A Hierarchical Piecewise Approximation Framework for Pattern-Based Sequence Learning

Patrick Van der Spiegel[0000−0001−9783−3778] and Johan Loeckx[0000−0003−4208−1411]

Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene, Belgium

Abstract. Several existing anomaly detection (AD) algorithms that work on sequence data employ similar pipelines that convert subsequences into piecewise approximations before aggregating these into a model. However, choices in sequence segmentation, representation, distance calculation, and modelling approaches affect detection performance across different anomaly types. We propose a framework that formalises these pipeline components and their parameters, enabling systematic evaluation of pipeline configurations across specific sequence learning tasks. Through this formalisation, we introduce a novel hierarchical piecewise approximation model based on nested words. As far as we know, we are the first to use nested words in this context. Using the UCR *Time Series Anomaly Archive*, we demonstrate that specific framework configurations effectively identify subsequences corresponding to specific anomaly definitions. On this benchmark, the pipelines implemented using our framework achieve performance comparable to state-of-the-art methods, including deep learning approaches. We provide our framework as an open-source Python module to facilitate the exploration of pipeline configurations across various sequence learning tasks.

Keywords: pattern recognition · sequence learning · anomaly detection

1 Introduction

In various domains and applications, sequence data are gathered by observing processes and systems through both sensors and event logs. This collection of data can be used for pattern mining and sequence learning, for which we usually create models based on sets of legitimate sequences [30]. One important sequence learning task is anomaly detection (AD), where we detect and flag data points that deviate from expected normal behaviour described in legitimate sequences.

However, both the expected normal behaviour and the definition of an anomaly strongly depend on the processes or systems under observation, the nature of the data, the domain, and the specific application. As there is no universal definition of normal behaviour and anomalies, there is also no one-size-fits-all AD algorithm [25]. Algorithms that work under the "independent and identically distributed assumption" [21], for example, fail to detect anomalous patterns involving the order of data points.

There is a large variety of AD approaches for sequence data described in the literature [7]. Among these, some algorithms [29,32] use similar pipelines that transform subsequences into piecewise approximations. These approaches use these approximations to build a model of normal behaviour as a baseline to detect deviations in newly observed data, or they use properties of the models themselves to flag potential anomalies in previously collected data. However, there are numerous approaches to segment sequences, to represent or approximate subsequences, to calculate distances between subsequences or their representations, and to model these subsequences. Many design choices in these pipelines–including their segmenting, approximating, and aggregating strategies–implicitly influence which subsequences are considered anomalous.

As distance-based approaches to AD are competitive with the state-of-the-art [23], we aim to enable the exploration of pipeline configurations that capture different patterns of interest in input data. Therefore, we generalise these similar AD algorithms in a framework, with the goal of making pipeline design choices explicit. We aim to provide a structured way to explore the space of possible sequence learning pipelines that work with hierarchical representations of subsequences for all types of sequence data. By applying different framework configurations on the UCR *Time Series Anomaly Archive* [18], we also show that there exist unexplored component combinations that can detect patterns corresponding to different anomaly definitions.

We show that framework configurations based on the "hierarchical pattern matching" AD algorithm (HPM) [32] correctly identify anomalies in the UCR benchmark at a rate similar to that of the state-of-the-art in time series AD, including deep learning approaches. We do this by comparing our results to a recent benchmark paper for time series AD [5].

Contributions: We generalise a common pipeline used across different AD algorithms by formalising its components and parameters, and provide a framework for exploring configurations of this pipeline for different sequence learning tasks. In this work, we also introduce a novel hierarchical piecewise approximation model based on nested words which has, as far as we know, not yet been explored for AD or sequence learning. We apply our framework to the UCR dataset, comparing it against an existing state-of-the-art benchmark. Together with this paper, we release our framework as an open-source Python module[1], allowing other researchers to explore pipeline configurations across different sequence learning tasks.

2 Preliminaries

2.1 Sequences

Sequence (or series) data are ordered sets of elements that may be continuous or discrete. This overarching term includes sequences of real-valued variables and sequences of discrete symbols [17].

[1] `https://github.com/pvdsp/pbsf-lib`

Formally, a **sequence** $S = (s_1, s_2, \ldots, s_n)$ is an ordered set of $n \in \mathbb{N}$ sequential observations s_i. A **symbolic sequence** $S = (s_1, s_2, \ldots, s_n)$ is a sequence of discrete observations $s_i \in \Sigma^p$, where $p \in \mathbb{N}$ is the dimensionality of each observation and Σ is a finite set of discrete symbols. A **time series** $S = (s_{t_1}, s_{t_2}, \ldots, s_{t_n})$ is a sequence of real-valued observations $s_{t_i} \in \mathbb{R}^p$, where each data point s_{t_i} is observed at time point t_i with $t_1 < t_2 < \ldots < t_n$. In this paper, we primarily focus on univariate ($p = 1$) time series.

In many cases, we are not interested in properties of the full sequence but rather in local parts of the sequence. Given a sequence S of length n, a **contiguous subsequence** $S_{[i,j]} = (s_i, \ldots, s_j)$ of S, with $1 \leq i \leq j \leq n$, is a sequence consisting of $j - i + 1$ consecutive elements from positions i to j of S. *Figure 1* shows an example of a contiguous time series subsequence. For the rest of this paper, we use the term subsequences to refer to contiguous subsequences.

Being able to determine the similarity between sequences is crucial for tasks involving sequence data [6]. Given two sequences $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$, the **distance** between P and Q, written as $dist(P, Q)$, is a measure of how dissimilar these sequences are. We require distances to be non-negative ($dist(P, Q) \geq 0$) and symmetric ($dist(P, Q) = dist(Q, P)$). There are many possible distances that we can calculate between two sequences, each with its own specific properties [11]. The choice of an appropriate distance measure depends on the application and the nature of the data.

Finally, **motifs** refer to sets of frequently recurring subsequences with a low pairwise distance to each other, found within a single sequence or across a larger set of sequences [10]. Subsequences that are maximally different from other time series subsequences are called **discords** [20]. *Figure 1* shows a time series subsequence with its discord highlighted.

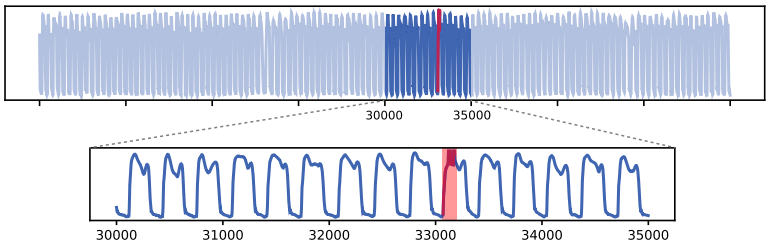

Fig. 1. The subsequence $S_{[30k,35k]}$ of the `gaitHunt1` time series from the UCR *Time Series Anomaly Archive*, representing the gait of someone walking on a force plate. The plot shows elapsed time since the start of the recording on the x-axis, and ground reaction force on the y-axis. The discord within this subsequence, corresponding to the ground-truth anomaly caused by a faulty sensor, is highlighted in red.

2.2 Sequence Learning

Sequence learning consists of sequence prediction, generation, and recognition [30]. For **sequence generation**, we generate sequence elements in their natural order. For **sequence prediction**, we predict future sequence elements based on one or more preceding elements of the sequence:

$$(s_i, s_{i+1}, \ldots, s_j) \longrightarrow (s_{j+1}, \ldots, s_{j+k}), \text{ where } 1 \leq i \leq j < \infty \text{ and } k \geq 1.$$

Sequence recognition consists of determining whether a sequence is legitimate, given a set of known legitimate sequences or knowledge of the underlying generative process:

$$(s_i, s_{i+1}, \ldots, s_j) \longrightarrow a \text{ with } a \in \{0, 1\}.$$

We can reframe various tasks on sequence data as sequence learning tasks. Our main focus in this paper is on the detection of novel subsequences and structural anomalies. For **novelty detection**, we flag subsequences that do not match known patterns within a given sequence:

$$S = (s_i, s_{i+1}, \ldots, s_j) \longrightarrow \{(k, l) \mid S_{[k,l]} \to 0\}$$

Similarly, **anomaly detection** for structural anomalies assigns scores to subsequences, where subsequences that deviate from expected patterns receive lower scores and are flagged as potentially anomalous:

$$S = (s_i, s_{i+1}, \ldots, s_j) \longrightarrow \{(S_{[k,l]}, a) \mid a \in [0, 1]\}$$

2.3 Sequence Approximation

Through sequence approximation, we create a sequence representation with a strongly reduced dimensionality compared to the original input sequence. These representations should efficiently approximate sequences by describing their essential features, thus preserving their characteristics [11].

A **representation** of a sequence S is a model $\bar{S}$ of reduced dimensionality $\bar{d}$ with $\bar{d} < n$ such that $\bar{S}$ approximates S. [14]. Methods used for this dimensionality reduction include Singular Value Decomposition (SVD), Discrete Fourier Transform (DFT) and its variants, and Discrete Wavelet Transform (DWT) [19].

Various sequence approximation techniques partition sequences into a number of non-overlapping subsequences (or *frames*) for approximation. Piecewise Linear Approximation (PLA) approximates each frame using a straight line [14]; Piecewise Aggregate Approximation (PAA) approximates frames using their mean [19]; and Symbolic Aggregate Approximation (SAX) maps the results of PAA to a discrete representation through equiprobable regions [22].

Uniform segmentation partitions the time series into equally-sized frames, whilst non-uniform segmentation picks breakpoints as to best fit the shape of the segment [6]. We show examples of piecewise approximation methods with uniform segmentation in *Figure 2*.

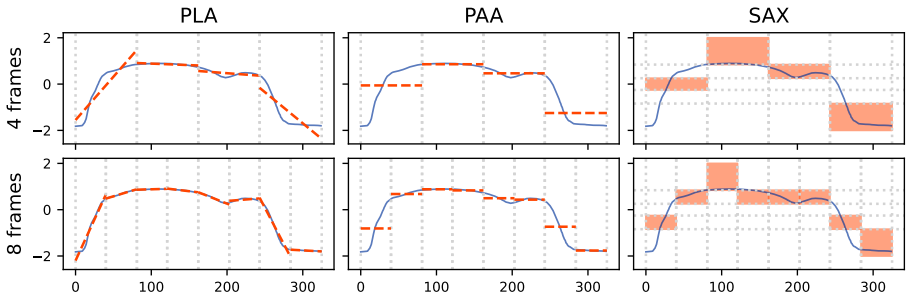

Fig. 2. Multiple representations of a subsequence from the `gaitHunt1` time series shown in *Figure 1* using PLA, PAA, and SAX. The subsequence is split into respectively 4 and 8 non-overlapping and equally-sized frames, and each frame is approximated through a straight line (PLA), its mean (PAA), or an equiprobable region (SAX). The higher the number of frames we use to approximate the sequence, the more detailed the representation of the sequence.

Multi-level or hierarchical representations of time series data also exists. The Hierarchical Piecewise Linear Approximation (HPLA) [6] technique approximates a time series subsequence by determining multiple sets of breakpoints to create variable-length frames, and approximates these frames using straight lines. HPLA aims to maximise the accuracy of the representation for a given number of frames.

Given two sequences P and Q, and their respective representations $\bar{P}$ and $\bar{Q}$, there exist distance measures that compare time series in the representation space, which we write as $dist(\bar{P}, \bar{Q})$. Preferably, these distance measures allow **lower bounding**, which guarantees that $dist(\bar{P}, \bar{Q}) \leq dist(P, Q)$ [11].

One such distance measure for PLA-based representations that is a lower bound of the Euclidean distance between two sequences is the PLA lower bound distance function $dist_{PLA}(\bar{P}, \bar{Q})$ [9]. If a_i and b_i respectively are the difference between slopes and the difference between intercepts of the i-th frame's approximation in $\bar{P}$ and $\bar{Q}$, and each frame has length l, then:

$$dist_{PLA}(\bar{P}, \bar{Q}) = \sqrt{\sum_{i=1}^{\bar{d}} \sum_{j=1}^{l} (a_i \cdot j + b_i)^2}$$

3 Related Work

Multiple frameworks and tools for sequence learning, pattern mining, and AD for the identification of structural anomalies in sequence data exist. In this section, we go over related frameworks and the pipelines they provide, the AD algorithms that we generalise, and possible sequence learning models.

Pattern Mining Frameworks One specific pattern mining and AD framework for sequence data discretises subsequences of time series data to build a transaction database, and provides state-of-the-art pattern mining algorithms to retrieve sequential patterns in that database for AD [16]. The same authors introduced a pattern-based AD method for mixed-type time series, which is the augmentation of time series data with related discrete event logs [15].

Discord-Based Anomaly Detection Time series discords [20], typically found through *matrix profiling* methods, are also used for time series AD. Recently, new parameter-free discord discovery algorithms named MERLIN and MERLIN++ were introduced [23], making repeated use of another discord discovery algorithm named DRAG to find discords. Other discord detection algorithms discretise the time series and hash these discretisations, where unique discretisations correspond to discords [23]. We categorise these as approximation-based methods.

Approximation-Based Methods Multiple AD algorithms make use of subsequence representations to detect anomalies. The HPM-based algorithm [32] discretises subsequences into hierarchical representations using repeated application of PLA and stores these discretisations in a tree structure. New data is compared against this tree, with dissimilar patterns flagged as anomalies.

Grammar-based compression time series AD [29] discretises subsequences using SAX to build context-free grammars via SEQUITUR [24]. Infrequently used grammar rules indicate potential anomalies. The authors of this AD method proposed a generic framework that incorporates their grammar-driven mining AD approach for both motif and discord discovery [28].

However, choices in subsequence approximation and comparison strategies influence anomaly detection results, which remains unexplored in these works.

Sequence Learning Models In terms of models for sequence data, finite automata and their variants are a natural data structure for modelling and recognising sets of related discrete sequences. Therefore, they can be used as a sequence learning model. For AD in the cybersecurity domain, for example, finite automata have been used to model normal system call behaviour of software programs, where unknown sequences of system calls are considered anomalous [27]. Other possible sequence learning models include hidden Markov models and recurrent neural networks [30].

Deep learning (DL) models have also been used to learn representations of time series [34], successfully modelling temporal dependencies in sequence data for AD. Interpretability into why something is marked as an anomaly is crucial in critical applications, however, and deep learning further complicates this process [34]. DL-based algorithms also require lots of critical parameters to be tuned which causes potential issues for practical use of these algorithms [23]. Additionally, AD in a production environment also require methods to be lightweight [32]. For these reasons, we focus on hierarchical piecewise aggregate approaches instead of DL for sequence learning.

4 Framework for Pattern-based Sequence Learning

We propose a sequence learning framework for pipelines that build models of sequences based on hierarchical piecewise approximations of their subsequences. This framework allows us to build various pipelines for different sequence learning tasks that require sensitivity or invariance to specific patterns. This framework consists of four main types of components:

1. **Segmenters** split input sequences into ordered sequences of overlapping or non-overlapping segments.
2. **Discretisers** take segments as input and return multiple approximations of these segments in various granularities, ranging from coarse- to fine-grained.
3. **Approximations** represent a segment at one specific level of granularity, and each type of approximation implements their own distance function.
4. **Models** are an aggregation of segment approximations, which we can use to investigate properties of the input data or to compare new data to this model.

Figure 3 shows how these types of components relate to each other. In the next subsections, we go over each of the components of our framework and discuss possible implementations with their specific parameters and strategies. Depending on the implementation and parameter choices of the components, the same high-level pipeline might learn different features of the input data. If these different pipelines are employed for AD, for example, they mark different subsequences as potentially anomalous.

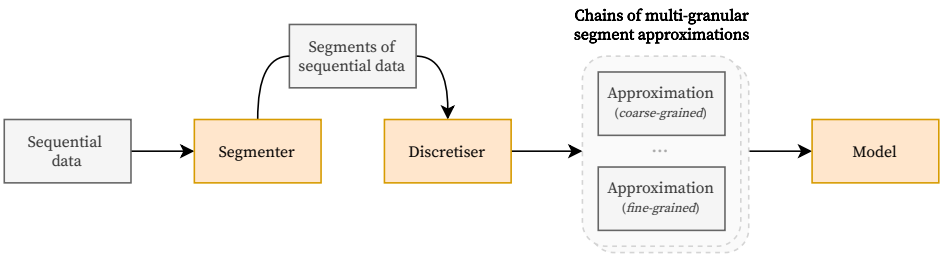

Fig. 3. Diagram that illustrates a pipeline using the proposed framework components. The pipeline partitions sequential input data into different segments. These segments are approximated using chains, which are sequences that contain multiple segment representations at different levels of detail or granularity. These multi-granular segment representations are combined into models that can be applied in various sequence learning tasks, including AD and sequence classification. The orange boxes depict the different framework components, and the gray boxes depict their input and output.

4.1 Segmenting

Segmenters split large input sequences S into ordered sequences of preprocessed segments $(S_1, S_2, \ldots, S_n)$. These segments could be equally-sized or variably-sized, and overlapping or non-overlapping. The preprocessing step for segments of time series might include z-score normalisation (which ensures invariance to amplitude shift and scale while preserving the shape of a segment [6]) or differentiation (which calculates the difference between successive data points, allowing us to focus on relative changes [5]).

One straightforward example of a segmenter implementation that returns equally-sized, overlapping segments is a **sliding window**: given a sequence S of length n, a sliding window of length k extracts all contiguous subsequences $S_{[i,i+k-1]}$ of length k, with $1 \leq i \leq n-k$. Optionally, the sliding window can also have a step size $j \geq 1$, such that it skips certain overlapping subsequences. *Algorithm 1* shows this approach to segmenting.

Algorithm 1 Sliding Window

Require: Sequence S of length n, window size k, step size j, preprocessing function
1: $position \leftarrow 1$
2: $segments \leftarrow ()$
3: **while** $position + k - 1 \leq n$ **do**
4: $\quad segment \leftarrow$ **preprocessing**$(S_{[position:position+k-1]})$
5: $\quad segments \leftarrow$ **append**$(segment)$
6: $\quad position \leftarrow position + j$
7: **end while**
8: **return** *segments*

One of the difficulties with sliding windows is deciding which window size k is appropriate for the input data and application. Strategies for deciding on an optimal window size based on properties of the input data can be employed. One approach involves analysing the autocorrelation of the input sequence to identify subsequence periodicities, since specific shifts corresponding with the length of a repeated pattern results in high autocorrelation values [13]. This shift size can be used as the size of a sliding window.

In this paper, we specifically look at segmenters that return equally-sized, overlapping, preprocessed segments. Other possible segmenting strategies could dynamically adapt their window size based on properties of the data [33]. An example of such segmenting strategy is the Stagger algorithm, which makes use of multiple expanding sliding windows for multiple periodicity mining [12].

The segmenting step could also transform snapshots of all sorts of sequential data types, including dynamic graphs [1], into real-valued or symbolic segments. This would essentially allow us to apply the framework on other sequential data types, as long as there is an appropriate translation technique from this data type to real-valued or symbolic segments.

4.2 Discretising

Discretisers transform segments into sequences of d_{max} piecewise approximations that increase in granularity, from coarse- to fine-grained. We call this sequence a **chain**. To obtain the l-th approximation in the chain (with $1 \leq l \leq d_{max}$), the segment is first split into a set of non-overlapping frames using breakpoints $f(l) = \{x_1, \ldots, x_k\}$. *Algorithm 2* shows how a segment S is represented using a chain of approximations by following a specific frame splitting, approximation, and depth strategy.

Frame Splitting For each approximation in the chain, we decide in how many non-overlapping frames we split a segment of length n. This number of frames is decided based on the approximation's position l in the chain. The higher l, the higher the number of frames $|f(l)| + 1$ used to approximate the segment, with $1 \leq |f(l)| + 1 \leq n$. The higher the number of frames, the more fine-grained we represent a single segment. For a high number of frames, noise might have an impact on the resulting approximation, however. Currently, we only consider equally-sized frames of length $\lfloor \frac{n}{|f(l)|+1} \rfloor$, but we could also extend this to allow strategies that result in variably-sized frames as employed by HPLA [6].

Approximation We represent segments through their frames using one of the approximation strategies discussed in *Subsection 2.3*. Possible representations include sequences of slopes and intercepts (PLA), means (PAA), or sequences of symbols (SAX), for example. Some sequence approximation strategies are more computationally expensive than others, however: using PLA to fit a line through each frame of a segment using the least squares method requires more work than PAA, which only calculates the mean of each frame. This representation accuracy and efficiency trade-off must be considered when designing a pipeline.

Depth We also have to decide on the number of approximations for a single segment, which we call the **maximum depth** d_{max} of the chain. For each of the levels l ranging from 1 to d_{max}, we approximate a single segment using $f(l)$ frames. Deeper chains require us to store more representations, and generally also require us to approximate more frames.

Algorithm 2 Discretisation Process

Require: Segment S of length n, frame splitting $f(l)$, depth d_{max}, approximation
1: $chain \leftarrow ()$
2: **for** $l = 1$ to d_{max} **do**
3: $\quad frames \leftarrow (S_1, \ldots, S_{|f(l)|+1})$
4: $\quad node \leftarrow$ **approximate**$(frame)$ **for** $frame$ **in** $frames$
5: $\quad chain \leftarrow$ **append**$(node)$
6: **end for**
7: **return** $chain$

4.3 Segment Approximations

Depending on which properties of a segment we are interested in, we store or calculate various properties of the discretisation for use in comparing representations. The method for comparing representations and determining when two representations are considered equivalent is defined by the equivalence strategy.

Equivalence Strategies To decide if two segments S_1 and S_2 are equivalent, we can compare their segment approximations $\bar{S}_1$ and $\bar{S}_2$ at a specific level of granularity l. If we approximate a segment in $|f(l)| + 1$ number of frames using PLA, for example, we can store two sequences of $|f(l)| + 1$ slopes and intercepts. By comparing the sequences of slopes and intercepts of these two representations, we decide if we regard $\bar{S}_1$ and $\bar{S}_2$, and thus S_1 and S_2, as equivalent. For PLA, one possible distance measure is the PLA lower bound distance function $dist_{PLA}$ [9] as discussed in *Subsection 2.3*.

No matter which distance function we use, however, exact equality between two representations is often not what we want. If we require that a certain distance $dist(\bar{S}_1, \bar{S}_2) = 0$ before we consider $\bar{S}_1$ and $\bar{S}_2$ as equivalent, small variations in segments could lead to two distinct representations and thus to an explosion in size of our aggregation model later on. Often, we want to determine when the overall shape of two segments is similar, disregarding small distortions. For this reason, we often make use of a distance threshold ϵ: two segments are therefore considered equivalent if $dist(\bar{S}_1, \bar{S}_2) \leq \epsilon$. The higher this ϵ, the lower the number of segment approximations we consider as distinct.

Although we currently focus on univariate sequences, our framework can be naturally extended to handle multivariate input by introducing component implementations that directly work with this type of data. Another option is to execute a pipeline for univariate sequences on each of the k rows of a multivariate sequence separately up until aggregation. To proceed with the rest of the pipeline, we interpret the k different segment approximations at a given level (where each level can take one of v different approximations), as a single approximation that can take one of v^k values. This approach might introduce computational issues, however, due to explosion in the number of possible segment approximations [26].

4.4 Aggregation Models

We aggregate chains into models based on the distance between corresponding segment approximations. There is a large variety of possible models in which we can store and combine chains, each with their own properties.

Sets The simplest model that can store approximations is a set. Either the approximation has been observed before, and is part of the set; or it has not yet been observed, and is not part of the set. We can store full chains in a set, or store each of the representations individually. The first approach causes

lots of duplicate representations to be stored into the model, while the latter disregards the hierarchical relationships between representations if $d_{max} > 1$. For this reason, we look into trees and graphs which use a similar principle to sets while explicitly modelling the hierarchical links between representations through their edges.

Trees and Graphs We can use trees and graphs $G = (V, E)$ to store chains of segment representations without duplicates. In these models, vertices V correspond to the representations, and the edges $E = (v_i, v_j)$ with $v_i, v_2 \in V$ correspond to the direct hierarchical relationship of representations as found in our set of chains. Chains for which their prefixes overlap share paths in these models, and leaves correspond to the finest-grained representation of a segment.

To check if a chain is part of a tree or graph, we simultaneously traverse the chain and the model. We start from the root node of the tree (or one of the vertices with in-degree zero that match the first representation in the chain), and compare each of the children of that node or vertex with the next representation in the chain. Depending on the equivalence strategy and the implementation of the model, we can traverse the model by greedily taking first matches, or find the children with the smallest distance.

One of the advantages of this approach is that we can abort the traversal at the first mismatch. However, the order in which we have observed the chains is disregarded using this approach. To solve this issue, we introduce a model based on nested words.

Nested Words Generally, when approximating consecutive overlapping segments, their chains first show differences at the finest level of granularity before a mismatch at higher levels of granularity occurs. To efficiently represent these transitions from one approximation to the other, we introduce a novel hierarchical piecewise approximation model based on nested words. Nested words allow us to model both the hierarchical and linear order in sequences of chains.

Nested words are a representation for data with both a linear ordering and hierarchical nested structure, which includes sequences of program execution and XML documents [3]. Formally, if $w = s_1 \dots s_\ell$ is a sequence of symbols from an alphabet Σ and $\rightsquigarrow$ is a matching relation, which is a subset of $\{-\infty, 0, 1, \dots, \ell\} \times \{0, 1, \dots, \ell, +\infty\}$, then $n = (w, \rightsquigarrow)$ is a nested word [2]. We write $i \rightsquigarrow j$, with call position i and return position j, if the nesting (i, j) is part of the matching relation. Positions that are not part of any matching are called internal positions. Every nested word maps to a unique tagged word, where $\langle\sigma$ and $\sigma\rangle$ respectively denote call and return positions labelled with $\sigma \in \Sigma$.

We represent a chain $(\bar{S_1}, \dots, \bar{S_{d-1}}, \bar{S_d})$ as the tagged word $\langle\bar{S_1}\langle \dots \langle\bar{S_{d-1}}\bar{S_d}$, where each $\langle\bar{S_i}$ denotes a pending call position. When a new discretisation chain $(\bar{S_1}, \dots, \bar{S_{d-1}}, \bar{S'}_d)$ is observed, where $\bar{S_d} \neq \bar{S'}_d$, we append the mismatch to the nested word as an internal position:

$$\langle\bar{S_1}\langle \dots \langle\bar{S_{d-1}}\bar{S_d}\bar{S'}_d$$

If the mismatch happens earlier in the chain however, with $(\bar{S_1}, \bar{S'_2}, \ldots, \bar{S'}_d)$ and $\bar{S_2} \neq \bar{S'_2}$ for example, we close all pending call positions after and including the mismatch, and append the rest of the new chain using call positions:

$$\langle \bar{S_1} \langle \bar{S_2} \langle \ldots \langle S^-_{d-1} \bar{S}_d S^-_{d-1} \rangle \ldots \rangle \bar{S_2} \rangle \langle \bar{S'_2} \langle \ldots$$

Given a certain context size, we can build sets of nested words that describe all observed approximation transitions within this context size. Using the set of nested words that represent observed sequences of chains, we can learn a NWA [4] or HAA [31] that recognises the regular nested language of observed chains.

5 Experimental Validation of Configurations

5.1 Framework configurations

When using specific component implementations and strategies, the framework described in *Section 4* corresponds to the approximation-based AD algorithms described in *Section 3*:

- The HPM AD algorithm [32] first uses a sliding window of a fixed size k to obtain segments. Next, HPM approximates these segments in 2^{l-1} frames, doubling the number of frames for each level l in the chain, using PLA as its approximation strategy. Based on the segment size k, each chain in HPM has a maximum depth of $d_{max} = \lfloor \log(k) \rfloor$. Segment approximations of HPM store the standard deviation of the full segment, together with the slope and intercept of each frame in the z-score normalised segment. Two representations are considered equivalent if their structural and prominence distance–two specific distance measures that compare slopes and intercepts, and the standard deviations respectively–are under specific thresholds. Chains are stored in a tree structure. Newly observed segments are discretised, and if their representation are not present in the tree, they are marked as anomalous.
- AD with grammar-based compression [29] uses a fixed-size sliding window to obtain segments. Segments are discretised to a fixed word size w with SAX as its approximation strategy. Next, the algorithm uses SEQUITUR [24], a bottom-up grammar induction approach, to build a context-free grammar based on the repetition of SAX words. Anomalies are identified using the *rule density curve*, which is used to identifying subsequences that correspond to unique or uncommon rules in the grammar.

Using our framework, we can explore new variants of the AD algorithms described above. We can decide on an appropriate segmenter window size k based on specific sequence properties or use differentiation as a preprocessing step in our segmenter. For HPM specifically, we can use PAA or SAX instead of PLA, use $dist_{PLA}$ instead of the structural and prominence distance, and use a nested word-based model instead of a pattern tree. In the next subsection, we apply HPM and an HPM-based variant on the UCR *Time Series Anomaly Archive*.

5.2 Benchmarking on the UCR Archive

The UCR *Time Series Anomaly Archive* is a dataset consisting of 250 univariate time series from various domains, where each sequence contains a single ground-truth anomaly of which we have the start and end point. These time series are diverse–differing in length and structure–and anomalies can be of any type, ranging from simple peaks and drops in value to complex anomalous patterns that are harder to detect [5].

We evaluate HPM-based configurations of our framework, shown in *Figure 4*, on the UCR dataset. For evaluation, UCR requires AD algorithms to return a single point in the testing subsequence of a time series as the suspected anomaly, after training on the anomaly-free training subsequence of that time series. If this suspected anomaly point falls within a certain range of the ground-truth anomaly region, that specific detection is considered successful. If we repeat this for all time series in the UCR dataset, we can take the ratio of successful detections over the total number of time series in the dataset as a single score for that algorithm.

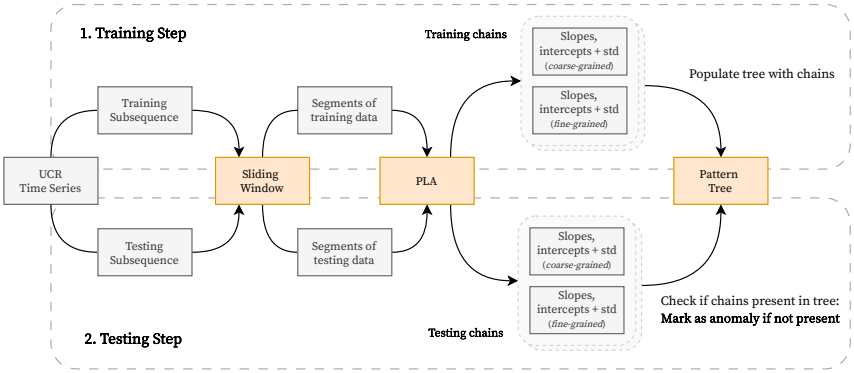

Fig. 4. Diagram that illustrates HPM as implemented using our framework. Each time series of the UCR dataset is split in training and testing at a predetermined point. First, the training subsequence is partitioned using a fixed-size sliding window, discretised using PLA, and aggregated into the pattern tree. Afterwards, the first two steps of the pipeline are repeated for the testing subsequence. Instead of aggregating the testing data into the tree, however, we only check if the testing chain is present in the pattern tree. If the discretisation of a test segment is not present in the pattern tree, the corresponding subsequence is marked as an anomaly.

We assign a score $a \in [0, 1]$ to each point of the testing subsequence based on the chains of all segments that include this point. This score corresponds to the mean of all relevant segment recognition scores. If all segments that include this point are present in the aggregation model, the point receives the maximum

score. If all segments are absent, the point receives the minimum score. The point with the lowest overall recognition score is our suspected anomaly.

If we apply the framework configuration that corresponds to the regular HPM algorithm (with a fixed window size of 200 and distance thresholds of 0.25) on the UCR benchmark, we see that HPM manages to correctly detect 100 of the 250 anomalies in the UCR dataset, corresponding to a score of 0.400. When comparing this result to other recent time series AD benchmarks [5], we see that this score outperforms all other algorithms except for *DeepSVDD* when no preprocessing is applied. However, when using differentiation as a preprocessing step, HPM achieves a score of 0.604, outperforming *DeepSVDD*'s best score of 0.520.

Using the HPM configuration with our nested words-based model slightly improves the initial 0.400 score to 0.416 with context size 2, and to 0.428 with context size 3. The pipelines with a nested words-based model correctly identify 13 additional time series anomalies that were initially missed with the original pattern tree, while missing 6 anomalies that were originally detected.

These preliminary results demonstrate that the detection of some anomalies benefit from contextual information about transitions between patterns, while the detection of other anomalies is hindered by this additional context. Examples of the former and the latter are respectively given in *Figure 5* and *6*. These figures show that choosing an appropriate configuration depends on the anomaly definition and nature of the data. There is no one-size-fits-all AD solution: by incorporating context into the model, we are more sensitive to one type of pattern over other types of patterns.

6 Discussion and Future Work

We introduced a framework for pattern-based sequence learning tasks by generalising a set of AD pipelines. Our key contribution is the formalisation of components and parameters which enables the exploration of pipeline variants. We also introduced a novel sequence learning model based on nested words.

Using the UCR *Time Series Anomaly Archive*, we demonstrated that pipeline configuration choices impact performance on sequence learning tasks, as different configurations are sensitive to different types of patterns. Initial results on HPM-based configurations for AD show competitive performance with state-of-the-art time series AD methods. However, our evaluation represents only a very limited exploration of possible framework configurations on one specific task and benchmark. We plan to perform a thorough evaluation in future work.

We also want to apply and evaluate our framework on other sequence learning tasks, including sequence classification, clustering, change point detection, and time series semantic segmentation [8]. We also plan to investigate the sensitivity and invariance of framework configurations on certain pattern types in sequences.

We release our framework as an open-source Python module, which allows researchers to explore the framework configurations that we have formalised, potentially discovering new effective combinations of components for specific sequence learning applications.

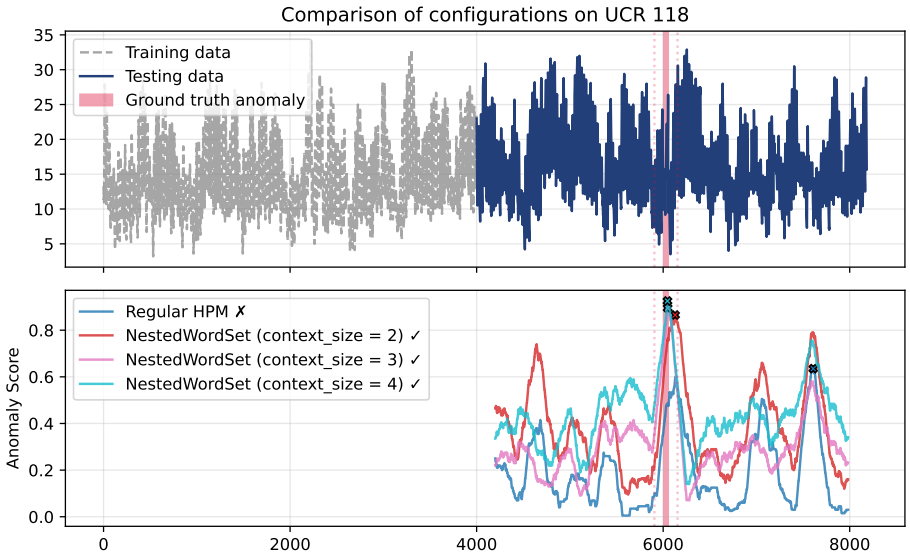

Fig. 5. Anomaly detection results on UCR 118. The top plot shows training and testing data with the ground-truth anomaly region. The bottom plot shows anomaly scores from different methods. Nested word configurations successfully detect the ground-truth anomaly, while standard HPM fails to identify it.

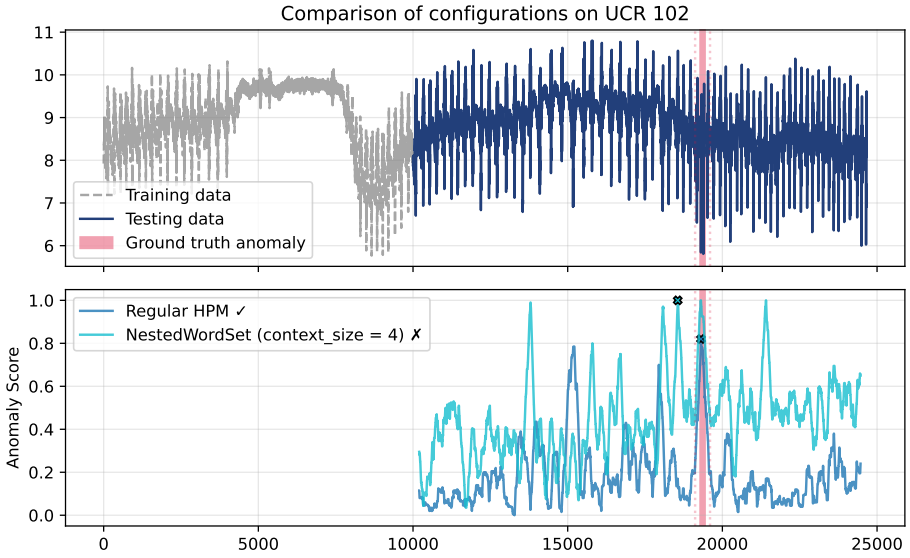

Fig. 6. Anomaly detection results on UCR 102. The top plot shows training and testing data with the ground-truth anomaly region. The bottom plot shows anomaly scores from different methods. Nested word configurations fail to identify the ground-truth anomaly, while standard HPM correctly identifies it.

Acknowledgements This research received funding from the Flanders Research Foundation through FWOAL1118 (MEMOBRAIN).

Disclosure of Interests The authors of this paper declare to have no conflict of interest.

Reproduction The code associated with the framework introduced in this paper can be found at `https://github.com/pvdsp/pbsf-lib`.

References

1. Akoglu, L., Tong, H., Koutra, D.: Graph based anomaly detection and description: A survey. Data Mining and Knowledge Discovery **29**(3), 626–688 (May 2015). https://doi.org/10.1007/s10618-014-0365-y
2. Alur, R.: Marrying words and trees. In: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 233–242. PODS '07, Association for Computing Machinery, New York, NY, USA (Jun 2007). https://doi.org/10.1145/1265530.1265564
3. Alur, R.: Nested Words and Visibly Pushdown Languages. https://www.cis.upenn.edu/~alur/nw.html (2016)
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM **56**(3), 1–43 (May 2009). https://doi.org/10.1145/1516512.1516518
5. Baldán, F.J., García-Gil, D.: Benchmarking Anomaly Detection Methods: Insights From the UCR Time Series Anomaly Archive. Expert Systems **42**(2), e13767 (2025). https://doi.org/10.1111/exsy.13767
6. Bettaiah, V.: The Hierarchical Piecewise Linear Approximation of Time Series Data. Thesis, The University of Alabama in Huntsville (2014)
7. Blázquez-García, A., Conde, A., Mori, U., Lozano, J.A.: A Review on Outlier/Anomaly Detection in Time Series Data. ACM Comput. Surv. **54**(3), 56:1–56:33 (Apr 2021). https://doi.org/10.1145/3444690
8. Carpentier, L., Feremans, L., Meert, W., Verbeke, M.: Pattern-based Time Series Semantic Segmentation with Gradual State Transitions. In: Proceedings of the 2024 SIAM International Conference on Data Mining (SDM), pp. 316–324. Proceedings, Society for Industrial and Applied Mathematics (Jan 2024). https://doi.org/10.1137/1.9781611978032.36
9. Chen, Q., Chen, L., Lian, X., Liu, Y., Xu Yu, J.: Indexable PLA for Efficient Similarity Search. In: VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria (2007)
10. Chiu, B., Keogh, E., Lonardi, S.: Probabilistic discovery of time series motifs. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 493–498. KDD '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/956750.956808, https://doi.org/10.1145/956750.956808
11. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: Experimental comparison of representations and distance measures. Proc. VLDB Endow. **1**(2), 1542–1552 (Aug 2008). https://doi.org/10.14778/1454159.1454226

12. Elfeky, M.G., Aref, W.G., Elmagarmid, A.K.: STAGGER: Periodicity Mining of Data Streams Using Expanding Sliding Windows. In: Sixth International Conference on Data Mining (ICDM'06). pp. 188–199 (Dec 2006). https://doi.org/10.1109/ICDM.2006.153
13. Ermshaus, A., Schäfer, P., Leser, U.: Window Size Selection in Unsupervised Time Series Analytics: A Review and Benchmark. In: Guyet, T., Ifrim, G., Malinowski, S., Bagnall, A., Shafer, P., Lemaire, V. (eds.) Advanced Analytics and Learning on Temporal Data. pp. 83–101. Springer International Publishing, Cham (2023). https://doi.org/10.1007/978-3-031-24378-3_6
14. Esling, P., Agon, C.: Time-series data mining. ACM Computing Surveys **45**(1), 1–34 (Nov 2012). https://doi.org/10.1145/2379776.2379788
15. Feremans, L., Vercruyssen, V., Cule, B., Meert, W., Goethals, B.: Pattern-Based Anomaly Detection in Mixed-Type Time Series. In: Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M., Robardet, C. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 240–256. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-46150-8_15
16. Feremans, L., Vercruyssen, V., Meert, W., Cule, B., Goethals, B.: A Framework for Pattern Mining and Anomaly Detection in Multi-dimensional Time Series and Event Logs. In: Ceci, M., Loglisci, C., Manco, G., Masciari, E., Ras, Z. (eds.) New Frontiers in Mining Complex Patterns. pp. 3–20. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-48861-1_1
17. Han, J., Kamber, M., Pei, J.: Data Mining Trends and Research Frontiers. In: Data Mining, pp. 585–631. Elsevier (2012). https://doi.org/10.1016/B978-0-12-381479-1.00013-7
18. Keogh, E., Roy, T.D., Naik, U., Agrawal, A.: Multi-dataset time series anomaly detection competition. https://compete.hexagon-ml.com/practice/competition/39/ (2021)
19. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. Knowledge and Information Systems **3**(3), 263–286 (Aug 2001). https://doi.org/10.1007/PL00011669
20. Keogh, E., Lin, J., Lee, S.H., Herle, H.V.: Finding the most unusual time series subsequence: Algorithms and applications. Knowledge and Information Systems **11**(1), 1–27 (Jan 2007). https://doi.org/10.1007/s10115-006-0034-6
21. Li, B.: Unsupervised Temporal Anomaly Detection. Ph.D. thesis, TU Dortmund (2024)
22. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: A novel symbolic representation of time series. Data Mining and Knowledge Discovery **15**(2), 107–144 (Oct 2007). https://doi.org/10.1007/s10618-007-0064-z
23. Nakamura, T., Mercer, R., Imamura, M., Keogh, E.: MERLIN++: Parameter-free discovery of time series anomalies. Data Mining and Knowledge Discovery **37**(2), 670–709 (Mar 2023). https://doi.org/10.1007/s10618-022-00876-7
24. Nevill-Manning, C.G., Witten, I.H.: Identifying Hierarchical Structure in Sequences: A linear-time algorithm. Journal of Artificial Intelligence Research **7**, 67–82 (Sep 1997). https://doi.org/10.1613/jair.374
25. Schmidl, S., Wenig, P., Papenbrock, T.: Anomaly detection in time series: A comprehensive evaluation. Proceedings of the VLDB Endowment **15**(9), 1779–1797 (May 2022). https://doi.org/10.14778/3538598.3538602
26. Sebastiani, P., Ramoni, M., Cohen, P.: Sequence Learning via Bayesian Clustering by Dynamics. In: Sun, R., Giles, C.L. (eds.) Sequence Learning: Paradigms, Algorithms, and Applications, pp. 11–34. Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44565-X_2

27. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001. pp. 144–155 (May 2001). https://doi.org/10.1109/SECPRI.2001.924295
28. Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A.P., Chen, C., Frankenstein, S., Lerner, M.: GrammarViz 2.0: A Tool for Grammar-Based Pattern Discovery in Time Series. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 468–472. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44845-8_37
29. Senin, P., Xing, W., Lin, J., Oates, T., Gandhi, S.: Time series anomaly discovery with grammar-based compression. In: 18th International Conference on Extending Database Technology (EDBT). Brussels, Belgium (Mar 2015)
30. Sun, R., Giles, C.L.: Sequence Learning: Paradigms, Algorithms, and Applications. Springer (Jun 2003)
31. Van der Spiegel, P., Loeckx, J., Arco, L.: Hierarchical-Alphabet Automata: Leveraging Hierarchy between Symbols in Sequences (Jun 2025). https://doi.org/10.2139/ssrn.5293094
32. Van Onsem, M., De Paepe, D., Vanden Hautte, S., Bonte, P., Ledoux, V., Lejon, A., Ongenae, F., Dreesen, D., Van Hoecke, S.: Hierarchical pattern matching for anomaly detection in time series. Computer Communications **193**, 75–81 (Sep 2022). https://doi.org/10.1016/j.comcom.2022.06.027
33. Verwiebe, J., Grulich, P.M., Traub, J., Markl, V.: Survey of window types for aggregation in stream processing systems. The VLDB Journal **32**(5), 985–1011 (Sep 2023). https://doi.org/10.1007/s00778-022-00778-6
34. Zamanzadeh Darban, Z., Webb, G.I., Pan, S., Aggarwal, C., Salehi, M.: Deep Learning for Time Series Anomaly Detection: A Survey. ACM Comput. Surv. **57**(1), 15:1–15:42 (Oct 2024). https://doi.org/10.1145/3691338