem commonly encountered in deep networks, while the convolutional layers capture temporal patterns from the time series. Finally, the extracted features are aggregated using a Global Average Pooling (GAP) layer, and a fully connected softmax layer produces the class predictions, with the number of neurons corresponding to the number of target classes.

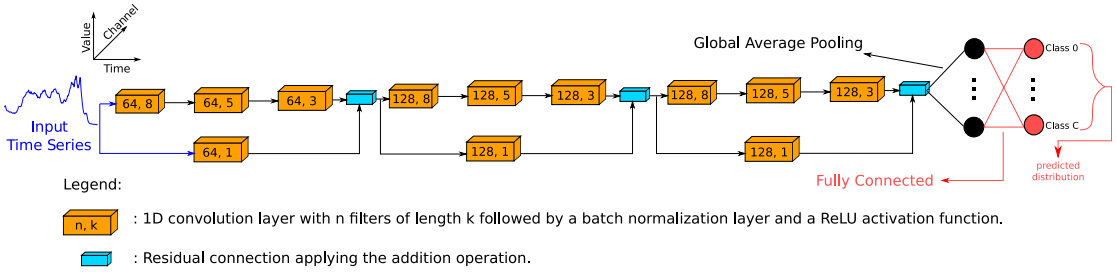

Figure 14: Overview of the ResNet structure. Image taken with permission of Matthew Middlehurst from [3].

InceptionTime

InceptionTime is a deep learning model introduced by Fawaz et al. [81], designed specifically for time series classification. It comprises an ensemble of five identical

deep neural networks, each sharing the same architecture but initialized with different random weights to encourage diversity.

The model is organized into two residual blocks. Each block processes the input through a sequence of Inception modules and includes a skip connection that adds the block input to its output, facilitating optimization in deep networks. Within each Inception module, a bottleneck transformation first reduces dimensionality to improve computational efficiency. The module then applies multiple convolutional filters with different kernel sizes in parallel, capturing patterns at multiple temporal scales. After the residual blocks, global average pooling aggregates the learned representations and a final classifier produces the prediction.

2.3.7 Hybrid approaches

Hybrid approaches combine two or more of the time series transformation strategies described in previous sections—such as distance-based, interval-based, shapelet-based, dictionary-based, convolution-based, or deep learning models. The choice to adopt a hybrid strategy is often motivated by the heterogeneity of the data and the absence of prior knowledge about which representation will be most effective for a given task.

In general, hybrid models seek to leverage the complementary strengths of different representations to improve classification accuracy and robustness. They may do so either explicitly, by combining the outputs of models built on distinct transformations, or implicitly, by integrating multiple representation mechanisms within a unified pipeline.

2.4 Random Forests for Time Series Classification

In this section, we present the fundamental concepts underlying Random Forests. We begin by introducing the general principles of ensemble learning and the role of Decision Trees as the base learners on which Random Forests are built. We then describe the learning mechanism of Random Forests, detailing how they combine multiple trees through Bagging and Randomization to achieve high predictive performance. Finally, we discuss several extensions of Random Forests specifically designed for time series data, which are particularly relevant for the methods explored later in this thesis.

Ensemble learning [87, 88] is a general meta-approach in pattern recognition and machine learning that aims to improve predictive performance by combining the outputs of multiple models. These meta-learning techniques aggregate diverse individual learners to achieve better accuracy, robustness, and stability than any single model could provide. By leveraging the complementary strengths of weakly correlated models, ensemble methods can produce more reliable and effective predictions. This foundational idea has been formalized and validated extensively in the literature [88, 89], and ensemble techniques have become central in supervised and unsupervised learning tasks.

A key motivation for ensemble methods arises from the sensitivity of high-variance learners to small changes in the training data, which can lead to significantly different model structures. Ensemble strategies address this problem by building and combining multiple models (weak learners) to produce a final prediction that is more robust to changes in any single training set. Depending on how the ensemble is constructed and how the base models are diversified, ensemble methods are generally categorized into three main types: bagging, boosting, and randomization. Bagging (bootstrap aggregating) builds multiple models independently on different random subsets of the data and averages their predictions to reduce variance. Boosting trains models sequentially, where each new model focuses on correcting the errors of the previous ones, often reducing bias. Randomization introduces stochastic elements into the training process (such as randomly selecting features or data splits) to increase model diversity and reduce correlation between base learners.

2.4.1 Bagging, Boosting, and Randomization

As introduced earlier, ensemble methods can be broadly grouped into three main types: bagging, boosting, and randomization. While bagging and boosting are the most widely studied, randomization offers an alternative strategy for creating diverse base learners. The following sections describe each approach in more detail. Bagging, short for bootstrap aggregating, is a variance-reduction technique introduced by Leo Breiman [89]. The core idea of bagging is to generate multiple versions of the weak learner by training each on a different bootstrap sample (i.e., random samples drawn with replacement) from the original training dataset. Each model in the ensemble is trained independently, and for classification tasks, the final prediction is typically obtained through majority voting, while for regression tasks, the ensemble prediction is computed as the average of individual outputs. Because the models are trained on slightly different data, they tend to learn different decision boundaries, and aggregating their predictions helps cancel out individual variances. In contrast, boosting is a sequential ensemble technique that focuses on bias reduction. Instead of training models independently, boosting builds a sequence of models in which each successive model attempts to correct the errors made by its predecessor. Algorithms such as AdaBoost [90] and Gradient Boosting Machines [91] are examples of this strategy. Each new model is trained on a modified version of the data, where examples that were previously misclassified are given greater weight or importance. Boosting tends to produce strong predictive performance but is more prone to overfitting if not regularized properly, especially in the presence of noise. Randomization takes a different approach by injecting stochasticity directly into the model construction process to promote diversity among base learners. Instead of altering the data distribution (as in bagging) or weighting samples (as in boosting), randomization alters the learning algorithm itself, for example, by randomly selecting subsets of features or splitting criteria when building decision trees. This randomness reduces correlation between base models, which can significantly improve ensemble performance when combined through averaging or voting.

2.4.2 Decision Trees

Before discussing the Random Forest model in detail, it is essential to understand the weak learner on which it is built: the Decision Tree (see Figure 15). Decision trees, in their standard form built on vectorial data, are non-parametric models that recursively partition the feature space into increasingly homogeneous subsets based on feature values. Each internal node in the tree represents a decision based on a particular feature and threshold, while each leaf node corresponds to a prediction (either a class label in classification or a continuous value in regression). To build a decision tree, the training data is repeatedly split into subsets by selecting, at each node, the feature and threshold that best separate the data according to an impurity criterion. This process continues recursively until a stopping condition is reached, such as a maximum tree depth or a minimum number of samples in a leaf. To use a decision tree for prediction, a new instance is passed from the root down through the tree: at each internal node, the algorithm evaluates the corresponding feature and follows the branch that matches the condition, until it reaches a leaf node that provides the predicted output. The strength of decision trees lies in their interpretability, minimal data preprocessing requirements, and ability to model complex, non-linear interactions between features. They are capable of handling both numerical and categorical data, missing values, and features with different scales. Furthermore, decision trees can naturally capture feature interactions and hierarchical decision rules. However, decision trees have a key limitation: they tend to overfit the training data. Because trees can grow arbitrarily deep and complex to capture all differences in the data, they often exhibit high variance, leading to poor generalization to new data [87]. Ensemble methods such as Bagging, therefore, offer a compelling solution to mitigate the high variance of decision trees while preserving their strengths as base learners. This idea is realized in the Random Forest algorithm: a powerful and widely used ensemble method that extends bagging with additional mechanisms for improving performance.

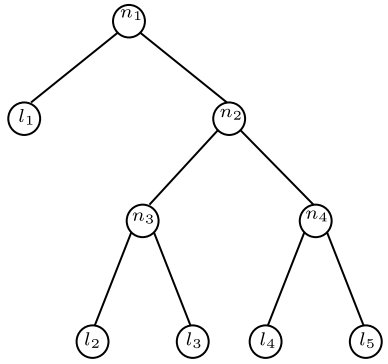

Figure 15: A schematic of a binary decision tree.

2.4.3 The Random Forest Algorithm

In their original formulation, Random Forests, introduced by Leo Breiman [26], are a robust ensemble learning method that combines the principles of Bagging with an additional source of randomness through randomized feature selection. Like bagging, Random Forests generate multiple decision trees by training each tree on a

bootstrap sample of the training data. However, Random Forests introduce a key innovation during the construction of each tree: at every candidate split in the tree, only a random subset of features is considered for splitting, rather than all available features.

This process, referred to as feature subspace sampling, introduces further diversity into the ensemble by reducing the correlation between individual trees. In standard bagging, especially when certain features are particularly strong predictors, many trees tend to make similar splits near the root, leading to correlated errors. Randomizing feature selection mitigates this effect, encouraging trees to explore different parts of the feature space and discover complementary patterns. The resulting ensemble is more diverse, which, as Dietterich [88] has shown, is a critical factor in ensemble effectiveness.

Formally, for a dataset with N observations and f features, the Random Forest algorithm constructs T trees, each trained on a bootstrap sample of the data. At each split in the tree, a subset of m features (where $m < f$) is chosen uniformly at random, and the best split is found among these. For classification tasks, the final prediction is determined by majority vote across all trees, while for regression, it is the average of their predictions.

Random Forests offer several benefits over traditional decision trees:

- **Improved generalization** By averaging over many uncorrelated trees, Random Forests substantially reduce overfitting, yielding better test performance than individual trees.
- **Internal estimates of generalization error** Using out-of-bag (OOB) samples—observations not included in the bootstrap sample for a particular tree—Random Forests can estimate error without the need for a separate validation set.
- **Feature importance** The algorithm provides intrinsic measures of feature importance, often computed based on the decrease in node impurity or prediction accuracy when a feature is permuted.
- **Scalability and parallelism** Trees can be trained independently and in parallel, making Random Forests well-suited to large-scale learning.

Since their introduction, Random Forests have become one of the most widely used machine learning algorithms, with the original paper by Breiman having accumulated over 160,000 citations (according to Google Scholar, accessed 09/2025). They have been successfully applied across a wide range of domains, including bioinformatics [92], remote sensing [93], and medical data [94], demonstrating their versatility and robustness.

Over the years, many task-specific adaptations of Random Forests have been developed, as well as numerous algorithmic variants designed to enhance performance or handle specialized data types—for example, Extremely Randomized Trees [95], Oblique Random Forests [96], and survival forests for censored data [97].

These variants extend the core Random Forest framework to better fit particular learning scenarios or data characteristics.

2.4.4 Isolation Forest for Anomaly Detection

While Random Forests are primarily designed for supervised classification and regression tasks, tree-based ensemble methods have also proven effective for unsupervised anomaly detection. Isolation Forest (iForest), introduced by Liu et al. [98], is an ensemble algorithm specifically designed for anomaly detection in high-dimensional data. Unlike traditional anomaly detection approaches that explicitly model normal behavior, Isolation Forest exploits the key observation that anomalies are rare and different from normal instances, making them inherently easier to isolate in partitioned spaces.

The algorithm constructs an ensemble of isolation trees by recursively partitioning the data using randomly selected features and thresholds. Because anomalous points are few and distinct, they tend to be separated with fewer partitions, resulting in shorter path lengths from the root to the leaf node. In contrast, normal instances, being more densely distributed and similar to each other, require more splits to be isolated. This fundamental property allows the algorithm to assign anomaly scores based on the average path length across all trees in the ensemble: shorter average paths indicate more unusual behavior.

Isolation Forest has several advantages: it operates in linear time with respect to the number of samples, scales well to high dimensions, and does not require explicit distance metrics or density estimations. These characteristics make it particularly suitable for time series anomaly detection, where dimensions are often large and computational efficiency is important. Extensions of Isolation Forest to time series data, which incorporate temporal and structural information about sequences, are explored in detail in later sections of this thesis and will be thoroughly examined in the anomaly detection chapter.

In the next section, we introduce the most prominent Random Forest variants specifically designed for time series classification, which adapt the core algorithm to handle sequential and temporal dependencies.

2.4.5 Towards Forest-Based Models for Time Series

Random Forest is a powerful ensemble method originally designed for fixed-length feature vectors. Applying it to time series, which are sequential and often variable in length, is not straightforward. Their temporal structure, order-dependence, and potential misalignments challenge the core assumptions of standard Random Forests. This has motivated several strategies to make Random Forests more suitable for time series classification.

Broadly, three main approaches can be distinguished. The most direct one is to treat each time series as a long feature vector, where each timestamp becomes a separate input variable. While this requires no extra processing and fits existing

implementations, it ignores temporal order, struggles with misalignments, and suffers from high dimensionality, often leading to poor generalization.

A second approach is to first extract descriptive features from the series (e.g. statistical summaries, frequency components, shape descriptors) and then apply a standard Random Forest to this new representation. This can reduce dimensionality and highlight relevant patterns, making learning easier and often more interpretable. However, it relies heavily on the choice of features: if they fail to capture key temporal dynamics, important information may be lost.

Finally, Random Forests can be adapted to operate inherently on time series. This can involve building trees that split on discriminative time intervals or subsequences, or using time series distance measures (such as Dynamic Time Warping) to guide their structure. Such models preserve temporal relationships and can handle misalignments or variable lengths more naturally, though they are more complex to design and computationally more demanding.

In the following section, we will explore these specialized forest-based models in detail, starting from interval-based methods and moving toward distance-based approaches that directly exploit the sequential nature of time series.

Time Series Forest (TSF)

Time Series Forest (TSF) is the foundational interval-based ensemble method [19]. An interval here refers to a contiguous segment of the time series — for example, a short window spanning from time step i to time step j. Instead of treating each timestamp as an independent feature, TSF extracts information from several such intervals, which allows it to capture local temporal patterns while keeping the feature space manageable.

Each tree in the forest randomly selects $\sqrt{N}$ intervals from the full series of length N. For each chosen interval, three simple summary statistics are computed: mean, variance, and slope. These values are concatenated to form a fixed-length feature vector that represents the whole series from the perspective of those sampled intervals. This vector is then used to train a decision tree — called a time series tree — which considers all available features at each split and chooses the best one according to a standard criterion (Gini impurity for classification or squared error for regression). Building a TSF thus follows the same principles as a standard Random Forest: bootstrap sampling is used to create different training sets for each tree, and random interval selection injects additional diversity. During training, each tree learns thresholds on the interval-based features to partition the training data into increasingly homogeneous subsets.

When classifying a new time series, the same procedure is applied: the model samples the same intervals that were used during training, computes their mean, variance, and slope, and feeds these features down each tree. Traversing a tree works like in a standard decision tree — at each node, the test instance follows the branch determined by the feature threshold until it reaches a leaf node that stores a

class distribution (or a prediction value for regression). The final TSF prediction is then obtained by majority voting across all trees in the ensemble.

Random Interval Spectral Ensemble (RISE)

RISE builds on the interval framework by incorporating frequency-domain features [99]. Each tree selects a single random interval from the time series and extracts features based on the periodogram, which represents how the signal's power is distributed across different frequencies, and the auto-correlation function, which measures how the series relates to itself at different time lags. These features are concatenated into a vector used to train a standard decision tree.

A RISE model is built as an ensemble of such trees, each trained on different randomly chosen intervals and their corresponding spectral and auto-correlation features. To classify a new time series, it is passed through every tree in the ensemble: for each tree, the same interval selection and feature extraction process is applied to the new series, producing features that are passed down the tree to obtain a class prediction. The final output of RISE is typically obtained by majority voting across all trees in the ensemble.

Supervised Time Series Forest (STSF)

STSF introduces supervision into interval selection [100]. Intervals are extracted from three representations: the raw series, the periodogram, and first-order differences. For each tree, an initial split point is chosen at random. Each resulting segment is bisected, and the half with the highest Fisher score—a statistical measure of class separability computed as the ratio of between-class variance to within-class variance—is retained. This procedure is recursively repeated until a minimum length is reached. Seven summary statistics are computed for each selected interval.

RSTSF [101] introduces stochasticity by selecting split points randomly and including auto-regressive features. Features from multiple representations are concatenated and used in an Extremely Randomized Trees [95].

To build an STSF or RSTSF model, many trees are grown independently, each selecting supervised (or randomized, for RSTSF) intervals and extracting features from them. At prediction time, a new series is processed through each tree: the same representations are computed, the same supervised interval-selection rules are applied, and features from the chosen intervals are extracted and passed through the trained decision trees. The final class is obtained by aggregating the predictions from all trees, typically by majority vote.

Canonical Interval Forest (CIF)

CIF extends Time Series Forest (TSF) by incorporating more informative and diverse features [20]. In addition to mean, standard deviation, and slope, it includes CATCH22 features—a curated set of 22 time series descriptors capturing a broad range of statistical, temporal, and frequency-based properties (summarised in Table 2). For each tree, $k = \sqrt{N} \cdot \sqrt{d}$ intervals are randomly selected (where d is the number of dimensions), and a attributes from a set of 25 are randomly chosen.

These are combined into a $k \cdot a$ feature vector per series. For multivariate series, a random dimension is selected per interval. DrCIF, a further extension, integrates three representations: the raw series, periodograms, and first-order differences [102]. For each representation, $\frac{4+\sqrt{N}\sqrt{d}}{3}$ intervals are sampled (where N is the length of the series in that representation). The resulting features from all representations are concatenated into a joint feature vector used to train each tree. Both CIF and DrCIF are built as ensembles of many such trees. During training, each tree selects its own random intervals (and for CIF, random attributes) to form feature vectors, which are used to grow standard decision trees. At prediction time, a new series is processed through every tree: the same random intervals (fixed from training) are used to extract features, which are passed down the trained decision trees to yield predictions. The final output is obtained by majority voting over all trees in the ensemble.

Distance-based classifiers operate differently from feature-based approaches: instead of extracting summary statistics or spectral features, they rely directly on pairwise similarities between series. Similarity Forest (SimF) is a perfect example of this approach.

Similarity Forest (SimF)

Similarity Forest belongs to the family of distance-based ensemble classifiers, but with a distinct approach to leveraging pairwise similarities [103]. Instead of relying on exemplars to define splits, SimF constructs decision trees based on the relative distances between objects. At each node, a subset of objects is selected as references, and each candidate split is evaluated according to how well it separates the classes based on the distribution of distances to these references. By recursively applying this process, each tree organizes the series hierarchically according to similarity patterns rather than absolute features. For prediction, a new series is compared to the references at each node, traversing the tree according to which side of the split it is closest to. The ensemble aggregates predictions from all trees, typically via majority vote, yielding a robust classification that captures nuanced relationships between time series.

Proximity Forest (PF)

PF is an ensemble method composed of multiple Proximity Trees [104], each of which leverages a diverse pool of distance functions to capture various aspects of similarity between time series.

During training, each node of a Proximity Tree randomly selects a distance function from this pool and fixes its hyperparameters. For every class, one exemplar time series is drawn at random. The node then evaluates several candidate combinations of distance function, parameter values, and class exemplars, selecting the combination that best separates the classes according to the Gini impurity. Once the best split is chosen, each time series is assigned to the branch corresponding to the exemplar it is closest to. This recursive process continues until the leaves are pure, meaning all series in a node belong to the same class.

At prediction time, a new series traverses each tree in a similar manner: at every node, distances to the stored exemplars are computed using the node's distance function, and the series follows the branch of the nearest exemplar. Once it reaches a leaf, the tree produces a class prediction, and the ensemble combines predictions from all trees, typically through majority voting, to determine the final label. In this way, PF captures the intrinsic structure of the data through distances rather than handcrafted or learned features, providing a flexible approach to time series classification.

Proximity Isolation Forest

Proximity Isolation Forest, as described by Mensi et al. [105], adapts the isolation forest concept to time series using a distance matrix as input. Unlike the previous distance-based classifiers, the forest does not require class labels during training. Each tree in the forest recursively partitions the training series by randomly selecting a pivot object and a threshold distance, isolating series based on whether their distance to the pivot is smaller or larger than the threshold. Over many trees, time series that are "easier to isolate" appear in shorter paths, while more typical series require longer paths to be separated.

At test time, a new series is incorporated by computing distances to all training series used in the building of the tree, effectively adapting the distance matrix to include the new object. The time series is then passed down each tree, and the length of the path it follows is recorded. Averaging these path lengths across the forest produces an anomaly score or measure of "isolation", where shorter average paths indicate more atypical series.

2.5 Clinical and Healthcare Applications

The methodological approaches presented throughout this chapter have demonstrated substantial clinical value and real-world impact in medical and healthcare settings. Time series classification and anomaly detection are particularly relevant for patient monitoring, diagnostic support, and surgical safety. In cardiology, electrocardiography (ECG) signals are routinely analyzed to detect cardiac arrhythmias, a critical application where distance-based methods and ensemble approaches have achieved cardiologist-level performance [106]. In neurosurgical contexts, intraoperative motor evoked potentials (MEPs), which are time series recordings of muscle responses to transcranial electrical stimulation, require rapid and accurate classification to alert surgeons to potential neurological complications. Recent evidence demonstrates that machine learning methods, including Random Forests and ensemble classifiers, achieve expert-level accuracy in classifying muscular responses during real-time monitoring [34], [107]. Beyond MEP classification, anomaly detection techniques play a critical role in continuous patient monitoring systems, enabling the early identification of abnormal patterns in vital signs and physiological parameters that could indicate deteriorating patient status [63]. In clinical neurophysiology and rehabilitation medicine, surface electromyography (EMG) signals require accurate detection and classification of muscular activity patterns. Ensemble methods and distance-based measures have

been successfully applied to support clinical decision-making in these domains [108], [109]. These diverse clinical applications underscore how advances in time series analysis techniques directly translate to improved patient safety, diagnostic accuracy, and surgical outcomes across multiple medical specialties.

3 Classification of Intraoperative Motor Evoked Potentials

In the following sections, we first introduce the concept of Motor Evoked Potentials (MEPs) (Section 3.1.1), describing their physiological basis and the motor pathways involved in their generation. We then discuss the stimulation and recording techniques used to elicit and capture these signals (Section 3.1.2). Building on this foundation, we finally introduce the automated muscle identification problem and its clinical relevance in Section 3.2.

3.1 Motor Evoked Potentials

Motor evoked potentials (MEPs) are electrical signals recorded from muscle tissue in response to stimulation of the motor cortex. The stimulation may be magnetic or electrical and applied directly to the motor cortex or transcranially through the skull [110, 111].

In the following, we provide an overview of the foundations relevant to understanding how MEPs are generated, focusing first on the underlying motor pathways.

3.1.1 Motor Pathways

MEPs are used to monitor the functional integrity of the descending motor pathways [112], which mediate voluntary movement of the face, limbs, and torso. The descending motor pathways can be divided into a lateral system and a medial system. The lateral system includes the corticospinal tract and a smaller rubrospinal tract. The corticospinal pathway originates mainly in the primary motor cortex (Brodmann's area 4), but fibers from other regions, such as the premotor, the supplemental motor, and somatosensory cortices contribute as well. The descending motor pathways consist of upper and lower motor neurons, as well as interneurons at the level of the spinal cord. The cell bodies of the upper motor neurons lie in the motor cortices, whereas the cell bodies of lower motor neurons lie in the brain stem and spinal cord. Each lower motor neuron gives rise to one axon that exits the spinal cord and passes through a ventral nerve root. The ventral nerve roots form a fiber bundle with the dorsal nerve roots, which together form a peripheral nerve bundle. Each lower motor axon passes through the peripheral nerve bundle and then arborizes to innervate multiple muscle fibers. Each synaptic connection forms an excitatory synapse. One lower motor neuron and the muscle fibers that it innervates is known as a motor unit. There are hundreds of motor units within a single muscle, each of which occupies a space of approximately 5 to 11 mm in diameter [113]. These descending pathways can be grouped into several major systems based on their anatomical course and the muscles they primarily control:

- **The corticospinal system.** The corticospinal pathway is composed primarily of a lateral system and a smaller anterior system, and these innervate the distal limbs. The axons of the lateral tract descend through the internal capsule and project to the lower medulla. The upper motor neurons of the lateral tract cross the midline to the contralateral side and descend to the lateral column of the

dorsal horn. After reaching the ventral horn of the spinal cord, these motor neuron axons form synaptic connections with lower motor neurons, often via local interneurons. The axons of the anterior tract do not appear to cross the midline.

- **The medial system.** The medial system innervates the trunk and proximal limb muscles. Fibers in the motor cortices descend bilaterally and do not cross the midline.
- **The corticobulbar system.** The upper motor neurons of the corticobulbar tract descend to the brain stem and innervate several nuclei for cranial nerves that control the facial muscles. In most cases, the corticobulbar tract innervates the facial motor nuclei bilaterally, including those involved in swallowing, speech, chewing, and tongue movement. In contrast, upper motor neurons innervate the lower facial nuclei and the hypoglossal nerves on the contralateral side of the body.
- **The basal ganglia.** The basal ganglia are a group of brain structures that help control movement. They are considered part of the upper motor system because they influence motor pathways. The motor cortex and substantia nigra pars compacta (SNc) send signals to the striatum. The striatum connects to the globus pallidus internus (GPi) and substantia nigra pars reticulata (SNr) through two routes: the direct and indirect pathways. The direct pathway uses neurons with D1 dopamine receptors, while the indirect pathway uses neurons with D2 receptors and also involves the globus pallidus externus (GPe) and subthalamic nucleus. In the end, the GPi and SNr send constant inhibitory signals to the motor cortex through the thalamus.

This brief anatomical overview provides the necessary background to understand how external stimulation techniques can activate these pathways to evoke MEPs, as discussed next.

3.1.2 Stimulation and Recording Techniques

Motor evoked potentials (MEPs) are generated by activating the descending motor pathways, and their reliable measurement depends on both the method of stimulation and the approach used to record the resulting muscle activity. Understanding these techniques is essential for interpreting MEP characteristics and for ensuring accurate intraoperative monitoring. In the following, we first describe the main stimulation methods used to evoke MEPs and then discuss the recording strategies and practical considerations necessary for capturing these signals reliably.

MEPs can be elicited through several stimulation techniques, each leveraging different physical principles to activate the motor pathways described previously.

1. **Transcranial electrical stimulation (TES)**. TES is a commonly used, non-invasive technique for generating MEPs. Stimulating electrodes for TES are placed on the scalp or subcutaneously above the primary motor cortex. An electrical current is applied to the scalp to modulate cortical neuronal activity.

However, due to the thickness and high resistance of the skull bone, only a small percentage of current reaches the brain tissue. Therefore, to record MEPs from muscle tissue, the stimulus strength must be set high enough to overcome that resistance and activate the underlying motor pathways. This type of stimulation is termed supra-maximal, as the stimulus exceeds that required to recruit all muscle fibers around the recording electrode. Train stimulation is required to elicit reliable MEPs under anesthesia, necessitating the use of a multi-pulse stimulator.

2. **Transcranial magnetic stimulation (TMS)**. TMS is another non-invasive technique for generating MEPs. By applying a magnetic field over the scalp, TMS can induce electrical currents in brain tissue in awake patients. One disadvantage of TMS, however, is that TMS-induced MEPs are suppressed under anesthesia.
3. **Direct cortical stimulation (DCS)**. For some brain surgeries, such as tumor resections, the surgeon will directly stimulate the motor cortex or adjacent subcortical tissue to elicit an MEP. Direct cortical stimulation requires much lower stimulation intensity because of the absence of the skull. In most cases, it is necessary to find the lowest stimulation threshold for MEPs, which provides important information on the proximity of the neural probe to the motor pathways. The lower the stimulation intensity, the closer the probe is to the motor pathways. The position of the neural probe relative to the homunculus normally dictates which muscle groups are activated.

Figure 16 illustrates a comparison between direct cortical stimulation (DCS) and transcranial electrical stimulation (TES) for eliciting MEPs in the Abductor Pollicis Brevis muscle. Although the waveform shapes are largely similar for both methods, it is evident that the amplitude of the MEP is considerably higher with DCS (left) than with TES (right). This reflects the reduced resistance to current in DCS, allowing for more efficient activation of the underlying motor pathways at lower stimulation intensities.

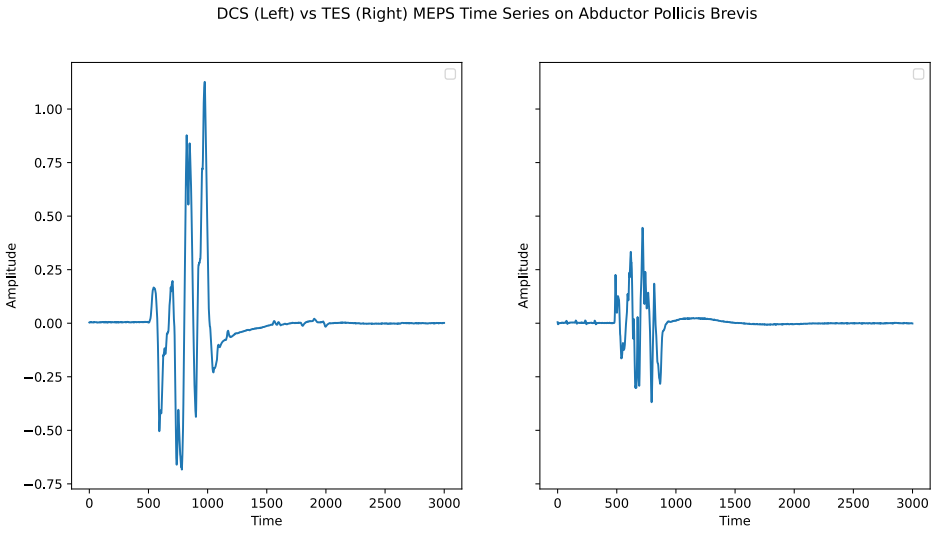

Figure 16: Comparison of intensity of direct cortical vs. transcranial electrical stimulation of Abductor Pollicis Brevis.

Understanding these stimulation methods provides the foundation for interpreting MEP waveforms. Equally important is the method used to record these signals reliably.

Reliable MEP recording depends critically on proper electrode placement, adherence to standardized protocols, and consideration of factors that influence signal quality. Two sets of considerations are particularly relevant:

1. **Technical considerations.** MEPs are typically recorded from muscles (myogenic MEPs) as compound muscle action potentials. Myogenic MEPs are recorded from muscle groups associated with each myotome (a set of muscle groups innervated by a single motor neuron). The placement of recording electrodes on specific muscles depends on the site of the surgical procedure and which nerve roots are potentially at risk. Signals are filtered at high and low frequencies to eliminate electrical artifacts, and recording electrode resistance should be kept low.
2. **Surgical and medical considerations.** Some considerations for recording MEPs include the use of anesthetics and neuromuscular blocking agents. For example, myogenic MEPs are very sensitive to inhalation anesthetics, which limits the use of these anesthetics during IOM. Inhalation anesthetics reduce the amplitude and prolong the latency of MEPs, and this effect becomes more pronounced when multiple synapses are involved, as is the case for myogenic MEPs, which involve the upper and lower motor neurons as well as interneurons in the spinal column. Intravenous anesthetics have a similar effect on MEPs but to a lesser degree. Total intravenous anesthesia is preferred in most cases, particularly those involving the spinal cord at L2 vertebrae and above. MEPs are also influenced by neurological disorders, such as myasthenia gravis, botulinum toxin treatments for dystonia, and muscular dystrophy.

Building on the foundations of MEP generation and stimulation described above, it becomes evident that accurate interpretation of MEPs is critical for patient safety during neurosurgical procedures. This chapter develops and evaluates machine learning approaches for automated muscle identification from intraoperative MEPs, comparing their performance to human experts. We formalize the medical problem and establish formal computational definitions, then describe our experimental methodology, present comprehensive results, and discuss clinical implications.

3.2 Introduction

The monitoring of motor evoked potentials (MEPs) during neurosurgical procedures is a well-established technique for real-time functional integrity assessment [114–116]. In intraoperative monitoring (IOM), MEPs are elicited by electrical stimulation of motor pathways and recorded from target muscles using subdermal electrodes; the physiological and technical background is reviewed in Section 3.1 [117–119].

In this chapter, however, the focus is not on alarm criteria but on a different and more basic task: identifying which muscle generated a recorded MEP. This step is still largely manual in clinical practice. As illustrated in Figure 17, interpretation is complicated by factors such as surgical manipulation, anesthetic management, the patient's overall condition, equipment reliability, and inter-rater variability, all of which can affect the recorded signals.

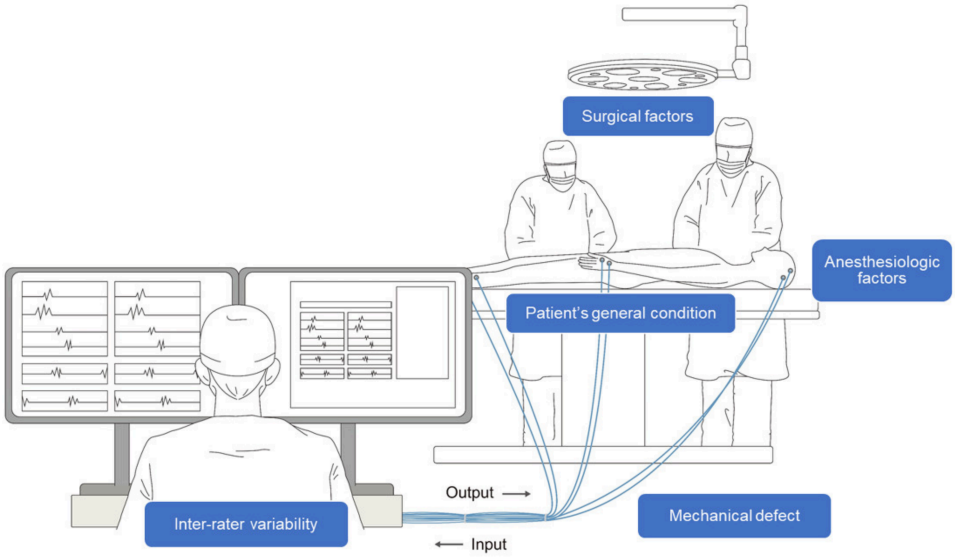

Figure 17: Multiple factors influencing intraoperative monitoring (IOM) outcomes. These include surgical factors, anesthetic management, the patient's general condition, mechanical issues with equipment, and inter-rater variability in interpreting signals. All these can affect MEP amplitude and should be considered when evaluating alarm criteria. Image taken with permission of Dr. Dougho Park [5].

Correct implementation of monitoring settings and accurate interpretation of motor outputs are essential to warn of signal changes that could lead to clinically meaningful deficits [115, 120, 121]. Current clinical interpretation relies on semi-quantitative evaluation by experienced neurophysiologists, who assess MEP amplitude, latency, morphology, and threshold changes. However, MEPs are complex biological signals with inherent variability within and between subjects due to anatomical differences, physiological variation, and influences from anesthesia and surgical manipulation [122–124]. This complexity makes muscle identification particularly challenging, especially in modern surgical environments where one neurophysiologist may monitor multiple patients simultaneously.

Recent developments in machine learning and signal analysis have proven effective for extracting clinically meaningful information from medical data, including electrophysiological signals [125–129]. Successful examples include classification algorithms for cardiac arrhythmias from ECG, seizure detection from EEG, and muscular activity recognition from surface EMG [106, 108, 109]. However, there is a currently untapped opportunity in applying these powerful techniques to

intraoperative neuromonitoring, with the potential to increase surgical safety and refine diagnostic criteria.

In this chapter, we develop and test machine learning models for identifying different muscles from intraoperative MEPs and compare their performance to human experts. We conduct a systematic evaluation under three distinct experimental scenarios of increasing complexity, evaluating multiple signal representations and classifier types to comprehensively assess classification capability and identify optimal approaches for clinical application.

3.3 Medical Problem

The fundamental clinical challenge addressed in this chapter is the reliable identification of muscles from intraoperative motor evoked potentials during neurosurgical procedures. Muscle identification is essential for interpreting motor evoked potential data throughout the surgical procedure. At present, this task is entirely manual and depends on the correct labeling of muscle-electrode connections before surgery. Any mislabeling of these connections could lead to misinterpretation of motor status and, potentially, to severe clinical consequences such as undetected motor pathway injury.

The problem is compounded by signal complexity and variability. MEPs vary significantly within patients (trial-to-trial variability), across patients (anatomical and physiological differences), and across protocols (TES vs DCS produce different signal characteristics). Neurophysiologists currently rely on visual inspection and limited quantitative parameters to identify muscles, but this approach is subjective and vulnerable to error, particularly under time pressure.

A reliable automated system capable of identifying which of six target muscles (biceps brachii, extensor digitorum communis brevis, abductor pollicis brevis, quadriceps femoris, tibialis anterior, abductor hallucis) produced a given MEP, across different stimulation protocols, and generalizing to previously unseen patients would serve as a critical clinical decision support tool. Such a system could verify correct muscle-electrode labeling during IOM setup, preventing labeling errors and enhancing patient safety.

3.4 Machine Learning Formalization

We formalize the muscle identification problem as a supervised multi-class classification task. Let $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ be a training dataset where N is the number of MEP signals, x_i represents a preprocessed MEP signal, and $y_i \in \{1, 2, 3, 4, 5, 6\}$ denotes the class label corresponding to one of six target muscles.

Given a training set D, the objective is to learn a classification model m such that for a new MEP signal x_{test}:

$$\hat{y} = m(x_{\text{test}}) \in \{1, 2, 3, 4, 5, 6\} \tag{23.}$$

We evaluate this model under three distinct experimental scenarios, defined by different patterns of data distribution between training and test sets:

Task 1 (Intra-patient): All training and test data originate from the same patient via a stratified k-fold (k=5), with MEPs from TES and DCS protocols considered separately. This provides an upper bound on classification performance, demonstrating the maximum discriminative power available from muscle-specific MEP patterns within a single subject. This scenario tests whether patterns are learnable and repeatable within individual patients but does not address generalization to other subjects.

Task 2 (Inter-patient): Following a leave-one-patient-out approach, models are trained on data from all patients except one, then tested exclusively on the held-out patient's data. This directly assesses whether a model trained on a specific patient population generalizes to a completely unseen subject with different anatomy and physiology. This is the clinically most relevant scenario, as future clinical deployment must classify MEPs from patients not present in training data. Two complementary variants evaluate important design tradeoffs: Task 2.1 maximizes patient count with two muscle classes (41 patients, upper limb muscles only), and Task 2.2 maximizes muscle count with four muscles (21 patients, upper and lower limbs).

Task 3 (Inter-protocol): Training and test data combine MEPs from both TES and DCS protocols with all six muscle classes. This evaluates protocol-agnostic classification, whether muscle identification is achievable independent of the stimulation method, critical for clinical applicability across diverse surgical scenarios.

For each task, we evaluate two signal representations: the whole signal database (including temporal latency information from stimulus to MEP onset) and the no-latency database (MEP waveform only). This allows assessment of whether temporal information is necessary or whether waveform characteristics alone suffice for muscle identification.

3.5 Methods

3.5.1 Dataset Preparation

Patient Selection and MEP Acquisition

Patients were retrospectively selected from the neurosurgery department database at Verona Regional Hospital based on: a) brain or spinal cord lesions requiring surgical treatment, b) TES, DCS, or both used during IOM, c) monitoring of all six target muscles, d) presence of monitorable responses. Pediatric patients were excluded. All procedures followed institutional guidelines and obtained IRB approval (protocol CRMS 23038, approved June 5, 2024).

MEP acquisition protocols followed standard intraoperative monitoring procedures as detailed in Section 3.1. For TES, anodal electrical stimulation was applied via corkscrew electrodes at C1 and C2 (10-20 International System); for DCS, anodal

stimulation was applied directly to exposed cortex using a strip electrode as the anode and a scalp electrode at Fz as the cathode. High-frequency short train stimulation employed five pulses with interstimulus intervals of 4 ms (TES) and 2 ms (DCS), at intensities of 50–150 mA (TES) and 5–20 mA (DCS). MEPs were recorded using subdermal needle electrode pairs from six target muscles: biceps brachii, extensor digitorum communis brevis, and abductor pollicis brevis (upper limbs); quadriceps femoris, tibialis anterior, and abductor hallucis (lower limbs).

Signal Representation and Feature Extraction

The MEP database comprised signals extracted by experienced neurophysiologists and neurosurgeons, excluding signals with excessive noise, silent responses, or latencies outside the physiological ranges for specific muscles. The database was then duplicated to create two versions: (1) the whole-signal database, containing complete signals including latency, and (2) the no-latency database, containing waveform information only.

Five signal representations were generated from both databases:

- **RAW:** Unprocessed signal
- **NORM:** Signal normalized to range [−1, +1]
- **TRAD:** Eight traditional neurophysiological features (amplitude, area, duration, thickness, size index, number of phases, number of turns, number of satellites) [130]
- **TSFRESH:** 789 features produced by a time series feature extraction model
- **TSFRESH_FS:** Top 10 features from TSFRESH, obtained by applying the TSFRESH feature selection method (yielding 182 features) followed by Recursive Feature Elimination [131] to reach a feature count comparable to TRAD

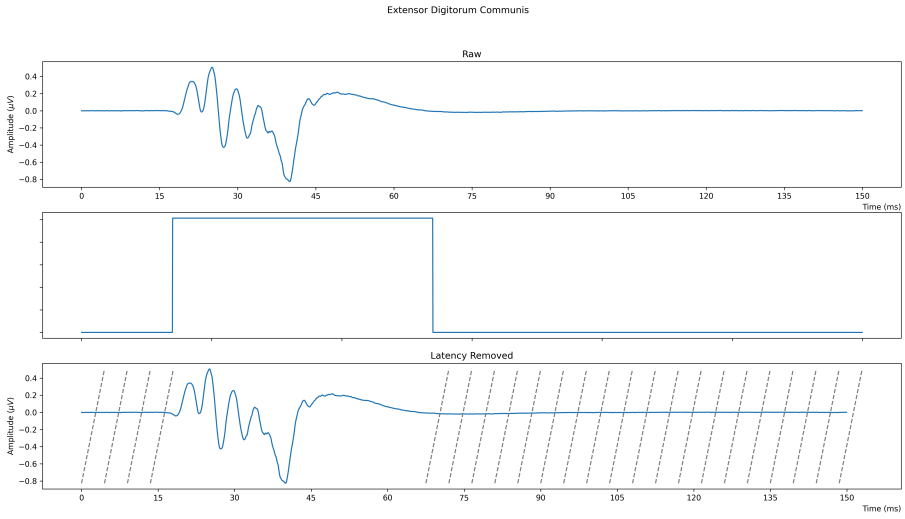

Figure 18: Whole signal and no-latency MEP example. Left: raw signal from whole signal database. Middle: interval showing signal waveform extraction. Right: final extraction as example from no-latency database. X-axis: time in milliseconds, Y-axis: MEP amplitude in microvolts.

Data Summary

We included 54 patients (26 TES, 28 DCS) who underwent oncological, vascular, or trauma surgery to brain or spine. Total dataset: 25,423 MEPs (14,476 TES, 10,947 DCS), variably distributed across patients and muscles as shown in Table 3. Feature categories in TSFRESH include: signal summary statistics (max, min, length), sample distribution characteristics (symmetry, entropy, energy), regression/correlation models, time-domain dynamics (first peak index, first derivative, maximal length over mean), and frequency-domain dynamics (frequency analysis, Fourier coefficients, wavelets). The top 10 TSFRESH_FS features are reported in Table 4.

	TES total	DCS total	Task 2.1 TES	Task 2.1 DCS	Task 2.2 TES	Task 2.2 DCS
Biceps Brachii	2438	2208	/	/	2167	904
Extensor Digitorum Communis Brevis	2714	3128	2714	3101	1833	1601
Abductor Pollicis Brevis	3433	3781	2971	3664	2179	1865
Quadriceps Femoris	1593	312	/	/	/	/
Tibialis Anterior	2311	702	/	/	1189	/
Abductor Hallucis	1987	816	/	/	/	804
Total	14476	10947	5695	6765	7368	5165

Table 3: Composition of the MEP database by muscle and IOM protocol.

Position	Whole signal	No-latency
1	ratio beyond r sigma (r=1)	autocorrelation (lag=8)
2	autocorrelation (lag=9)	agg. autocorrelation(aggtype=median, maxlag=40)
3	index mass quantile (q=0.1)	agg. autocorrelation (aggtype=mean, maxlag=40)
4	spkt welch density (coeff=8)	change quantiles (qh=0.2, ql=0.0)
5	lempel-ziv complexity (bins=100)	ar coefficient (coeff=3, k=10)
6	index mass quantile (q=0.2)	agg. linear trend (chunklen=10)
7	index mass quantile (q=0.3)	fft coefficient (attr=image, coeff=1)
8	index mass quantile (q=0.4)	index mass quantile (q=0.1)

Position	Whole signal	No-latency
9	energy ratio by chunks (n.seg=10, seg.foc=0)	energy ratio by chunks (n.seg=10, seg.foc=0)
10	fft aggregated (aggtype=variance)	cwt coefficients (coeff=10)

Table 4: Top 10 TSFRESH_FS features for the whole-signal and no-latency databases.

3.5.2 Classification Models and Training

Baseline Machine Learning Approaches

Five classifiers were trained and tested: k-Nearest Neighbor with k=1 (NN), k-Nearest Neighbor with k=10 (kNN), linear SVM (linSVM), radial basis function SVM (rbfSVM), and Random Forest (RF).

k-Nearest Neighbors (k-NN): An instance-based learner that classifies data points according to the labels of their nearest neighbors, assuming that similar instances cluster in feature space. k-NN identifies the k closest training examples and assigns the most frequent label. We employed standard Euclidean distance with $k = 1$ (nearest-neighbor rule) and $k = 10$ without hyperparameter tuning, to maintain consistent baseline approaches.

Support Vector Machines (SVM): Supervised models that identify hyperplanes maximizing the margin between classes [51], with non-linear decision functions obtained via the kernel trick. We employed two variants: linSVM with a linear kernel and rbfSVM with a Radial Basis Function kernel. The cost parameter C (controlling the trade-off between generalization and training error) was tuned via 5-fold cross-validation on the training set, together with the parameter σ for the RBF kernel. This ensures a fair comparison across different signal representations and study designs.

Random Forests (RF): Ensemble methods that combine multiple decision trees trained on bootstrap-aggregated data with random feature subsets [26], using majority voting for classification. This ensemble approach reduces the overfitting often encountered in medical data classification. We employed 100 trees trained using the Gini impurity criterion.

3.5.3 Expert Neurophysiologist Comparison

Ten expert neurophysiologists independently classified 50 randomly selected MEPs from the whole signal database. Their performance, measured as the percentage of correctly classified signals, was compared with that of the best-performing model on the most difficult task (Task 3).

3.6 Results

3.6.1 Task 1: Intra-patient Classification

In intra-patient classification with whole signal database, classification performances expressed as balanced accuracy reached maximum of 99% in both TES and DCS groups using Random Forest on TSFRESH representation (Table 5). Lowest

performances were rbfSVM on TRAD representation: 49% (TES), 58% (DCS). With no-latency database, Random Forest on TSFRESH achieved 99% (DCS) and 97% (TES) accuracy (Table 5). Lowest performance was rbfSVM on TSFRESH_FS: 48% (TES), 55% (DCS).

	TES		DCS	
	Whole signal	No-latency	Whole signal	No-latency
Balanced Accuracy				
kNN (1)	0.96 (raw/ ts_fs)	0.93 (raw)	0.97 (raw/ norm)	0.97 (raw/ norm)
kNN (10)	0.91 (ts_fs)	0.82 (norm)	0.90 (ts_fs)	0.88 (norm)
linSVM	0.93 (ts)	0.88 (norm)	0.97 (norm/ts)	0.96 (norm)
rbfSVM	0.89 (norm)	0.89 (norm)	0.93 (raw/ norm)	0.95 (norm)
RF	0.99 (ts)	0.97 (ts)	0.99 (ts)	0.99 (ts)
F1 Score				
kNN (1)	0.97 (raw/ norm/ts_fs)	0.95 (raw/ norm)	0.98 (raw/ norm)	0.98 (norm)
kNN (10)	0.95 (ts_fs)	0.89 (norm)	0.94 (ts_fs)	0.95 (norm)
linSVM	0.96 (ts)	0.91 (raw/ norm)	0.98 (norm/ts)	0.98 (norm)
rbfSVM	0.95 (norm)	0.95 (norm)	0.97 (raw/ norm)	0.98 (norm)
RF	0.99 (ts)	0.98 (ts)	0.99 (raw/ts/ ts_fs)	0.99 (raw/ts/ ts_fs)
MCC				
kNN (1)	0.96 (raw/ ts_fs)	0.92 (raw/ norm)	0.97 (raw/ norm)	0.97 (norm)
kNN (10)	0.93 (ts_fs)	0.85 (norm)	0.92 (ts_fs)	0.92 (norm)
linSVM	0.94 (ts)	0.87 (norm)	0.97 (norm/ts)	0.97 (norm)
rbfSVM	0.92 (norm/ ts_fs)	0.92 (norm)	0.96 (raw/ norm)	0.97 (norm)
RF	0.99 (ts)	0.97 (ts)	0.99 (ts)	0.99 (ts)

Table 5: Peak performances per model and signal representation in Task 1 (intra-patient classification). Representation abbreviations: raw = RAW, norm = NORM, ts = TSFRESH, ts_fs = TSFRESH_FS. The first block reports balanced accuracy.

3.6.2 Task 2: Inter-patient Classification

Task 2.1 (Maximize Patients: 41 patients, 2 muscles)

Two upper limb muscles classification (extensor digitorum communis brevis, abductor pollicis brevis). TES group: Random Forest achieved highest performance

(89% whole signal and no-latency databases on RAW and TSFRESH_FS). DCS group: all models performed similarly high in whole signal database (91–93% on TSFRESH_FS and RAW). No-latency database: Random Forest on TSFRESH and TSFRESH_FS achieved 90% and 89% (Table 6). Lowest performances: linSVM on RAW representation (38–45%).

Task 2.2 (Maximize Muscles: 21 patients, 4 muscles)

Four-muscle classification (biceps brachii, extensor digitorum communis brevis, abductor pollicis brevis, abductor hallucis). TES group: Random Forest achieved highest performance in both databases (83% TSFRESH_FS whole signal, 82% TSFRESH no-latency). DCS group: Random Forest in both databases (78% TSFRESH_FS whole signal, 68% TSFRESH no-latency) (Table 7). Lowest performances: linSVM in both groups and databases (29% to 21% DCS, 18% to 25% TES).

Latency inclusion improved performance for all muscles, especially abductor hallucis (39.23% no-latency to 60.04% whole signal). Misclassification patterns show biceps brachii and abductor hallucis frequently misclassified as extensor digitorum communis brevis, indicating MEP similarities. Abductor pollicis brevis showed distinctive characteristics with lower misclassification rates.

Task 2.1: Inter-patient, 2 muscles, 41 patients				
	TES		DCS	
	Whole signal	No-latency	Whole signal	No-latency
Balanced Accuracy				
kNN (1)	0.82 (ts_fs)	0.72 (raw)	0.89 (ts_fs)	0.73 (raw)
kNN (10)	0.85 (ts_fs)	0.72 (raw/ts)	0.90 (ts_fs)	0.73 (norm)
linSVM	0.88 (ts_fs)	0.87 (ts_fs)	0.91 (ts_fs)	0.86 (ts_fs)
rbfSVM	0.88 (norm)	0.74 (norm)	0.90 (ts_fs)	0.76 (raw)
RF	0.89 (raw/ts_fs)	0.90 (raw/ts)	0.93 (raw)	0.90 (ts)
F1 Score				
kNN (1)	0.82 (ts_fs)	0.71 (raw)	0.89 (ts_fs)	0.71 (raw)
kNN (10)	0.85 (ts_fs)	0.71 (norm)	0.89 (ts_fs)	0.73 (norm)
linSVM	0.88 (ts_fs)	0.87 (ts_fs)	0.91 (ts_fs)	0.85 (ts_fs)
rbfSVM	0.88 (ts_fs)	0.73 (norm)	0.89 (ts_fs)	0.76 (raw)
RF	0.89 (raw/ts_fs)	0.91 (ts)	0.92 (raw)	0.88 (ts_fs)
MCC				
kNN (1)	0.66 (ts_fs)	0.45 (raw)	0.80 (ts_fs)	0.47 (raw)
kNN (10)	0.72 (ts_fs)	0.46 (raw)	0.82 (ts_fs)	0.48 (norm)
linSVM	0.77 (ts_fs)	0.46 (raw)	0.84 (ts_fs)	0.76 (ts_fs)
rbfSVM	0.76 (ts_fs)	0.48 (norm)	0.81 (ts_fs)	0.54 (raw)
RF	0.80 (raw/ts_fs)	0.84 (ts)	0.85 (raw)	0.82 (ts)

Table 6: Peak performances per model and signal representation in Task 2.1 (inter-patient classification). Representation abbreviations: raw = RAW, norm = NORM, ts = TSFRESH, ts_fs = TSFRESH_FS. The first block reports balanced accuracy.

Task 2.2: Inter-patient, 4 muscles, 21 patients				
	TES		DCS	
	Whole signal	No-latency	Whole signal	No-latency
Balanced Accuracy				
kNN (1)	0.76 (ts_fs)	0.55 (raw)	0.73 (ts_fs)	0.49 (raw)
kNN (10)	0.78 (ts_fs)	0.50 (ts)	0.71 (ts_fs)	0.47 (norm)
linSVM	0.76 (ts_fs)	0.69 (ts_fs)	0.67 (ts_fs)	0.54 (ts_fs)
rbfSVM	0.79 (ts_fs)	0.59 (norm)	0.74 (ts_fs)	0.50 (norm)
RF	0.83 (ts_fs)	0.82 (ts)	0.78 (ts_fs)	0.69 (ts)
F1 Score				
kNN (1)	0.72 (ts_fs)	0.57 (raw/norm)	0.70 (ts_fs)	0.49 (raw)
kNN (10)	0.74 (ts_fs)	0.57 (norm)	0.67 (ts_fs)	0.46 (ts)
linSVM	0.70 (ts_fs)	0.69 (ts_fs)	0.63 (ts_fs)	0.57 (ts_fs)
rbfSVM	0.74 (ts_fs)	0.59 (norm)	0.70 (ts_fs)	0.47 (raw)
RF	0.85 (ts)	0.83 (ts)	0.75 (ts)	0.70 (ts)
MCC				
kNN (1)	0.65 (ts_fs)	0.41 (raw)	0.60 (ts_fs)	0.30 (raw)
kNN (10)	0.68 (ts_fs)	0.42 (norm)	0.59 (ts_fs)	0.27 (norm)
linSVM	0.66 (ts_fs)	0.62 (ts_fs)	0.56 (ts_fs)	0.44 (ts_fs)
rbfSVM	0.70 (ts_fs)	0.45 (ts_fs)	0.63 (ts_fs)	0.30 (raw)
RF	0.80 (ts)	0.78 (ts)	0.69 (ts_fs)	0.62 (ts)

Table 7: Peak performances per model and signal representation in Task 2.2 (inter-patient classification). Representation abbreviations: raw = RAW, norm = NORM, ts = TSFRESH, ts_fs = TSFRESH_FS. The first block reports balanced accuracy.

3.6.3 Task 3: Inter-protocol Classification

Combined TES and DCS signals with all six muscles. To prevent subject information leakage in inter-protocol classification, stratified k-fold cross-validation ensured that all MEPs from each patient were grouped together in the same fold, so no patient's signals appeared in both training and test sets. This approach maintains protocol diversity (mixing TES and DCS) while preventing model exposure to test subjects during training.

In the whole-signal database, the best-performing models were Random Forest on TSFRESH, RAW, and TSFRESH_FS, all reaching 78% balanced accuracy (Table 8). The lowest result was obtained by linSVM on TSFRESH (16%). In the no-latency database, Random Forest achieved 66% on TSFRESH, whereas the lowest result was 18%, obtained by linSVM on TSFRESH_FS. Across all three tasks, the differences

among balanced accuracy, weighted F1, and MCC were negligible, indicating consistent performance across evaluation metrics.

	Whole signal	No-latency
Balanced Accuracy		
kNN (1)	70% (ts_fs)	44% (raw)
kNN (10)	72% (ts_fs)	43% (norm)
linSVM	64% (ts_fs)	53% (ts_fs)
rbfSVM	72% (ts_fs)	49% (norm)
RF	78% (raw/ts/ts_fs)	66% (ts)
F1 Score		
kNN (1)	71% (ts_fs)	45% (raw)
kNN (10)	75% (ts_fs)	46% (norm)
linSVM	67% (ts_fs)	57% (ts_fs)
rbfSVM	76% (ts_fs)	51% (norm)
RF	81% (ts)	68% (ts)
MCC		
kNN (1)	65% (ts_fs)	31% (raw)
kNN (10)	70% (ts_fs)	32% (raw/norm)
linSVM	63% (ts_fs)	49% (ts_fs)
rbfSVM	71% (ts_fs)	39% (norm)
RF	76% (raw/ts)	61% (ts)

Table 8: Peak performances per model and signal representation in Task 3 (inter-protocol classification with all six muscles). Representation abbreviations: raw = RAW, norm = NORM, ts = TSFRESH, ts_fs = TSFRESH_FS. The first block reports balanced accuracy.

3.6.4 Expert Classification Performance

Ten experts achieved an overall accuracy of 47.4 ± 11.9% on 50 randomly selected MEPs, whereas the best-performing Task 3 model (mixed protocols, all six muscles) achieved 78% accuracy with Random Forest, substantially exceeding expert performance.

3.7 Discussion

This work shows that machine learning can effectively classify intraoperative MEPs and identify the muscle that generated a given response during neurosurgical procedures. Our evaluation across three experimental scenarios of increasing complexity reveals both the strengths and the limitations of this approach. The best-performing model, Random Forest applied to the whole-signal database with the TSFRESH representation, achieved 93% accuracy on two-muscle upper-limb classification, 83% on four-muscle mixed-limb classification, and 78% on the most challenging six-muscle mixed-protocol scenario. Importantly, this model

substantially outperformed expert neurophysiologists, who achieved 47.4% accuracy on the same task. This gap highlights the potential clinical value of automated approaches.

Interpretation of biological electrical signals such as MEPs is inherently challenging because of their complexity and their marked variability both within and across subjects. Much of this variability arises from individual anatomical and physiological differences, anesthesia effects, and the unavoidable perturbations introduced by surgical manipulation. Despite this complexity, classical machine learning algorithms proved capable of extracting discriminative information that substantially exceeds what can be achieved by unaided visual inspection. Whereas human operators may be affected by fatigue, distraction, or cognitive biases that accumulate during hours of surgical monitoring, algorithms apply consistent decision rules to every signal. This finding has direct clinical implications: automated muscle identification could serve as a robust decision-support tool for verifying correct muscle-electrode labeling during intraoperative monitoring setup, thereby helping prevent labeling errors with potentially severe consequences for patient safety.

An unexpected finding emerged regarding latency, a parameter considered crucial by trained neurophysiologists. We observed minimal performance differences between whole-signal (including latency) and no-latency (waveform-only) databases in most classification scenarios. In fact, two-muscle and four-muscle TES classification showed negligible differences, and models occasionally performed slightly better without latency when discriminating muscles from the same limb. However, latency retained informative value in Task 3, the most difficult setting: the whole-signal set improved performance by approximately 11% compared with the waveform-only representation. This differential importance of latency across tasks suggests that, while waveform shape carries sufficient discriminative information for simpler classification problems, temporal information becomes valuable when discrimination must occur across dissimilar contexts (mixed protocols and distributed anatomical regions).

Raw signals and mathematically derived features (TSFRESH) consistently outperformed traditional neurophysiological parameters (TRAD). This result suggests that machine learning algorithms capture information in forms that differ fundamentally from those traditionally used by human experts. Whereas traditional features describe human-interpretable concepts such as amplitude and area, TSFRESH captures statistical and temporal dynamics that may lack an intuitive neurophysiological interpretation but remain highly discriminative. This divergence is neither surprising nor problematic; rather, it reflects the fact that algorithms and humans may excel at different forms of pattern recognition. For clinical deployment, TSFRESH also offers practical advantages: its compact representation can improve generalization to unseen data by mitigating the curse of dimensionality, and feature-selection methods make the most relevant features explicit, thereby improving explainability.

Despite the apparent clinical relevance of automated MEP analysis, the literature on machine learning applications to intraoperative neurophysiology remains sparse. Prior work by Wermelinger and colleagues [132] explored standard machine learning approaches for transcranial MEP classification with 36 patients, achieving 89% accuracy on two-muscle discrimination, 97% on different limbs, and 83% on four-muscle classification. Our work advances this research in two important directions. First, we systematically investigated signal representation effects throughout the machine learning pipeline. While prior work applied classification to preprocessed signals using standard feature choices, we conducted comprehensive evaluation of five distinct signal representations applied to the same models and data. This revealed that advanced time series feature extraction substantially improved accuracy while reducing computational requirements and enhancing interpretability. Second, we expanded the scope of evaluation across multiple challenging scenarios: beyond quantifying how easily models generalize to new subjects (leave-one-patient-out cross-validation), we evaluated model performance within single subjects and across different stimulation protocols. Crucially, we evaluated both complete signals and waveform-only representations, demonstrating the possibility of muscle discrimination using waveform characteristics alone, a capability enabling broader clinical application in conditions where latency may be compromised but waveform characteristics remain intact.

The clinical potential of automated muscle identification extends substantially beyond initial labeling verification. Real-time classification could enhance interpretation of signal changes throughout the surgical procedure, providing explicit alerts when unexpected muscle responses occur or when expected responses disappear. The capability to rapidly detect anatomical variations, such as anomalous patterns, could enable rapid protocol adjustment during surgery.

3.8 Conclusion

This chapter evaluated machine learning approaches for automated muscle identification from intraoperative MEPs. Random Forest classifiers achieved 99% accuracy on intra-patient classification, 83–89% on inter-patient tasks, and 78% on the most challenging multi-muscle, multi-protocol scenario, substantially exceeding human expert performance (47.4%). Waveform characteristics alone often suffice for muscle identification, while advanced time series feature extraction (TSFRESH) outperformed traditional neurophysiological parameters.

These results have direct clinical implications: automated muscle identification could verify electrode labeling during IOM setup, detect real-time signal changes, and identify unexpected response patterns. Algorithm consistency contrasts favorably with human susceptibility to fatigue and error, offering significant potential to enhance patient safety.

Future work should address several directions to advance clinical deployment. Access to datasets from multiple institutions would increase cohort size beyond the current 54 patients and enhance generalizability across different equipment,

protocols, and patient populations. Finally, extending these methods to other intraoperative monitoring applications, such as somatosensory evoked potential analysis, could expand the scope of automated decision support in neuromonitoring.

4 Detection and Early Detection of Pathological Motor Evoked Potentials

Following the automated muscle identification task in Section 3, we now address the complementary problem of detecting and anticipating pathological changes in intraoperative MEPs during monitoring.

In this section, we provide a detailed explanation of our approach to intraoperative MEP anomaly detection. First, we formally define the detection and early detection tasks, establishing the problem formulation both from the medical (see Section 4.1) and computational (see Section 4.1.1) point of view. Then, in Section 4.2, we introduce our Canonical Isolation Forest (CISOF), which serves as the foundation for our anomaly detection framework.

4.1 Medical Problem

Motor evoked potentials (MEPs) are routinely used in intraoperative monitoring (IOM) to assess the integrity of motor pathways during neurosurgical procedures (see Section 3.2 for a clinical overview). After anesthesia induction and before major surgical manipulation, a baseline MEP response is acquired and used as a reference. As surgery progresses, newly recorded MEPs are compared against this baseline to detect changes that may indicate potential neurological injury and warrant immediate corrective action.

Because MEPs are influenced by multiple confounding factors (e.g., anesthesia, physiological variability, surgical manipulation, and technical artifacts), robust automated methods for detecting and anticipating pathological changes are clinically valuable. In the next section (Section 4.1.1), we formalize this problem from a computational perspective, defining the detection and early-detection tasks.

4.1.1 Machine Learning Formalization

We define a dataset of intraoperative motor evoked potentials (MEPs) as:

$$D = \{x_1, x_2, ..., x_N\} \tag{24.}$$

where N is the number of MEPs in D. Each x_i represents a MEP signal:

$$x_i = \left\{v_1^i, v_2^i, ..., v_{l^i}^i\right\} \tag{25.}$$

where v_j^i denotes the value of the i-th signal at time step j, and l^i denotes the length of the i-th time series. The dataset D consists of all MEP time series recorded before a reference time point λ, corresponding to the phase before the procedure starts, in which baseline patterns are learned. In the remainder of this section, we introduce both the Detection and Early Detection tasks.

After λ, a new time series is recorded:

$$x_{TEST} = \left\{v_1, v_2, ..., v_{\{l^{TEST}\}}\right\} \tag{26.}$$

where l^{TEST} is the length of the newly observed MEP time series.

The objective is to determine whether x_{TEST} exhibits anomalous behavior when compared to the learned distribution of normal MEPs in D. Specifically, we seek to develop a model m that, given D and x_{TEST}, outputs a binary decision:

$$m(D, x_{TEST}) = \begin{cases} 1 \text{ if } x_{TEST} \text{ is an anomalous MEP} \\ 0 \text{ if } x_{TEST} \text{ is a normal MEP} \end{cases}, \tag{27.}$$

An anomalous time series is defined as one that deviates significantly from the learned distribution of normal MEPs in D, capturing potential pathological intraoperative events that could indicate neurological injury. Our approach extends Isolation Forest to the time series domain, as it is a state-of-the-art method for anomaly detection [133, 134]. This unsupervised algorithm is particularly suited to MEP monitoring because it requires only baseline normal data, produces interpretable anomaly scores suitable for clinical decision support, and scales efficiently to real-time surgical scenarios. While we evaluate multiple baseline methods, including distance-based approaches (such as LOF-cDTW), we exclude TSRF-Dist (a novel distance measure introduced in Section 5), since the isolation-based framework provides a more appropriate foundation for this application. Our method aims to identify such anomalies early, providing real-time alerts to neurophysiologists and surgeons for timely intervention.

This example in Figure 19 illustrates a real but straightforward case where the anomaly is visually evident. However, in practical scenarios, distinguishing anomalies is more challenging due to the high variability of MEP signals. Hence, developing robust models capable of accurate detection is crucial for intraoperative neurophysiological monitoring.

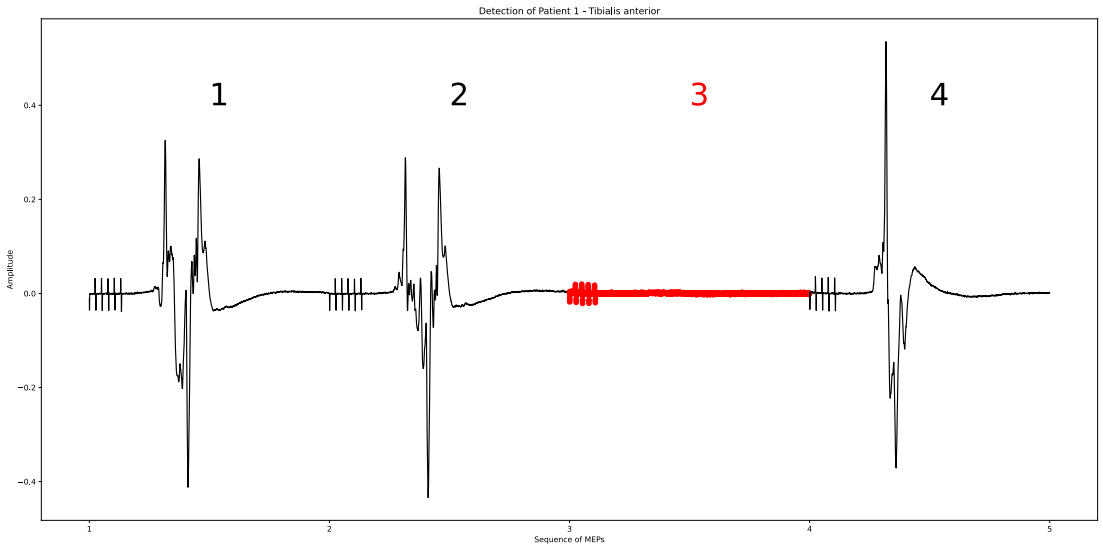

Figure 19: Example of MEP sequences in the testing phase. Each MEP is numbered according to its temporal order. The MEP labelled as #3 (highlighted in red and with a bigger line width) is an anomaly. In the detection task, the goal is to determine whether MEP #3 is anomalous.

CISOF operates under two distinct tasks: Detection and Early Detection. While the Detection task follows the standard formulation described in Equation 27, the Early Detection task aims to predict future anomalies before they manifest. Rather than determining whether the current MEP time series is anomalous, the goal is to predict whether the next MEP time series will be anomalous. Formally, given an observed MEP time series x_t, the model predicts the anomaly status of the subsequent time series $x_{\{t+1\}}$, thereby enabling proactive intervention. The model m' is trained to learn the temporal dependencies between consecutive MEP signals and outputs:

$$m'(D, x_t) = \begin{cases} 1 \text{ if } x_{TEST} \text{ is expected to be anomalous} \\ 0 \text{ if } x_{TEST} \text{ is expected to be normal} \end{cases}, \tag{28.}$$

The early detection task is particularly crucial in intraoperative monitoring, as it allows for timely intervention before significant neurological deterioration occurs. By leveraging the temporal structure of MEP sequences, the model can anticipate pathological deviations, reducing the risk of irreversible damage.

4.2 Canonical Isolation Tree and Forest

This section presents the forest-based approach: we first introduce how a tree works and then how to build a forest of this specific tree. The Canonical Isolation Forest (CISOF) represents an ensemble of Canonical Isolation Trees (CISOT).

A CISOT is a binary tree structure in which each node has either no children — making it a leaf — or exactly two children — making it an internal node (an illustration is provided in Figure 20). As in interval-based forests such as TSF and CIF [19, 20], each internal node applies a test $t = (f, v, I)$ to a time series $x = \{v_1, v_2, ..., v_l\}$: the interval I is extracted from x, a feature f is computed on that interval, and the result x_I^f is compared with a threshold v to send the series to the left or right child. If fixed-length series are assumed, $I = [I_s, I_e]$ can be defined through absolute positions; for variable-length series, it can be specified relative to the series length.

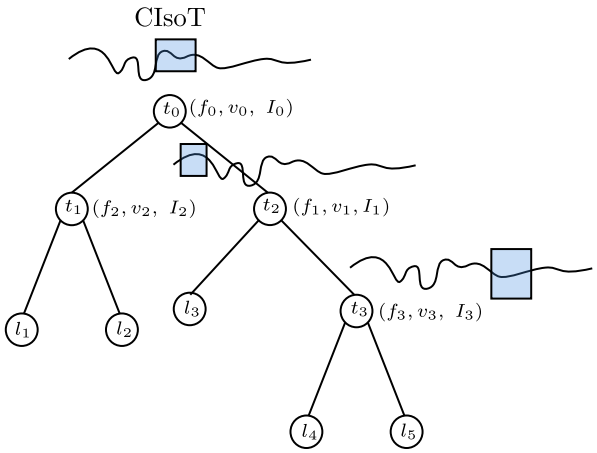

Figure 20: The Canonical Isolation Tree. In each node n the decision is taken i) by extracting the interval I_n from the time series (highlighted in light blue); ii) by

extracting from it the feature f_n; iii) by checking if $x_{\{I_n\}}^{\{f_n\}} < v_n$, making the time series following the left branch if true, to the right one if false.

4.2.1 Features

The feature f is chosen from a set of 25 measures that can be computed from a time series. Specifically, the first 22 are drawn from CATCH22 [1] (summarized in Table 2), while the remaining three — mean, standard deviation, and slope of the least squares regression line — follow the Time Series Forest classifier [19]. This is the same feature pool later used in the forest-based distance model of Section 5.

4.2.2 Training of a CISOT

The Canonical Isolation Tree (CISOT) is constructed using a standard top-down, recursive approach. Beginning with the full training set $D = \{x_1, ..., x_N\}$, consisting of N time series, a test $t = (f, v, I)$ is selected for the root node. This test partitions D into two subsets, D_L and D_R, based on the outcomes of the test for each time series. These subsets are then used to recursively define the tests for the left and right child nodes. The recursion continues until a predefined maximum tree height is reached or until a node contains fewer than a minimum number of samples. Since searching for the "best" question cannot be performed using traditional criteria (such as the Gini criterion in classification trees [135]) due to the absence of labels, we implement two distinct split methods. The first follows the standard random selection approach used in the Isolation Forest [27], where a feature and a split threshold are chosen at random. The second method aims to minimize variance, as proposed in [28]. Specifically, we select a feature and evaluate a limited number of candidate thresholds (as commonly done in the RF paradigm [26]), choosing the one that minimizes the sum of variances in the resulting partitions D_L and D_R. This variance-minimizing approach ensures that the data is divided into more homogeneous subsets. More in detail, in our CISOT, given a set D_n of time series reaching the node n, the test $t = (f, v, I)$ of that node is determined in the following way:

- The interval I is selected from a group of randomly generated intervals, with a different group created for each tree. Intervals are generated by randomly choosing starting points and lengths[1].
- Feature f is randomly selected from a random subset of 25 pre-defined features.
- A split threshold v in the range $(f_{\min}, f_{\max})$ is selected differently based on the split method chosen. Once defined $f_{\min} = \min_{j=1...N} \left(x_j\right)_I^f$ and $f_{\max} = \max_{j=1,...N} \left(x_j\right)_I^f$, we select v in two ways[2]:
  - randomly in $(f_{\min}, f_{\max})$,
  - minimizing the variance of the children by $\arg\min_{v \in (f_{\min}, f_{\max})} \left(\sigma^2_{\{D_L\}} + \sigma^2_{\{D_R\}}\right)$. The pseudocode is provided in Algorithm 7.

[1] Note that certain CATCH22 features require the interval to have a minimum length of 20.

[2] $\left(x_j\right)_I^f$ represents the value of feature f computed on the subseries corresponding to interval I, extracted from the time series x_j.

MinVarianceSplit: Obtaining best split threshold(D_I^f):

let $minVariance \leftarrow \inf$
for v **in** D_I^f:
    let $D_L \leftarrow \left\{x \in D \mid x_I^f < v\right\}$
    let $D_R \leftarrow \left\{x \in D \mid x_I^f \geq v\right\}$
    let $variance \leftarrow |D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R)$
    if $variance < minVariance$:
        let $v_{best} \leftarrow v$
        let $minVariance \leftarrow variance$
return v_{best}

Algorithm 7: Training Set D_I^f.

The complete training procedure for the CISOT is provided in Algorithm 8.

buildCISOT: Fit a CISOT(D, h, hl, F, IS, m):

if $h \geq hl$ **or** $|D| < 2$:
    return $leaf\{size : |D|\}$
else:
    let $D_L \leftarrow \emptyset$
    let $D_R \leftarrow \emptyset$
    let Select a random interval $I \in IS$
    let Select a random feature $finF$
    let D_I^f is the set composed by computing x_I^f **for** every $x \in D$
    if $m = minvariance$:
        let $v = \text{MinVarianceSplit}\left(D_I^f\right)$
    else:
        let $min_v = \min\left(D_I^f\right)$
        let $max_v = \max\left(D_I^f\right)$
        let $v = \text{RandomSample}(min_v, max_v)$
        // Perform the split using v as threshold
    let $D_L \leftarrow \left\{x \in D \mid x_I^f < v\right\}$
    let $D_R \leftarrow \left\{x \in D \mid x_I^f \geq v\right\}$
    return node $\begin{cases} \text{left: buildCISOT}(D_L) \\ \text{right: buildCISOT}(D_R) \\ \text{split\_feature:} f \\ \text{split\_value:} v \\ \text{interval:} i \end{cases}$

Algorithm 8: Training Set D, Current Height h, Height Limit hl, CATCH22 Feature Subset F, Interval Subset IS, Split Method m

4.2.3 Training of a CISOF

The CISOF (see Algorithm 9) represents an ensemble of CISOTs. Following the classical procedure of Isolation Forest, each CISOT is created, with the procedure described previously, starting from a random subsampling of the training set. We need to set n_f, which represents the cardinality of the subset of CATCH22 features to randomly sample. This approach aligns with the common practice in the random forest paradigm, where each tree is trained on a random subsample of the features to enhance diversity and robustness. Finally, n_I, which represents the number of intervals generated for each tree.

buildCISOF: Fit a CISOF(D, n_t, n_I, n_f, m):

let $Forest \leftarrow []$
for $i \leftarrow 0$ **to** n_t:
    let $D_j \leftarrow \text{Subsample}(D)$
    $hl \leftarrow \log_2(|D_j|)$ // generate a subsample of D
    let $F_j \leftarrow \text{SampleFeatures}(n_f)$
    let $I_j \leftarrow \text{SampleInterval}(n_I)$
    let $Forest[j] \leftarrow$ buildCISOT$(D_j, 0, hl, F_j, I_j, m)$
    // build a tree and add it to the forest
    return Forest

Algorithm 9: Training Set D, Number of trees n_t, Number of intervals n_I, Number of features n_f, Split Method m

4.2.4 Detection of Anomalies

Once the model is trained, we can detect anomalies by thresholding time series according to their anomaly scores, and anomalies are time series that produce a higher score. We define anomaly score (AS - see Equation 29) as in the work of Liu et al. [27]. The isolation mechanism exploits the fact that anomalies are easier to isolate than normal instances. Isolation is achieved by recursively partitioning the data using the test splits. Since anomalies are few and different, they tend to be isolated in fewer splits, leading to shorter path lengths in the trees. In contrast, normal instances require more splits due to their higher density and similarity to other points. This fundamental property allows the model to distinguish anomalies based on their expected path length in the constructed trees.

$$AS(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{29.}$$

Where x is a time series, n represents the size of the subsample set used in the training phase, $E(h(x))$ is the average depth of x from a CISOF and $c(n)$ the average path length of unsuccessful search in a Binary Search Tree [136].

4.2.5 Early Detection of Anomalies

To better understand how the mechanism works for early detection, as detailed in Equation 29, we attempt to retrieve the anomaly score of the MEP that precedes the actual anomaly. By doing so, we aim to distinguish between the anomaly scores of a normal MEP and those of MEPs that occur just before a true anomaly. In this way, we make an early prediction, attempting to anticipate the anomaly before it occurs, as detailed in Section 4.3.1.

4.3 Experimental Details

4.3.1 Datasets Creation

The dataset used in this study consists of data collected from four patients undergoing neurosurgical procedures. The protocol employed to elicit MEPs is transcranial electrical stimulation (TES). MEPs were evoked via transcranial anodal electrical stimulation with corkscrew electrodes placed on the scalp at C1 and C2 in the 10-20 International System. The high-frequency short train technique was used by applying a train of five pulses with an interstimulus interval of 4 ms. Intensity ranged from 50 to 150 mA. MEPs were recorded from the same target muscles in all patients, with pairs of subdermal needle electrodes placed into the muscle belly of abductor pollicis brevis (APB) for upper limbs and tibialis anterior (TA) and abductor hallucis (AH) for lower limbs. All procedures were performed in compliance with institutional guidelines and have been approved by the appropriate institutional committee. The study obtained IRB approval with protocol ID: CRMS 23038 on June 5, 2024. These muscles were chosen as they are the only ones consistently present across all patients, ensuring uniformity in the dataset. For each patient-muscle combination, we created separate training and testing datasets for the task of anomaly detection. Figure 21 illustrates that simple statistical measures, such as the mean, are insufficient to effectively characterize an MEP. As shown in Figure 21, the red crosses represent anomalies, but relying on basic statistics does not provide a clear linear separation between normal and anomalous data.

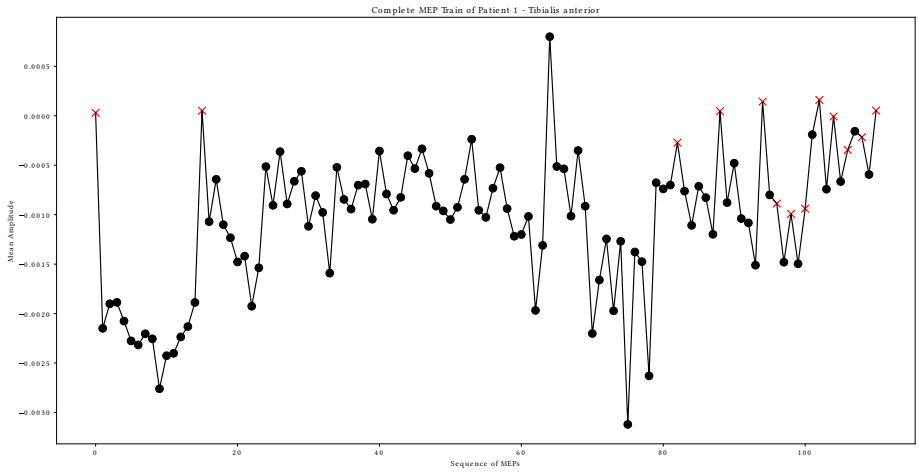

Figure 21: Plot of the mean amplitude for each MEP in the testing set of the Detection task, following the temporal order of the procedure. Red crosses indicate anomalies. This plot highlights that simple metrics cannot fully support the surgeon during the procedure.

The methodology used to split the signals was designed to simulate real-world operating-room conditions as closely as possible. The first anomaly within the sequence of MEP signals recorded during the procedure was identified and used as a reference point. The MEP signals preceding this event constituted the training

dataset, while those following it, including the anomaly itself, formed the testing dataset. This approach preserves the natural chronological order of events, ensuring that the training and testing sets reflect a real-time clinical scenario.

To implement early detection, the datasets were further processed. Specifically, the first MEP signal in the testing set was removed and replaced with the last signal from the training set, which was then marked as anomalous. In this way, we labelled the signal immediately preceding an anomaly as anomalous, while removing the actual anomaly from the dataset. This transformation simulates an early-detection setting in which the system learns to recognize warning signs before a full anomaly occurs. We then repeated this procedure sequentially for all subsequent anomalies.

Table 9 and Table 10 summarize the datasets for the Detection and Early Detection tasks, respectively. In each cell, the first number is the number of samples in the training set, the second is the number of samples in the test set, and the third is the proportion of anomalies in the test set.

Patient	AH	APB	TA
1	130 - 366 (0.16)	25 - 94 (0.06)	130 - 111 (0.12)
2	16 - 239 (0.18)	12 - 326 (0.17)	33 - 616 (0.01)
3	11 - 21 (0.38)	13 - 145 (0.01)	19 - 77 (0.12)
4	11 - 868 (0.13)	41 - 1571 (0.04)	12 - 148 (0.30)

Table 9: Cardinality of the datasets for detection.

Patient	AH	APB	TA
1	129 - 308 (0.19)	24 - 89 (0.07)	129 - 99 (0.13)
2	15 - 198 (0.21)	11 - 270 (0.21)	32 - 609 (0.01)
3	10 - 14 (0.57)	12 - 145 (0.01)	18 - 69 (0.13)
4	10 - 757 (0.15)	40 - 1505 (0.04)	11 - 105 (0.42)

Table 10: Cardinality of the datasets for early detection

4.3.2 CISOF in detail

In this section, we outline the experimental setup used to evaluate the proposed method, detailing the configurations, split methods, evaluation metric, and repetition strategy employed. We designed three CISOF configurations to evaluate the trade-off between computational efficiency and predictive performance: **H**eavy, **L**ight, and **U**ltra **L**ight.

- In the **H**eavy configuration, we set the number of trees (n_t) to 500, the number of features (n_f) to 4, and the number of intervals (n_i) in I to $\texttt{sqrt}\{N\}$, where N is the sequence length.
- In the **L**ight configuration, $n_t = 200$, $n_f = 4$, and $n_i = \log_2\{N\}$.
- In the **U**ltra **L**ight configuration, $n_t = 100$, $n_f = 4$, and $n_i = \log_{\{10\}}\{N\}$.

We evaluated two different split methods (both described in Section 4.2):

- Random: standard method in Isolation Forest [27].

- MinVariance: unsupervised split technique that can be optimized.

To evaluate the performance of our method, we used the Receiver Operating Characteristic Area Under the Curve (ROC-AUC), a widely used metric for anomaly detection problems [27, 137]. For all the statistical tests we set the value of α to 0.05. Finally, to assess the impact of randomization on model performance, we repeated all experiments ten times for each dataset, configuration, and split method, always reporting the average over these repetitions. This procedure allows us to quantify how the inherent randomization affects the performance of the proposed method.

4.4 Results and Discussion

4.4.1 Analysis of the different variants of CISOF

As an initial analysis, we compared the different configurations of CISOF to evaluate their performance and computational efficiency.

	UL	L	H
Detection	0.6991	0.7243	0.7720
Early Detection	0.5437	0.5833	0.5908

Table 11: Average ROC-AUC of the configuration (UL, L, H) for Detection and Early Detection tasks

Table 11 compares the three configurations in terms of performance on the detection and early-detection tasks. The reported values correspond to the average ROC-AUC across all patient-muscle combinations. The **H**eavy configuration achieves the best performance on both tasks, whereas the **U**ltra **L**ight configuration yields the lowest scores. The **L**ight configuration offers a compromise between computational efficiency and predictive performance, improving on the **U**ltra **L**ight variant while maintaining a lower computational burden than the **H**eavy configuration.

To assess the statistical significance of our comparisons, we conducted a Friedman test, followed by a post-hoc pairwise analysis using Nemenyi's test for multiple comparisons of mean rank sums [138]. The Friedman test determines whether there are significant overall differences in the rankings of the evaluated methods across different datasets. If a significant difference is found, Nemenyi's test is applied to examine pairwise differences in mean ranks. The results are then visualized using a critical difference diagram (see Figure 22), where the mean ranks of the methods across datasets are displayed. The red lines connect rankings that do not present statistically significant differences.

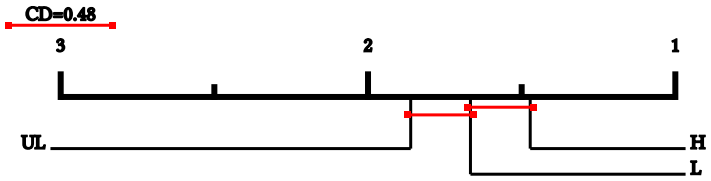

Figure 22: Critical Difference Diagram comparing the ROC-AUC of the 3 configurations: **UL**, **L**, **H** over all datasets.

Figure 22 shows the Critical Difference diagram, where configurations are ranked according to their average performance across datasets. The numbers 1–3 indicate the average ranks, increasing from best to worst (left to right). Two configurations must differ by at least 0.48 (the critical difference) to be considered statistically different. The **H**eavy and **L**ight configurations, which achieve the best performance, do not exhibit statistically significant differences. By contrast, the **U**ltra **L**ight configuration is statistically different from the **H**eavy configuration. Overall, the **L**ight configuration represents a good compromise between performance and computational requirements.

In the next part, we present the performance of CISOF models for both detection and early detection tasks, using the best-performing configuration. Two tables summarize the results: Table 12 for detection and Table 13 for early detection. Each table includes both split methods explained above, hereinafter denoted CISOF-R and CISOF-MV for the Random and MinVariance variants, respectively. Each table evaluates the models over four patients (rows 1–4) and the three muscle groups previously described. Overall, the proposed model performs well, achieving ROC-AUC values greater than 0.80 in 14 out of 24 datasets in the Detection task, as visible in Table 12. Additionally, the MinVariance variant demonstrates strong performance, obtaining a ROC-AUC higher than 0.80 in 12 datasets. In the next section, we provide a detailed analysis of these results to further examine the model's behavior across different scenarios. Table 12 - **Random** displays the performance of CISOF-R. The highest scores, on average, are observed for APB, with values ranging from 0.8034 to 0.9645. APB also achieves consistently high performance, with scores exceeding, on average, 0.80. However, TA shows significantly lower scores, particularly in patients 3 and 4, where it falls around 0.50. Table 12 - **MinVariance** represents the performance of CISOF-MV, which aims to improve the quality of child nodes. This modification leads to substantial performance gains across most metrics, particularly for TA and APB. For instance, TA improves from 0.4188 for patient 3 to 0.6326, while APB sees consistent

improvements across all patients, with scores above 0.85. These results suggest that minimizing the variance in splits better captures the structure of the underlying data, leading to improved detection performance.

Random				**MinVariance**			
	APB	TA	AH		APB	TA	AH
1	0.9645	0.7062	0.9824	1	0.9729	0.9038	0.9871
2	0.9482	0.6192	0.6419	2	0.9705	0.7256	0.8831
3	0.8034	0.4188	0.9708	3	0.8538	0.6326	0.9873
4	0.8395	0.5164	0.6459	4	0.9596	0.6160	0.6362

Table 12: ROC-AUC comparison between CISOF-R and CISOF-MV on all patient-muscle combinations in the Detection task.

Table 13 evaluates the models' ability to perform early detection. Table 13 - **Random** shows the performance of CISOF-R in this setting. While APB and AH maintain reasonable scores (e.g., 0.8232 and 0.7217 for patient 1), the performance for TA is relatively low, particularly in patients 2 and 4, with scores below 0.55.

Table 13 - **MinVariance** demonstrates the performance of CISOF-MV for early detection. Compared to Table 13 - **Random**, the scores for TA and AH show notable improvements. For example, TA increases from 0.4522 to 0.5088 in patient 2, and APB achieves more consistent performance across all patients.

Random				**MinVariance**			
	APB	TA	AH		APB	TA	AH
1	0.8232	0.5994	0.7217	1	0.8126	0.6408	0.7237
2	0.7262	0.4522	0.5572	2	0.6848	0.5088	0.5925
3	0.5448	0.8681	0.7368	3	0.6771	0.9465	0.6327
4	0.5936	0.5386	0.5734	4	0.6526	0.5886	0.5498

Table 13: ROC-AUC comparison between CISOF-R and CISOF-MV on all patient-muscle combinations in the Early Detection task.

Compared to the detection task, where our model achieved strong performance with high ROC-AUC scores, the early detection task proved to be more challenging, as expected. While CISOF exceeds a ROC-AUC of 0.70 in 8 out of 24 datasets, and the random split variant does so in 5 out of 12 cases, these results are less consistent than in the detection setting. These are promising results, demonstrating that our approach achieves good predictive capability even in this more difficult scenario, where identifying anomalies at an earlier stage is inherently more challenging.

We compared the two different split methods of CISOF, namely **Random** and **MinVariance**. Specifically, we averaged the results across the three previously described configurations and all datasets. These results are presented in Table 14. To determine whether the observed differences were statistically significant, we applied

the Wilcoxon signed-rank test [139], a non-parametric test that evaluates whether two paired samples drawn from related distributions differ significantly. The test results indicate a significant difference between the **MinVariance** and Random splitting strategies. Moreover, **MinVariance** consistently outperformed Random, confirming its statistical superiority.

	Random	**MinVariance**
Detection	0.6855	0.7781
Early Detection	0.5772	0.5681

Table 14: Average ROC-AUC of the split methods (**Random** and **MinVariance**) for Detection and Early Detection

4.4.2 Comparison with literature alternatives

In this section, we present the results of our experiments comparing CISOF with alternative approaches to abnormal time series detection [137]. Since no standard solution exists for this problem, we evaluate our method against several competing strategies. We identified a related formulation in [140], but, as discussed in the introduction, deep learning techniques are not suitable for our setting. We therefore considered two families of alternatives. The first transforms time series into feature vectors using statistical and temporal descriptors, specifically CATCH22 [1], and then applies standard anomaly-detection models. The second computes distances between time series, in our case using constrained Dynamic Time Warping (cDTW) with a constraint value equal to 5% of the time series length, and then applies a distance-based anomaly-detection model.

The selected models cover a diverse range of anomaly detection methodologies. One-Class Support Vector Machine (OC-SVM) [141] follows a boundary-based approach, learning a hyperplane that encapsulates normal data distribution in a high-dimensional space. Subspace Outlier Detection (SOD) [142] is designed for high-dimensional datasets, detecting anomalies within relevant subspaces. LOF-cDTW [13, 143] extends the classic Local Outlier Factor (LOF) by incorporating cDTW as a dissimilarity measure for time series data. Histogram-Based Outlier Score (HBOS) [144] is a non-parametric method that models feature distributions using histograms and assigns anomaly scores based on deviations from expected densities. Empirical Cumulative Distribution Outlier Detection (ECOD) [145] is a fully unsupervised approach that estimates the empirical cumulative distribution function and detects anomalies based on extreme deviations. Finally, Minimum Covariance Determinant (MCD) [146] is a robust statistical technique that identifies anomalies by computing the most representative subset of the data and detecting points that significantly deviate from it. We utilized the implementations available in the PyOD Python library, which provides a comprehensive suite of anomaly detection tools.

Table 15 presents the ROC-AUC comparison between our proposed model and the other anomaly detection methods on the Detection Task. Our approach outperforms all competitors in 9 out of 12 cases, demonstrating its superior performance across

most scenarios. Among the alternative models, ECOD achieves the second-best results overall, performing best in 3 cases.

		IForest	OC-SVM	SOD	LOF-cDTW	HBOS	ECOD	MCD	CISOF-R	CISOF-MV
1	AH	0.9169	0.7245	0.6468	0.6601	0.8132	0.9474	0.9195	0.9824	**0.9871**
2		0.6135	0.3639	0.3520	0.1053	0.8131	0.8106	0.3816	0.6419	**0.8831**
3		0.6964	0.5441	0.6340	0.2304	0.4085	0.8431	0.4613	0.9708	**0.9873**
4		0.5396	0.5101	0.5227	0.5804	0.5780	0.6014	0.5967	**0.6459**	0.6362
1	APB	0.8965	0.8908	0.2195	0.5062	0.8270	0.8551	0.7378	0.9645	**0.9729**
2		0.7869	0.4181	0.4037	0.4152	0.4364	0.7368	0.5506	0.9482	**0.9705**
3		0.3519	0.5000	0.5192	0.2692	0.8269	0.5769	0.6000	0.8034	**0.8538**
4		0.4198	0.5354	0.5736	0.4115	0.3308	0.5514	0.6974	0.8395	**0.9596**
1	TA	0.2767	0.4110	0.5663	0.3617	0.2841	0.6951	0.5928	0.7062	**0.9038**
2		0.6837	0.5262	0.6423	0.3893	0.2809	**0.8055**	0.7453	0.6192	0.7256
3		0.0389	0.0139	0.5694	0.0069	0.0278	**0.7569**	0.1250	0.4188	0.6326
4		0.6076	0.6160	0.4414	0.5524	0.7327	**0.7608**	0.6969	0.5164	0.6160
Mean		0.5690	0.5045	0.5076	0.3741	0.5299	0.7451	0.5921	0.7548	**0.8440**

Table 15: ROC-AUC comparison of CISOF-R, CISOF-MV, and the competing models on the Detection task.

Table 16 presents the ROC-AUC comparison for the Early Detection Task, which is the most critical evaluation in our study. CISOF achieves the highest performance in 8 out of 12 cases. In contrast, ECOD wins 2 times, while LOF-cDTW and SOD achieve the highest score in just a single case each. This further reinforces the advantage of our approach over traditional anomaly detection methods. On average, CISOF-MV achieves a mean ROC-AUC of 0.6675, which is approximately 7% higher than ECOD's mean score of 0.5969, further highlighting its superior effectiveness in early anomaly detection.

		IForest	OC-SVM	SOD	LOF-cDTW	HBOS	ECOD	MCD	CISOF-R	CISOF-MV
1	AH	0.5794	0.4267	0.5653	0.5818	0.6011	0.6950	0.5699	0.7217	**0.7237**
2		0.4342	0.3696	**0.7169**	0.4056	0.6624	0.6741	0.3221	0.5572	0.5925
3		0.5561	0.5500	0.5352	0.4639	0.4444	0.5148	0.6572	**0.7368**	0.6327
4		0.5000	0.4784	0.6043	**0.6393**	0.4620	0.5674	0.5325	0.5734	0.5498
1	APB	0.5900	0.6778	0.6072	0.6784	0.5664	0.6315	0.6571	**0.8232**	0.8126
2		0.6771	0.5618	0.5737	0.5743	0.6235	0.6824	0.5748	**0.7262**	0.6848
3		0.3740	0.5000	0.3958	0.6667	0.5000	0.5833	0.5312	0.5448	**0.6771**
4		0.3959	0.4978	0.4482	0.2720	0.4208	0.4737	0.4835	0.5936	**0.6526**
1	TA	0.5534	0.5843	0.6747	0.4297	0.5341	**0.6847**	0.6767	0.5994	0.6408
2		0.4618	0.5193	0.5842	0.4308	0.4139	**0.6836**	0.6394	0.4522	0.5088

3		0.0333	0.0069	0.6181	0.0139	0.0347	0.3958	0.0431	0.8681	**0.9465**
4		0.5086	0.5281	0.4987	0.5363	0.5009	0.5764	0.5841	0.5386	**0.5886**
Mean		0.4720	0.4751	0.5685	0.4744	0.4804	0.5969	0.5227	0.6446	**0.6675**

Table 16: ROC-AUC comparison of CISOF-R, CISOF-MV, and the competing models on the Early Detection task.

To statistically assess the performance of CISOF, we selected its best variant (CISOF-MV) and compared it against the best competitor ECOD. We then conducted a Wilcoxon signed-rank test to evaluate whether CISOF-MV significantly outperforms the alternative. The test results confirmed that CISOF-MV passed the significance threshold, demonstrating that it is statistically superior to the best competing model.

4.5 Conclusions

This study presents a novel approach to the detection and early detection of pathological intraoperative motor evoked potentials by introducing the Canonical Isolation Forest (CISOF), an extension of Isolation Forest specifically tailored to time series biomedical signals. Our results demonstrate that the proposed method effectively identifies deviations from patient-specific baseline MEP patterns and provides meaningful early warning signals prior to clinically confirmed pathological events. A key strength of the proposed framework is its compatibility with real intraoperative constraints. CISOF operates in a patient-specific, unsupervised manner, requires only baseline data acquired at the beginning of the procedure, and produces interpretable anomaly scores without relying on real-time labeling, online retraining, or automated decision-making. The system is intended as a decision-support tool that augments, rather than replaces, the expertise of neurophysiologists and surgeons. Nevertheless, several limitations should be acknowledged. First, the study is based on a limited number of patients, reflecting the intrinsic difficulty of collecting intraoperative MEP data. While our results are encouraging, larger and more diverse cohorts are required to fully assess robustness across surgical types, anatomical targets, and anesthetic protocols. Second, early detection is evaluated retrospectively through an anticipatory labeling protocol; prospective validation is necessary to confirm the utility of early warning signals in real-time surgical decision-making. Future work will investigate prospective clinical validation, integration with multimodal intraoperative monitoring signals, adaptive threshold selection strategies, robustness to non-pathological physiological variations, and the exploration of distance-based anomaly detection methods as a complementary approach. These directions are essential to bridge the gap between methodological development and safe, effective deployment in clinical practice.

5 TSRF-Dist: A novel Time Series distance

After addressing classification and anomaly detection on intraoperative MEPs in Section 3 and Section 4, we now turn to a general methodological contribution: a novel Random Forest distance for time series.

This section presents TSRF-Dist, a novel distance between time series based on Random Forests (RFs). We extend to the time series domain the concepts and tools of RF distances, a recent class of robust data-dependent distances defined for vectorial representations, thereby proposing the first RF distance for time series. The distance is obtained in two steps: first, an RF is trained to model a set of time series; second, the trained RF is exploited to quantify similarity between time series. For the first step, we introduce the Extremely Randomized Canonical Interval Forest (ERCIF), a novel extension of Canonical Interval Forests that can model time series and can be trained without labels. We then consider three distance schemes inspired by the vectorial case. The resulting variants are evaluated on 128 datasets from the UCR archive, showing promising results compared with literature alternatives.

5.1 Introduction

The analysis of time series is a fundamental topic in Pattern Recognition and Data Mining [147], [148], [36] and is crucial in many application domains, including signal processing [149], medical data analysis [38], [39], weather forecasting [150], human activity recognition [151], and bioinformatics [152]. In this context, defining an appropriate distance between time series has received substantial attention in the literature, since distance is often the basis for more complex operations such as classification or clustering. Several measures have been proposed in the past [48], [153], [154], characterized by different features in terms of employed concepts, obtained performances, and computational requirements — see Section 2.2 for a detailed overview. In this section, we contribute to this field by proposing a novel distance between time series based on Random Forests (RFs) [26, 155]. These models, which represent ensembles of decision trees [156], are typically employed in supervised tasks like regression or classification. In recent years, however, great interest has arisen in the exploitation of Random Forests also in unsupervised contexts, such as outlier detection [27, 98], clustering [29, 31, 32, 157–160], and, crucially, distance computation. Breiman, in his seminal work [26], was the first to suggest the possibility of deriving powerful data-dependent similarity measures from Random Forests. After his intuition, many other RF distances have been proposed (see, e.g., [31, 32, 160–165]), shown to be effective in classification, clustering, outlier detection and other tasks. All these distances are defined on vectors, i.e., they permit the derivation of a distance measure between objects described with a vectorial representation. Among these, Proximity Forest [104] extends RF-based approaches to time series classification by leveraging distance-based splits, while Proximity Isolation Forest [105] extends the isolation mechanism to define a proximity-based anomaly detection method applicable to time series. In this section, we take one step forward and export concepts and tools for RF

distances to the time series domain, proposing the first RF distance for time series, which we call TSRF-Dist.

Following the classical scheme for the definition of vectorial RF distances, the TSRF-Dist distance is derived in two steps: first, we create a Random Forest starting from a training set of time series; then, given two time series, we compute their distance through the trained Forest. More in detail, in the first step, we define the Extremely Randomized Canonical Interval Forest (ERCIF), a Random Forest which can model time series and can be trained without labels. Our model starts from the Canonical Interval Forest [20], an RF for time series defined for classification, and extends it in different directions. The main change is to adapt it to the unsupervised case with a training scheme that follows the Extremely Randomized Trees (ERT) paradigm [95], in which the tests in the nodes of the trees are defined with an extreme degree of randomness. ERT-based schemes have shown to be appropriate to compute RF distances [166], also in tasks in which labels are available [167]. Given the trained ERCIF, the distance between two time series is defined by simply exploiting one of the schemes already defined for the vectorial case. For example, following the definition of Breiman [26], the distance is computed by letting the two time series traverse all the trees of the ERCIF, and counting how many times they fall in different leaves. The main intuition is that if two time series are clearly different, they will answer the tests in the nodes of the trees in different ways, thus ending in different leaves. In this section, we investigated three different distance schemes: the classic Breiman scheme, the more recent scheme introduced in [31], which exploits path overlaps, and the very recent scheme [32], which is based on the Tversky ratio model.

Within this framework, the main novelties of TSRF-Dist can be summarized as follows:

- TSRF-Dist exploits the ERT paradigm, shown to exhibit good theoretical properties for the derivation of RF distances [166], to provide an RF-based distance for time series.
- TSRF-Dist relies on ERCIF, which is unsupervised, whereas the other models proposed in the literature are supervised and designed for classification. Consequently, ERCIF can be trained without labels and can also be exploited in unsupervised contexts such as time series clustering or anomaly detection.

Due to its unsupervised nature, the proposed distance has been thoroughly investigated in a clustering setting on 128 datasets from the UCR archive [168], testing different variants and parameterizations with an average-link Agglomerative Clustering algorithm. We also compared our distance with literature alternatives, including the measures investigated in the recent review [48] on clustering. The results are promising and suggest that TSRF-Dist is a valid alternative to both classical and advanced time series distances.

The rest of the section is organized as follows. The proposed distance is introduced in Section 5.2, the empirical evaluation is described in Section 5.3, and Section 5.4 concludes the chapter and outlines future perspectives.

5.2 Proposed distance

In this section, we propose the distance TSRF-Dist. As typically done in the field of vectorial Random Forest distances, TSRF-Dist is obtained by following two steps:

- In the first step, a Random Forest for time series is built, starting from a set of training time series. Since no labels are available, classical classification forests for time series, such as those in [19] and [20], cannot be used. We introduce here a novel RF, which we call Extremely Randomized Canonical Interval Forest (ERCIF), which i) can model time series, and ii) can be trained without labels.
- In the second step, given two time series, the TSRF-Dist is defined through the learned ERCIF, in particular by exploiting the information obtained from the paths the two time series are taking in each tree of the forest.

5.2.1 Extremely Randomized Canonical Interval Tree and Forest

The Extremely Randomized Canonical Interval Forest (ERCIF) represents an ensemble of Extremely Randomized Canonical Interval Trees (ERCITs). As in the interval-based trees discussed in the background chapter and in the CISOF model of Section 4, an ERCIT is a binary tree whose internal nodes apply tests of the form $t = (f, v, i)$ to a time series $x = \{x_1, x_2, ..., x_T\}$. The interval i is extracted from the series, a feature f is computed on it, and the resulting value $x_f^{(i)}$ is compared with a threshold v to choose the left or right branch. For fixed-length series, i can be expressed through absolute positions $[i_l, i_u]$; for variable-length series, it can be defined relative to the series length.

- **Features.** Feature f is selected among the same 25 interval features adopted in CISOF: 22 CATCH22 descriptors [1], summarized in Table 2, together with mean, standard deviation, and slope from the TSF classifier [19]. This choice keeps the tree structure aligned with CIF-style interval forests while adapting it to the unsupervised setting.
- **Training of an ERCIT.** We follow a classical top-down recursive strategy to build an Extremely Randomized Canonical Interval Tree (ERCIT). We start with the full training set $D = \{x_1, ..., x_n\}$ composed of n time series, and choose a test $t = (f, v, i)$ for the root. The test splits D into D_L and D_R according to the answers given by the time series in D; these sets are then used to determine the tests in the left and right children of the root, recursively continuing until the tree reaches a height limit or a node contains fewer than a minimum number of samples.

To determine the test adopted at a given node, we resort to the paradigm employed in Extremely Randomized Trees (ERT - [95]): instead of searching for the “best” question, as in classification trees using the Gini criterion [135], the tests are chosen in an extremely randomized way. This paradigm has proven effective for

classification [95], anomaly detection [98], clustering [30, 157, 158], and, crucially, for the derivation of vectorial RF distances [161–165]. It offers an unsupervised, simple, and efficient way to derive a forest that often defines distances outperforming more sophisticated schemes, even in supervised settings [169, 170]. A theoretical characterization of the favorable behavior of ERT for RF distances has also recently been provided in [166]. More specifically, in our ERCIT, given a set $D = \{x_1, ..., x_n\}$ of time series reaching a node, the test $t = (f, v, i)$ is determined as follows:

- Interval i is defined by selecting an interval from the set I. The set I is computed by randomly choosing starting points and random lengths[3].
- Feature f is randomly selected from a random subset of 25 pre-defined features.
- A split threshold v is randomly selected from the set $D_f^{(i)}$ of feature values computed at that node. Let:

$$f_{\min} = \min_{j=1\cdots n} x_{jf}^{(i)}, \quad f_{\max} = \max_{j=1,\cdots n} x_{jf}^{(i)}, \tag{30.}$$

where $x_{jf}^{(i)}$ represents the value of feature f computed on the interval i extracted from the time series x_j.

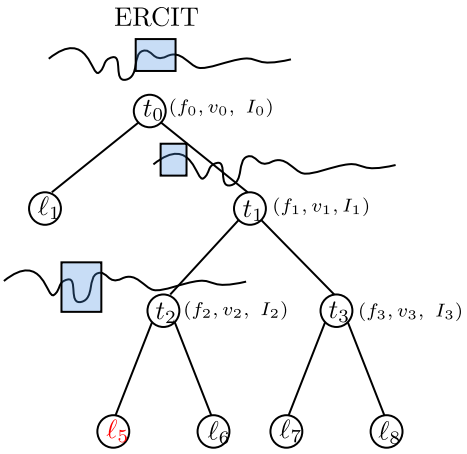

Figure 23: An Extremely Randomized Canonical Interval Tree. In each node i, the decision is taken as follows: i) we extract the interval I_i from the time series (highlighted by the shaded box); ii) on I_i we compute the feature f_i; iii) if $f_i < v_i$, the time series is sent to the left else to the right, till we reach a leaf (in red).

Figure 23 illustrates the tree-building process for time series. At each node, a random interval and a random feature are selected, and the sample is sent to the left or right child according to its response to the test.

Algorithm 10 contains the pseudo-code of the complete training procedure for the ERCIT.

[3]Please note that to compute some of the CATCH22 features, the interval should have a minimal length of 20.

ERCIT Fitting: buildERCIT(D, F, I):

if $|D| < 2$:
  return leaf{size: $|D|$}
else:
  $D_L \leftarrow \emptyset$
  $D_R \leftarrow \emptyset$
  $i \underset{R}{\in} I$ // select a random interval from I
  $f \underset{R}{\in} F$ // select a random feature from F
  let $D_f^{(i)}$ be the set composed by computing $x_f^{(i)}$ **for** every $x \in D$
  $v \underset{R}{\in} D_f^{(i)}$ // Randomly select v from $D_f^{(i)}$
  for $j \leftarrow 0$ **to** $|D|$: // Test Split
    if $x_{jf}^{(i)} < v$:
      $D_L \leftarrow D_L \cup \{x_j\}$
    else:
      $D_R \leftarrow D_R \cup \{x_j\}$
  return node $\begin{cases} \text{left:} buildERCIT(D_L) \\ \text{right:} buildERCIT(D_R) \\ \text{split\_feature:} f \\ \text{split\_value:} v \\ \text{interval:} i \end{cases}$

Algorithm 10: Training Set D, CATCH22 Feature Subset F, Interval Subset I

- **From Trees to Forest.** The Extremely Randomized Canonical Interval Forest (ERCIF - see Algorithm 11) represents an ensemble of ERCITs. Following the classic recipe of Random Forests [26], each ERCIT is created, with the procedure described previously, starting from a random bootstrapping of the training set. To control the computational requirements for the training of ERCIT, while keeping a high level of randomization, we adapt to our case some of the mechanisms typically employed for classification trees. For example, in a CART (Classification and Regression Tree) [135], it is common that the optimal feature in a node is selected among only a subset of all possible features, which is randomly chosen and kept fixed over the whole tree construction. In our case, at the beginning of the ERCIT training, we randomly select a set of n_i intervals and a set of n_f features; then, during the recursive training, feature f and interval I in each test $t = (f, v, I)$ are randomly chosen within these pre-selected sets. By efficiently pre-computing all the selected features on all the selected intervals for all the training time series, we reduce the overload of the recursive training procedure. At the same time, since this procedure is repeated for every tree in the forest, different trees work on different intervals and use different features, maintaining a high degree of randomization, which is typical in ERT-inspired schemes. Figure 24

offers a visual depiction of the training procedure for the presented forest-based model.

ERCIF Construction: buildERCIF(D, F, I):

```
1  let Forest← []
2  for i ← 0 to n_t:
3      D_i ← bootstrap(D)
4      I_i ← sample_intervals (D)          // generate random intervals set
5      Forest[i] ← buildERCIT(D_i, F, I_i)
6  return Forest
```

Algorithm 11: Training Set D, Number of trees n_t, Set of features F.

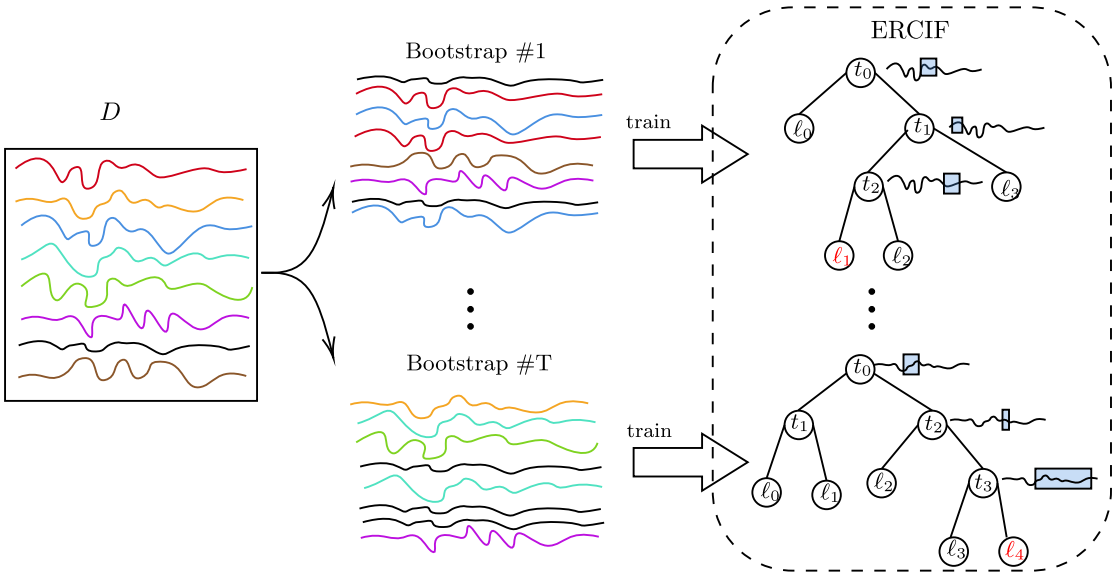

Figure 24: Training phase of the ERCIF model. The original unlabelled dataset is bootstrapped T times, and each bootstrap sample is used to train a single ERCIT.

5.2.2 Computing the distance

Given the trained ERCIF and two time series, the TSRF-Dist is then computed following the typical schemes used for vectorial RF distances [31, 32, 160–165]. The main idea, which is also behind all RF distances, is the following. In an ERCIT, a node contains a test on a specific feature extracted from a specific interval of the time series. If the two time series provide the same answer to this test, they share a similarity in that feature and interval; otherwise, if they respond differently, they do not. This can be generalized at the tree level since a time series follows a path in the tree, i.e., it has to answer many tests. By aggregating and comparing these answers, we can define a similarity measure in a tree, which can finally be aggregated at the forest level (graphical example in Figure 25). For example, in his seminal work [26], Breiman defined a tree-based similarity measure that compares two objects: if the objects fall in different leaves, their similarity is 0; if they fall in the same leaf, their similarity is 1. This tree-based measure is then extended to a forest-based similarity by averaging the 0/1 similarities across all trees in the forest. As a result, the forest-

based similarity between two objects is computed as the proportion of trees in which the two objects end up in the same leaf, effectively measuring how often they provide identical answers to all tests along their paths through the trees. The simplicity of Breiman's approach offers computational efficiency and robustness to minor path variations, but it loses information about partial agreement: two paths that diverge at different depths are treated identically. More sophisticated variants address this trade-off. The Zhu measure considers the depth of the lowest common ancestor, capturing how far paths agree before diverging. The RatioRF measure, based on the Tversky ratio model, evaluates the proportion of common decisions along both entire paths, providing finer-grained discrimination at a higher computational cost. More precisely, given an ERCIF composed by T ERCITs, and given two time series x, y, the TSRF-Dist $d^{TS-RF}(x, y)$ is defined as:

$$d^{TS-RF}(x, y) = 1 - \frac{1}{T}\sum_{i=1}^{T} s^i(x, y), \tag{31.}$$

where $s^i(x, y)$ corresponds to the chosen similarity measure, evaluated on the i^{th} ERCIT. Thus, the resulting distance measure varies according to the chosen definition of tree similarity. In this section, we consider three distinct similarity measures, each entailing a different distance measure:

- The first measure adopts the variant introduced by Breiman [26], described above. The corresponding Breiman tree similarity $s(x, y)$ with respect to a given ERCIT is defined as:

$$s_{\text{Breiman}}(x, y) = \begin{cases} 1 \text{ if } \quad \ell(x) = \ell(y) \\ 0 \text{ otherwise} \end{cases}, \tag{32.}$$

where $\ell(x)$ and $\ell(y)$ represent the leaves, in the considered ERCIT, where the time series x and y fall, respectively.

- The second measure adopts ClustRF-Strct definition proposed in [31], here simply referred to as Zhu from the name of the first author. The Zhu measure is defined as the depth of the lowest common ancestor (LCA) of the two leaves where the two time series fall: the deeper this node, the larger the number of tests to which the two time series answered in the same way. Formally,

$$s_{\text{Zhu}}(x, y) = \frac{h(LCA(\ell(x), \ell(y)))}{\max(h(\ell(x)), h(\ell(y)))}, \tag{33.}$$

where $h(u)$ represents the depth of a node u, and $LCA(u, v)$ represents the lowest common ancestor between nodes u and v.

- The third measure employs the RatioRF definition, the most recent in this field [32], which has shown to outperform many alternatives in the clustering scenario. RatioRF defines a similarity that follows the axiomatic definition given by the Tversky's Ratio model [171]. The main idea is to consider, among all the tests in the paths of the two time series x and y, how many times x and y provide the

same answer. Formally, if we define the set $P(x)$ of tests in the path the time series x is following in the given ERCIT and denoting as $t(x)$ the answer provided by the time series x to a given test t (and similarly for time series y), the RatioRF similarity is defined as

$$s_{\text{RatioRF}}(x, y) = \frac{|\{t \in P(x) \cup P(y) \mid t(x) = t(y)\}|}{|\{P(x) \cup P(y)\}|}, \tag{34.}$$

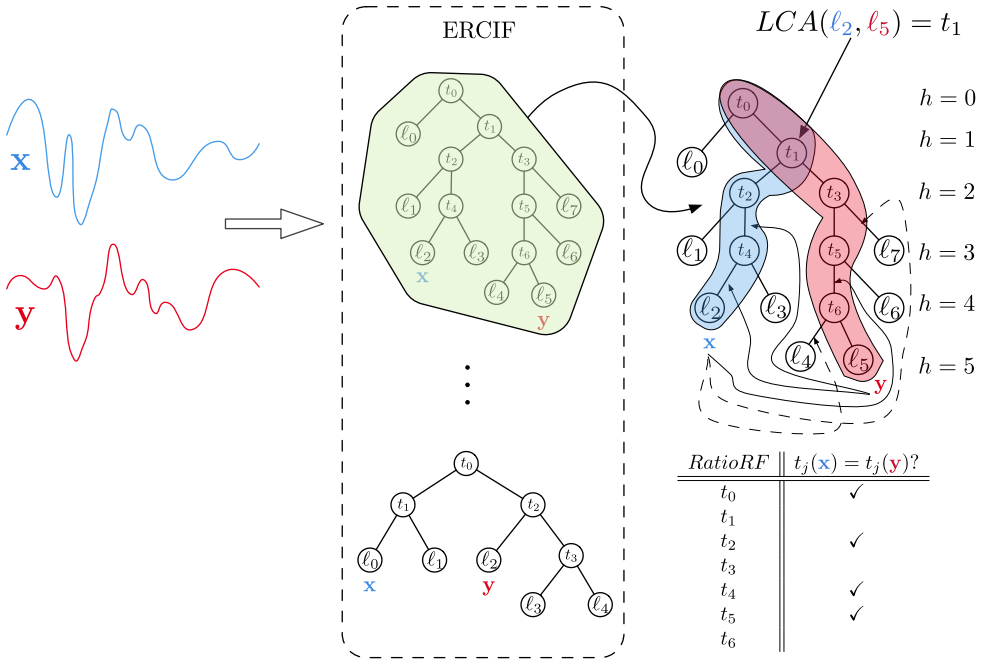

Figure 25: Visual representation of the computation of the TSRF-Dist in Breiman, Zhu, and RatioRF versions.

We conclude this section with an example to illustrate the computation of the TSRF-Dist distances. Figure 25 demonstrates how these distances are computed. In this process, two time series are fed to each ERCIT of the trained ERCIF, and their root-to-leaf paths are compared. As shown in the right part of Figure 25, considering the ERCIT in the green area, the blue time series x reaches ℓ_2 at depth 4, while the red time series y terminates in leaf ℓ_5. The node t_1 serves as their Lowest Common Ancestor (LCA) at depth 1. Based on these paths, we can compute the three distances. The Breiman distance, being a binary measure, simply checks if the two time series end up in the same leaf node. In this case, since x and y land in different leaves (ℓ_2 and ℓ_5 respectively), the Breiman similarity for this tree, $s_{\text{Breiman}}(x, y)$, is 0.

The Zhu similarity, on the other hand, considers the depth of the LCA relative to the maximum depth reached by either time series. Here, the LCA (t_1) is at depth 1, while the maximum depth between ℓ_2 and ℓ_5 is 5. Thus, the Zhu similarity is quantified as $s_{\text{Zhu}}(x, y) = \frac{1}{5} = 0.2$, representing the ratio of the LCA depth to the maximum depth between the leaves determined by the two samples. Lastly, the RatioRF similarity examines the proportion of common decisions along the paths of

both time series. In our example, we assume that paths of x and y agree on 4 decisions (at nodes t_0, t_2, t_4, and t_5, as reported in the table in Figure 25), out of a total of 7 decisions encountered. This results in a RatioRF similarity of $s_{\text{RatioRF}}(x, y) = \frac{4}{7} \approx 0.57$. To compute the final TSRF-Dist measure, each of these three similarities would be averaged across all trees in the forest.

5.3 Experimental Evaluation

This section deals with the evaluation of the proposed distances. In particular, after introducing our benchmark dataset, the UCR archive [168], in Section 5.3.1, we present the experimental setup in Section 5.3.2. We then compare the different variants of TSRF-Dist in Section 5.3 and compare our approach with literature alternatives in Section 5.3.4.

5.3.1 Benchmark Dataset: The UCR Time Series Archive

The UCR archive is a comprehensive collection of time series data that serves as the benchmark for evaluating TSRF-Dist. Our evaluation encompasses 128 datasets from the UCR archive [168]. To provide a high-level overview of the data types represented in the archive, we grouped the datasets into three categories: Multimedia, Sensors/Devices, and Biomedical & Others. Table 17 summarizes the number of datasets and associated UCR tags for each category.

Category	#Datasets	UCR Tags
Multimedia	34	Image, Audio
Sensors/Devices	43	Spectro, Sensor, Device, Traffic
BioMed. & Others	51	ECG, Hemodynamics, EOG, EPG, Simulated, Other, HAR, Motion

Table 17: Details of each category.

Each UCR dataset is originally organized as a supervised problem and is therefore provided as separate training and test files. Since our focus is the unsupervised task of clustering, we merged these files into a single dataset for each UCR problem. The original labels were retained and used as external criteria for evaluating clustering performance, as discussed further in Section 5.3.2.

After merging, each dataset is characterized by two key parameters: the number of samples and the length of the time series. Our assessment focuses on two aspects: effectiveness, measured by clustering performance, and efficiency, measured by computational cost.

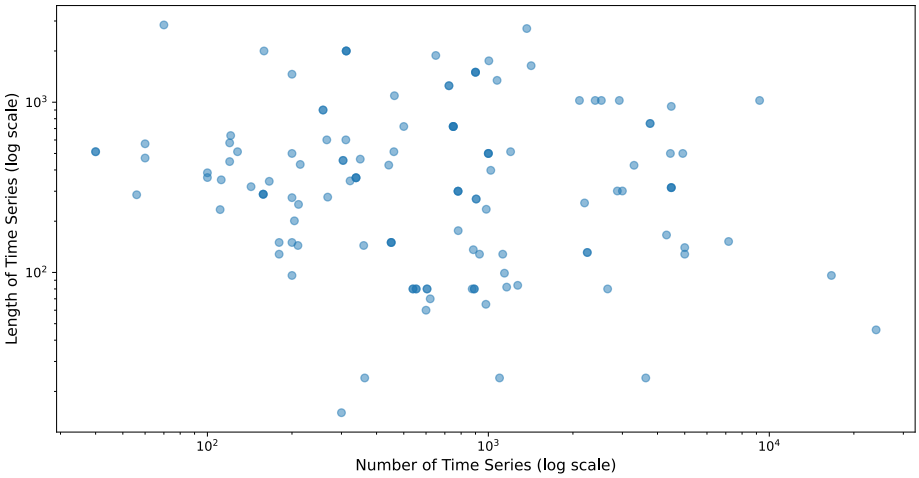

Figure 26: Scatter plot of dataset size vs time series length for the datasets in the UCR archive. The x-axis shows the total dataset size (train + test) on a logarithmic scale, and the y-axis shows the time series length on a logarithmic scale.

Figure 26 presents a scatter plot of the number of samples and time-series length for all datasets in the UCR archive. Blue dots denote the full archive, whereas red stars highlight the representative subset later reused for the implementation benchmarks in Section 6. The clustering experiments reported in Sections Section 5.3.2, Section 5.3, and Section 5.3.4 were conducted on the full archive.

For a more comprehensive overview of the datasets, including specific categories, names, and cardinalities, readers can refer to UCR Archive Datasets.

5.3.2 Experimental details

The experimental evaluation was conducted in comparison with established time series distances from the literature, providing a comprehensive view of the main properties of TSRF-Dist. To compare TSRF-Dist with existing distances on the UCR datasets, we implemented the workflow presented in Figure 27. As is common in the evaluation of time series distances [48], the assessment is performed in a clustering setting. We employed Agglomerative Clustering [172] with average linkage, which starts with each object forming its own cluster and then iteratively merges clusters based on similarity. In our case, the process terminates when the number of clusters equals the number of classes in the problem. As the final step, we compute the Adjusted Rand Index (ARI) [173]. ARI is a corrected-for-chance version of the Rand index that compares two partitions of a set. It is commonly used either to measure agreement between two different clusterings of the same dataset or to assess the similarity between clustering results and known ground-truth labels. In this study, we focus on the latter use. ARI ranges from -1 to 1, where 1 indicates perfect agreement between the partitions, 0 represents random labelling, and negative values indicate less agreement than expected by chance. In our case, ARI quantitatively assesses how well the clustering results match the known class structure of each dataset. Figure 27 depicts the formula for computing the ARI, as

well as a graphical description of our entire experimental process, including the clustering and evaluation steps. This approach allows us to account for the possibility of agreement occurring by chance while providing a robust metric for comparing the performance of different time series distance measures across UCR datasets. Our experimental setup involves two distinct configurations of the ERCIF model: **H**eavy and **L**ight. The **H**eavy configuration closely resembles that employed in the work introducing the Canonical Interval Forest [20], while the **L**ight configuration explores the possibility of reducing computational requirements. In particular, in the **H**eavy we set the number of ERCITs $T = 500$, the number of features $n_f = 8$ and the number of intervals $n_i = \sqrt{N}$, where N is the sequence length. In the **L**ight configuration, we set $T = 200$, $n_f = 8$ and $n_i = \log_2(N)$. For both configurations, we computed ARIs over datasets with the three distances described in Section Section 5.2, namely $Breiman$, Zhu, and $RatioRF$ distances, for a total of six variants of TSRF-Dist.

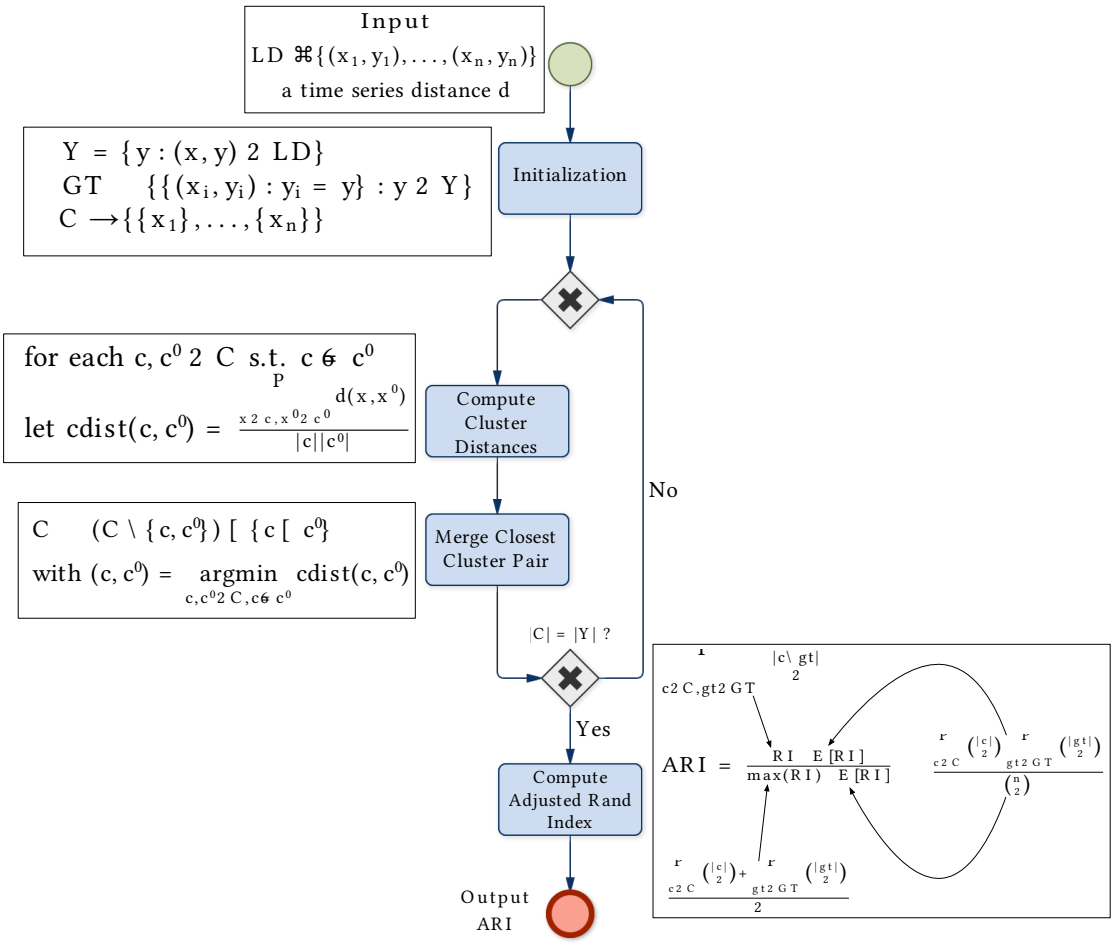

Figure 27: Our experimental workflow for evaluating clustering by a given time series distance d against a dataset with a given external criteria.

To assess the impact of randomization on the performance of the ERCIF model, we iterated the entire workflow described in Figure 27 ten times for each dataset and variant. This repetition allows us to measure how the inherent randomization affects the performance of our proposed method. We then report the average ARI across

these iterations, providing a more robust evaluation of the performance of the model. Additionally, we applied the same workflow to all datasets in the UCR archive using the time series distances introduced in Section 2.2. Due to the deterministic nature of these distances, a single iteration of the workflow was sufficient for each dataset. The results of these comparisons between TSRF-Dist and the time series distances introduced in Section 2.2 are presented and analyzed in Section 5.3.4.

5.3.3 Comparison between different variants

As a first analysis, we performed a comparison of the different variants of TSRF-Dist. First, we analyze the difference between the two configurations **L**ight and **H**eavy. In particular, for each dataset, we computed the average among $Breiman$, Zhu, and $RatioRF$ of the ARI of the two configurations. The results, averaged over the datasets of the different domains, are presented in Table 18. The last column indicates the p-value of a Wilcoxon signed-rank test [139], used to check whether the null hypothesis that two related paired samples are drawn from the same distribution can be rejected. For all statistical tests, the significance level has been set to 0.05.

	L	H	p-value
Multimedia	0.3022	**0.3072**	0.0147
Sensors/Devices	**0.2113**	0.2084	0.4328
BioMed. & Others	0.2608	**0.2653**	0.0009

Table 18: ARI-based comparison between the Light (L) and Heavy (H) forest configurations. The fourth column reports the p-value of a Wilcoxon signed-rank test, between L and H.

As evident from the table, the **H**eavy configuration performs slightly better than the **L**ight one across all categories. For Sensors/Devices, however, this difference is not statistically significant. By contrast, the BioMed. & Others and Multimedia categories do exhibit statistically significant differences, although the gaps in mean ARI remain limited. Despite its higher computational demands, the superior performance of the **H**eavy configuration suggests that it is worth considering when the small gain in accuracy is important.

Our second analysis focuses on the various distances used, which is crucial for understanding performance variations across different scenarios. We computed the average ARI for the $Breiman$, Zhu, and $RatioRF$ variants across the two ERCIF configurations and presented the results in a critical difference diagram (Figure 28). To have an idea of the statistical significance of the comparison, we performed a Friedman test, followed by a post-hoc pairwise test for multiple comparisons of mean rank sums (Nemenyi's test) [138]. The Friedman test checks if there are overall significant differences among the ranks of the analyzed methods in different datasets; if the test is passed, post-hoc Nemenyi mean rank sums assess pairwise differences. Finally, a critical difference diagram is derived, in which the mean ranks

across datasets represent the different methods. In all critical difference diagrams throughout this thesis, rankings with a statistically significant difference are connected by red lines. Figure 28 clearly shows that the *Breiman* variant performs the best; additionally, the differences between the other distances are all statistically significant.

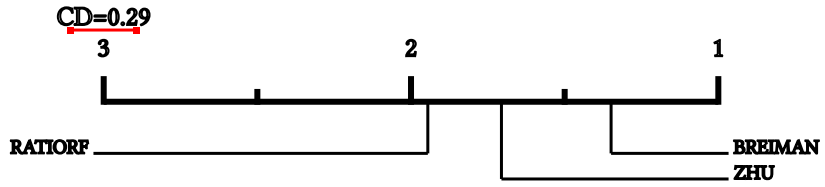

Figure 28: Critical Difference Diagram comparing the ARIs of distances employed over the UCR archive datasets.

5.3.4 Comparison with state of the art

In this part, we compare the proposed approach with alternative distances developed for time series in recent years, here collectively referred to as elastic distances. In particular, we used as competitors the distances described in Section 2.2, namely Euclidean, CATCH22Eucl, ERP, LCSS, DTW, cDTW, DDTW, WDTW, WDDTW, ADTW, MSM, TWE, SBD, and MPDist. A subset of these distances has been considered in a recent review on clustering with time series distances [48]; we include Euclidean distance as a baseline. In our experiments, for standard elastic distances, we used the implementation included in the aeon Python library https://www.aeon-toolkit.org, which provides a comprehensive set of tools for time series distance computation and comparison. We used average-link agglomerative clustering as implemented in sklearn [174]. Obtained ARIs, computed using the workflow depicted in Figure 27 and averaged over the three categories of Table 17, are displayed in Table 19. In the last row, we report the best result of TSRF-Dist (Breiman variant; best among **L**ight and **H**eavy configurations). As shown in Table 19, TSRF-Dist outperforms, on average across the three groups, all competitors.

	Multimedia	Sensors/Devices	BioMed. & Others
EUCLIDEAN	0.1213	0.0848	0.1553
CATCH22EUCL	0.0463	0.0340	0.0619
ERP	0.1632	0.1326	0.2283
LCSS	0.1036	0.0771	0.1585

	Multimedia	Sensors/Devices	BioMed. & Others
DTW	0.1722	0.0960	0.1982
cDTW	0.1802	0.0940	0.2075
DDTW	0.1172	0.0588	0.0444
WDTW	0.1803	0.1061	0.2176
WDDTW	0.1309	0.0454	0.0521
ADTW	0.1602	0.1018	0.2237
MSM	0.1848	0.0960	0.1842
TWE	0.1940	0.1006	0.1812
SBD	0.1258	0.0932	0.1365
MPDIST	0.0471	0.0492	0.0327
TSRF-Dist	**0.3612**	**0.2711**	**0.3145**

Table 19: Average ARI comparison between TSRF-Dist and the other time-series distances across the three dataset categories.

Figure 29 depicts the obtained critical diagram. The complete ARI results, including standard deviations, for TSRF-Dist and all other compared methods, from which the critical difference diagrams in Figure 29 are derived, are reported in Table 24 in Appendix Appendix A.

Figure 29 is derived using the mean ARI over 10 distinct random initializations for TSRF-Dist, providing a direct comparison with other distance measures. The results are extremely promising. Indeed, we can observe that TSRF-Dist represents the method that ranks best on all the datasets used; moreover, the differences to ERP (the second most accurate method) are statistically significant, thus confirming that the proposed approach represents a valid alternative to the standard as well as advanced time series distances found in the literature.

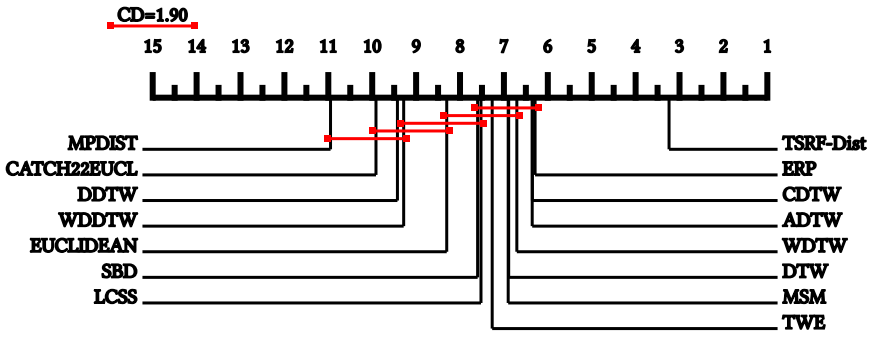

Figure 29: Critical Difference Diagram comparing TSRF-Dist with other time series distances. Comparison using mean ARI values for TSRF-Dist.

5.4 Discussion and conclusions

In this chapter, we introduced TSRF-Dist, a novel unsupervised distance for time series based on Random Forests. The key contribution is extending RF distance concepts from the vectorial to the time series domain through the Extremely Randomized Canonical Interval Forest (ERCIF), which can be trained without labels. We investigated three distance variants—Breiman, Zhu, and RatioRF—each representing different complexity-sensitivity trade-offs, and evaluated them on 128 UCR datasets.

TSRF-Dist's unsupervised nature makes it particularly valuable for clinical and biomedical applications where labeled data is scarce or expensive. The learned, data-dependent distance can capture domain-specific patterns better than fixed measures. The multiple variants provide flexibility: Breiman offers computational efficiency, while RatioRF provides fine-grained discrimination. Empirical results demonstrate consistent superiority over established alternatives including DTW variants and ERP.

While TSRF-Dist could, in principle, be integrated into the intraoperative MEP case study presented in Section 3 and Section 4 (e.g., by replacing cDTW in distance-based baselines or by using TSRF-Dist within a k-NN classifier), we did not include a dedicated evaluation on MEP datasets in this thesis. The MEP chapters focus on feature-based Random Forest classifiers and isolation-based anomaly detection, which better match the real-time and interpretability constraints of intraoperative monitoring. A careful study of TSRF-Dist on MEPs would require additional design choices (e.g., strict patient-level splits to avoid subject leakage, latency handling, and real-time distance computation) and is left as future work.

For clinical deployment, several considerations remain important. Clinical validation on actual biomedical datasets (ECG, MEP, EEG) is essential, as medical signals have characteristics that may differ from general time series benchmarks. The interpretability of forest-based distances, while improvable through feature importance analysis, is lower than traditional measures. Real-time clinical applications may require careful tuning of forest parameters to balance accuracy and computational demands. Finally, regulatory validation and integration into clinical decision-support systems represent necessary steps toward practical medical application.

6 tsdistances: A High-Performance Python Library

This chapter focuses on the algorithmic foundations of the *tsdistances* library[4]. A broader review of Euclidean distance, DTW, and related elastic measures is provided in Section 2.2. Here, we introduce only the notation and dynamic programming machinery needed to explain the optimizations discussed in Section Section 6.2.

6.1 Introduction

We denote time series by lowercase letters x and y, with $x = \{x_1, ..., x_N\}$ and $y = \{y_1, ..., y_M\}$. Many elastic distances can be written as dynamic programs over a matrix D, where each entry stores the optimal value of the subproblem up to indices (i, j). In the formulation relevant to this chapter, the recurrence has the generic form

$$f\big(x_i, y_j, D[i-1, j-1], D[i-1, j], D[i, j-1]\big) \tag{35.}$$

where $f(...)$ depends on the particular distance. The full methodological background is given in Section 2.2; here we focus on this generic DP structure because it underlies both the CPU and GPU implementations in *tsdistances*.

In Algorithm 12, we report the standard row-wise DP scheme. Ignoring the band constraint for the moment, the matrix is filled from $[0, 0]$ to $[N, M]$, where the final distance is stored. This requires $\mathcal{O}(NM)$ time, since one constant-time update is performed per cell, and $\mathcal{O}(NM)$ space if the entire matrix is stored.

Because this baseline quickly becomes expensive for long sequences, practical implementations rely on pruning and structural constraints. One of the most common constraints is the Sakoe-Chiba band [55], which limits computation to cells within a width b around the main diagonal. In Algorithm 12, line 1 checks whether the indices fall inside this band; the corresponding logic is shown in Algorithm 13, and a visualization is given in Figure 30. Under this constraint, the computation drops to $\mathcal{O}(Nb)$ time.

Standard Dynamic Programming Algorithm$(x, y, band, m)$:

```
1  Let D ← matrix(N, M)                                  // Initialised to m
2  D[0, 0] ← 0
3  for i ← 1 to N:
4      for j ← 1 to M:
5          if inbound(i, j, band):
6              D[i, j] ← f(x_i, y_j, D[i − 1, j − 1], D[i −
               1, j], D[i, j − 1])
7  return D[N, M]
```

[4]The *tsdistances* library is publicly available on PyPI (https://pypi.org/project/tsdistances/) and on GitHub (https://github.com/irazza/tsdistances/)

Algorithm 12: Time series x and y of lengths N and M, a window band and an initialization value m

```
INBOUND FUNCTION(i, j, b):
1  if |i − j| < b:
2      return true
3  else:
4      return false
```

Algorithm 13: Indices i, j and bandwidth parameter b

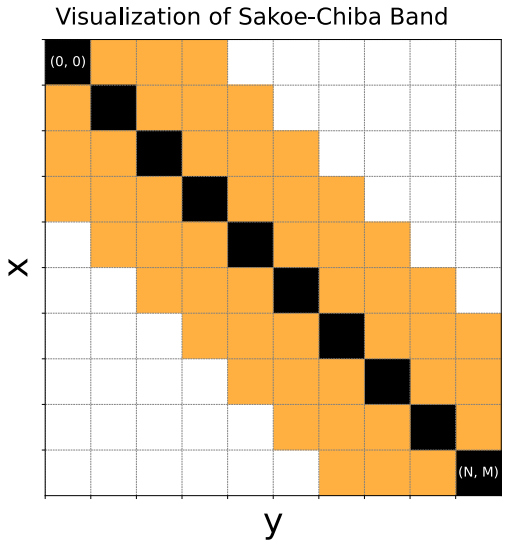

Figure 30: Sakoe-Chiba band computed on time series x and y of length 10. The window size is 3. The orange area represents the Sakoe-Chiba band, while the black area is the diagonal of the matrix. The white areas are the cells excluded from the computation.

6.2 Dynamic Programming Optimizations for Time Series Distances

In this section, we present two complementary approaches for optimizing time series distance computations. The first, described in Section Section 6.2.1, builds on the work of Hughey [175] by exploiting anti-diagonal traversal of the dynamic programming matrix. Since the cells along each anti-diagonal are independent, they can be computed simultaneously using single-instruction multiple-data (SIMD) operations, thereby maximizing CPU utilization while reducing memory overhead. The second, presented in Section Section 6.2.2, builds on the ideas of Castells-Rufas [176] to extend these concepts to massively parallel GPU architectures, enabling efficient processing of large-scale time series datasets through workload distribution and synchronization techniques. It employs a diamond-shaped dependency pattern that reorganizes the matrix traversal to expose parallelism and reduce redundant computations. The GPU implementation also incorporates a tiling strategy that

leverages shared memory and parallel kernel execution to fully exploit modern GPU architectures, achieving high throughput even for long input sequences.

6.2.1 CPU Implementation

In this section, we describe the efficient implementation of elastic distances based on anti-diagonal traversal of the dynamic programming matrix with SIMD parallelization. Following the approach introduced by Hughey [175], our optimization approach addresses two key performance bottlenecks in elastic distance computation: memory usage and sequential processing dependencies. By traversing the dynamic programming matrix along its anti-diagonals and carefully managing memory access patterns, we achieve both reduced memory footprint and enhanced opportunities for parallel execution through SIMD operations.

(A)

2.2	2.4	3.0	3.6	4.1	5.4	5.6
3.6	4.5	5.1	6.0	6.2	5.6	6.1
4.3	5.6	6.1	(6.3)	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0

(B)

2.2	2.4	3.0	3.6	4.1	(5.4)	0.0
3.6	4.5	5.1	6.0	(6.2)	0.0	0.0
4.3	5.6	6.1	(6.3)	0.0	0.0	0.0
4.4	6.3	(6.4)	0.0	0.0	0.0	0.0
6.2	(6.5)	0.0	0.0	0.0	0.0	0.0
(7.6)	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 20: Comparison of dynamic programming (DP) matrix processing strategies. (A) Standard row-by-row computation, where only one cell can be updated at a time based on its dependencies. (B) Anti-Diagonal computation using SIMD operations, which enables simultaneous updates of all cells along the current anti-diagonal. Red cells indicate the active cell(s) being computed. Green cells represent the dependencies required for computing the current cell or anti-diagonal. Orange cells are kept in memory for subsequent computation. Blue cells are those that are no longer relevant for the current computation step.

An important optimization for elastic distance computation is the reduction of space complexity from quadratic to linear. Table 20 illustrates two approaches to dynamic programming: row-wise processing (A) and anti-diagonal processing (B). In the row-wise case, each cell $D[i, j]$ depends on its left neighbor $D[i, j-1]$, imposing strict sequential dependencies. In contrast, anti-diagonal processing exploits the fact that all cells on the same anti-diagonal are independent and can be computed in parallel. Conceptually, the computation of each cell depends only on values from the previous two anti-diagonals. Hence, three anti-diagonals need to be stored in memory simultaneously: the two just described and one for the current computation. For efficiency, however, the implementation compacts the last two previously computed anti-diagonals into a single array. In this layout, odd indices

store values from the second-to-last anti-diagonal, while even indices store values from the most recent one. When computing the current anti-diagonal, the algorithm overwrites in the array *diag* (see line 13) the values of the older of the two anti-diagonals, either the odd or even positioned values, replacing them with the newly computed ones of the current anti-diagonal. Algorithm 14 provides pseudocode to illustrate this technique.

```
ANTI-DIAGONAL DYNAMIC PROGRAMMING(x, y, f):
 1  let maxK ← N + M − 1
 2  let midpoint ← max(N, M)                  // Index offset correction
 3  let diag ← array(2 * max(N, M))           // empty array of length 2*max(N,M)
 4  for k = 0 to maxK:
 5      let (i_start, i_end) ←
        diag_bounds(k, midpoint)
 6      for t ← i_start to i_end + 1 by 2:    // starting point at correct parity
 7          let i ← (i_end − t)/2
 8          let j ← (t − i_start)/2
 9          if k > midpoint:
10              i ← i + (k − midpoint)        // second half of the matrix
11              j ← j + (k − midpoint)
12          let a ← get(diag, t − 1)
13          let b ← get(diag, t)              // D[i][j-1]
14          let c ← get(diag, t + 1)          // D[i-1][j-1]
15          diag[t] ← f(x[i], y[j], a, b, c)  // D[i-1][j]
16  let d ← diag[midpoint]
17  return d
```

Algorithm 14: Time series x, y of length N, M, step function f

```
DIAG_BOUNDS(k, midpoint):
1  if k ≤ midpoint:
2      i_start ← midpoint − k                 // First half of the matrix
3      i_end ← midpoint + k
4  else:
5      i_start ← k − midpoint + 2
6      i_end ← 2 * midpoint − (k − midpoint + 1)
7  return (i_start, i_end)
```

Algorithm 15: Step index k and $midpoint\}$

Initially, Algorithm 14 computes the total number of anti-diagonals as $N + M - 1$, where N and M denote the lengths of the two input sequences. Rather than storing

the entire DP matrix, the algorithm maintains only a compact diagonal representation as described previously. This reduces memory consumption from quadratic to linear in the sum of the input lengths. The main loop proceeds over each anti-diagonal index k. For every k, the helper function "diag_bounds" (see Algorithm 15) determines the range on which the anti-diagonal array is computed (i_{start}, i_{end}). Depending on whether we are in the first or second half of the DP matrix, we compute these bounds differently. The main idea is the following. During the first half, the number of cells to be computed in sub-sequent anti-diagonals grow until reaching the middle diagonal, then when moving to the second half, instead, the anti-diagonals are going to shrink in size. The index bounds always depend on the *midpoint* and on the anti-diagonal counter. Each cell along the current anti-diagonal is then computed in parallel. For every cell, the corresponding matrix coordinates (i, j) are derived. Boundary cases, such as cells on the first row or column, are handled separately since their values depend on initialization constraints (i.e. cells initialized to $+\inf$). For all other cells, the dynamic programming recurrence relation is applied, using values from adjacent cells. The results overwrite the appropriate positions in the $diag$ array. After all anti-diagonals have been processed, the algorithm returns the final distance value, located at the bottom-right cell of the DP matrix, which in this compact representation corresponds to the *midpoint* of the diagonal array.

In addition to memory reduction and anti-diagonal traversal, we implement two further optimizations to accelerate distance computations. Following the strategy of Silva and Batista [177], we incorporate an early abandoning mechanism based on the main diagonal upper bound. Specifically, we compute the distance constrained to the diagonal path and use this value as an initial upper bound. During the full dynamic programming computation, any partial path whose cumulative cost exceeds this bound can be safely pruned, since it cannot contribute to the final minimum distance. This optimization substantially reduces the number of matrix cells that must be evaluated, particularly for long time series with low alignment variability. Furthermore, when computing distances between sets of time series, we exploit task-level parallelism. Each pairwise distance is independent, allowing us to distribute computations across multiple threads. This parallelization reduces time almost linearly with the number of available cores, providing significant performance gains for large datasets.

6.2.2 GPU Implementation

In this section, we present the implementation for Graphics Processing Units (GPUs) that enhances the anti-diagonal algorithm approach by exploiting the massive parallel processing capabilities of modern graphics hardware. Following the work of Castells-Rufas [176], our GPU implementation divides the computation into independent tiles that can be processed simultaneously by warp threads. The GPU implementation partitions the dynamic programming (DP) matrix into smaller submatrices, referred to as tiles. Each tile corresponds to a subproblem of the larger computation, is processed by the $compute\_tile$ kernel function, and requires two

time series slices together with a diagonal array. This kernel uses the relevant slices of the input series to compute the corresponding portion of the sub-DP matrix in an anti-diagonal fashion and outputs the diagonal array. Its behavior closely follows Algorithm 14, with one key difference: instead of returning only the final distance, it returns the entire diagonal array.

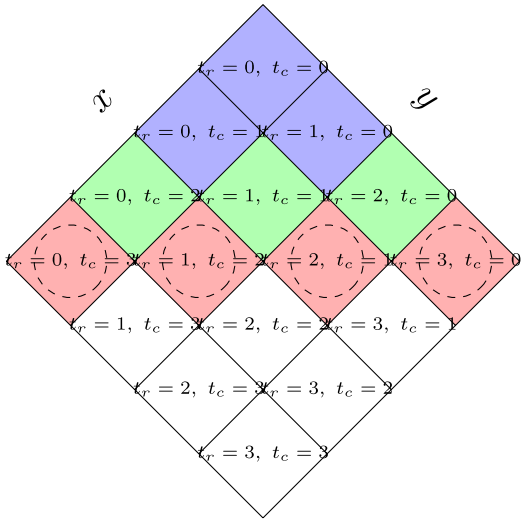

Figure 31: Schematic representation of tiled GPU-based distance matrix computation. Each cell corresponds to a tile, with colors indicating their computational role. Blue tiles contain no useful data for the current computation. Green tiles contribute only boundary values needed by their neighbor tiles. Red tiles are computed during the current iteration and processed in parallel. The matrix has been rotated, with respect to (B) in Fig. Table 20, to highlight the anti-diagonals.

This diagonal array conceptually consists of two halves: the first contains the last row of the tile, while the second stores the last column, as shown in Fig. Figure 32, which depicts a single tile from Fig. Figure 31.

In Algorithm 16, the inputs are x_{slice} and y_{slice} of length B, together with the diagonal array $diag$ of length $2B - 1$. On the first call to $compute\_tile$, $diag$ is initialized to zeros. On subsequent calls, it is initialized with the intermediate left and right diagonal parts computed in the previous anti-diagonal iteration.

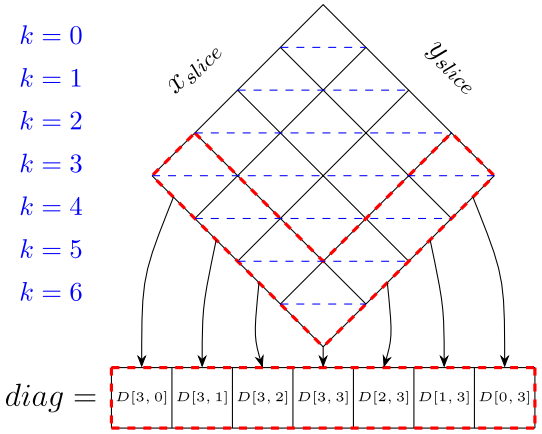

Figure 32: Example of a tile with warp size $B = 4$. Each iteration k corresponds to the computation of one anti-diagonal within the tile. The array $diag$ collects the values from the last row and last column of the tile, storing them as intermediate results for subsequent computations.

From Fig. Figure 32, it can also be observed that each iteration of k in Algorithm 16 does not overwrite the values corresponding to the last row and column. For example, when $k = 3$, the first and last entries of $diag$ remain unchanged for the subsequent iterations, and the same behavior applies to the other edge values as the diagonal advances.

```
COMPUTE_TILE(x_slice, y_slice, diag):
maxK ← 2 * B − 1
midpoint ← B                                  // Index offset correction
for k ← 0 to maxK − 1:
    (i_start, i_end) ←
    diag_bounds(k, midpoint)                  // Starting point at correct parity
    par for t ← i_start to i_end step 2:
        i ← (i_end − t) / 2                   // Launch B warp threads
        j ← (t − i_start) / 2
        if k > midpoint:
            i ← i + (k − midpoint)            // Second half of the matrix
            j ← j + (k − midpoint)
        a ← get(diag, t − 1)                  // D[i][j − 1]
        b ← get(diag, t)                      // D[i − 1][j − 1]
        c ← get(diag, t + 1)                  // D[i − 1][j]
        diag[t] ←
        f(x_slice[i], y_slice[j], a, b, c)
        Synchronize warp threads
return diag
```

Algorithm 16: Time series x_{slice}, y_{slice} of length B, and diagonal $diag$

In Algorithm 17, the DP matrix is partitioned into tiles, each identified by a tile row index t_r and tile column index t_c. The main function proceeds with the computation tile-by-tile along the anti-diagonals of the tile grid. The critical insight is that all tiles along the same anti-diagonal w, defined by:

$$t_r + t_c = w \tag{36.}$$

can be computed in parallel because the DP dependencies for each tile only reference tiles from previous anti-diagonals where $t_r + t_c < w$. The GPU architecture allows spawning a massive number of threads simultaneously, in this case equal to $w \cdot B$.

Once the first tile $(t_r \leftarrow 0, t_c \leftarrow 0)$ has been computed, its last row is passed to the neighboring tile $(t_r = 0, t_c = 1)$ as the boundary row, while its last column is passed to the neighboring tile $(t_r = 1, t_c = 0)$ as the boundary column. These boundary values are stored in their respective positions so that, at each new iteration, the part of the diagonal just computed can be correctly placed as either a row or a column in Algorithm 17 at line 15 for the next anti-diagonal computation.

GPU-BASED DISTANCE COMPUTATION(x, y):

1 $B \leftarrow 32$ // Warp (tile) size
2 $num\_tiles\_row \leftarrow \lceil N/tile\_size \rceil$
3 $num\_tiles\_col \leftarrow \lceil M/tile\_size \rceil$
4 $diagonal\_parts \leftarrow [zeros(1, B), zeros(1, B)]$
5 **for** $w \leftarrow 0$ **to** $num\_tiles\_row + num\_tiles\_col - 1$:
6 $new\_diagonal\_parts \leftarrow []$
7 **for** each (t_r, t_c) such that $t_r + t_c = w$:
8 $left \leftarrow t_r - t_c$
9 $right \leftarrow left + 1$
10 $diag \leftarrow concat(diagonal\_parts[left], diagonal\_parts[right])$
11 $x\_offset \leftarrow t_c * B$
12 $y\_offset \leftarrow t_r * B$
13 $diag \leftarrow compute\_tile\big(x\big[x_{offset} : x_{offset} + B\big], y\big[y_{offset} : y_{offset} + B\big], diag, B\big)$
14 $new\_diagonal\_parts[left] \leftarrow diag[: B + 1]$
15 $new\_diagonal\_parts[right] \leftarrow diag[B :]$
16 $diagonal\_parts \leftarrow new\_diagonal\_parts$
17 $d \leftarrow diagonal\_parts[0][B]$
18 **return** d

Algorithm 17: Time series x, y of lengths N, M

After computing all the anti-diagonals and tiles, the final distance can be obtained from the last value of the last computed anti-diagonal. For the final tile, this corresponds to $diagonal\_parts[0][B]$.

This GPU implementation achieves significant acceleration over the CPU version by combining two key strategies: parallel processing of multiple tiles and efficient anti-diagonal computation within each tile that can exploit the large number of threads available. The performance benefits of this approach are demonstrated in Section , which presents comprehensive benchmarking results.

As a final remark, choosing an appropriate tile size is critical for achieving optimal performance. Ideally, the tile size should align with the number of threads per GPU

workgroup (block), typically 32 or 64 depending on the vendor. If the tile size is too small, GPU resources are underutilized; if too large, shared memory limits may be exceeded, reducing overall efficiency.

6.3 Experimental Evaluation

In this section, we evaluate the performance and capabilities of *tsdistances*. We first describe the datasets used in our experiments (see Section 6.3.1) and provide detailed information about the experimental setup (see Section 6.3.2). Next, we present a more in-depth view of the library's implementation and optimizations (see Section), highlighting features that contribute to its efficiency and versatility. We then compare *tsdistances* with state-of-the-art alternatives from the literature (see Section). In particular, we assess (i) DTW computation, emphasizing speed and pruning strategies, and (ii) the computation of multiple distance measures, demonstrating both efficiency and the broader functionality of the library. These experiments collectively illustrate the advantages of our implementation in terms of runtime performance, scalability, and support for a wide range of time series distances.

6.3.1 Datasets

The UCR Time Series Archive served as our primary benchmark dataset for evaluating the performance of *tsdistances*. As shown in Figure 33, which presents the relationship between dataset size (sum of the train and test sets) and time series length across the archive, the collection encompasses a wide range of configurations. The figure represents the 128 datasets in the archive, specifically those containing time series of equal length, as required by our GPU implementation. The datasets vary significantly in both dimensions, with time series lengths ranging from 24 to 2000 points and set sizes from 16 to 6164 series. This diversity makes the archive particularly suitable for comprehensive performance evaluation of time series distance computations. While experiments were conducted on the entire archive (128 equal-length datasets), one dataset (*StarLightCurves*, the 108th in alphabetical order) could not be successfully computed and was therefore excluded from all analyses, leaving 127 valid datasets. To keep the main text readable, the tables in this section report results for a representative subset of 23 benchmark datasets, chosen to cover a wide range of dataset sizes and time series lengths. The complete results for all 127 datasets are provided in Appendix B. Figure 33 shows the distribution of the selected datasets within the archive. We selected a subset that spans a range of properties, including varying numbers of time series and different sequence lengths. Table 26 lists the dataset IDs, along with the dimensions of the two sets between which distances are computed, and the length of the time series.

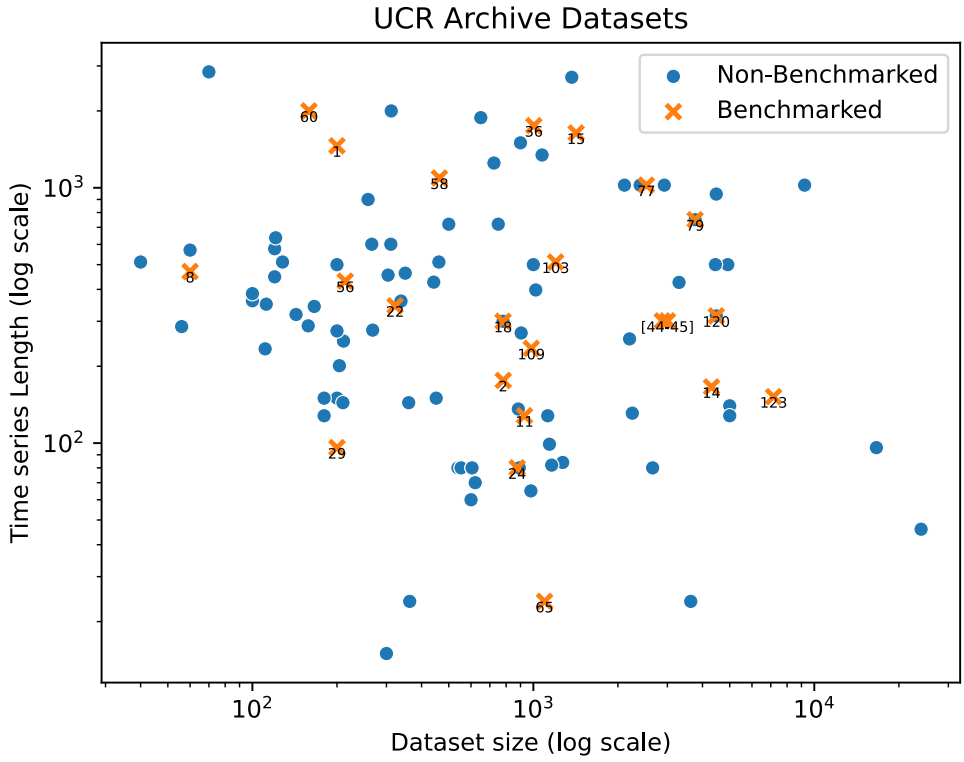

Figure 33: Dataset size vs. time-series length in the UCR datasets. The numbers refer to the dataset IDs in Table 26.

6.3.2 Experimental Details

All experiments were conducted on a workstation equipped with an Intel(R) Core(TM) i9-10980HK CPU running at 2.40GHz, featuring 8 physical cores and 16 threads. The system also included an NVIDIA GeForce RTX 2080 Super with Max-Q Design (driver version 575.57.08) and 32 GB of RAM, operating on the x86_64 architecture. The software environment consisted of Ubuntu 24.04.2 LTS as the operating system, Python 3.12 for scripting and evaluation, Rust 1.88 as the systems programming language used for backend components, Vulkan 1.4.3 for cross-platform GPU acceleration, and CUDA 12.9 for comparisons with CUDA-based baselines. All components were compiled in release mode with full optimization enabled.

To establish a robust baseline, we compared our implementation, that can be found at github.com/irazza/tsdistances, against several widely used libraries for time series distances, ensuring that each was evaluated under conditions that reflected its maximum performance. We included **DynamicAxisWarping**, a Julia-based implementation of DTW, representing the state of the art within the Julia ecosystem. For Python-based libraries, we considered **DTAIdistance**, which provides a highly optimized DTW implementation [178]. In our experiments, we enabled its most efficient configuration, namely $fast\_c$ and $use\_pruning$, to ensure that its performance was not underestimated. We also evaluated **tslearn**, a popular Python package that implements DTW and related measures such as LCSS [179]. We

further included **aeon**, which currently implements the largest collection of time series distance measures in a single library. Building upon and extending **sktime**, **aeon** incorporates numerous optimizations, most notably the use of *numba* for JIT compilation, making it both easy to understand and efficient [180]. For this reason, we excluded **sktime** from our experiments and considered only **aeon** as the more advanced option. Finally, to provide a fair comparison with GPU-accelerated approaches, we benchmarked against **cudtw**, the CUDA-based implementation of DTW described by Schmidt and Hundt in [181]. This represents, to the best of our knowledge, the only available open-source GPU implementation of DTW.

For benchmarking *tsdistances* and *aeon*, we evaluated three representative elastic distances: ERP, DTW, and ADTW. For each dataset in the UCR Archive, pairwise distances were computed between the training and test sets. Prior to timing, a warm-up phase using a small subset (at most 10 samples) was performed to eliminate initialization effects. Subsequently, 5 independent runs were executed, and we report the mean over these runs. The *tsdistances* library was evaluated under three configurations: single-threaded (`par=False`), parallel (`par=True`), and GPU (`device='gpu'`). For *aeon*, the default pairwise distance implementation was used. Numerical equivalence between the two libraries was verified with a tolerance of 10^{-8}. All timing results were stored as NumPy arrays for reproducibility. As noted above, one dataset (*StarLightCurves*) was excluded due to computation failure.

tsdistances in detail

Table 21 reports the computation times of our implementation across 23 benchmark datasets. We compare the single-threaded baseline (*sthread*), the parallelized CPU implementation (*par*), and the GPU implementation (*gpu*). On average, parallelization yields a speedup of $5.25\times$ compared to the single-threaded version, while GPU acceleration achieves a $26.59\times$ speedup. The gains are particularly pronounced for large datasets such as *NonInvasiveFetalECGThorax1*, where the runtime decreases from 3499.72 seconds to 396.61 seconds on the GPU.

	sthread	*par*	*gpu*
ACSF1	33.90	6.05	**1.00**
Adiac	5.42	0.89	**0.48**
Beef	0.47	0.08	**0.06**
CBF	1.52	0.27	**0.10**
CC	131.31	29.87	**4.64**
CCECGT	485.65	117.14	**6.79**
CX	47.57	8.33	**1.04**
DSR	0.58	**0.10**	0.24
DPOC	2.64	**0.39**	0.73
ECG200	0.29	0.05	**0.04**
EL	913.01	218.28	**79.16**

	sthread	*par*	*gpu*
FRT	91.46	21.28	**2.68**
FST	17.64	3.10	**0.55**
Ham	5.16	0.91	**0.18**
Haptics	148.35	33.44	**2.90**
HT	65.12	11.60	**0.84**
IPD	0.15	**0.03**	0.07
MSST	781.23	191.35	**97.64**
NIFECGT	3499.72	813.13	**396.61**
SA	311.14	64.15	**3.92**
Strawberry	19.27	2.94	**1.15**
UWGLX	1002.87	226.73	**52.80**
Wafer	438.83	94.93	**29.13**

Table 21: Computation times (in seconds) of our method across 23 datasets, comparing single-threaded, parallelized, and GPU implementations. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

The complete results for all 127 successfully computed UCR datasets are reported in Table 27 in Appendix B.

Figure 34 further illustrates the effect of parallelization by showing the elapsed time required to compute the DTW distance on the ACSF1 dataset (TRAIN vs. TEST), averaged over 10 repetitions, for varying numbers of CPU threads. The results indicate that runtime decreases almost linearly with the number of threads up to the 8 physical cores available on our CPU, confirming that the implementation achieves near-ideal scaling under parallel workloads. The GPU implementation consistently outperforms the CPU variants on larger datasets, although for very small datasets (e.g., *ECG200*, *ItalyPowerDemand*) GPU overhead can make the parallel CPU implementation competitive. In some cases (e.g., *DiatomSizeReduction*), the GPU is even slower than the parallel CPU version because initialization and data-transfer costs dominate the computation.

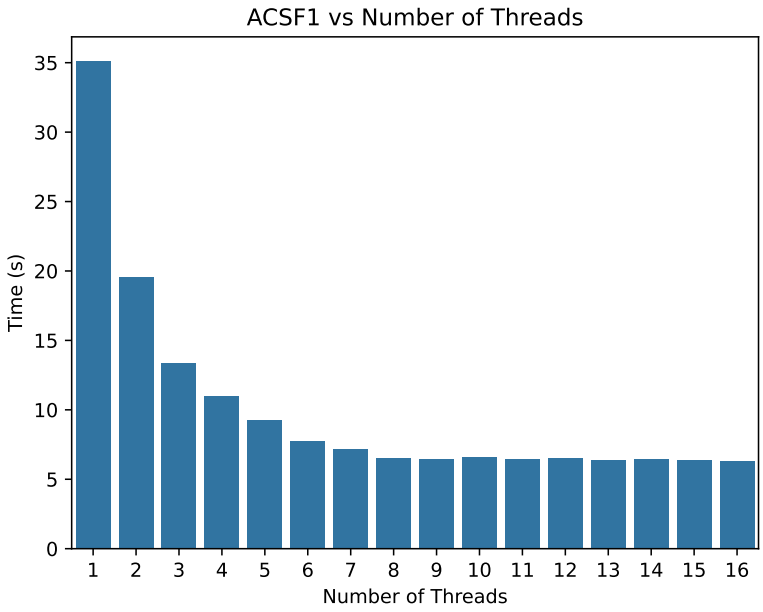

Figure 34: Elapsed time for computing the DTW distance on the ACSF1 dataset (TRAIN vs. TEST) with varying numbers of threads.

Overall, these results confirm that our library provides substantial computational benefits, making large-scale time series distance computations feasible in practice, while also highlighting the importance of selecting the appropriate backend (CPU vs. GPU) depending on dataset size and complexity.

Comparison with literature alternatives

In this subsection, we perform two sets of experiments. The first compares the running times of DTW computation on the 23 datasets introduced in Section 6.3.1. The second considers a different setting, in which we compare running times on a broader subset of distances supported by our library. In the first experiment, we compare our tool with the current state-of-the-art implementations listed in Section 6.3.2. In the second, we compare only against aeon, since it is the only competing tool that implements the full set of distances considered here.

Comparison on DTW In Table 22, we report the running times of the competing tools and our implementation. In this experiment, we used only one thread, since most tools do not support multi-threading. DTAI is the only competitor besides our implementation that supports parallelization, but it can only be enabled when computing every pairwise distance within a single set of time series, whereas our experiments compare two distinct sets of time series.

The results show that tslearn is the slowest competitor, largely because it is implemented in pure Python. DynamicAxisWarping (DAW) is often faster, sometimes by a large margin (e.g., 4× on Wafer). Although also written in Python, aeon relies on JIT compilation and is therefore substantially faster than tslearn. DTAI is the strongest competitor and remains the closest to our implementation, but our tool is still the fastest on every dataset. DTAI is a highly optimized C++

implementation of DTW that also employs the pruning technique of Silva and Batista [177]. The last row of Table 22 summarizes the average speedup of our implementation relative to the competitors. On average, our tool requires less than half the time needed by aeon and 24% less time than the highly optimized DTAI implementation.

This comparison validates the speedups achieved through our algorithm engineering techniques. As noted, these experiments used a single thread; our implementation supports multiple threads, which can further amplify the computational gains of *tsdistances*.

	DAW	DTAI	tslearn	aeon	our
ACSF1	130.14	37.82	144.73	100.03	**33.90**
Adiac	34.02	6.27	134.97	21.78	**5.42**
Beef	3.28	0.55	2.66	0.92	**0.47**
CBF	7.22	1.86	17.58	2.03	**1.52**
CC	551.12	156.52	1826.94	234.59	**131.31**
CCECGT	1052.81	585.27	824.01	710.22	**485.65**
CX	125.34	56.54	251.19	63.22	**47.57**
DSR	5.75	0.65	9.30	2.72	**0.58**
DPOC	10.72	3.51	77.33	5.32	**2.64**
ECG200	2.82	0.38	5.20	0.42	**0.29**
EL	4706.81	1103.76	5513.71	3714.13	**913.01**
FRT	256.55	116.12	654.96	178.32	**91.46**
FST	49.78	21.62	118.34	33.55	**17.64**
Ham	18.42	6.34	28.53	9.81	**5.16**
Haptics	357.45	188.50	278.38	271.11	**148.35**
HT	137.01	79.64	124.55	98.72	**65.12**
IPD	2.57	0.64	14.66	0.20	**0.15**
MSST	1592.58	974.75	1254.12	1175.93	**781.23**
NIFECGT	13783.42	4206.07	19324.78	9459.00	**3499.72**
SA	642.04	392.03	1118.10	435.69	**311.14**
Strawberry	92.54	23.40	262.73	59.67	**19.27**
UWGLX	2144.70	1385.62	5120.33	1466.56	**1002.87**
Wafer	1001.75	558.36	4462.68	644.39	**438.83**
Avg. speedup (%)	234	24	419	134	–

Table 22: Computation times (in seconds) of DTW across 23 datasets. All comparisons were performed on a single thread. The last column reports the results for our method. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

In addition, we considered GPU-based implementations. The main available competitor is cudtw, which is on average 4.61 times faster than our implementation. However, we do not provide direct comparison results with cudtw due to concerns about the correctness of its output. For CPU-based implementations, all tools produced identical results for the computed distances. Our GPU implementation, which uses 32-bit floating-point arithmetic, deviates by at most 0.67% from the CPU version (64-bit floating-point arithmetic) when evaluated using a 1% relative threshold on the distance matrix. In contrast, cudtw shows an error of 3.5% compared to the CPU version, even after accounting for the expected numerical differences from 32-bit operations (approximately 1%). Moreover, comparing the output of cudtw with our GPU implementation reveals discrepancies of up to 3%. This is unexpected, as both implementations use 32-bit floating-point arithmetic and should therefore produce much more consistent results. Furthermore, cudtw has several limitations: it only supports time series of maximum length 2047, and the series length must belong to a fixed set of values (127, 255, 512, 1023, or 2047), typically obtained through padding. It runs only on NVIDIA GPUs, and is restricted to query-vs-database tasks, which limited our ability to perform broader comparisons.

Comparison on multiple distance measures Beyond DTW, a major advantage of our implementation is its support for a broad range of time series distances. Most existing libraries, including the highly optimized DTAI and cudtw, are limited to DTW computations. Among the libraries that support multiple distances, aeon is the most relevant competitor.

To illustrate both speed and versatility, we evaluated our implementation against aeon on three representative distance measures: ERP (one of the oldest), DTW (a classical distance), and ADTW (a recent variant). Table 23 reports the running times (in seconds) on various benchmark datasets.

The table highlights two main observations: (i) our implementation consistently outperforms aeon across all considered distances and datasets, often by substantial margins; and (ii) our library supports a wider range of distances while maintaining efficient computation, thereby enabling analyses that go beyond standard DTW comparisons.

	ERP		DTW		ADTW	
	aeon	our	aeon	our	aeon	our
ACSF1	125.78	**40.14**	100.03	**33.90**	118.85	**23.42**
Adiac	26.83	**7.33**	21.78	**5.42**	25.74	**3.40**
Beef	1.14	**0.57**	0.92	**0.47**	1.10	**0.30**
CBF	2.51	**2.03**	2.03	**1.52**	2.38	**1.39**
CC	281.49	**162.33**	234.59	**131.31**	276.22	**77.56**
CCECGT	859.73	**599.88**	710.22	**485.65**	848.59	**485.08**
CX	79.68	**54.91**	63.22	**47.57**	75.26	**45.82**

DSR	3.31	**0.77**	2.72	**0.58**	3.21	**0.34**
DPOC	6.64	**2.94**	5.32	**2.64**	6.20	**1.60**
ECG200	0.52	**0.34**	0.42	**0.29**	0.49	**0.22**
EL	4503.56	**663.45**	3714.13	**913.01**	4411.40	**322.79**
FRT	216.62	**106.51**	178.32	**91.46**	211.14	**67.85**
FST	42.00	**19.70**	33.55	**17.64**	40.50	**12.73**
Ham	12.07	**7.03**	9.81	**5.16**	11.56	**3.12**
Haptics	325.19	**172.69**	271.11	**148.35**	311.06	**93.68**
HT	114.24	**71.29**	98.72	**65.12**	108.61	**70.92**
IPD	0.26	**0.18**	0.20	**0.15**	0.23	**0.13**
MSST	1476.08	**1030.31**	1175.93	**781.23**	1399.52	**745.55**
NIFECGT	11421.30	**3729.05**	9459.00	**3499.72**	11062.56	**1614.11**
SA	574.28	**390.06**	435.69	**311.14**	531.51	**277.86**
Strawberry	72.76	**19.76**	59.67	**19.27**	65.92	**8.21**
UWGLX	1832.05	**1311.82**	1466.56	**1002.87**	1779.51	**942.48**
Wafer	791.78	**497.43**	644.39	**438.83**	765.82	**393.79**

Table 23: Running times (in seconds) of aeon and our implementation on three representative distance measures: ERP, DTW, and ADTW. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

Full results for all 127 datasets are reported in Table 28 in Appendix B.

6.4 Conclusion

In this chapter, we presented *tsdistances*, a high-performance Python library for computing elastic distances between time series. Our implementation demonstrates significant performance improvements over existing solutions, as evidenced by comprehensive benchmarks across the UCR Time Series Archive. The library achieves these improvements through two main technical innovations.

First, we implemented an optimized CPU version utilizing anti-diagonal traversal of the dynamic programming matrix, enabling efficient SIMD parallelization and reduced memory usage. This approach provides substantial speedup for sequential processing while maintaining exact results. Second, we developed a novel GPU implementation that extends the anti-diagonal algorithm approach through tile-based parallelization, effectively leveraging modern graphics hardware capabilities. Our careful attention to memory management and synchronization strategies enables the processing of large-scale time series datasets that were previously computationally prohibitive.

Experimental results demonstrate that our GPU implementation consistently outperforms both sequential and parallel CPU implementations, with particularly dramatic improvements for larger datasets. The parallel CPU implementation also shows significant speedup over sequential processing, providing a valuable option when GPU hardware is unavailable.

6.4.1 Clinical Decision-Support Integration

The performance improvements demonstrated by *tsdistances* have direct implications for real-world clinical applications. Medical decision-support systems, particularly those operating in intraoperative environments, require sub-second response times for real-time feedback to surgeons. The 26.59× GPU speedup makes large-scale distance-based classification and anomaly-detection pipelines substantially more feasible when such approaches are adopted.

Consider distance-based MEP analysis pipelines, where distances between a new signal and large reference sets may need to be computed quickly to support timely feedback during surgery. The computational gains achieved by *tsdistances* could support this type of real-time analysis once integrated and validated within a clinical workflow. Furthermore, as clinical datasets grow, the ability to scale to larger patient populations and longer monitoring periods becomes critical. The library's support for multiple distance measures allows researchers to select distance metrics optimized for specific biomedical signals (EEG, MEP, EMG) and adapt them to evolving clinical requirements.

Integration of *tsdistances* into clinical AI pipelines offers several practical benefits: (1) reduced computational burden on hospital infrastructure, enabling deployment on standard clinical workstations rather than requiring specialized computing resources; (2) faster model training and validation, accelerating the development cycle for new clinical decision-support tools; (3) improved patient throughput in time-critical scenarios such as intraoperative monitoring, where faster analysis reduces surgical time without compromising accuracy.

Several directions for future work emerge from this research. One important avenue is the extension of GPU support to handle variable-length time series, as the current implementation requires equal-length sequences. Additionally, the investigation of hybrid CPU-GPU execution strategies could potentially optimize performance further based on dataset characteristics. Finally, the development of auto-tuning capabilities could help users automatically select the most efficient implementation based on their specific hardware configuration and problem parameters.

Through its combination of algorithmic innovation and careful implementation, *tsdistances* makes elastic distance computation practical for larger datasets than previously feasible, while maintaining the accuracy and flexibility required for real-world applications.

7 Conclusions

The analysis of time series remains a central challenge in artificial intelligence because many practical domains require models that are accurate, computationally efficient, and usable under limited data or real-time constraints. This thesis addressed that challenge through four complementary contributions, united by the use of forest-based ideas in both methodological development and clinical application.

First, we showed that machine learning models can reliably identify the muscle of origin of intraoperative motor evoked potentials (MEPs), supporting a clinically relevant task that is still largely manual. Second, we introduced CISOF, extending the isolation principle to anomaly detection and early anomaly detection in intraoperative MEP monitoring. Third, we proposed TSRF-Dist, a novel unsupervised time series distance derived from Random Forest concepts. Fourth, we developed *tsdistances*, a high-performance software library that makes large-scale elastic distance computation more practical on both CPU and GPU hardware. Taken together, these contributions show that forest-based approaches can support not only prediction, but also anomaly detection, distance design, and efficient algorithmic implementation.

7.1 Future Work

While the contributions of this thesis are substantial, they also highlight several limitations and open avenues for future research.

An example is the focus on univariate time series. Many real-world applications involve multivariate or structured data, where interactions between multiple variables must be modeled jointly. Extending TSRF-Dist and CISOF to multivariate settings is a natural next step. One possible approach, inspired by interval-based forests, is to incorporate a selection mechanism at each node that chooses both the interval, the dimension (or channel), and the feature of the multivariate series to split.

Another avenue for future work involves exploring alternative feature extraction strategies or subsets. For instance, frequency-domain features or different subsets of the hctsa library could be employed in the construction of the forests, potentially improving performance or interpretability.

Finally, a broader challenge is to make time series analysis methods more accessible and scalable. Maintaining and extending the tsdistances library developed in this thesis is essential. Future updates could include the integration of additional distance measures on the GPU, such as shape-based distances or MP distances, ensuring the library remains a versatile tool for the time series research community.

Overall, this thesis advances time series analysis through a combination of methodological innovation, domain-specific validation, and practical tool building. Beyond the specific results reported in each chapter, its broader message is that classical ensemble ideas, when carefully adapted to sequential data, remain a

powerful foundation for building accurate, interpretable, and scalable time series methods.

Bibliography

1. Lubba CH, Sethi SS, Knaute P, et al (2019) catch22: CAnonical Time-series CHaracteristics - Selected through highly comparative time-series analysis. Data Min Knowl Discov 33:1821–1852. https://doi.org/10.1007/s10618-019-00647-x

2. Azzari A, Bicego M, Combi C, et al TSRF-Dist: a novel time series distance based on extremely randomized canonical interval forests. Data Mining and Knowledge Discovery 39:27. https://doi.org/10.1007/s10618-025-01098-3

3. Middlehurst M, Schäfer P, Bagnall A Bake off redux: a review and experimental evaluation of recent time series classification algorithms. Data Mining and Knowledge Discovery 38:1958–2031. https://doi.org/10.1007/s10618-024-01022-1

4. Schäfer P, Leser U (2017) Fast and accurate time series classification with weasel. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. pp 637–646

5. Park D, Kim I (2022) Application of Machine Learning in the Field of Intraoperative Neurophysiological Monitoring: A Narrative Review. Applied Sciences 12:7943. https://doi.org/10.3390/app12157943

6. Tsay RS (2005) Analysis of financial time series. John wiley & sons

7. Mondal P, Shit L, Goswami S (2014) Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. International Journal of Computer Science, Engineering and Applications 4:13

8. Zeger SL, Irizarry R, Peng RD (2006) On time series analysis of public health and biomedical data. Annu Rev Public Health 27:57–79

9. Sternickel K (2002) Automatic pattern recognition in ECG time series. Computer methods and programs in biomedicine 68:109–115

10. Kashpruk N, Piskor-Ignatowicz C, Baranowski J (2023) Time series prediction in industry 4.0: A comprehensive review and prospects for future advancements. Applied sciences 13:12374

11. Mehdiyev N, Lahann J, Emrich A, et al (2017) Time series classification using deep learning for process planning: A case from the process industry. Procedia Computer Science 114:242–249

12. Abanda A, Mori U, Lozano JA (2019) A review on distance based time series classification. Data Mining and Knowledge Discovery

13. Berndt DJ, Clifford J (1994) Using dynamic time warping to find patterns in time series. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining. pp 359–370

14. Keogh EJ, Pazzani MJ (2001) Derivative Dynamic Time Warping. In: Proceedings of the First SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, April 5-7, 2001

15. Herrmann M, Webb GI (2023) Amercing: An intuitive and effective constraint for dynamic time warping. Pattern Recognit

16. Chen L, Ng RT (2004) On The Marriage of Lp-norms and Edit Distance. In: (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004

17. Paparrizos J, Gravano L k-Shape: Efficient and Accurate Clustering of Time Series. SIGMOD Rec

18. Christ M, Braun N, Neuffer J, Kempa-Liehr AW (2018) Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh - A Python package). Neurocomputing 307:72–77. https://doi.org/10.1016/j.neucom.2018.03.067

19. Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Information Sciences 239:142–153

20. Middlehurst M, Large J, Bagnall A (2020) The canonical interval forest (CIF) classifier for time series classification. In: 2020 IEEE international conference on big data (big data). pp 188–195

21. Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: a novel symbolic representation of time series. Data Mining and knowledge discovery 15:107–144

22. Hills J, Lines J, Baranauskas E, et al (2014) Classification of time series by shapelet transformation. Data mining and knowledge discovery 28:851–881

23. Schäfer P, Högqvist M (2012) SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th international conference on extending database technology. pp 516–527

24. Middlehurst M, Large J, Cawley G, Bagnall A (2021) The temporal dictionary ensemble (TDE) classifier for time series classification. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I. pp 660–676

25. Dempster A, Petitjean F, Webb GI (2020) ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery 34:1454–1495

26. Breiman L (2001) Random forests. Machine learning 45:5–32

27. Liu FT, Ting KM, Zhou Z-H (2008) Isolation forest. In: 2008 eighth ieee international conference on data mining

28. Mensi A, Bicego M, Tax DMJ (2020) Proximity Isolation Forests. In: 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021. Ieee, pp 8021–8028

29. Bicego M (2019) K-Random Forests: a K-means style algorithm for Random Forest clustering. In: Proc. Int. Joint Conf. on Neural Networks (IJCNN2019)

30. Bicego M, Escolano F (2020) On learning Random Forests for Random Forest-clustering. In: 25th International Conference on Pattern Recognition

31. Zhu X, Loy CC, Gong S (2014) Constructing Robust Affinity Graphs for Spectral Clustering. In: Proc. Int. Conf. on Computer Vision and Pattern Recognition, CVPR 2014

32. Bicego M, Cicalese F, Mensi A (2023) RatioRF: a novel measure for random forest clustering based on the Tversky's ratio model. IEEE Transactions on Knowledge and Data Engineering

33. Boaro A, Azzari A, Nunes S, et al (2022) Machine learning approaches for the automated classification of intraoperative motor evoked potentials. A pilot study. Brain and Spine 2:101359

34. Boaro A, Azzari A, Basaldella F, et al (2024) Machine learning allows expert level classification of intraoperative motor evoked potentials during neurosurgical procedures. Computers in Biology and Medicine 180:109032. https://doi.org/https://doi.org/10.1016/j.compbiomed.2024.109032

35. Azzari A, Bicego M (2024) An empirical characterization of the stability of isolation forest results. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). pp 166–176

36. Ruiz AP, Flynn M, Large J, et al (2021) The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery 35:401–449. https://doi.org/10.1007/s10618-020-00727-3

37. Calvano E, Calzolari G, Denicoló V, Pastorello S (2021) Algorithmic collusion with imperfect monitoring. International Journal of Industrial Organization 79:102712. https://doi.org/https://doi.org/10.1016/j.ijindorg.2021.102712

38. Morid MA, Sheng ORL, Dunbar J (2023) Time Series Prediction Using Deep Learning Methods in Healthcare. ACM Trans Manage Inf Syst 14:1–29

39. Shehab M, Abualigah L, Shambour Q, et al (2022) Machine learning in medical applications: A review of state-of-the-art methods. Computers in Biology and Medicine 145:105458. https://doi.org/10.1016/j.compbiomed.2022.105458

40. Chatfield C, Xing H (2019) The analysis of time series. Chapman, Hall/CRC, Seventh edition. | Boca Raton, Florida : CRC Press, [2019] |

41. Ratanamahatana CA, Lin J, Gunopulos D, et al (2010) Mining Time Series Data. In: Maimon O, Rokach L (eds) Data Mining and Knowledge Discovery Handbook. Springer US, Boston, MA, pp 1049–1077

42. Chatzigeorgakidis G, Skoutas D, Patroumpas K, et al (2017) Indexing Geolocated Time Series Data. In: Hoel E, Newsam SD, Ravada S, et al (eds) Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. Association for Computing Machinery, Redondo Beach, CA, USA, pp 1–10

43. Holder C, Middlehurst M, Bagnall A A review and evaluation of elastic distance functions for time series clustering. Knowledge and Information Systems 66:765–809. https://doi.org/10.1007/s10115-023-01952-0

44. Li W, Law KLE (2024) Deep Learning Models for Time Series Forecasting: A Review. IEEE Access 12:92306–92327. https://doi.org/10.1109/access.2024.3422528

45. Amaral K, Li Z, Ding W, et al (2022) SummerTime: Variable-length Time Series Summarization with Application to Physical Activity Analysis. ACM Trans Comput Healthcare 3:1–15. https://doi.org/10.1145/3532628

46. Shaukat K, Alam TM, Luo S, et al (2021) A Review of Time-Series Anomaly Detection Techniques: A Step to Future Perspectives. In: Arai K (ed) Advances in Information and Communication. Springer International Publishing, Cham, pp 865–877

47. Keogh E, Chu S, Hart D, Pazzani M (2004) Segmenting time series: A survey and novel approach. Data mining in time series databases 57:1–22

48. Holder C, Middlehurst M, Bagnall A (2024) A review and evaluation of elastic distance functions for time series clustering. Knowledge and Information Systems

49. Keogh EJ, Pazzani MJ (1999) Relevance feedback retrieval of time series data. In: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. pp 183–190

50. Teng M (2010) Anomaly detection on time series. In: 2010 IEEE International Conference on Progress in Informatics and Computing. pp 603–608

51. Cortes C, Vapnik V (1995) Support-vector networks. Machine learning 20:273–297

52. Schölkopf B, Smola A, Müller K-R (1997) Kernel principal component analysis. In: International conference on artificial neural networks. pp 583–588

53. Lines J, Bagnall AJ (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Discov 29:565–592. https://doi.org/10.1007/s10618-014-0361-2

54. Ratanamahatana C (Ann), Keogh EJ (2005) Three Myths about Dynamic Time Warping Data Mining. In: Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005

55. Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing 26:43–49. https://doi.org/10.1109/tassp.1978.1163055

56. Itakura F (1975) Minimum prediction residual principle applied to speech recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing 23:67–72. https://doi.org/10.1109/tassp.1975.1162641

57. Jeong Y-S, Jeong MK, Omitaomu OA (2011) Weighted dynamic time warping for time series classification. Pattern Recognit

58. Chen L, Özsu MT, Oria V (2005) Robust and Fast Similarity Search for Moving Object Trajectories. In: Özcan F (ed) Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005. Acm, pp 491–502

59. Stefan A, Athitsos V, Das G (2013) The Move-Split-Merge Metric for Time Series. IEEE Trans Knowl Data Eng

60. Marteau P-F (2009) Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. IEEE Trans Pattern Anal Mach Intell

61. Gharghabi S, Imani S, Bagnall A, et al (2018) Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios. In: 2018 IEEE International Conference on Data Mining (ICDM). pp 965–970

62. Fulcher BD, Little MA, Jones NS (2013) Highly comparative time-series analysis: the empirical structure of time series and their methods. Journal of the Royal Society Interface 10:20130048

63. Mietus JE (2002) The pNNx files: re-examining a widely used heart rate variability measure. Heart

64. Wang X, Wirth A, Wang L (2007) Structure-based statistical features and multivariate time series clustering. In: Proceedings—IEEE International Conference on Data Mining, ICDM

65. Fisher RA (1922) On the Interpretation of $\chi 2$ from Contingency Tables, and the Calculation of P. Journal of the Royal Statistical Society 85:87–94

66. Massey Jr FJ (1951) The Kolmogorov-Smirnov test for goodness of fit. Journal of the American statistical Association 46:68–78

67. Kendall MG (1938) A new measure of rank correlation. Biometrika 30:81–93

68. Benjamini Y, Yekutieli D (2001) The control of the false discovery rate in multiple testing under dependency: The Annals of Statistics. Institute of Mathematical Statistics 29:1165–1188

69. Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data mining and knowledge discovery 22:149–182

70. Large J, Lines J, Bagnall A (2019) A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates. Data mining and knowledge discovery 33:1674–1709

71. Bostrom A, Bagnall A, Lines J (2016) Evaluating improvements to the shapelet transform. In: Knowledge Discovery and Data Mining, in Workshop on Mining and Learning from Time Series

72. Bostrom A, Bagnall A (2017) Binary shapelet transform for multiclass time series classification. Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXII: Special Issue on Big Data Analytics and Knowledge Discovery 24–46

73. Rodriguez JJ, Kuncheva LI, Alonso CJ (2006) Rotation forest: A new classifier ensemble method. IEEE transactions on pattern analysis and machine intelligence 28:1619–1630

74. Karlsson I, Papapetrou P, Boström H (2016) Generalized random shapelet forests. Data mining and knowledge discovery 30:1053–1085

75. Le Nguyen T, Gsponer S, Ifrim G (2017) Time series classification by sequence learning in all-subsequence space. In: 2017 IEEE 33rd international conference on data engineering (ICDE). pp 947–958

76. Le Nguyen T, Gsponer S, Ilie I, et al (2019) Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data mining and knowledge discovery 33:1183–1222

77. Le Nguyen T, Ifrim G (2022) Fast time series classification with random symbolic subsequences. In: International Workshop on Advanced Analytics and Learning on Temporal Data. pp 50–65

78. Guillaume A, Vrain C, Elloumi W (2022) Random dilated shapelet transform: A new approach for time series shapelets. In: International Conference on Pattern Recognition and Artificial Intelligence. pp 653–664

79. Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery 29:1505–1530

80. Grabocka J, Schilling N, Wistuba M, Schmidt-Thieme L (2014) Learning time-series shapelets. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp 392–401

81. Ismail Fawaz H, Lucas B, Forestier G, et al (2020) Inceptiontime: Finding alexnet for time series classification. Data Mining and Knowledge Discovery 34:1936–1962

82. Ismail Fawaz H, Forestier G, Weber J, et al (2019) Deep learning for time series classification: a review. Data mining and knowledge discovery 33:917–963

83. Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: A strong baseline. In: 2017 International joint conference on neural networks (IJCNN). pp 1578–1585

84. Mohammadi Foumani N, Miller L, Tan CW, et al (2024) Deep learning for time series classification and extrinsic regression: A current survey. ACM Computing Surveys 56:1–45

85. Woo G, Liu C, Kumar A, et al (2024) Unified training of universal time series forecasting transformers

86. Ansari AF, Stella L, Turkmen C, et al (2024) Chronos: Learning the language of time series. arXiv preprint arXiv:240307815

87. Hastie T, Tibshirani R, Friedman JH, Friedman JH (2009) The elements of statistical learning: data mining, inference, and prediction. Springer

88. Dietterich TG (2000) Ensemble methods in machine learning. In: International workshop on multiple classifier systems. pp 1–15

89. Breiman L (1996) Bagging predictors. Machine learning 24:123–140

90. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences 55:119–139

91. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Annals of statistics 1189–1232

92. Boulesteix A-L, Janitza S, Kruppa J, König IR (2012) Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2:493–507

93. Gislason PO, Benediktsson JA, Sveinsson JR (2006) Random forests for land cover classification. Pattern recognition letters 27:294–300

94. Sarica A, Cerasa A, Quattrone A (2017) Random forest algorithm for the classification of neuroimaging data in Alzheimer's disease: a systematic review. Frontiers in aging neuroscience 9:329

95. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Machine learning 63:3–42

96. Menze BH, Kelm BM, Splitthoff DN, et al (2011) On oblique random forests. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp 453–469

97. Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS (2008) Random survival forests

98. Liu FT, Ting KM, Zhou Z-H (2012) Isolation-Based Anomaly Detection. ACM Trans Knowl Discov Data

99. Flynn M, Large J, Bagnall T (2019) The contract random interval spectral ensemble (c-RISE): The effect of contracting a classifier on accuracy. In: Hybrid Artificial Intelligent Systems: 14th International Conference, HAIS 2019, León, Spain, September 4–6, 2019, Proceedings 14. pp 381–392

100. Cabello N, Naghizade E, Qi J, Kulik L (2020) Fast and accurate time series classification through supervised interval search. In: 2020 IEEE international conference on data mining (ICDM). pp 948–953

101. Cabello N, Naghizade E, Qi J, Kulik L (2024) Fast, accurate and explainable time series classification through randomization. Data Mining and Knowledge Discovery 38:748–811

102. Middlehurst M, Large J, Flynn M, et al (2021) HIVE-COTE 2.0: a new meta ensemble for time series classification. Machine Learning 110:3211–3243

103. Sathe S, Aggarwal CC (2017) Similarity forests. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp 395–403

104. Lucas B, Shifaz A, Pelletier C, et al Proximity forest: an effective and scalable distance-based classifier for time series. Data Mining and Knowledge Discovery 33:607–635. https://doi.org/10.1007/s10618-019-00617-3

105. Mensi A, Tax DMJ, Bicego M (2023) Detecting outliers from pairwise proximities: Proximity isolation forests. Pattern Recognit 138:109334. https://doi.org/10.1016/j.patcog.2023.109334

106. Hannun AY, Rajpurkar P, Haghpanahi M, et al (2019) Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature medicine 25:65–69

107. MacDonald DB (2006) Intraoperative motor evoked potential monitoring: overview and update. Journal of clinical monitoring and computing 20:347–377

108. Quitadamo L, Cavrini F, Sbernini L, et al (2017) Support vector machines to detect physiological patterns for EEG and EMG-based human–computer interaction: a review. Journal of neural engineering 14:11001

109. Di Nardo F, Nocera A, Cucchiarelli A, et al (2022) Machine learning for detection of muscular activity from surface EMG signals. Sensors 22:3393

110. Rossini PM, Barker A, Berardelli A, et al (1994) Non-invasive electrical and magnetic stimulation of the brain, spinal cord and roots: basic principles and procedures for routine clinical application. Report of an IFCN committee. Electroencephalography and clinical neurophysiology 91:79–92

111. Patton HD, Amassian VE (1954) Single-and multiple-unit analysis of cortical stage of pyramidal tract activation. Journal of neurophysiology 17:345–363

112. Legatt AD (2002) Current practice of motor evoked potential monitoring: results of a survey. Journal of clinical neurophysiology 19:454–460

113. Leppanen RE (2005) Intraoperative monitoring of segmental spinal nerve root function with free-run and electrically-triggered electromyography and spinal cord function with reflexes and F-responses: a position statement by the American Society of Neurophysiological Monitoring. Journal of Clinical Monitoring and Computing 19:437–461

114. Deletis V, Sala F (2008) Intraoperative neurophysiological monitoring of the spinal cord during spinal cord and spine surgery: A review focus on the corticospinal tracts. Clinical Neurophysiology 119:248–264. https://doi.org/10.1016/j.clinph.2007.09.135

115. Sala F, Bricolo A, Faccioli F, et al (2007) Surgery for intramedullary spinal cord tumors: the role of intraoperative (neurophysiological) monitoring. European Spine Journal 16:130–139. https://doi.org/10.1007/s00586-007-0423-x

116. Sala F, Manganotti P, Tramontano V, et al (2007) Monitoring of motor pathways during brain stem surgery: What we have achieved and what we still miss?. Neurophysiologie Clinique/Clinical Neurophysiology 37:399–406. https://doi.org/10.1016/j.neucli.2007.09.013

117. Levy W (1987) Transcranial stimulation of the motor cortex to produce motor-evoked potentials.. Medical instrumentation 21:248–254

118. TSuTSui S, Yamada H (2016) Basic principles and recent trends of transcranial motor evoked potentials in intraoperative neurophysiologic monitoring. Neurologia medico-chirurgica 56:451–456

119. Kombos T, SüSS O (2009) Neurophysiological basis of direct cortical stimulation and applied neuroanatomy of the motor cortex: a review. Neurosurgical focus 27:E3

120. Ringel F, Sala F (2015) Intraoperative mapping and monitoring in supratentorial tumor surgery.. Journal of neurosurgical sciences 59:129–139

121. Boaro A, Sala F (2022) Intraoperative neurophysiology during intramedullary spinal cord tumor surgery. Koht, Sloan, Toleikis's Monitoring the Nervous System for Anesthesiologists and Other Health Care Professionals 635–645

122. Goetz SM, Luber B, Lisanby SH, Peterchev AV (2014) A novel model incorporating two variability sources for describing motor evoked potentials. Brain stimulation 7:541–552

123. Goetz SM, Alavi SM, Deng Z-D, Peterchev AV (2019) Statistical model of motor-evoked potentials. IEEE Transactions on Neural Systems and Rehabilitation Engineering 27:1539–1545

124. Machetanz K, Gallotti AL, Leao Tatagiba MT, et al (2021) Time-frequency representation of motor evoked potentials in brain tumor patients. Frontiers in neurology 11:633224

125. Senders JT, Staples PC, Karhade AV, et al (2018) Machine learning and neurosurgical outcome prediction: a systematic review. World neurosurgery 109:476–486

126. Boaro A, Kaczmarzyk JR, Kavouridis VK, et al (2022) Deep neural networks allow expert-level brain meningioma segmentation and present potential for improvement of clinical practice. Scientific Reports 12:15462

127. Goedmakers CM, Lak AM, Duey AH, et al (2021) Deep learning for adjacent segment disease at preoperative MRI for cervical radiculopathy. Radiology 301:664–671

128. Lega C, Pirruccio M, Bicego M, et al (2020) The topography of visually guided grasping in the premotor cortex: a dense-transcranial magnetic stimulation (tms) mapping study. Journal of Neuroscience 40:6790–6800

129. Boaro A, Leung J, Reeder HT, et al (2021) Smartphone GPS signatures of patients undergoing spine surgery correlate with mobility and current gold standard outcome measures. Journal of Neurosurgery: Spine 35:796–806

130. Stålberg E, Erdem H (2002) Quantitative motor unit potential analysis in routine.. Electromyography and clinical neurophysiology 42:433–442

131. Guyon I, Weston J, Barnhill S, Vapnik V (2002) Gene selection for cancer classification using support vector machines. Machine learning 46:389–422

132. Wermelinger J, Parduzi Q, Sariyar M, et al (2023) Opportunities and challenges of supervised machine learning for the classification of motor evoked potentials according to muscles. BMC Medical Informatics and Decision Making 23:198

133. Marcelli E, Barbariol T, Sartor D, Susto GA (2024) Active Learning-based Isolation Forest (ALIF): Enhancing anomaly detection with expert feedback. Information Sciences 678:121012

134. Kumar A, Kumar A, Raja R, et al (2025) Revolutionising anomaly detection: a hybrid framework for anomaly detection integrating isolation forest, autoencoder, and Conv. LSTM. Knowledge and Information Systems 67:11903–11953

135. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and Regression Trees

136. Preiss BR (1999) Data Structures and Algorithms with Object-Oriented Design Patterns in Java. Wiley

137. Samariya D, Thakkar A (2023) A Comprehensive Survey of Anomaly Detection Algorithms. Annals of Data Science

138. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research

139. Wilcoxon F (1945) Individual Comparisons by Ranking Methods. Biometrics Bulletin

140. Li G, Jung JJ (2023) Deep learning for anomaly detection in multivariate time series: Approaches, applications, and challenges. Information Fusion 91:93–102. https://doi.org/https://doi.org/10.1016/j.inffus.2022.10.008

141. Schölkopf B, Platt JC, Shawe-Taylor J, et al (2001) Estimating the Support of a High-Dimensional Distribution. Neural Comput 13:1443–1471

142. Kriegel H-P, Kröger P, Schubert E, Zimek A (2009) Outlier Detection in Axis-Parallel Subspaces of High Dimensional Data. In: Theeramunkong T, Kijsirikul B, Cercone N, Ho TB (eds) Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, 2009, Proceedings. Springer, pp 831–838

143. Breunig MM, Kriegel H-P, Ng RT, Sander J (2000) LOF: Identifying Density-Based Local Outliers. In: Chen W, Naughton JF, Bernstein PA (eds) Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA. Acm, pp 93–104

144. Goldstein M, Dengel A (2012) Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. KI-2012: poster and demo track

145. Li Z, Zhao Y, Hu X, et al (2023) ECOD: Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions. IEEE Trans Knowl Data Eng 35:12181–12193. https://doi.org/10.1109/tkde.2022.3159580

146. Rousseeuw PJ, Driessen K van (1999) A Fast Algorithm for the Minimum Covariance Determinant Estimator. Technometrics 41:212–223

147. Mishra K, Basu S, Maulik U (2022) Graft: A graph based time series data mining framework. Engineering Applications of Artificial Intelligence 110:104695. https://doi.org/10.1016/j.engappai.2022.104695

148. Zhao K, Wulder MA, Hu T, et al (2019) Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment 232:111181. https://doi.org/10.1016/j.rse.2019.04.034

149. Lu S, Lu J, An K, et al (2023) Edge Computing on IoT for Machine Signal Processing and Fault Diagnosis: A Review. IEEE Internet of Things Journal

150. Karevan Z, Suykens JA (2020) Transductive LSTM for time-series prediction: An application to weather forecasting. Neural Networks

151. Chen K, Zhang D, Yao L, et al (2021) Deep Learning for Sensor-based Human Activity Recognition: Overview, Challenges, and Opportunities. ACM Comput Surv

152. Mitra R, MacLean AL (2021) RVAgene: generative modeling of gene expression time series data. Bioinformatics

153. Wang X, Mueen A, Ding H, et al (2013) Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery

154. Esling P, Agon C (2012) Time-series data mining. ACM Computing Surveys (CSUR)

155. Criminisi A, Shotton J, Konukoglu E (2012) Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Foundations and Trends in Computer Graphics and Vision

156. Quinlan JR (1993) C4.5: Programs for Machine Learning

157. Moosmann F, Triggs B, Jurie F (2006) Fast Discriminative Visual Codebooks using Randomized Clustering Forests. In: Advances in Neural Information Processing Systems 19

158. Shotton J, Johnson M, Cipolla R (2008) Semantic texton forests for image categorization and segmentation. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)

159. Yan D, Chen A, Jordan MI (2013) Cluster forests. Computational Statistics & Data Analysis

160. Shi T, Horvath S (2006) Unsupervised Learning With Random Forest Predictors. Journal of Computational and Graphical Statistics

161. Ting KM, Zhu Y, Carman M, et al (2016) Overcoming Key Weaknesses of Distance-based Neighbourhood Methods Using a Data Dependent Dissimilarity Measure. In: Proc. Int. Conf. on Knowledge Discovery and Data Mining

162. Aryal S, Ting KM, Washio T, Haffari G (2020) A comparative study of data-dependent approaches without learning in measuring similarities of data objects. Data Min Knowl Discov

163. Ting KM, Zhu Y, Carman M, et al (2019) Lowest probability mass neighbour algorithms: relaxing the metric constraint in distance-based neighbourhood algorithms. Machine Learning

164. Aryal S, Ting KM, Haffari G, Washio T (2014) mp-dissimilarity: A data dependent dissimilarity measure. In: 2014 IEEE International Conference on Data Mining

165. Aryal S, Ting KM, Washio T, Haffari G (2017) Data-dependent dissimilarity measure: an effective alternative to geometric distance measures. Knowledge and information systems

166. Bicego M, Cicalese F (2023) On the Good Behaviour of Extremely Randomized Trees in Random Forest-Distance Computation. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases

167. Ting KM, Zhu Y, Zhou Z-H (2018) Isolation kernel and its effect on SVM. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining

168. Dau HA, Bagnall A, Kamgar K, et al (2019) The UCR time series archive. IEEE/CAA Journal of Automatica Sinica

169. Breiman L (2000) Some infinity theory for predictor ensembles

170. Davies A, Ghahramani Z (2014) The random forest kernel and other kernels for big data from random partitions. arXiv preprint arXiv:14024293

171. Tversky A (1977) Features of similarity. Psychological review

172. Ran X, Xi Y, Lu Y, et al (2023) Comprehensive survey on hierarchical clustering algorithms and the recent developments. Artif Intell Rev

173. Hubert L, Arabie P (1985) Comparing partitions. Journal of classification

174. Pedregosa F, Varoquaux G, Gramfort A, et al (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12:2825–2830

175. Hughey R (1996) Parallel hardware for sequence comparison and alignment. Comput Appl Biosci 12:473–479

176. Castells-Rufas D (2023) GPU acceleration of Levenshtein distance computation between long strings. Parallel Comput 116:103019. https://doi.org/10.1016/j.parco.2023.103019

177. Silva DF, Batista GEAPA (2016) Speeding Up All-Pairwise Dynamic Time Warping Matrix Calculation. In: Venkatasubramanian SC, Jr. WM (eds) Proc. of the 2016 SIAM International Conference on Data Mining. SIAM, pp 837–845

178. Meert W, Hendrickx K, Van Craenendonck T, et al (2020) DTAIDistance (Version v2)

179. Tavenard R, Faouzi J, Vandewiele G, et al (2020) Tslearn, A Machine Learning Toolkit for Time Series Data. Journal of Machine Learning Research 21:1–6

180. Middlehurst M, Ismail-Fawaz A, Guillaume A, et al (2024) aeon: a Python Toolkit for Learning from Time Series. Journal of Machine Learning Research 25:1–10

181. Schmidt B, Hundt C (2020) cuDTW++: Ultra-Fast Dynamic Time Warping on CUDA-Enabled GPUs. In: Malawski M, Rzadca K (eds) Euro-Par 2020: Parallel Processing - 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24-28, 2020, Proceedings. Springer, pp 597–612

Appendix

A TSRF-Dist: A novel Time Series distance

	TSRF-Dist	ERP	WDTW	ADTW	cDTW	TWE
ACSF1	**0.4287 (0.0257)**	0.0233	0.0051	0.0069	0.0099	0.0252
Adiac	**0.4498 (0.0144)**	0.0259	0.0795	0.0231	0.0797	0.0224
AGWX	**0.1253 (0.0109)**	0.0137	0.0123	0.0131	0.0074	0.0033
AGWY	**0.0955 (0.0086)**	0.0067	0.0119	0.0113	0.0087	0.0021
AGWZ	**0.1077 (0.0112)**	0.0099	0.0113	0.0113	0.0085	0.0064
AH	**0.2979 (0.0807)**	−0.0028	−0.0037	−0.0000	−0.0031	0.0042
BME	**0.2882 (0.0717)**	0.1376	0.1366	0.1404	0.1404	0.1373
Beef	**0.1414 (0.0146)**	0.0629	0.0348	0.0658	0.0658	0.0305
BF	**0.5299 (0.1949)**	0.0052	−0.0093	0.0722	0.0052	0.3438
BC	**0.0506 (0.0665)**	0.0000	−0.0221	−0.0150	0.0240	−0.0221
CBF	0.4964 (0.2020)	0.1419	0.2769	0.7377	0.2288	0.6521
Car	**0.1682 (0.0661)**	0.1134	0.1058	0.1067	0.1061	0.1146
Chinatown	**0.5628 (0.3547)**	0.0456	0.0164	0.0164	0.0211	0.0456
CC	0.0131 (0.0179)	0.0193	0.0056	0.0092	0.0056	0.0101
CCECGT	**0.9978 (0.0093)**	0.0237	0.2480	0.0116	0.4308	0.0003
Coffee	**0.7272 (0.1067)**	−0.0087	0.0158	0.0026	0.0158	0.0026
Computers	**0.1705 (0.0472)**	0.0049	−0.0004	−0.0002	−0.0005	0.0018
CX	**0.1609 (0.0139)**	0.1010	0.1166	0.1201	0.0857	0.0552
CY	**0.1984 (0.0177)**	0.1444	0.1204	0.1354	0.0781	0.1155
CZ	**0.1620 (0.0195)**	0.0656	0.1122	0.1134	0.1131	0.0746
Crop	**0.2427 (0.0353)**	0.2418	0.2105	0.2331	0.2291	0.2126
DSR	**1.0000 (0.0081)**	0.0123	0.0123	0.0123	0.0190	0.0123
DPOAG	0.2130 (0.0000)	0.4220	0.4220	0.4185	0.4220	0.4169
DPOC	**0.0086 (0.0276)**	0.0014	0.0014	0.0027	0.0027	0.0014
DPTW	0.3274 (0.0001)	0.6733	0.5882	0.5938	0.6710	0.6610
DLD	0.2007 (0.0449)	0.1786	0.2033	−0.0002	0.2023	0.2027
DLG	0.0046 (0.0183)	0.0047	0.0010	0.0010	−0.0009	0.0010
DLW	0.8508 (0.0041)	0.9229	−0.0074	−0.0074	−0.0074	−0.0074
ECG200	**0.2767 (0.0604)**	0.0242	0.0098	0.0242	0.0507	0.0098
ECG5000	0.4420 (0.0545)	0.7128	0.7291	0.8222	0.0177	0.0016
ECGFD	**0.4449 (0.0238)**	0.0000	0.0138	0.0009	0.0033	0.0059
EOGHS	**0.2675 (0.0045)**	0.1707	0.0889	0.1551	0.1479	0.0004
EOGVS	**0.2111 (0.0313)**	0.0218	0.0304	0.0511	0.0230	0.0004
Earthquakes	−0.0330 (0.0202)	−0.0032	−0.0889	−0.0032	−0.0032	−0.0033
ED	**0.3436 (0.0308)**	0.1489	−0.0001	0.1794	0.0031	0.0907
EL	**0.0050 (0.0298)**	0.0011	0.0003	0.0000	0.0020	0.0018

	TSRF-Dist	ERP	WDTW	ADTW	cDTW	TWE
FA	**0.6074 (0.0583)**	0.0877	0.1105	0.1172	0.0450	0.0860
FF	0.6560 (0.0017)	0.1458	0.1365	0.1458	0.1365	0.6964
FUCR	**0.6095 (0.0141)**	0.0834	0.1105	0.1172	0.0450	0.0781
FW	0.4635 (0.0248)	0.2680	0.5525	0.5909	0.5329	0.5291
Fish	**0.3937 (0.0412)**	0.0010	0.0013	0.0053	0.0013	0.0013
FA	**0.0205 (0.0309)**	−0.0000	0.0000	−0.0000	0.0000	−0.0001
FB	**0.0854 (0.0453)**	−0.0001	−0.0001	0.0000	−0.0001	−0.0000
FRT	**0.2814 (0.0309)**	0.0002	0.0002	0.0002	0.0002	0.0001
FST	**0.2860 (0.0427)**	0.0002	0.0002	0.0002	0.0002	0.0002
Fungi	0.9271 (0.0394)	0.9568	0.7665	0.7238	0.8380	0.8221
GMAD	**0.3826 (0.0456)**	0.0324	0.1059	0.1406	0.1014	0.0152
GMAD	**0.3221 (0.0007)**	0.1253	0.2308	0.2141	0.2491	0.0122
GMAD	**0.2430 (0.0183)**	0.0270	0.0416	0.0357	0.0295	0.0078
GPZ	0.3767 (0.0142)	0.4038	0.3895	0.1192	0.2666	0.0516
GPZ	0.3890 (0.0091)	0.4038	0.3895	0.1192	0.2666	0.0516
GP	−0.0050 (0.0536)	0.0308	−0.0049	0.0308	0.0091	0.0091
GPAS	**0.0744 (0.0552)**	−0.0017	−0.0017	−0.0017	−0.0017	−0.0017
GunPMVF	0.0428 (0.0000)	0.2340	0.2340	0.2340	0.2340	0.2340
GunPOVY	0.0447 (0.0029)	0.2384	0.2384	0.2384	0.2384	0.2384
Ham	**0.0147 (0.0023)**	0.0007	0.0007	0.0007	0.0007	0.0007
HO	**0.1842 (0.0021)**	0.0160	0.0022	0.0011	0.0011	0.0011
Haptics	**0.0895 (0.0195)**	0.0011	0.0020	0.0023	0.0036	0.0023
Herring	**−0.0040 (0.0884)**	−0.0053	−0.0053	−0.0210	−0.0053	−0.0122
HT	**0.3635 (0.0101)**	0.0071	0.0034	−0.0027	0.0201	−0.0027
IS	**0.0216 (0.0071)**	0.0073	0.0084	0.0089	0.0093	0.0096
InsectEPGRT	0.4211 (0.2009)	1.0000	1.0000	1.0000	1.0000	1.0000
InsectEPGST	0.4460 (0.0049)	1.0000	1.0000	1.0000	1.0000	1.0000
InsectWS	**0.3714 (0.0425)**	0.0161	0.2002	0.2763	0.2949	0.1069
IPD	0.0025 (0.0232)	0.0005	−0.0004	−0.0003	0.0221	0.0124
LKA	0.0266 (0.0189)	−0.0000	0.0030	−0.0000	0.0364	−0.0000
Lightning2	0.0846 (0.0023)	0.1252	0.1202	0.1202	0.1202	0.1097
Lightning7	0.3610 (0.0180)	0.4275	0.3541	0.4545	0.1938	0.3508
Mallat	**0.9724 (0.0552)**	0.5925	0.8654	0.7610	0.8569	0.5925
Meat	**0.8337 (0.0352)**	0.5316	0.5551	0.5551	0.5551	0.5316
MI	0.0592 (0.0518)	0.0022	0.1102	−0.0746	−0.0316	−0.0574
MP	**0.4288 (0.1080)**	0.1341	0.1418	0.1399	0.1436	0.0619
MPOAG	**0.4237 (0.0247)**	0.4040	0.4040	0.0040	0.4040	0.4084

	TSRF-Dist	ERP	WDTW	ADTW	cDTW	TWE
MPOC	−0.0054 (0.0171)	−0.0009	−0.0009	−0.0009	−0.0009	0.0014
MPTW	0.3208 (0.0023)	0.5175	0.5156	0.5204	0.5128	0.5239
MSRT	**0.6728 (0.0000)**	0.1174	0.1319	0.0165	0.3003	0.0007
MSST	**0.6544 (0.0531)**	0.1149	0.1249	0.0160	0.1256	0.0991
MS	**0.4879 (0.0712)**	0.0003	0.0005	0.0005	0.0005	0.0003
NIFECGT	**0.5998 (0.0564)**	0.1011	0.0738	0.0892	0.0807	0.0277
NIFECGT	**0.6992 (0.0875)**	0.1173	0.1175	0.1303	0.1295	0.0456
OSULeaf	**0.3301 (0.0225)**	0.0982	0.0523	0.0936	0.0155	0.1402
OO	**0.7145 (0.0147)**	0.5132	0.4615	0.4615	0.4615	0.5132
PLAID	**0.1367 (0.0276)**	0.0157	0.0181	0.0238	0.0206	0.0020
POC	−0.0008 (0.0240)	0.0008	0.0006	0.0006	0.0012	0.0006
Phoneme	**0.1334 (0.0138)**	0.0750	0.0285	0.0597	0.0359	0.0041
PGWZ	0.3410 (0.0000)	0.2514	0.3102	0.3290	0.3222	0.3596
PAP	0.0629 (0.0070)	0.0807	0.0783	0.0716	0.0719	0.0869
PAP	**0.4075 (0.0280)**	0.3136	0.2460	0.2696	0.2583	0.1617
PCVP	0.0821 (0.0025)	0.0590	0.0421	0.0984	0.0557	0.1158
Plane	0.9544 (0.0276)	0.8419	0.8464	0.7039	0.7002	1.0000
PC	**0.0590 (0.0085)**	0.0022	0.0000	0.0040	0.0074	0.0156
PPOAG	0.5315 (0.0700)	0.5395	0.5395	0.5411	0.5395	0.5411
PPOC	0.0409 (0.0420)	0.0745	0.0748	0.0050	0.0748	0.0760
PPTW	0.3555 (0.0892)	0.6779	0.6653	0.5864	0.6632	0.5677
RD	**0.0302 (0.0000)**	0.0097	−0.0000	−0.0001	−0.0000	0.0095
Rock	0.2343 (0.0244)	0.2784	0.3678	0.2643	0.1985	0.1679
ST	**0.0320 (0.0166)**	0.0003	0.0000	0.0000	−0.0000	0.0005
SemgHand-Gender Ch2	**0.0818 (0.0552)**	−0.0052	0.0034	0.0034	0.0034	−0.0050
SemgHand-Movement Ch2	**0.0932 (0.0199)**	0.0009	0.0398	0.0088	0.0444	0.0001
SemgHand-Subject Ch2	**0.2797 (0.0282)**	0.0002	0.0125	0.0009	0.0212	0.0001
SGWZ	**0.5230 (0.0181)**	0.2810	0.3140	0.2734	0.2009	0.3137
SS	**0.4763 (0.0548)**	0.0000	0.0000	0.0383	−0.0002	0.0000
SA	**0.4809 (0.0353)**	0.0833	0.1011	0.0948	0.1004	0.0969
SKA	**0.2059 (0.2793)**	0.0002	0.0000	0.0008	0.0014	0.0002
SS	0.6797 (0.0040)	0.9410	0.2169	0.8154	0.5361	0.9702

	TSRF-Dist	ERP	WDTW	ADTW	cDTW	TWE
SonyAIBORS	**0.6044 (0.0122)**	−0.0085	−0.0117	−0.0116	−0.0117	−0.0029
SonyAIBORS	**0.2977 (0.0203)**	0.0025	0.0025	0.0025	0.0012	0.0025
SLC	0.5160 (0.0778)	0.6764	0.6757	0.6743	0.6763	0.6720
Strawberry	**0.0417 (0.0523)**	−0.0186	−0.0270	−0.0315	−0.0186	−0.0264
SL	**0.6550 (0.0019)**	0.0360	0.0386	0.0568	0.0392	0.0668
Symbols	**0.9519 (0.0360)**	0.6757	0.6752	0.6752	0.4818	0.6744
SC	**0.7522 (0.0846)**	0.6268	0.6786	0.6654	0.6786	0.6261
TS	**0.0598 (0.0380)**	0.0245	0.0119	0.0015	−0.0035	0.0477
TS	0.0310 (0.0154)	0.0466	0.0699	0.0699	0.0093	0.0466
Trace	0.3880 (0.0393)	0.6649	0.7047	0.6304	0.4225	0.3984
TLECG	**0.0138 (0.0559)**	0.0000	0.0000	0.0000	0.0002	0.0002
TP	0.0094 (0.0572)	0.9995	0.0001	0.1339	0.0839	0.0226
UMD	0.1649 (0.0566)	0.1561	0.1455	0.1131	0.1715	0.1115
UWGLA	**0.6819 (0.0188)**	0.1338	0.6375	0.3003	0.4618	0.2183
UWGLX	**0.4742 (0.0123)**	0.2541	0.3227	0.3399	0.3484	0.2300
UWGLY	0.3010 (0.0150)	0.2395	0.3105	0.2990	0.3301	0.2832
UWGLZ	**0.4504 (0.0286)**	0.3185	0.2737	0.2673	0.2947	0.2870
Wafer	**0.1432 (0.0461)**	0.0014	−0.0001	−0.0001	−0.0001	0.0008
Wine	−0.0004 (0.0340)	0.0050	0.0050	0.0050	0.0050	0.0050
WS	0.2740 (0.0215)	0.1430	0.2846	0.2733	0.3665	0.3117
Worms	0.1767 (0.0002)	0.0594	0.0646	0.1341	0.1191	0.1788
WTC	**0.0147 (0.0113)**	−0.0042	−0.0032	−0.0116	−0.0148	−0.0117
Yoga	0.0024 (0.0130)	−0.0034	−0.0022	0.0064	−0.0021	−0.0001

Table 24: Complete ARIs results from our analysis of TSRF-Dist and the top 5 literature alternatives (with higher mean ARIs over all datasets), including the standard deviations. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

B tsdistances

BA Appendix Tables

This appendix reports the complete timing results for all 127 UCR datasets with equal-length series used in our experiments. Table 25 provides the mapping between the original UCR dataset names and the shortened labels used throughout the thesis tables. Table 26 summarizes dataset sizes and series lengths, while Table 27 and Table 28 provide the full runtime comparisons. Each timing table ends with the column means and the average speedup relative to the baseline implementation to summarize overall performance.

Dataset Name	Short	Dataset Name	Short
ACSF1	ACSF1	Adiac	Adiac
AllGestureWiimoteX	AGWX	AllGestureWiimoteY	AGWY
AllGestureWiimoteZ	AGWZ	ArrowHead	AH
BME	BME	Beef	Beef
BeetleFly	BF	BirdChicken	BC
CBF	CBF	Car	Car
Chinatown	Chinatown	ChlorineConcentration	CC
CinCECGTorso	CCECGT	Coffee	Coffee
Computers	Computers	CricketX	CX
CricketY	CY	CricketZ	CZ
Crop	Crop	DiatomSizeReduction	DSR
DistalPhalanxOutlineAge-Group	DPOAG	DistalPhalanxOutlineCor-rect	DPOC
DistalPhalanxTW	DPTW	DodgerLoopDay	DLD
DodgerLoopGame	DLG	DodgerLoopWeekend	DLW
ECG200	ECG200	ECG5000	ECG5000
ECGFiveDays	ECGFD	EOGHorizontalSignal	EOGHS
EOGVerticalSignal	EOGVS	Earthquakes	Earthquakes
ElectricDevices	ED	EthanolLevel	EL
FaceAll	FA	FaceFour	FF
FacesUCR	FUCR	FiftyWords	FW
Fish	Fish	FordA	FA
FordB	FB	FreezerRegularTrain	FRT
FreezerSmallTrain	FST	Fungi	Fungi
GestureMidAirD1	GMAD	GestureMidAirD2	GMAD
GestureMidAirD3	GMAD	GesturePebbleZ1	GPZ
GesturePebbleZ2	GPZ	GunPoint	GP
GunPointAgeSpan	GPAS	GunPointMaleVersusFe-male	GPMVF

Dataset Name	Short	Dataset Name	Short
GunPointOldVersusYoung	GPOVY	Ham	Ham
HandOutlines	HO	Haptics	Haptics
Herring	Herring	HouseTwenty	HT
InlineSkate	IS	InsectEPGRegularTrain	IEPGRT
InsectEPGSmallTrain	IEPGST	InsectWingbeatSound	IWS
ItalyPowerDemand	IPD	LargeKitchenAppliances	LKA
Lightning2	Lightning2	Lightning7	Lightning7
Mallat	Mallat	Meat	Meat
MedicalImages	MI	MelbournePedestrian	MP
MiddlePhalanxOutlin-eAgeGroup	MPOAG	MiddlePhalanxOut-lineCorrect	MPOC
MiddlePhalanxTW	MPTW	MixedShapesRegularTrain	MSRT
MixedShapesSmallTrain	MSST	MoteStrain	MS
NonInvasiveFetalECGThorax1	NIFECGT	NonInvasiveFetalECGThorax2	NIFECGT
OSULeaf	OSULeaf	OliveOil	OO
PLAID	PLAID	PhalangesOutlinesCorrect	POC
Phoneme	Phoneme	PickupGestureWiimoteZ	PGWZ
PigAirwayPressure	PAP	PigArtPressure	PAP
PigCVP	PCVP	Plane	Plane
PowerCons	PC	ProximalPhalanxOutlin-eAgeGroup	PPOAG
ProximalPhalanxOut-lineCorrect	PPOC	ProximalPhalanxTW	PPTW
RefrigerationDevices	RD	Rock	Rock
ScreenType	ST	SemgHandGenderCh2	SHGC
SemgHandMovementCh2	SHMC	SemgHandSubjectCh2	SHSC
ShakeGestureWiimoteZ	SGWZ	ShapeletSim	SS
ShapesAll	SA	SmallKitchenAppliances	SKA
SmoothSubspace	SS	SonyAIBORobotSurface1	SAIBORS
SonyAIBORobotSurface2	SAIBORS	StarLightCurves	SLC
Strawberry	Strawberry	SwedishLeaf	SL
Symbols	Symbols	SyntheticControl	SC
ToeSegmentation1	TS	ToeSegmentation2	TS
Trace	Trace	TwoLeadECG	TLECG
TwoPatterns	TP	UMD	UMD
UWaveGestureLibraryAll	UWGLA	UWaveGestureLibraryX	UWGLX
UWaveGestureLibraryY	UWGLY	UWaveGestureLibraryZ	UWGLZ
Wafer	Wafer	Wine	Wine

Dataset Name	Short	Dataset Name	Short
WordSynonyms	WS	Worms	Worms
WormsTwoClass	WTC	Yoga	Yoga

Table 25: Mapping between the original UCR dataset names and the shortened labels used throughout the thesis tables.

Dataset Name	ID	Train Size	Test Size	Time Series Length
ACSF1	1	100	100	1460
Adiac	2	390	391	176
Beef	8	30	30	470
CBF	11	30	900	128
CC	14	467	3840	166
CCECGT	15	40	1380	1639
CX	18	390	390	300
DSR	22	16	306	345
DPOC	24	600	276	80
ECG200	29	100	100	96
EL	36	504	500	1751
FRT	44	150	2850	301
FST	45	28	2850	301
Ham	56	109	105	431
Haptics	58	155	308	1092
HT	60	40	119	2000
IPD	65	67	1029	24
MSST	77	100	2425	1024
NIFECGT	79	1800	1965	750
SA	103	600	600	512
Strawberry	109	613	370	235
UWGLX	120	896	3582	315
Wafer	123	1000	6164	152

Table 26: UCR dataset information. The table shows the number of time series in the training and test sets, as well as the length of the time series for each dataset. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

Dataset	*sthread*	*par*	*gpu*
ACSF1	33.90	6.05	**1.00**
Adiac	5.42	0.89	**0.48**
AGWX	177.23	37.24	**3.23**
AGWY	173.30	37.35	**3.22**

Dataset	*sthread*	*par*	*gpu*
AGWZ	163.90	33.73	**3.22**
AH	0.79	0.13	**0.08**
BME	0.22	**0.04**	**0.04**
Beef	0.47	0.08	**0.06**
BF	0.36	0.07	**0.04**
BC	0.33	0.06	**0.04**
CBF	1.52	0.27	**0.10**
Car	2.04	0.37	**0.11**
Chinatown	0.02	**0.00**	0.01
CC	131.31	29.87	**4.64**
CCECGT	485.65	117.14	**6.79**
Coffee	0.08	**0.01**	0.02
Computers	106.37	21.23	**1.72**
CX	47.57	8.33	**1.04**
CY	46.36	9.87	**1.04**
CZ	46.10	10.80	**1.04**
Crop	959.46	224.62	**183.14**
DSR	0.58	**0.10**	0.24
DPOAG	0.79	**0.12**	0.23
DPOC	2.64	**0.39**	0.73
DPTW	0.78	**0.12**	0.23
DLD	1.48	0.26	**0.21**
DLG	0.66	0.13	**0.07**
DLW	0.68	0.13	**0.07**
ECG200	0.29	0.05	**0.04**
ECG5000	141.84	32.65	**25.40**
ECGFD	1.17	**0.20**	0.22
EOGHS	680.51	149.11	**10.05**
EOGVS	688.03	151.87	**10.00**
Earthquakes	39.84	9.21	**4.64**
ED	2282.74	551.85	**135.54**
EL	913.01	218.28	**79.16**
FA	58.35	**10.20**	11.14
FF	0.96	0.17	**0.11**
FUCR	25.05	5.56	**4.69**
FW	53.11	8.69	**1.21**
Fish	9.20	1.43	**0.40**

Dataset	*sthread*	*par*	*gpu*
FA	4077.15	972.35	**156.02**
FB	2506.91	600.43	**127.80**
FRT	91.46	21.28	**2.68**
FST	17.64	3.10	**0.55**
Fungi	0.43	0.09	**0.05**
GMAD	10.20	1.80	**0.26**
GMAD	9.69	1.92	**0.28**
GMAD	10.23	2.23	**0.28**
GPZ	15.65	2.85	**0.33**
GPZ	15.90	3.64	**0.33**
GP	0.37	**0.06**	**0.06**
GPAS	2.82	0.50	**0.16**
GPMVF	2.82	0.50	**0.15**
GPOVY	2.85	0.50	**0.16**
Ham	5.16	0.91	**0.18**
HO	4298.44	1026.24	**320.99**
Haptics	148.35	33.44	**2.90**
Herring	1.35	0.24	**0.11**
HT	65.12	11.60	**0.84**
IS	508.83	126.03	**8.76**
IEPGRT	17.90	3.95	**0.36**
IEPGST	5.14	1.08	**0.14**
IWS	101.89	17.07	**2.10**
IPD	0.15	**0.03**	0.07
LKA	253.64	58.64	**3.80**
Lightning2	4.95	0.87	**0.13**
Lightning7	1.72	0.30	**0.08**
Mallat	221.49	52.03	**6.65**
Meat	0.82	0.14	**0.09**
MI	10.23	1.78	**0.60**
MP	7.08	**1.27**	**1.27**
MPOAG	0.68	**0.11**	0.12
MPOC	1.99	0.31	**0.21**
MPTW	0.74	**0.11**	0.12
MSRT	3978.99	959.84	**178.83**
MSST	781.23	191.35	**97.64**
MS	0.63	0.13	**0.12**

Dataset	*sthread*	*par*	*gpu*
NIFECGT	3499.72	813.13	**396.61**
NIFECGT	3586.58	846.22	**308.21**
OSULeaf	29.07	6.56	**0.57**
OO	0.15	**0.02**	0.06
PLAID	1745.13	413.14	**95.21**
POC	18.58	3.78	**1.63**
Phoneme	1431.49	353.63	**163.58**
PGWZ	1.10	0.18	**0.16**
PAP	279.33	62.85	**3.83**
PAP	307.77	63.28	**3.78**
PCVP	288.73	70.66	**3.83**
Plane	0.71	0.12	**0.06**
PC	2.35	0.42	**0.11**
PPOAG	0.87	**0.13**	0.14
PPOC	1.77	0.29	**0.23**
PPTW	0.93	**0.13**	0.14
RD	245.55	59.26	**3.83**
Rock	24.60	5.57	**0.36**
ST	242.62	57.52	**3.82**
SHGC	1343.59	326.42	**81.91**
SHMC	1540.76	371.19	**64.17**
SHSC	1570.85	362.06	**66.83**
SGWZ	1.19	0.22	**0.18**
SS	3.09	0.62	**0.39**
SA	311.14	64.15	**3.92**
SKA	247.60	59.49	**3.82**
SS	0.02	**0.00**	0.02
SAIBORS	0.23	**0.04**	0.05
SAIBORS	0.42	0.08	**0.07**
Strawberry	19.27	2.94	**1.15**
SL	12.58	2.13	**0.62**
Symbols	10.56	2.04	**0.30**
SC	1.31	0.20	**0.11**
TS	2.36	0.45	**0.11**
TS	1.86	0.36	**0.07**
Trace	2.13	0.38	**0.12**
TLECG	0.39	**0.08**	**0.08**

Dataset	*sthread*	*par*	*gpu*
TP	243.64	45.53	**5.80**
UMD	0.34	0.06	**0.05**
UWGLA	9411.49	2195.47	**419.83**
UWGLX	1002.87	226.73	**52.80**
UWGLY	964.24	220.41	**54.80**
UWGLZ	986.12	222.02	**74.44**
Wafer	438.83	94.93	**29.13**
Wine	0.13	**0.02**	0.10
WS	41.06	7.48	**5.86**
Worms	37.11	7.03	**4.58**
WTC	37.39	6.85	**4.63**
Yoga	484.01	106.95	**70.54**
Mean	432.19	101.26	**26.33**
Avg. speedup (%)	–	520	2419

Table 27: DTW computation times (in seconds) of our implementation across all 127 UCR datasets, comparing single-threaded, parallelized, and GPU configurations. Values are means over 5 runs. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.

	ERP		DTW		ADTW	
	aeon	our	aeon	our	aeon	our
ACSF1	125.78	**40.14**	100.03	**33.90**	118.85	**23.42**
Adiac	26.83	**7.33**	21.78	**5.42**	25.74	**3.40**
AGWX	314.91	**215.19**	241.41	**177.23**	293.94	**48.00**
AGWY	295.92	**212.36**	241.56	**173.30**	294.39	**39.79**
AGWZ	303.07	**200.47**	239.51	**163.90**	287.02	**59.83**
AH	2.46	**1.01**	1.99	**0.79**	2.15	**0.50**
BME	0.42	**0.25**	0.34	**0.22**	0.40	**0.18**
Beef	1.14	**0.57**	0.92	**0.47**	1.10	**0.30**
BF	0.60	**0.43**	0.49	**0.36**	0.60	**0.36**
BC	0.60	**0.42**	0.53	**0.33**	0.63	**0.32**
CBF	2.51	**2.03**	2.03	**1.52**	2.38	**1.39**
Car	6.76	**3.08**	5.52	**2.04**	7.20	**1.35**
Chinatown	0.03	**0.02**	0.02	0.02	0.02	0.02
CC	281.49	**162.33**	234.59	**131.31**	276.22	**77.56**
CCECGT	859.73	**599.88**	710.22	**485.65**	848.59	**485.08**
Coffee	0.35	**0.08**	0.28	**0.08**	0.34	**0.03**
Computers	182.78	**127.90**	148.29	**106.37**	177.23	**104.41**

CX	79.68	**54.91**	63.22	**47.57**	75.26	**45.82**
CY	77.79	**55.51**	63.26	**46.36**	77.14	**45.63**
CZ	76.72	**59.01**	62.90	**46.10**	80.67	**46.02**
Crop	1511.59	**1082.79**	1179.73	**959.46**	1373.29	**325.23**
DSR	3.31	**0.77**	2.72	**0.58**	3.21	**0.34**
DPOAG	2.07	**0.87**	1.82	**0.79**	2.12	**0.51**
DPOC	6.64	**2.94**	5.32	**2.64**	6.20	**1.60**
DPTW	2.04	**0.93**	1.70	**0.78**	1.99	**0.48**
DLD	2.92	**1.22**	2.39	**1.48**	2.83	**1.54**
DLG	1.29	**0.53**	1.06	**0.66**	1.28	**0.69**
DLW	1.29	**0.55**	1.05	**0.68**	1.24	**0.69**
ECG200	0.52	**0.34**	0.42	**0.29**	0.49	**0.22**
ECG5000	249.04	**168.94**	206.65	**141.84**	238.19	**120.66**
ECGFD	2.26	**1.36**	1.64	**1.17**	1.94	**0.91**
EOGHS	1163.03	**767.08**	986.53	**680.51**	1126.52	**670.77**
EOGVS	1198.34	**759.84**	3843.53	**688.03**	1160.46	**715.30**
Earthquakes	66.80	**47.51**	54.08	**39.84**	64.58	**41.32**
ED	3660.05	**2776.98**	2971.16	**2282.74**	3466.19	**2350.64**
EL	4503.56	**663.45**	3714.13	**913.01**	4411.40	**322.79**
FA	94.30	**68.51**	73.66	**58.35**	87.64	**54.42**
FF	1.60	**1.05**	1.19	**0.96**	1.42	**0.82**
FUCR	39.73	**29.56**	31.74	**25.05**	38.68	**24.59**
FW	90.74	**60.60**	68.75	**53.11**	82.14	**48.60**
Fish	40.00	**13.29**	32.45	**9.20**	38.46	**5.60**
FA	6730.03	**4875.07**	5504.26	**4077.15**	6530.12	**4112.13**
FB	4208.86	**2955.52**	3453.49	**2506.91**	4000.69	**2557.78**
FRT	216.62	**106.51**	178.32	**91.46**	211.14	**67.85**
FST	42.00	**19.70**	33.55	**17.64**	40.50	**12.73**
Fungi	0.77	**0.51**	0.63	**0.43**	0.80	**0.45**
GMAD	19.70	**14.77**	16.67	**10.20**	19.10	**11.03**
GMAD	19.75	**10.99**	16.12	**9.69**	20.71	**10.02**
GMAD	19.35	**14.67**	15.84	**10.23**	19.64	**10.49**
GPZ	26.50	**18.12**	21.62	**15.65**	25.31	**16.31**
GPZ	26.70	**18.07**	21.99	**15.90**	26.10	**16.63**
GP	0.95	**0.48**	0.79	**0.37**	0.95	**0.29**
GPAS	5.34	**2.38**	4.35	**2.82**	5.14	**2.90**
GPMVF	5.42	**2.28**	4.39	**2.82**	5.23	**2.89**
GPOVY	5.51	**2.32**	4.64	**2.85**	5.45	**3.04**

Ham	12.07	**7.03**	9.81	**5.16**	11.56	**3.12**
HO	16637.68	**5639.31**	13942.64	**4298.44**	16116.38	**1767.73**
Haptics	325.19	**172.69**	271.11	**148.35**	311.06	**93.68**
Herring	6.13	**1.87**	4.96	**1.35**	5.88	**0.74**
HT	114.24	**71.29**	98.72	**65.12**	108.61	**70.92**
IS	1135.43	**642.08**	939.24	**508.83**	1113.99	**426.58**
IEPGRT	31.41	**16.30**	26.54	**17.90**	30.44	**10.50**
IEPGST	9.04	**4.43**	7.04	**5.14**	9.18	**2.81**
IWS	174.59	**122.65**	132.95	**101.89**	161.59	**96.16**
IPD	0.26	**0.18**	0.20	**0.15**	0.23	**0.13**
LKA	423.07	**300.48**	353.41	**253.64**	406.94	**256.17**
Lightning2	8.92	**5.85**	6.98	**4.95**	8.34	**4.32**
Lightning7	3.10	**2.10**	2.46	**1.72**	2.98	**1.56**
Mallat	786.96	**222.88**	636.63	**221.49**	758.05	**93.06**
Meat	4.29	**0.63**	3.55	**0.82**	4.06	**0.28**
MI	16.84	**11.76**	14.59	**10.23**	17.02	**8.82**
MP	11.95	**8.11**	9.23	**7.08**	10.42	**7.43**
MPOAG	2.31	**0.89**	1.86	**0.68**	2.18	**0.43**
MPOC	6.82	**2.28**	5.31	**1.99**	6.16	**1.22**
MPTW	2.54	**0.78**	2.04	**0.74**	2.37	**0.47**
MSRT	7362.95	**5047.94**	5986.67	**3978.99**	7144.96	**3737.63**
MSST	1476.08	**1030.31**	1175.93	**781.23**	1399.52	**745.55**
MS	1.07	**0.73**	0.86	**0.63**	0.96	**0.53**
NIFECGT	11421.30	**3729.05**	9459.00	**3499.72**	11062.56	**1614.11**
NIFECGT	11395.95	**3617.70**	9088.76	**3586.58**	11229.54	**1812.06**
OSULeaf	49.81	**37.85**	43.59	**29.07**	49.89	**30.70**
OO	1.66	**0.14**	1.36	**0.15**	1.61	**0.10**
PLAID	3056.40	**1921.65**	2570.20	**1745.13**	2995.24	**717.85**
POC	57.20	**21.64**	46.05	**18.58**	57.11	**11.43**
Phoneme	2431.42	**1750.13**	1975.04	**1431.49**	2388.77	**1501.38**
PGWZ	2.05	**1.16**	1.66	**1.10**	1.97	**0.43**
PAP	523.27	**324.00**	419.13	**279.33**	497.73	**279.10**
PAP	542.70	**130.36**	417.39	**307.77**	499.24	**300.35**
PCVP	510.86	**278.29**	425.21	**288.73**	503.00	**302.84**
Plane	1.37	**0.89**	1.07	**0.71**	1.25	**0.53**
PC	3.81	**2.59**	3.15	**2.35**	3.64	**2.26**
PPOAG	3.20	**0.92**	2.58	**0.87**	2.93	**0.48**
PPOC	6.62	**1.95**	5.49	**1.77**	6.22	**1.04**

PPTW	3.37	**0.95**	2.57	**0.93**	3.01	**0.51**
RD	410.76	**305.20**	337.18	**245.55**	415.53	**254.37**
Rock	49.92	**22.84**	41.02	**24.60**	47.51	**25.55**
ST	414.45	**298.19**	338.02	**242.62**	433.98	**236.92**
SHGC	2381.25	**1434.43**	1953.26	**1343.59**	2296.90	**1397.65**
SHMC	2759.95	**1587.26**	2256.98	**1540.76**	2737.60	**1662.34**
SHSC	2714.52	**1619.64**	2203.47	**1570.85**	2739.85	**1585.47**
SGWZ	2.03	**1.53**	1.84	**1.19**	2.00	**0.68**
SS	5.10	**3.63**	4.50	**3.09**	5.32	**3.48**
SA	574.28	**390.06**	435.69	**311.14**	531.51	**277.86**
SKA	413.55	**301.19**	359.39	**247.60**	399.70	**270.60**
SS	0.04	**0.03**	0.03	**0.02**	0.03	**0.01**
SAIBORS	0.38	**0.25**	0.31	**0.23**	0.32	**0.17**
SAIBORS	0.65	**0.49**	0.52	**0.42**	0.61	**0.39**
Strawberry	72.76	**19.76**	59.67	**19.27**	65.92	**8.21**
SL	28.18	**16.71**	23.09	**12.58**	27.02	**9.58**
Symbols	21.77	**13.58**	18.27	**10.56**	21.40	**9.18**
SC	2.07	**1.41**	1.65	**1.31**	1.92	**1.30**
TS	3.88	**3.12**	3.19	**2.36**	3.86	**2.37**
TS	3.10	**2.26**	2.54	**1.86**	3.01	**1.83**
Trace	4.29	**2.41**	3.50	**2.13**	4.21	**1.76**
TLECG	1.01	**0.51**	0.90	**0.39**	1.05	**0.34**
TP	394.64	**288.75**	299.72	**243.64**	356.05	**237.75**
UMD	0.66	**0.38**	0.54	**0.34**	0.64	**0.27**
UWGLA	16324.77	**11351.34**	13041.89	**9411.49**	15979.62	**9264.18**
UWGLX	1832.05	**1311.82**	1466.56	**1002.87**	1779.51	**942.48**
UWGLY	1765.39	**1267.35**	1540.11	**964.24**	1746.20	**917.81**
UWGLZ	1805.14	**1254.91**	1516.63	**986.12**	1742.72	**960.22**
Wafer	791.78	**497.43**	644.39	**438.83**	765.82	**393.79**
Wine	0.95	**0.11**	0.76	**0.13**	0.90	**0.07**
WS	69.50	**49.56**	57.05	**41.06**	67.70	**40.06**
Worms	63.42	**45.05**	51.91	**37.11**	63.94	**37.04**
WTC	63.71	**46.59**	52.13	**37.39**	61.54	**37.49**
Yoga	936.50	**622.63**	769.89	**484.01**	894.06	**441.50**
Mean	937.67	**504.93**	788.81	**432.19**	911.22	**357.66**
Avg. speedup (%)	210	–	192	–	331	–

Table 28: Running times (in seconds) of aeon and our implementation on ERP, DTW, and ADTW across all 127 UCR datasets, using a single thread. Values are means

over 5 runs. The full mapping between original UCR dataset names and shortened labels is reported in Table 25.