Data Mining and Knowledge Discovery (2026) 40:2
https://doi.org/10.1007/s10618-025-01170-y

Deep clustering for large-scale interpretable time series segmentation

Erick Draayer[1] · **Huiping Cao**[1] · **Qixu Gong**[1]

Received: 7 February 2024 / Accepted: 14 October 2025

Abstract

Time series segmentation (TSS) is often an unsupervised data mining task that partitions a given time series into homogeneous regions. Existing TSS algorithms either scale poorly or perform poorly on complex large-scale time series (TS) commonly observed in real-world applications. This paper introduces Deep Clustering for Time Series Segmentation (DC-TSS). DC-TSS is a domain-agnostic method that uses a three-phase neural-based model to segment a given time series. DC-TSS includes a carefully designed neural architecture and a newly designed data augmentation approach to efficiently learn TS representations, utilizes a neural-based clustering model to refine such representations, designs a novel efficient component to infer segments from clustered TS representation, and provides mechanisms to understand/interpret the segmentation results. We test DC-TSS on 27 multivariate time series datasets, which are much larger and more complex than others typically used in TSS studies. We also test DC-TSS on a more traditional repository of 98 simpler time series datasets. The experiments from both types of dataset provide an in-depth analysis of DC-TSS's performance and limitations. We compare five variations of our method against seven strong baselines. The results show that DC-TSS significantly outperforms other methods and scales well to larger and more complex datasets and shows some limitation on shorter simple datasets. DC-TSS addresses a growing need for unsupervised TSS algorithms designed to segment large-scale, complex datasets, which are becoming more common as evolving technology allows collecting and storing greater volumes of data.

Keywords Change point detection · Time series segmentation · Deep clustering

Extended author information available on the last page of the article

Published online: 08 November 2025

1 Introduction

This paper addresses the problem of time series segmentation (TSS). TSS may refer to several tasks, but in this work, it refers to partitioning a time series (TS) into similar data-generating regimes. This kind of segmentation is also known as "semantic segmentation" (Aminikhanghahi and Cook 2017). TSS is often an initial task to facilitate the discovery of underlying properties of a TS, especially in the absence of ground truth. TSS aims to detect and return a set of changepoints (CP) within a TS, referred to as a segmentation. A CP indicates a change in the data-generating state of the system. For example, a CP in a TS of brain wave activity may indicate a change in a person's emotional state. This problem is often synonymous with Change Point Detection (CPD) (Truong et al. 2020), and benchmark studies often test TSS and CPD algorithms together (Burg and Williams 2020). We focus on implementing and testing TSS in an offline setting.

It is common that transitioning short states exist between actual semantic segments. Examples of such short transition states include drinking from a water bottle, retying a shoelace, or taking short breaks. Such short states are considered higher levels of granularity closer to things like individual footsteps. We consider detecting short transition states like these to be outside the scope of semantic segmentation.

Many TSS/CPD algorithms exist. However, most are domain-specific and do not apply to different data domains. Current domain-agnostic state-of-the-art (SOTA) methods have not been shown to be suitable for handling complex large-scale TS datasets. For example, ClaSP (Ermshaus et al. 2023) was a recently proposed approach and performed well when tested on 98 different TS datasets. However, the TS datasets contain 40000 observations or less, and most of them have 3 or fewer CPs to detect. These types of datasets are a poor representation of a complex large-scale TS. In real-life applications, multiple sensors accumulate time series data over days, weeks, or months. These TS datasets are typically large-scale (much longer) and complex. This work focuses on **complex large-scale** time series characterized by long sequences (over 100000 observations), high dimensionality (several dozen to more than 100 variables), and numerous change points (CPs) (over a dozen). For example, the datasets analyzed here range from 170250 to 4949700 observations, span 14 to 148 variables, and contain 13 to 29 CPs, with the exception of one dataset. We found that competitive methods generally perform poorly or fail to scale on such complex large-scale time series.

This motivates us to propose more effective and efficient methods to segment complex large-scale time series recorded over long periods (leading to more CPs). The availability of algorithms that can handle large-scale, complex datasets is important because the collection and storage of data has become inexpensive. Hundreds of observations may be recorded from several sensors every second. Current TSS/CPD methods fail to be competitive at these larger scales or are domain-specific and can not be generally applied.

We introduce Deep Clustering for Time Series Segmentation (DC-TSS), a TSS method for segmenting complex large-scale TS datasets, building upon Xie et al. (2016); the differences between our approach and theirs are discussed in Sect. 3. We design the components of DC-TSS to maximize effectiveness and efficiency. First, it captures features at different abstraction levels (from initial mapping and refined clusters). The features with multiple abstraction levels can better differentiate the segments even though the segments are only created from the clusters. These features also allow our method to make no assumptions about the TS, including the number of CPs. Second, DC-TSS chooses less expensive operators or operators that are easily parallelized using GPU processing to improve computational efficiency. DC-TSS also provides a mechanism to help understand and interpret the results.

The **contributions of our research** are as follows:

- Through extensive experiments, we find that many SOTA methods demonstrated very poor performance when tested on complex large-scale TS datasets, highlighting a need for TSS research on these datasets.
- We introduce DC-TSS, a TSS method designed around complex large-scale TS datasets.
- We provide in-depth experiments to test our method against its variations and baselines including SOTA. Our analysis also provides an intuitive visualization of our method's performance and reveals interesting and interpretable underlying properties of the TS.
- We make all our code and datasets available online for other researchers and easy replication of our experiments (Draayer et al. 2025).

2 Background and related work

In this section, we review the necessary background, notation, definitions, and related work for time series segmentation (TSS).

2.1 Definitions and Statements

Definition 1 A *time series* $T = t_0^d, t_1^d, t_2^d, ..., t_{n-1}^d$ is a sequential order of observations of length *n* in equally spaced time intervals. Each observation, t_i^d contains *d* real values where *d* ($d \geq 1$) is the number of variables of the time series.

Definition 2 A *subsequence* $T_{j,L_s} = t_j^d, t_{j+1}^d, t_{j+2}^d, ..., t_{j+L_s-1}^d$ of T is a subset of sequential order observations from *T* with length L_s and $j + L_s \leq n$.

Definition 3 A *regime* is a discrete data generating state within *T* that corresponds to a specific physical or logical process. For example, a person walking might describe a specific regime within a TS of a person exercising.

Definition 4 A *changepoint* is an observation of *T* that separates two regimes.

Definition 5 A *segmentation* of T is a division of T into $m + 1$ regimes where the regimes' observations are described by the set of changepoints $t_{i_1}, ..., t_{i_m}$ $(1 < i_1 < .... < i_m < n - 1)$ found by the algorithm.

The goal of TSS is to find a segmentation of *T*, where *T* contains one or more regimes. A segmentation is considered correct when the proposed segments found by the TSS algorithm align with the true underlying regimes of *T*. Alternatively, the correct solution can be seen as finding the correct set of CPs of *T*.

2.2 Related work

Domain-agnostic TSS research is surprisingly sparse. Most recent TSS research focuses on domain-specific implementations. For example, the Gaussian process hidden semi-Markov model (HVGH) (Nagano et al. 2019) is designed around the TSS of human motion datasets. They rely on trajectories in a 3D space as a feature to find the best segmentation. Truong et al. (2020) discuss many domain-agnostic methods and create a typology based on their cost function, search method, and constraint. They recognize three main classes of domain-specific methods: (1) Graph-based, (2) Likelihood-based, and (3) Kernel-based. The graph-based methods infer a graph from the observations of the TS. The graph is split into sub-graphs based on a bespoke graph statistic (Chen and Chu 2023). Likelihood-based methods detect changepoints by subsequencing a given TS, estimating probability distributions from subsequences, and calculating differences between consecutive distributions. Kernel-based methods subsequence a given TS but use a kernel-based similarity metric to determine changepoints.

Hidden Markov Models (HMMs) have also been used for TSS. The construction of a HMM requires domain-specific knowledge. HMM architecture can be difficult to establish and is usually tailored to a specific domain or dataset. For example, Khalifa et al. (2021) review HMMs for event detection in biomedical signals. They discuss how biomedical signals usually have rhythms that can be identified and used to model the dynamic interactions between the HMM states.

The most recent domain-agnostic TSS research includes ClaSP (Ermshaus et al. 2023) and FLOSS (Gharghabi et al. 2019). ClaSP uses a self-supervised learning approach to detect CPs in an iterative manner by hierarchically splitting a TS into two segments and training a binary classifier to classify subsequences as left or right of the split. Split points with an accuracy above some threshold are detected as CPs. However, ClaSP was mainly tested on smaller datasets with 40000 observations and 6 or less CPs. ClaSP has also recently been shown to be competitive on more complex and MTS datasets (Ermshaus et al. 2023).

FLOSS uses the matrix profile, a data structure that stores information of the nearest neighbor of any given subsequence, to count the number of arcs that cross over an observation. Observations with few arc crosses are detected as CPs. Our experiments show that FLOSS performs poorly when there are repeating behaviors in the TS. ClaSP and FLOSS have been shown to outperform other recently proposed domain-agnostic TSS methods such as Autoplait (Matsubara et al. 2014) and Hierarchical Optimal Gaps (HOG) (Zhao and Itti 2016) by a significant margin over a wide range of TS datasets (Ermshaus et al. 2023; Gharghabi et al. 2017).

Time2State (Wang et al. 2023) is a recently proposed TS state discovery algorithm. Although Time2State is not directly designed for TSS, we can easily infer changes within data-generating states from its output. The design of Time2State also allows it to efficiently handle large-scale MTS datasets, which are the main focus of this research. Therefore, we consider Time2State to be a possible competitor.

Besides these methods, recent surveys (Burg and Williams 2020) compared several TSS/CPD algorithms and identified BOCPD (Adams and MacKay 2007) and PELT (Killick et al. 2012) as strong competing methods despite their age. BOCPD finds changepoints based on the probability of an observation belonging to distributions estimated from previous observations. PELT detects changepoints by minimizing the constrained sum of approximation errors.

Like our proposed method, previous works also explore autoencoders (AEs) for TSS purposes. De Ryck et al. (2021) proposes TIRE, an AE based TSS method to find features from both the time and frequency domains of TS. Lee et al. (2018) used an AE to automatically featurize subsequences of a TS but determined CPs based on distances between consecutive subsequences. In contrast, our method determines CPs through cluster analysis. The details of their architecture are unclear, and their code is not available, but we can still use their proposed algorithm for measuring the distance between embedded subsequences. Strommen et al. (2023) uses an AE and the matrix profile to find changepoints. We omitted their method from our study because their research focuses on real-time CP detection in the physiological domain, they did not publish their code, and many details of their AE architecture are not given.

3 DC-TSS

This section presents our Deep Clustering for Time Series Segmentation (DC-TSS) method. DC-TSS takes a given time series T with d variables and returns a set of changepoints signifying the start of new segments.

We design our method around monitoring scenarios that produce large-scale multivariate time series (MTS) with several CPs to detect between various data-generating states (e.g., a person's exercise, a song, or a mental state). Our method focuses on large-scale complex MTS because these datasets are becoming more common as technology allows for the inexpensive collection and storage of larger volumes of data. Figure 1 shows a flowchart of our method for reference.

DC-TSS works in three phases. The first phase learns to map the input to a lower dimensional latent space with a deep autoencoder (AE). The second phase optimizes the initial mapping and finds clusters by calculating auxiliary and target distributions and minimizing their distance using a Kullback–Leibler (KL) divergence loss function. The third phase analyzes the clusters to infer the changepoints of the TS. The first two phases are based on the deep clustering work of Xie et al. (2016), but with significant differences to make the model more scalable and applicable to complex TS

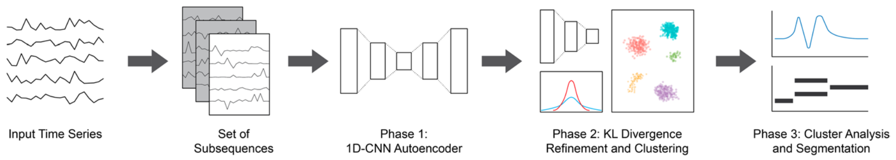

Input Time Series
Set of Subsequences
Phase 1: 1D-CNN Autoencoder
Phase 2: KL Divergence Refinement and Clustering
Phase 3: Cluster Analysis and Segmentation

Fig. 1 Flowchart of Deep Clustering for Time Series Segmentation

data. The method presented by Xie et al. is not a TSS method. Their research focuses on image clustering and relies on a completely different neural network architecture, making their method incompatible with the specific requirements of MTS segmentation. Although their clustering technique is adopted by DC-TSS, particularly in our second stage, their method cannot be applied directly to our task.

We highlight the key differences between DC-TSS and the method proposed in Xie et al. (2016) as follows. (i) The third phase of generating change points from clusters is completely new and is not based on any prior research. This phase recognizes the clusters found using technique in Xie et al. (2016) as tokens and interprets them in a novel way to perform TSS (Sect. 3.3). This phase includes multiple components and linear algorithms to derive change points. (ii) We design a different architecture to learn the representations of MTS. While our architectural design may seem simple at first glance, it is non-trivial due to the wide range of alternative architectures available. As shown in Sect. 4.5.1, our design outperforms other DC-TSS variants that adopt different architectures. (iii) We design a data augmentation strategy in the first phase to augment subsequences that correspond to sparse generation states, which helps improve the accuracy of detecting such states. (iv) We adopt learning-efficient design, guided by deliberate technical choices. In particular, our Phase 1 architecture uses 1-dimensional convolutional layers instead of the original 2D convolutions. This not only enables effective extraction of hidden features but also facilitates GPU acceleration. The third phase uses linear functions. Such design allows for fast and efficient processing of large-scale time series datasets, making our approach suitable for processing long TS with many variables and change points. (v) We utilize the deep clusters to gain insight into and interpret the properties of each segment (Sect. 4.7).

3.1 First phase - mapping initialization

The goal of the first phase is to initialize a non-linear mapping of the input to a lower dimensional latent space $f_\theta : X \rightarrow Y$ where X and Y are the input space and latent space respectively and f_θ is the non-linear mapping function with learnable parameters θ. To create this initial mapping, we use a 1-dimensional convolutional autoencoder. Autoencoders are an effective unsupervised method for automatically learning features to produce well-separated representations in lower dimensional latent spaces (Le 2013; Vincent et al. 2010). Thus, autoencoders are an obvious choice for creating an initial non-linear mapping of our input into a lower dimensional latent space. We explain the rationale of using a 1-dimensional convolutional encode shortly.

In order to map a given TS with d variables, we first divide the TS into smaller overlapping subsequences. The first phase introduces our first two hyperparameters: L_s and *overlap*, which refer to the length of the subsequences and the percentage they overlap, respectively. This process gives us a set of size $d \times L_s$ subsequences. This setting can easily accommodate the case where subsequences do not overlap ($overlap = 0$).

We design a data augmentation technique to increase the training set size in the first phase. Utilizing the data augmentation technique is particularly helpful to detect sparse states. For each subsequence, we shift the subsequence by $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ of its length (L_s). We roll elements over to the beginning of the subsequence. For example, given a subsequence $T_{1,200}$=$t_1, t_2, \cdots, t_{200}$, we create the subsequences $T^1_{1,200}$=$t_{151}, t_{152}, \cdots, t_{200}, t_1, t_2, \cdots, t_{150}$ (shifting $\frac{1}{4}$ of the length), $T^2_{1,200}$=$t_{101}, t_{102}, \cdots, t_{200}, t_1, t_2, \cdots, t_{100}$ (shifting $\frac{1}{2}$ of the length), and $T^3_{1,200}$ =$t_{51}, t_{52}, \cdots, t_{200}, t_1, t_2, \cdots, t_{50}$ (shifting $\frac{3}{4}$ of the length) by shifting and rolling over the values.

Our data augmentation technique quadruples the samples our autoencoder can train from. This technique helps ensure that sparse segments can still be well represented in the training data. For example, we may only have 10 min of a person vacuum cleaning within a 10-hour recording. The data augmentation technique of rolling each of the original subsequences associated with vacuum cleaning helps ensure this relatively short but important segment is not lost among more frequent segments. Our data augmentation approach may seem to degrade the TS because it distorts the temporal ordering. However, this is not the case. More explanations about this are provided after introducing the architecture that learns subsequence representations.

We view each subsequence as an image and, therefore, use a convolutional neural network autoencoder (Masci et al. 2011) (CAE) to capture their hidden features. Since the subsequences still pertain to TS data, we only apply the kernel to the temporal axis and thus use 1-dimensional convolutions in our encoder. The encoder creates a bottleneck latent space to capture features of each subsequence. We use 1-dimensional convolutional transpose layers to decode from the latent features. We apply leaky rectified linear functions (ReLU) (Xu et al. 2015) after each convolutional and convolutional transpose layer. This activation function is well suited for handling noise, a typical property of real-world TS data. The input channels of our convolutional layers in the encoder and decoder are kept constant and set to the number of variables of the TS, *d*. We use an ADAM optimizer (Kingma and Ba 2014) to train the AE by minimizing the least squared error: $||x - y||_2^2$ where x is the original subsequence and y is its reconstruction. Initial ad-hoc experimentation of the CAE architecture showed that keeping the number of input channels constant between convolutional layers and making the latent space 1% the size of the input space yielded the best segmentations.

Figure 2 shows a typical example of our autoencoder architecture. In this example, subsequences have 4 variables, each of which has a length of 160 observations. DC-TSS uses four 1-dimensional convolutional and four transposed convolutional layers. Each convolutional layer has a leaky ReLU activation function after it. We set the number of channels to be the same as the number of variables (i.e., 4) throughout the architecture. DC-TSS shrinks the original input to a 1×4 latent space (the smallest possible). DC-TSS reduces the space by about a third between convolutions according to the specified kernel size and stride length. We found best results with kernel lengths approximately 10% of the preceding space's length and a stride of ≤ 3.

Using only convolutional layers in our AE architecture allows us to take full advantage of graphics processing unit architecture to accelerate computation, making our method scalable to large datasets. Training time and memory usage are much smaller than more traditional fully connected (FC) AEs, which we test in Sect. 4.5.

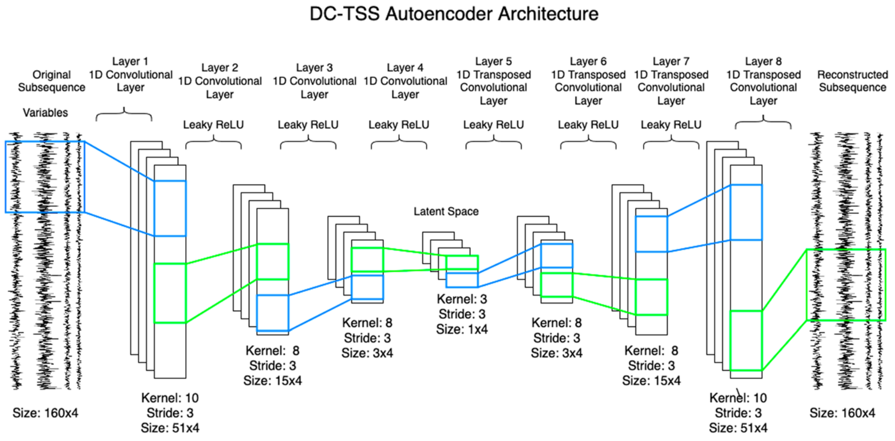

DC-TSS Autoencoder Architecture
Original Subsequence
Variables
Layer 1 1D Convolutional Layer
Layer 2 1D Convolutional Layer
Layer 3 1D Convolutional Layer
Layer 4 1D Convolutional Layer
Layer 5 1D Transposed Convolutional Layer
Layer 6 1D Transposed Convolutional Layer
Layer 7 1D Transposed Convolutional Layer
Layer 8 1D Transposed Convolutional Layer
Reconstructed Subsequence
Leaky ReLU
Latent Space
Size: 160x4
Kernel: 10 Stride: 3 Size: 51x4
Kernel: 8 Stride: 3 Size: 15x4
Kernel: 8 Stride: 3 Size: 3x4
Kernel: 3 Stride: 3 Size: 1x4
Kernel: 8 Stride: 3 Size: 3x4
Kernel: 8 Stride: 3 Size: 15x4
Kernel: 10 Stride: 3 Size: 51x4
Size: 160x4

Fig. 2 Architecture of DC-TSS autoencoder

After $n_{epochs1}$ number of epochs, the AE training is complete. We then discard the decoder, leaving only the encoder. The encoder is our initial non-linear mapping function $f_\theta : X \rightarrow Y$ that embeds each subsequence into some latent space. Each embedded subsequence, $y_i \in Y$, can also be seen as a set of features extracted from their corresponding subsequence $x_i \in X$. The second phase aims to refine this latent space and cluster the embedded subsequences.

Using the CAE architecture, the data augmentation strategy works effectively because of two reasons. The first reason relates to the underlying mechanism of the autoencoder. Rather than modeling *global* sequence dynamics or transitions between subsequences, the autoencoder focuses on reconstructing *individual* subsequences, aiming to capture meaningful *local* intra-subsequence structure. For example, given a subsequence of length 200 (e.g., $T_{1,200}$=$t_1, t_2, \cdots, t_{200}$) and the convolutional kernel size be 40.[1] Using such kernels, the hidden units of the first hidden layer aggregate subsequences of the kernel size. Consider the subsequence $T_{1,200}$, after a circular shift, we obtain a new subsequence $T^1_{1,200}$=$t_{151}, t_{152}, \cdots, t_{200}, t_1, t_2, \cdots, t_{150}$ (shifting $\frac{1}{4}$ of the length). The first layer aggregates length-40 subsequences in T_{1200} and T^1_{1200}. To simplify illustration, suppose that the convolutional kernels operate on non-overlapping subsequences, the kernels produce 5 hidden units in the first hidden layer from each input sequence since 50=$\frac{200}{40}$. Among the 5 hidden units derived from T^1_{1200}, only one is computed from a distorted subsequence, specifically, the wraparound segment $t_{191}, t_{192}, \cdots, t_{200}, t_1, t_2, \cdots, t_{30}$, which spans the boundary due to the shift. The remaining 4 hidden units learned from T^1_{1200} correspond to the same local patterns as those in the original subsequence T_{1200}. Thus, the majority of hidden units learned in the first layer still preserves the temporal order. In deeper layers, although the hidden unit derived from the corrupted subsequence may propagate through the network, its overall impact remains limited, as most hidden units are still

[1] This example does not use the kernel size specified in Fig. 2 to purposely create one problematic wraparound segment.

learned from valid local short subsequences. The second reason is related to the property of the data. Many time series domains exhibit cyclic or quasi-cyclic behaviors (such as repetitive physical motions, environmental cycles, or periodic mechanical patterns), where the local shape matters more than the absolute temporal phase within the subsequence window. In these cases, cyclic shifts preserve the essential shape and frequency content, which are exactly what an autoencoder is meant to capture.

3.2 Second phase: clustering

The second phase improves the initial mapping of the first phase encoder, f_θ, and assigns clusters to the embedded subsequences. The overall intuition of phase II is to refine the encoder of phase I by learning from a specific subset of subsequences to fine-tune the encoder. These subsequences a) do not belong to a large cluster, and b) only belong to a single cluster with c) a high probability. Learning from these subsequences creates a latent space of well-separated and smaller clusters. The learning from these subsequences also improves the encoding of uncertain subsequences (ones with low probability and bordering between two or more clusters) by encouraging a latent space with small and well-separated clusters. Splitting bigger clusters into smaller ones is not an issue for our TSS purposes because we rely on analyzing several sequential cluster assignments simultaneously for CP detection. We describe this in detail in Sect. 3.3.

This phase continues to refine the same set of parameters, θ, from the encoder of the first phase but with a different objective function by following the major steps in Xie et al. (2016). For completeness and self-containment, we briefly explain the calculations below.

The Kullback–Leibler (KL) divergence loss was used (Eq. 1) as the objective function (Xie et al. 2016). Minimizing the Kullback–Leibler (KL) divergence loss function depends on minimizing the differences between two probability distributions which we define as *P* and *Q*. *P* and *Q* are the target and auxiliary distributions, respectively. The terms p_{ij} and q_{ij} are the probabilities of the *i*-subsequence belonging to the *j*-th distribution of *P* and *Q* probability distributions, respectively.

$$Loss = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{1}$$

We define the probabilities of the auxiliary distribution, *Q*, based on the Student's t-distribution kernel. Eq. (2) defines auxiliary distribution probability calculation, q_{ij}, following (Xie et al. 2016). The probability q_{ij} requires the embedded *i*-th subsequence (y_i) and the *j*-th distribution centroid, μ_j. The encoder mapping of the *i*-th subsequence into the latent space gives y_i. We initialize μ_j using k-means clustering (Lloyd 1982) for the first epoch. The clustering k-means method introduces the third hyperparameter, *k*, which defines the maximum number of clusters we attempt to fit the embedded subsequences into. We use k-means for only the first iteration. Afterward, we determine clusters based on the highest probability calculation of q_{ij} for the *i*-th subsequence. We then calculate the average of the clusters to determine μ_j for all ensuing epochs. Since we rely on q_{ij} to determine our cluster assignments, the number of clusters we find may be less than k. The parameter α represents the statistical degrees of freedom, the maximum number of independent values. Previous research shows the value of this parameter to be insignificant (Van Der Maaten 2009), and therefore, we set $\alpha = 1$ for all experiments.

$$q_{ij} = \frac{(1 + ||y_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + ||y_i - \mu_{j'}||^2/\alpha)^{-\frac{\alpha+1}{2}}}, \text{ where } j' \neq j \tag{2}$$

Eq. (3) (also used in Xie et al. 2016) defines the probability, p_{ij}, of the target distribution *P*. The term $f_j = \sum_i q_{ij}$ is the frequency of the *j*-th auxiliary cluster. The learning process (minimizing loss of Eq. 1), targets a smaller loss. The target distribution emphasizes learning from subsequences with (a) a high probability (b) belonging to a single cluster and (c) belonging to a smaller cluster. Each quality leads to a smaller loss (explained below). The target distribution accomplishes this by first squaring the auxiliary probability term, q_{ij}. If q_{ij} is a high probability, p_{ij} will be as well, resulting in a lower loss in Eq. (1) than when q_{ij} is a low probability. The denominator of Eq. (1) is the sum of the probabilities from every probability distribution except the *j*-th distribution. If the *i*-th subsequence has a high probability with one or fewer distributions (cluster), this will cause p_{ij} to be bigger and thus result in a smaller loss in Eq. (1). Lastly, the numerator and denominator of Eq. (3) are normalized by dividing them by f_j. This division helps to discourage large groups from forming. f_j scales with the *j*-th cluster size and will cause p_{ij} to shrink as the *j* cluster grows, resulting in a greater loss calculation in Eq. (1). We stop training after $n_{epochs2}$ epochs or until the assignment of clusters between consecutive epochs remains stable. We use Stochastic Gradient Descent (SGD) for optimization.

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}}, \text{ where } j' \neq j \tag{3}$$

3.3 Phase 3: Cluster analysis for time series segmentation

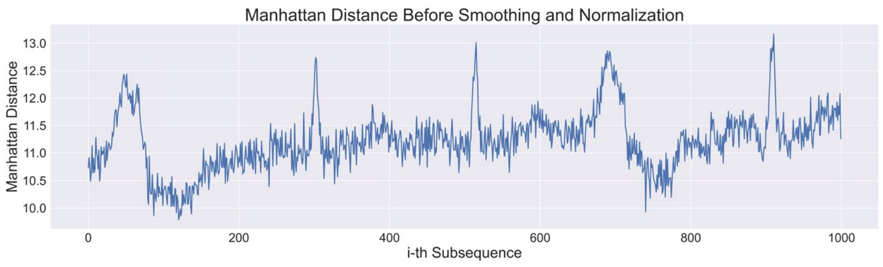
The third phase analyzes the clustered subsequences to determine CPs in the original TS. The clustered subsequences' temporal order can be considered a tokenized version of our original TS. A naive approach might be to determine segmentation based on a change in cluster label over time. For example, if subsequences x_0 to x_{50} are labeled "cluster 2" and x_{51} to x_{100} are labeled "cluster 5" then we might establish a CP between x_{50} and x_{51}. However, this would only work if each regime was associated with a single cluster. Our experiments found that regimes may be associated with several clusters. Subsequences may also rapidly change cluster assignments but still form recognizable and distinct regimes.

We design a strategy to analyze the clustered subsequences. This strategy uses a sliding window (Truong et al. 2020) to measure the similarity between halves of the window. At this stage, the TS reduces into a single sequence of tokens (cluster labels). The sliding window measures the similarity between subsets of tokens before and after some point. We define the similarity measure as Eq. 4. Equation 4 is the Manhattan distance between two vectors U and V. These vectors represent the first and second window halves, respectively. U and V are constructed based on the frequency of each cluster label within them. Each vector is of size k', the number of unique cluster labels within the whole sequence. The values of $u_i \in U$ and $v_i \in V$ are the frequencies of the *ith* cluster label within the vectors. For example, given the window halves *ABAA* and *BBCC*, $U = [3, 1, 0]$ and $V = [0, 2, 2]$, thus $M(U, V) = 6$.

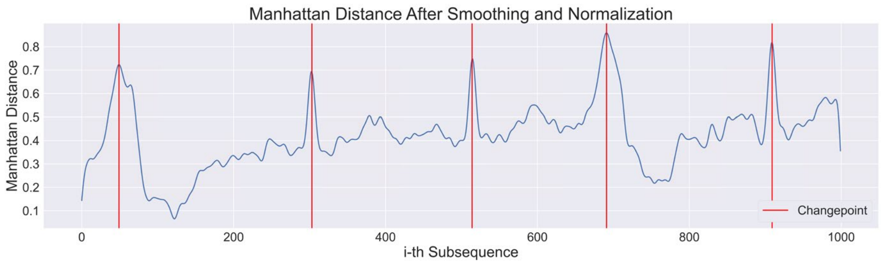
$$M(U,V) = \sum_{1}^{k'} |u_i - v_i| \tag{4}$$

The sliding window introduces the fourth and final hyperparameter, L_w, the length of the sliding window. The sliding window gives us a quantitative measure of similarity over time. We want to find where this measure peaks. A peak within this measure indicates a change in the data-generating state of the time series. Therefore, peaks indicate a CP in the overall TS.

To find peaks in the TS, we first normalize the similarity to be between 0 and 1, where greater values indicate a greater dissimilarity. We then smooth the similarity over time by convolving it with a Blackman window (Podder et al. 2014). The normalization prevents large peaks from dominating others. The smoothing removes jittering but preserves the overall trends, making peaks/CPs detection easier. Figure 3



(a) Sliding window Manhattan distance before Blackman window smoothing and normalization



(b) Sliding window Manhattan distance after Blackman window smoothing and changepoint detection

Fig. 3 Examples of sliding window Manhattan distances before and after smoothing and normalization

shows an example of a sliding window Manhattan distance over a tokenized version of a TS before and after normalization and smoothing. Figure 3a is the sliding window distance before Blackman window smoothing and normalization. The sliding window Manhattan distance contains a lot of minor fluctuations (jittering), making a peak detection algorithm prone to error. Figure 3b shows how our normalization and smoothing preserve the overall peak and valley trends while removing the minor fluctuations. This process helps the peak detection be less prone to error. The red vertical lines indicate the final changepoint predictions in Fig. 3b, corresponding to the most prominent peaks.

We have tried several methods for analyzing the token representation of TS for TSS. These included using different distance metrics for the sliding window, such as Euclidean, different constructions of U and V, and analyzing distances of the clusters in the latent space. These other strategies performed worse than analyzing the frequency of cluster labels with the Manhattan distance. Our analysis showed that some segments may have consecutive subsequences spread far apart in the latent space. Therefore, trying to incorporate the distance between clusters or subsequences will result in worse performance.

Overall, our method can quickly and accurately segment large-scale and complex TS datasets. Our cluster analysis also enables us to gain further insights into the underlying properties of the TS and validate our segmentation. We demonstrate this in Sect. 4.7 with a case study. Our method requires the user to set four hyperparameters. We discuss the intuition of these hyperparameters and their sensitivity in Sect. 4.6.

4 Experimental evaluation

We evaluate the effectiveness and running time of our method against four other variations of itself and seven other baselines on four challenging TS datasets. We also evaluate the influence of our method's main hyperparameters (Sect. 4.6) and present a case study to show how our method can reveal properties of the regimes (Sect. 4.7). All of our code, datasets, and documentation are publicly available (Draayer et al. 2025).

4.1 Setup

For all experiments, we set our parameters to the following. The first phase autoencoder was trained until 1200 epochs ($n_{epochs1} = 1200$). The second phase cluster refinement was trained until 8000 epochs ($n_{epochs2} = 8000$). Leaky ReLU slope is 0.1.

The specifications of the computer used in our experiments is the following: CPU: AMD EPYC 7282 2.8G, Memory: 16GB DDR4 RAM, OS: Red Hat Enterprise Linux release 8 (RHEL 8). GPU: Nvidia Tesla A100.

4.2 Datasets

We select TS datasets that reflect real-world prolonged multi-sensor monitoring situations capturing several segments that may repeat, resulting in complex large-scale MTS. We select 27 MTS datasets spanning three domains: human movement (**PAPAP2(1-9)**, **WESAD(1-15)**), **Sports**, brain wave activity (**EEG**), and music (**Music**). Twenty-five are real and two are semi-real.

Besides these complex and large-scale MTS datasets, we also test DC-TSS on a repository of 98 univariate TS datasets. This repository was proposed by Schäfer et al. (2021) and its subsets have been used in many other TSS studies (Gharghabi et al. 2017; Burg and Williams 2020).

Each dataset's information is detailed below. We then describe how the ground truth change points are obtained.

1. PAMAP2 (Reiss and Stricker 2012) contains real physiological and motion MTS datasets for 9 subjects. All the subjects' data are used in the experiments and referred to as PAMAP2(1), $\cdots$, PAMAP2(9). PAMAP2(1-8) are typical large-scale complex TS with 252833 to 447000 observations, and 18 to 27 CPs. PAMAP2(9) is an exception with only 8477 observations and 2 CPs. PAMAP2(1-9) all have 40 variables. PAMAP2(1-9) correspond to subjects 1-9 in the original repository. The data is sampled at 100 Hertz (Hz).
2. WESAD (Schmidt et al. 2018) is another real physiological and motion repository with 15 MTS datasets. All the subjects' data are used in the experiments and referred to as WESAD(1), $\cdots$, WESAD(15). Each of the MTS datasets contains 14 variables, 3656100 to 4949700 observations, and 14 to 16 CPs. The data is sampled at 700Hz. WESAD(1-15) correspond to subjects 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, and 17 in the original repository. The original repository omits data for subjects 1 and 12.
3. Daily and Sports Activity (Sports) (Altun et al. 2010) is a semi-real MTS dataset with 45 variables, 170,250 observations, and 29 regime changes. Sports is constructed from smaller labeled MTS segments from 17 different activities recorded at 25Hz by various motion capture sensors on a person.
4. EEG (Schalk et al. 2004) is a real MTS dataset with 64 variables, 259,520 observations, and 13 regime changes. The dataset contains brain wave recordings of a person performing motor/imagery tasks sampled at 160Hz.
5. Music (Defferrard et al. 2016) is a semi-real MTS dataset with 148 variables, 309,972 observations, and 29 regime changes. The Music dataset is constructed from 30 different songs from various genres and artists and is the power spectral representation of audio recordings sampled at 44,100Hz.
6. The Schäfer repository (Schäfer et al. 2021) is a collection of 98 univariate TS datasets from various domains. These TS datasets are much shorter than Sports, PAMAP2, EEG, Music with fewer CPs to detect.**Transition periods.** Semantic segmentation focuses on segmenting at low levels of granularity or on the overall data-generative states (Gharghabi et al. 2017). For example, a person walking, a song, or a patient in septic shock are all examples of semantic segments. In semantic segmentation, it is understood that certain short actions or events, like

a person stumbling or taking a sip of water, do not signify the transitions from one data-generating state to another. There is also an understanding that many of the transitions between segments are not instantaneous. Datasets like PAMAP2, EEG, and WESAD likely contain non-instantaneous transitions between semantic segments, as the physiological data were recorded at a high frequency.

Ground truth. For PAMAP2(1-9) (Reiss and Stricker 2012), EEG (Schalk et al. 2004), and WESAD(1-15) (Schmidt et al. 2018), the ground truth (i.e., CPs for semantic segmentations) of these datasets were provided by the dataset creators, it remains unclear how transition periods were handled - specifically, whether they were assigned to the preceding segment, the subsequent one, or treated as distinct intervals. For the Sports and Music datasets, we establish the ground truth by marking the observations where one regime is concatenated to another as CPs which represent the boundary between two different data-generative states. This follows the same gold standard convention established for PAMAP2, EEG, and WESAD.

Our proposed method solely identifies CPs. Once CPs are identified, our method does not examine whether a segment is super short and is transition period. Instead, the identified CPs are directly compared with the annotated ground truth to compute evaluation metrics. The way a transition-period segment (in human perception) is identified depends on which points are identified as CPs by the method. Our approach does not include any post-processing to adjust or handle very short segments. This strategy of evaluating the quality of CPs (i.e., segments) has been utilized by all the previous work.

4.3 Competing methods

We compare our method with seven baseline methods.

- ClaSP (Schäfer et al. 2021) is a recent state-of-the-art method that has been shown to outperform many others. We select ClaSP as a main competitor due to its high performance in other recent studies (Harańczyk 2023) for domain agnostic semantic segmentation for univariate TS. Harańczyk (Harańczyk 2023) has recently created a version of ClaSP that can directly process MTS datasets. Given its strong performance but lack of available source code, we have implemented a very similar version of CLaSP-based method for segmenting MTS, following their methodology. We name our implementation as **CLaSP-MTS-ED**, whose design details are explained below. Compared with (Harańczyk 2023), CLaSP-MTS-ED also aggregates the CPs found across each variable in the MTS. However, (Harańczyk 2023) prunes CPs that fall too closely to an already detected CP. Our version combines them and uses their average to determine the final CP location.
- FLOSS (Gharghabi et al. 2019) is another relatively recent and well-performing semantic segmentation method for univariate TS. We select FLOSS as another possible main competitor due to this fact and its widespread use. We develop a variant of FLOSS for MTS segmentation, called **FLOSS-MTS**, and provide a detailed description below.

- Time2State (Wang et al. 2023) is a recently proposed data-generating state recognition algorithm for MTS. Although it is not directly designed for semantic segmentation, we can still derive segmentations from its output. We select Time2State because of its MTS design and its potential for accurate TSS.
- TIRE (De Ryck et al. 2021) is a segmentation method that uses an autoencoder model for the time-domain and frequency-domain to derive time-invariant features. We select TIRE because it relies on an autoencoder like our method, making it an interesting comparison to our method.
- PELT (Killick et al. 2012) is an older segmentation method designed around MTS data with several CPs. We recognize PELT as a potential competitor due to its ability to handle TS data with many variables and its ability to perform a complete search over the solution space.
- Bayesian Online Changepoint Detection (BOCPD) (Adams and MacKay 2007) is an older method segmentation method for univariate TS. However, we select it because recent benchmark studies (Burg and Williams 2020) still show it is competitive and is widely used. We also implement a version of BOCPD to segment MTS, referred to as BOCPD-MTS, following the details described below.
- Window-L_2 is a fast, simple, and straightforward sliding window-based segmentation algorithm. We select this method to serve as a bottom-line comparison. We follow the same implementation as (Truong et al. 2020).**Design of CLaSP-MTS-ED, BOCPD-MTS, and FLOSS-MTS**. The original design of the methods ClaSP, FLOSS, and BOCPD cannot directly process MTS datasets; they can only process univariate. In order to evaluate them on MTS datasets, we run an instance of each method on each variable of the MTS. We then combine and post-process the resulting segmentations across all variables to yield a final segmentation. We call the set of CPs across all variables $G' = \{G_1, G_2, \ldots, G_d\}$ where d is the number of variables and each $G_i \in G'$ is the set of CPs on the i-th variable. We combine each array and take the average of CPs near each other. For example, if $G_1 = [25, 68]$ and $G_2 = [25, 66, 120]$, their combination yields [25, 25, 66, 68, 120] and the averaging of the CPs near each other yields $G' = [25, 67, 120]$. We consider CPs to be near each other if their difference is less than the margin of error, E, for that dataset. The intuition of our approach is that a CP detected in any of the variables indicates a change in the whole system. This approach of adapting univariate TSS methods for MTS closely follows other implementations (Harańczyk 2023).

We also compare our method with several variations of DC-TSS itself.

- DC-TSS (FC). This variation of DC-TSS uses a fully connected AE instead of a CNN-based AE in Phase 1.
- AE-TSS (CNN). This variation removes the clustering analysis phase of DC-TSS and changes Phase 3 of segmentation to measure distances between embedded neighboring subsequences instead of using cluster analysis. A high distance between neighboring subsequences in the embedded space suggests a segmentation in the TS. In Phase 1, the AE uses a 1D-CNN. This variation is similar to previous work of (Lee et al. 2018).

- AE-TSS (FC). This variation is similar to AE-TSS (CNN) except that it uses a fully connected AE.
- DC-TSS Ablation. We provide an ablation test that removes the clustering refinement stage of our method and directly clusters on the embedded space using hierarchical, agglomerative clustering (Murtagh and Legendre 2014). We test alternative clustering algorithms such as k-means (Lloyd 1982), spectral clustering (Ng et al. 2001), and DBSCAN (Schubert et al. 2017) but these algorithms tend to yield slightly worse performance.

Table 1 shows the **hyperparameter settings** selected for each baseline. For DC-TSS we perform a grid search. The range and steps of the DC-TSS grid search are shown in Sect. 4.6 and Fig. 5. We follow the guidelines for setting hyperparameters outlined in their respective papers for each competitive method. We then adjust the hyperparameters of these competitive methods until we observe major decreases in accuracy. We use the same hyperparameter settings listed under DC-TSS for its variations, where applicable. The latest implementation of ClaSP is parameter-free (Ermshaus et al. 2023). We use its built-in capabilities to set this hyperparameter automatically. We use the same hyperparameter grid search for PAMAP2(2-9) as we have done for PAMAP2(1) in Sect. 4.6. For WESAD(1-15) our grid search tests L_s = [600, 900, 1200, 1500, 1900], Overlap $= [10, 20, 30, 40, 50]$, *k*=10, 40, 70, 100, 130, 160, and L_w = [90, 120, 150, 180, 210]

4.4 Evaluation metrics

We use covering (Everingham et al. 2010) and F1-score (Rijsbergen 1979) to evaluate method performance. These metrics encapsulate two prevailing views of TSS as either a clustering or classification problem. We also report running time, which includes the training time of the deep learner where applicable. Covering is a clustering metric that measures performance by calculating similarity between regimes of the TSS algorithm and ground truth. Eq. (5) defines covering where S and S' are the sets of regimes of the ground truth and method respectively. ||*T*|| is the length of the TS.

$$C(S, S') = \frac{1}{||T||} \sum_{s \in S} ||s|| * \max_{s' \in S'} \frac{||s \cap s'||}{||s \cup s'||} \tag{5}$$

When treating time series segmentation as a classification problem, each observation is classified as either a changepoint or not. In this scenario, the F_1-score can be used to evaluate method performance. Eq. (6) defines the F_1-score.

TP, *FP*, *FN*, and *FP* are true positive, false positive, false negative, and false positive respectively. A true positive, *TP*, is determined based on a margin of error, *E*. If the difference between the ground truth and changepoint is within *E* then it is labeled as a true positive unless a nearer changepoint is present. We set *E* = 200, 3000, 4200, 2400, and 21000 for Sports, PAMAP2(1-9), EEG, Music, and WESAD(1-15) respectively.

Table 1 The hyperparameter settings for each baseline to obtain results in Tables 2, 3, 4, and 5

Hyperparameter Settings for Each Baseline

	DC-TSS				FLOSS-MTS	Time2State		TIRE	PELT		BOCPD-MTS				Window-L_2
	L_s	Overlap	k	L_w	m	*WinSize*	Step	*WinSize*	Pen	Jump	μ	κ	β	α	L_{window}
PAMAP2(1-8)	100	40	70	150	200	12000	9000	100	25000	15	0	0.1	0.1	0.1	700
PAMAP2(9)	100	40	70	150	200	1200	200	100	25000	15	0	0.1	0.1	0.1	700
WESAD(1-15)	1200	30	60	150	300	84000	10000	700	300000	15	0	0.1	0.1	0.1	2000
Sports	25	20	70	200	50	1500	750	50	6600	15	0	0.1	0.1	0.1	200
EEG	160	10	100	150	200	8000	500	200	6800	15	0	0.1	0.1	1	250
Music	340	50	130	200	400	4000	1000	350	900000	15	0	0.01	0.1	0.1	300

$$F_1 = \frac{2PR}{P+R} \tag{6}$$

where

$$P = \frac{TP}{TP+FP}, \quad R = \frac{TP}{TP+FN}$$

4.5 Segmentation evaluation

Tables 2, 3, and 4 show covering, F1-score, and running time of our method, the variations of our method, and the baselines across each dataset respectively. Table 5 shows the number of CPs detected by each method and provides further insight into its performance. Bold values highlight the best scores or closest to ground truth depending on the context of the table. We average the F1-score and covering across five runs for DC-TSS and its versions. Averages are not used for the other baseline methods because they are deterministic. We report the running times for BOCPD-MTS, ClaSP-MTS-ED, and FLOSS-MTS under ideal circumstances. Since we run an independent instance of the method on each feature, we can easily parallelize this computation. Therefore the time to compute a solution for the entire MTS should be the time it takes to compute a solution for a single variable of the MTS. In reality, some datasets can have hundreds of variables, each requiring large allocations for memory or GPUs, making the task of running all instances in parallel unrealistic even with supercomputing resources. Despite this, we decide to report the running time, assuming perfect parallelization is achievable.

Overall, our method (DC-TSS) achieves the best covering and F1-score for the majority of the datasets, including near perfect F1-scores for Sports and Music. PELT also achieves high accuracy in many datasets, but overall poorer performance, especially in terms of running time. DC-TSS's running times also show its architecture allows it to scale well to large-scale datasets. We first compare DC-TSS's performance to its other variations in Sect. 4.5.1 and then compare to the other baselines in Sect. 4.5.3.

4.5.1 Comparison with different variations of DC-TSS

The fully connected (FC) versions of DC-TSS all perform worse than their CNN counterparts, especially for the EEG dataset. The result suggests that our 1-dimensional CNN architecture can learn important discriminative features and the complex relationships between the variables otherwise missed by the FC versions. Besides the better performance, the CNN versions scale much better, resulting in much lower running times for all datasets except Sports, where the FC version has a slight advantage. The difference in running time is most dramatic for our largest dataset, Music, which has a difference of over 250 minutes.

AE-TSS (FC) and AE-TSS (CNN) are the versions that replace our phase 3 cluster analysis of the subsequences with a distance metric proposed by Lee et al. (2018) that aims to determine CPs based on sudden increases in the distances of neighbor-

Table 2 Covering scores of all methods across each dataset

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
PAMAP2(1)	0.749	0.585	0.541	0.679	0.478	0.440	0.590	0.268	0.730	0.444	0.461	**0.765**
PAMAP2(2)	0.711	0.454	0.518	0.712	0.587	0.427	0.587	0.290	**0.753**	0.511	0.495	0.715
PAMAP2(3)	0.721	0.444	0.421	0.712	0.577	0.545	0.547	0.093	**0.744**	.412	0.250	0.730
PAMAP2(4)	0.712	0.517	.497	0.732	0.422	0.487	0.626	0.083	0.753	0.388	0.396	**0.766**
PAMAP2(5)	0.788	0.598	0.581	0.715	0.455	0.482	0.590	0.072	0.712	0.368	0.133	**0.792**
PAMAP2(6)	0.701	0.499	0.506	0.689	0.412	0.418	0.610	0.083	0.727	0.400	0.380	**0.732**
PAMAP2(7)	0.765	0.423	0.502	0.755	0.521	0.490	0.570	0.085	**0.773**	0.407	0.339	0.766
PAMAP2(8)	0.689	0.519	0.499	0.687	0.601	0.431	0.602	0.305	0.672	0.417	0.235	**0.706**
PAMAP2(9)	0.589	0.398	0.402	0.601	**0.746**	0.712	0.512	0.616	0.617	0.544	0.120	0.606
WESAD(1)	0.512	0.371	0.344	0.501	0.383	0.355	0.275	0.113	0.580	0.290	0.522	**0.599**
WESAD(2)	0.645	0.394	0.364	0.609	0.129	0.500	0.261	0.295	**0.507**	0.322	0.473	0.714
WESAD(3)	0.621	0.264	0.255	0.612	0.102	0.417	0.266	0.182	0.766	0.301	0.533	**0.636**
WESAD(4)	0.500	0.222	0.308	0.498	0.179	0.361	0.256	0.107	**0.566**	0.269	0.516	0.521
WESAD(5)	0.651	0.309	0.312	0.631	0.192	0.362	0.244	0.098	0.600	0.269	0.528	**0.680**
WESAD(6)	0.478	0.400	0.392	0.499	0.314	0.380	0.278	0.114	**0.674**	0.275	0.568	0.513
WESAD(7)	0.598	0.342	0.348	0.605	0.178	0.435	0.284	0.222	0.599	0.244	0.602	**0.659**
WESAD(8)	0.589	0.346	0.378	0.581	0.182	0.423	0.288	0.116	0.572	0.254	0.588	**0.612**
WESAD(9)	0.602	0.404	0.328	0.612	0.289	0.352	0.270	0.111	0.553	0.286	0.591	**0.653**
WESAD(10)	0.521	0.214	0.310	0.502	0.274	0.414	0.271	0.119	**0.582**	0.323	0.475	**0.534**
WESAD(11)	0.530	0.400	0.388	0.530	0.305	0.354	0.263	0.109	0.535	0.315	0.497	**0.551**
WESAD(12)	0.540	0.298	0.314	0.508	0.279	0.362	0.260	0.107	**0.645**	0.287	0.493	**0.544**
WESAD(13)	0.578	0.345	0.344	0.589	0.258	0.378	0.286	0.114	0.615	0.238	0.493	**0.618**
WESAD(14)	0.612	0.312	0.333	0.614	0.254	0.430	0.258	0.108	0.526	0.235	0.436	**0.646**
WESAD(15)	0.655	0.389	0.440	0.651	0.308	0.409	0.259	0.103	0.600	0.216	0.543	**0.678**
Sports	0.967	0.549	0.552	0.938	0.537	0.555	0.739	0.061	0.946	0.364	0.713	**0.970**

Table 2 (continued)

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
EEG	0.577	0.386	0.577	0.521	0.421	0.266	0.454	0.142	0.508	0.328	0.381	**0.642**
Music	0.951	0.744	0.519	0.933	0.556	0.652	0.733	0.198	**0.964**	0.536	0.523	0.953
# of wins/ties	0	0	0	0	1	0	0	0	9	0	0	17

Bold values highlight highest scores

Table 3 F1-scores of all methods across each dataset

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
PAMAP2(1)	0.794	0.478	0.558	0.746	0.364	0.448	0.635	0.195	0.737	0.412	0.303	**0.814**
PAMAP2(2)	0.755	0.448	0.398	0.711	0.593	0.398	0.506	0.216	**0.767**	0.360	0.393	0.764
PAMAP2(3)	0.698	0.428	0.435	0.601	0.476	0.488	0.506	0.083	**0.744**	0.500	0.250	0.722
PAMAP2(4)	0.711	0.526	0.477	0.702	0.364	0.400	0.550	0.133	0.727	0.448	0.396	**0.766**
PAMAP2(5)	0.805	0.511	0.479	0.788	0.550	0.510	0.597	0.114	0.690	0.469	0.133	**0.816**
PAMAP2(6)	0.691	0.389	0.367	0.682	0.444	0.556	0.638	0.129	0.717	0.475	0.380	**0.731**
PAMAP2(7)	0.784	0.404	0.378	0.785	0.578	0.622	0.515	0.121	**0.840**	0.421	0.339	0.800
PAMAP2(8)	0.779	0.421	0.479	0.725	0.571	0.424	0.525	0.313	0.724	0.391	0.235	**0.792**
PAMAP2(9)	0.500	0.167	0.182	0.500	0.749	0.749	0.503	0.800	**0.800**	0.500	0.200	0.500
WESAD(1)	0.483	0.289	0.302	0.429	0.300	0.421	0.230	0.102	**0.500**	0.222	0.268	**0.500**
WESAD(2)	0.587	0.344	0.328	0.586	0.272	0.487	0.291	0.222	0.350	0.247	0.269	**0.605**
WESAD(3)	0.501	0.301	0.321	0.498	0.260	0.510	0.180	0.222	0.688	0.142	0.267	**0.516**
WESAD(4)	0.457	0.366	0.358	0.457	0.320	0.350	0.246	0.103	**0.558**	0.196	0.280	0.512
WESAD(5)	0.516	0.403	0.374	0.533	0.261	0.500	0.228	0.045	0.500	0.235	0.212	**0.563**
WESAD(6)	0.414	0.298	0.314	0.400	0.320	0.368	0.147	0.121	**0.686**	0.226	0.359	0.424
WESAD(7)	0.512	0.310	0.391	0.618	0.272	0.436	0.271	0.117	0.615	0.189	0.438	**0.645**
WESAD(8)	0.552	0.378	0.355	0.483	0.333	0.343	0.210	0.118	0.564	0.227	0.415	**0.621**
WESAD(9)	0.429	0.308	0.324	0.500	0.385	0.333	0.165	0.133	0.526	0.275	0.415	**0.581**
WESAD(10)	0.333	0.218	0.201	0.400	0.273	0.329	0.169	0.108	**0.564**	0.189	0.368	0.437
WESAD(11)	0.400	0.201	0.194	0.400	0.273	0.378	0.188	0.129	**0.450**	0.185	0.267	0.424
WESAD(12)	0.371	0.303	0.290	0.371	0.385	0.407	0.231	0.114	**0.529**	0.222	0.237	0.457
WESAD(13)	0.500	0.381	0.345	0.452	0.385	0.479	0.229	0.108	**0.588**	0.318	0.368	0.529
WESAD(14)	0.444	0.320	0.312	0.444	0.272	0.333	0.208	0.111	0.378	0.264	0.286	**0.500**
WESAD(15)	0.500	0.307	0.288	0.429	0.261	0.439	0.229	0.103	0.571	0.261	0.289	**0.600**
Sports	0.984	0.368	0.313	0.966	0.262	0.473	0.627	0.337	0.966	0.160	0.556	**0.987**

Table 3 (continued)

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
EEG	0.486	0.429	0.486	0.500	0.255	0.261	0.470	0.143	0.258	0.071	0.381	**0.667**
Music	0.978	0.701	0.505	0.967	0.514	0.633	0.800	0.186	0.967	0.382	0.537	**0.989**
# of wins/ties	0	0	0	0	0	0	0	0	11	0	0	17

Bold values highlight highest scores

Table 4 Running time in minutes of all methods across each dataset

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
PAMAP2(1)	29.1	21.9	29.1	18.7	796.9	13.8	7.6	36.8	2952.2	94.7	0.1	21.9
PAMAP2(2)	34.4	24.2	34.4	21.8	823.3	14.7	7.7	43.3	3124.9	99.5	0.2	24.1
PAMAP2(3)	22.1	16.7	15.2	14.7	541.3	11.3	7.5	25.0	1984.9	66.6	2.8	16.7
PAMAP2(4)	27.3	21.2	19.5	17.3	702.7	13.1	7.6	36.4	2575.8	85.9	3.5	21.1
PAMAP2(5)	30.1	23.8	22.0	19.6	798.4	14.7	7.7	35.1	2933.4	98.8	3.7	22.8
PAMAP2(6)	29.8	22.1	22.3	19.9	770.9	15.3	7.6	30.5	2827.6	94.3	3.6	22.1
PAMAP2(7)	26.0	19.0	19.6	16.5	667.9	12.5	7.6	40.0	2461.5	81.6	3.6	20.2
PAMAP2(8)	30.3	22.4	30.2	19.1	821.4	14.4	7.7	40.0	3012.4	97.3	0.12	22.4
PAMAP2(9)	3.9	3.5	2.9	2.4	22.9	1.3	2.3	1.6	7.3	5.5	0.1	3.6
WESAD(1)	438.0	46.7	437.6	42.3	928.5	22.4	50.3	768.1	1858.1	376.9	1.0	46.8
WESAD(2)	457.6	57.2	457.6	44.1	923.8	22.1	50.8	820.5	2220.5	384.6	1.1	52.2
WESAD(3)	482.4	59.9	482.4	43.2	933.3	22.9	50.7	802.3	2158.2	404.8	1.1	52.9
WESAD(4)	438.0	48.1	438.2	40.5	954.9	20.1	50.6	789.4	2138.6	390.1	0.9	48.3
WESAD(5)	499.3	57.1	490.2	47.9	1079.1	21.9	51.1	886.4	2179.1	439.8	1.0	57.8
WESAD(6)	367.2	38.9	355.9	36.6	799.9	20.5	49.9	658.8	2058.6	327.6	1.0	40.1
WESAD(7)	504.3	63.4	504.3	45.9	958.4	24.2	50.0	690.9	2154.6	423.1	0.9	55.4
WESAD(8)	368.1	40.2	368.3	35.5	796.6	18.4	49.8	659.4	2200.9	326.6	1.1	41.0
WESAD(9)	386.3	41.0	386.3	37.5	840.4	19.4	50.1	690.7	2144.4	343.6	1.2	42.2
WESAD(10)	366.8	41.1	366.4	37.6	798.1	18.5	49.9	659.6	2096.3	324.4	0.9	41.1
WESAD(11)	391.2	42.3	391.1	35.7	846.6	20.5	50.1	700.3	2043.6	342.2	0.9	44.5
WESAD(12)	392.0	47.2	392.9	34.7	850.3	19.6	50.1	700.3	2169.2	345.9	1.1	47.6
WESAD(13)	371.1	40.8	370.4	34.7	804.0	18.5	49.9	671.2	2089.6	325.4	1.1	41.2
WESAD(14)	396.8	44.9	395.1	39.2	859.9	20.6	50.1	713.3	2011.8	351.1	1.0	45.3
WESAD(15)	420.2	46.1	419.7	39.2	905.1	21.9	50.2	744.1	2008.4	367.1	0.9	45.6
Sports	18.1	18.4	18.1	15.5	100.0	3.1	0.8	16.8	332.2	613.1	0.1	18.4

Table 4 (continued)

Dataset	DC-TSS Variants and AE-TSS				Baseline Methods							
	DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	DC-TSS
EEG	34.3	19.4	34.3	17.2	83.5	7.1	4.7	28.6	373.3	94.1	0.10	19.4
Music	274.1	45.4	274.1	32.4	643.5	9.2	36.3	202.9	3124.7	233.7	0.40	45.4

Table 5 Number of changepoint detections for each baseline

Dataset	Ground Truth	DC-TSS Variants and AE-TSS				Baseline Methods							DC-TSS
		DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	
PAMAP2(1)	25	29	42	44	29	30	33	**28**	7	30	48	39	31
PAMAP2(2)	27	**26**	44	44	24	30	29	29	15	31	44	32	**28**
PAMAP2(3)	18	**16**	28	28	15	26	25	29	4	23	29	100	**16**
PAMAP2(4)	22	19	38	39	19	**24**	**24**	27	6	31	38	72	**20**
PAMAP2(5)	24	22	40	40	22	20	27	27	9	32	40	4	**24**
PAMAP2(6)	24	**24**	39	39	25	27	21	28	5	27	33	53	26
PAMAP2(7)	23	21	36	35	21	21	21	28	8	25	34	34	**24**
PAMAP2(8)	25	27	44	39	23	21	31	**26**	11	31	41	7	**26**
PAMAP2(9)	2	**2**	**2**	**2**	**2**	4	4	8	1	1	6	26	**2**
WESAD(1)	14	15	27	26	**14**	6	24	76	23	20	31	81	16
WESAD(2)	16	**15**	32	30	13	6	19	78	20	22	29	86	19
WESAD(3)	16	12	29	24	12	7	**18**	77	21	**14**	30	42	13
WESAD(4)	16	**19**	32	38	**19**	9	24	77	21	25	35	82	21
WESAD(5)	16	13	28	36	**14**	7	20	78	26	22	35	114	**14**
WESAD(6)	16	13	14	30	14	9	22	77	**15**	**17**	37	21	**15**
WESAD(7)	16	13	33	28	10	6	18	78	**16**	21	34	46	14
WESAD(8)	16	**13**	31	30	**13**	8	**19**	77	12	21	28	35	**13**
WESAD(9)	16	12	25	29	12	10	20	75	**19**	20	35	35	**13**
WESAD(10)	16	**14**	27	28	**14**	6	19	77	13	21	37	20	**14**
WESAD(11)	16	14	39	37	14	6	21	78	17	**16**	38	27	15
WESAD(12)	16	**15**	32	33	**15**	10	19	77	18	22	29	100	**17**
WESAD(13)	16	15	30	28	15	10	19	77	19	**16**	28	20	**16**
WESAD(14)	16	11	27	31	11	6	20	78	**18**	19	37	10	10
WESAD(15)	16	12	25	28	12	7	25	78	21	**17**	30	58	12

Table 5 (continued)

Dataset	Ground Truth	DC-TSS Variants and AE-TSS				Baseline Methods							DC-TSS
		DC-TSS (FC)	AE-TSS (CNN)	AE-TSS (FC)	DC-TSS Ablation	ClaSP-MTS-ED	FLOSS-MTS	Time2State	TIRE	PELT	BOCPD-MTS	Window-L_2	
Sports	29	26	40	41	26	26	32	24	16	26	58	36	**28**
EEG	13	12	22	18	7	8	15	4	1	6	40	30	**13**
Music	29	**28**	35	42	27	33	31	22	10	**28**	50	47	**28**
# of wins/ties		10	1	1	7	1	3	2	4	6	0	0	17

The bold numbers are closest to the ground truth ing subsequences in the latent space. Both methods perform much worse on all the datasets compared to DC-TSS. The result suggests that our cluster analysis approach is better suited for semantic segmentation than previously proposed distance analysis measures. Table 5 shows the AE-TSS variations tend to over detect on each of the datasets compared to the DC-TSS variations.

4.5.2 Ablation study

We have conducted an ablation study that omits the second phase refinement. Our ablation study focuses only on the second phase due to the complications surrounding omitting the first or third phases. The third phase yields our final segmentation result. We cannot omit the third phase and still achieve a segmentation result. We can only test alternatives to the third phase analysis, which we do in the form of AE-TSS, a variation of our method (Sect. 4.3). We cannot omit the first phase because the second phase of DC-TSS requires an initial mapping from the input space into some sort of feature space. We can propose an alternative method to our autoencoder to establish this initial mapping, but this would be another variation of our method and not an ablation study. Devising an alternative method for the initial method is not trivial.

The ablation version performs worse for all 27 datasets. We see major differences in performance in several datasets such as EEG, PAMAP2(1), PAMAP2(8), WESAD(1) and WESAD(5), where Tables 2 and 3 show they differ by up to 7%. The ablation version has better running times than DC-TSS, but the differences are insignificant. Our ablation study shows how important the second phase cluster refinement is for finding well-separated clusters we can analyze for semantic segmentation purposes.

4.5.3 Comparison with baselines

All baseline methods except PELT display very poor performance. We attribute the poor performance of FLOSS-MTS and ClaSP-MTS-ED to the complexity of the datasets. When ClaSP was proposed, its largest dataset was univariate with 40000 observations and one CP. The newer version of ClaSP proposed by Harańczyk (2023) was tested on a repository of MTS datasets that have at most 12 variables and a median length of 5000 observations (Ermshaus et al. 2023). In comparison, our smallest dataset has 45 variables with 170250 observations and 29 CPs. FLOSS uses a subset of ClaSP's dataset repository in its evaluation. Although FLOSS has been applied to larger datasets, these tests are for demonstrating scalability. Table 5 shows the number of CPs detected by ClaSP-MTS-ED and FLOSS-MTS is close to the ground truth for most datasets with the exception that ClaSP-MTS-ED tendeds to under detect CPs for the WESAD(1-15) datasets. This result indicates that the changes they detect do not correspond to the ground truth established according to semantic segmentation notation. Time2State has some success but tends to under-detect CPs along with TIRE with the exception of the WESAD(1-15) datasets where it greatly over-detects. BOCPD-MTS and Window-L_2 often over-detect, according to Table 5.

We took great care to adjust all the hyperparameters associated with these methods as explained at the end of Sect. 4.3, but were unable to improve their accuracy.

For example, the latest implementation of ClaSP is parameter-free (Ermshaus et al. 2023). However, we still took the time to adjust the window length associated with the method to try and achieve better results. However, we ultimately used its built-in capabilities to set this hyperparameter automatically.

In our study, PELT achieves high performance in 9 and 11 datasets (Tables 2 and 3 respectively), but performs poorer than our method on the other datasets. PELT's running times are much longer compared to our method. The running time may be alleviated by adjusting a "jump" parameter that restricts PELT to only evaluate a subset of the observations but at risk of worse performance. PELT also does not provide any metric or visualization to help evaluate the integrity of the results, making it difficult to use in an unsupervised setting. As discussed in Sect. 4.7, our method provides visual aids to help interpret results. Our supporting repository contains code to replicate all baseline experiments (Draayer et al. 2025).

4.5.4 Testing on simpler univariate time series

We study the compatibility of DC-TSS on smaller univariate TS datasets like those used to study the performance of ClaSP and FLOSS. We test on the same repository of the 98 univariate TS datasets proposed by Schäfer et al. (2021). The longest TS in this repository is 40000 observations, but 96 of the TS datasets have 20000 observations or less. The number of CPs to detect ranges from 1 to 6.

Figure 4 shows the covering score boxplots for ClaSP, FLOSS, and DC-TSS. These boxplots reveal the distribution of the covering score results across the 98 TS datasets. We see that DC-TSS performs worse overall than ClaSP and FLOSS. DC-TSS has a median covering score of 0.63, while ClaSP and FLOSS have median covering scores of 0.76 and 0.96, respectively. Interestingly, two datasets where DC-TSS performs well include the longer TS datasets (TiltABP and TiltECG) with lengths of 40000 observations. DC-TSS achieves covering scores of 0.998 and 0.942, respec-

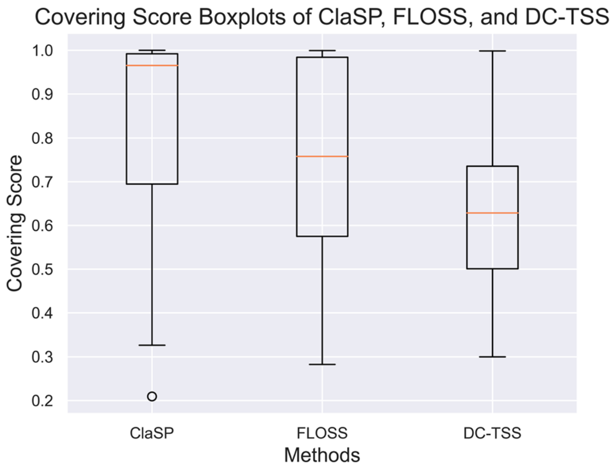


Fig. 4 Boxplots showing covering score distributions when using ClaSP, FLOSS, and DC-TSS to segment Univariate TS

tively. In comparison, ClaSP achieves covering scores of 0.209 and 0.615, while FLOSS achieves covering scores of 0.995 and 0.981 for the same datasets.

Overall, this experiment reveals DC-TSS has limitations on processing simpler/shorter TS. We attribute this to the *shorter* TS datasets not providing enough samples for DC-TSS to learn from. This is evidenced by DC-TSS still performing well on the two much longer TS datasets with 40000 observations.

4.6 Hyperparameter sensitivity and selection

We test the sensitivity of each hyperparameter and discuss the intuitiveness of their settings for our method, DC-TSS. Figure 5 shows a graph for each hyperparameter across each dataset. Each row of the figure corresponds to a specific hyperparameter and each column corresponds to a dataset. The four hyperparameters include subsequence length L_s (introduced in Sect. 3.1), overlap (introduced in Sect. 3.1), number of clusters k (introduced in Sect. 3.2), and the window length L_w (introduced in Sect. 3.3). We ran each hyperparameter experiment five times and averaged results.

4.6.1 Subsequence length

The first row of Fig. 5 shows the performance of DC-TSS as subsequence length (L_s) is changed. We centered L_s at the frequency at which the TS was recorded (1 s worth

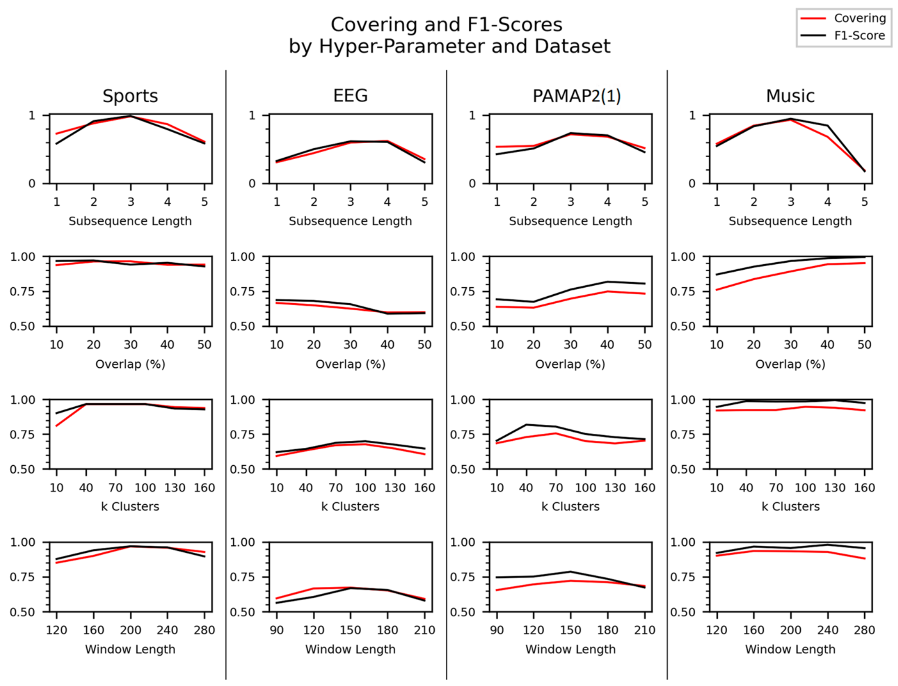


Fig. 5 Hyperparameter experiments across all four datasets. Each row represents the tuning of a different hyperparameter. Each column represents a different dataset

of observations). Our reasoning for this choice of length is to have subsequences represent the primitives that compose an overall segment. Examples of primitives include individual footsteps or the playing of a note, which would span about 1 s. DC-TSS's third phase can then analyze sets of primitives (cluster assignments) before and after a certain point in time to detect the changes between segments. The result is a span of $\frac{1}{4}\times$ to $4\times$ the original frequency. Our results show a peak in performance at the original recording frequency with poorer performance at the lowest and highest settings. This experiment indicates that the L_s hyperparameter, should be set close to the frequency at which the TS is recorded.

4.6.2 Overlap

The second row of Fig. 5 shows performance of DC-TSS across five settings of overlap (from 10% to 50%). Some datasets like Sports and EEG perform better with low overlap while PAMAP2(1) and Music perform better with high overlap. Based on our analysis we conclude this hyperparameter strongly influences the level of granularity of the segmentation. Setting a high overlap discourages clusters from breaking up during phase two refinement, therefore creating lower granularity clusters. For example, segmenting Music with a low overlap setting results in clusters corresponding to instruments or vocals rather than songs. However, either segmentation may be valid depending on the needs of the user. PAMAP2(1) seems to have poorer performance for a different reason. Notably, PAMAP2(1) contains very short regimes. Therefore, setting a low overlap reduces the number of subsequences associated with short regimes, making them more likely to be lost during phase two refinement. In conclusion the overlap should be kept low unless a TS is known to have short regimes or there is need for segmentation at a higher granularity.

4.6.3 k clusters

The third row of Fig. 5 shows the effect of *k* over six settings (10 to 160 clusters incremented by 30). We see the worst performance when *k* is low and slightly poor performance when *k* is high. These results show our method is less sensitive to higher values of *k* than to lower values. This may be because our method tries to fit the input into *k* clusters, but is allowed to find fewer. Therefore, it is better to overestimate than underestimate the hyperparameter *k*.

4.6.4 Window length

The final row of Fig. 5 shows the sensitivity of the phase 3 sliding window length, L_w. We tested a broad range of L_w centered around the highest accuracy according to Table 2. We conclude that the user should visually inspect the phase 3 sliding window distance (Sects. 3.3) to guide setting this hyperparameter. Overall, L_w affects the smoothness of the phase 3 distance. A large L_w analyzes too many subsequences and causes a lack of prominent peaks in the sliding window distance resulting in missed CP detections. A small L_w is sensitive to small changes due to events like a person stumbling over themselves while walking. This results in a noisy sliding window distance; a larger L_w should be used. However, the setting of L_w is robust according to Fig. 5. We expect this result because comparing 15 s of walking to 15 s of running in the respective first and second window halves is very similar to comparing 30 s of walking to 30 s of running, thus yielding a similar distance. The statement holds true if both window halves were of walking.

4.7 Further insights for interpretability

Our method can provide further insight into other underlying properties of a given TS and its regimes by analyzing the clusters and cluster assignments over time. We demonstrate this with one of our more challenging real TS datasets, PAMAP2(1). PAMAP2(1) is a multi-sensor continuous recording of a person performing a variety of chores and activities and contains ambiguous segments of transitioning. PAMAP2(1) also contains missing values and different modalities that require preprocessing, which may affect segmentation results.

To visualize the latent space of our deep neural network, we use t-SNE (Maaten and Hinton 2008) to project the embedded subsequences into two variables. We also test other methods, such as UMAP (McInnes et al. 2018), but found they did not represent the local structures of the clusters very well. Figure 6 shows the t-SNE projection of PAMAP2(1). We can infer properties and relationships between different segments. For example, activities like sitting or standing fall into only 2-3 clusters, while running falls into 5 or more, indicating its complexity. We also see activities like cycling and walking form circular clusters, revealing their cyclical nature. t-SNE does not sufficiently represent global structures such as cluster distances. Instead, we opt to do this analysis in the original feature space. The distance between clusters allows us to infer how closely clusters are related. For example, the ironing

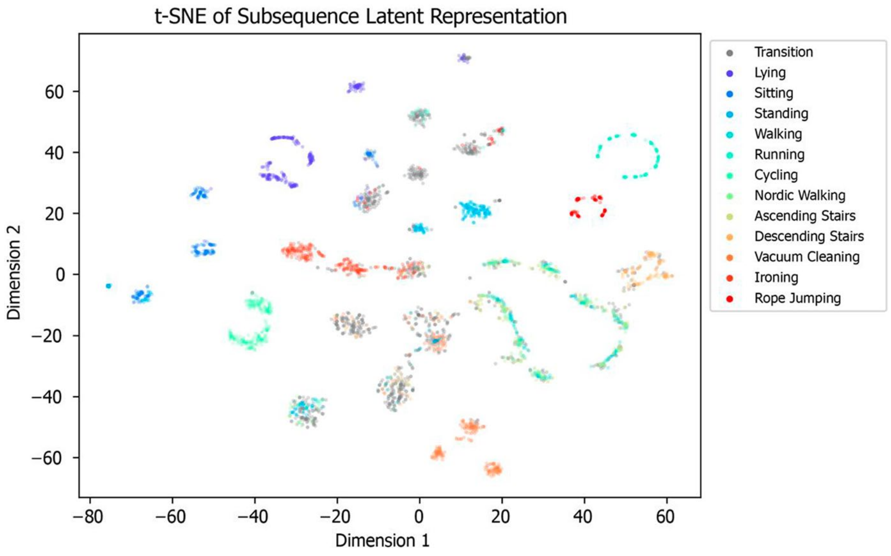


Fig. 6 t-SNE plot of embedded PAMAP2(1) subsequences after training of DC-TSS

clusters are closer in distance and, therefore, more similar to standing than to activities like running. Overall, the t-SNE projection of the embedded subsequences helps reveal many data characteristics that may be useful for other TS analytical tasks. For example, knowing that some activities have strong cyclical behavior may help motif discovery.

We can gain further insight by examining the clustered subsequences over time and the result of the sliding window. Figure 7 displays two plots. The top half shows the Manhattan distance of the sliding window algorithm on the clustered subsequences. We normalize distances between 0 and 1. The bottom half shows an event plot where each black bar is a subsequence, and the y-axis shows its cluster label. The cluster labels of the y-axis are nominal. The background colors indicate the ground truth of the dataset, and the vertical red dotted lines represent our method's segmentation and align with the peaks of the top plot. The sliding window (top) plot can help us determine the strength of some changepoint predictions over others based on the prominence of the peaks. A peak is determined to be prominent if the vertical distance between the peak and its lowest contour line is bigger than 0.07. The lowest contour line is determined by the lowest point between a peak and its neighboring peaks. For example, the two peaks around timestamps 750 in Fig. 7 have a very small valley between them. A horizontal line tangent to the lowest point in that valley is the lowest contour line for both peaks. The vertical distance between the peaks and that line is less than 0.07 and, therefore, both peaks are not prominent enough to be CPs. Additionally, short peaks with a distance smaller than 0.5 on the y-axis are excluded to avoid detecting events such as temporary shifts in a standing position as a transition into a new regime. Lastly, peaks must be separated by half the size of the window length (L_w). This prevented peaks around timestamps 1600, 2050, and 2700 from being detected as CPs.

The peaks also help us validate our results without ground truth, where the lack of prominent peaks suggests that our method does not work well. The periods of con-

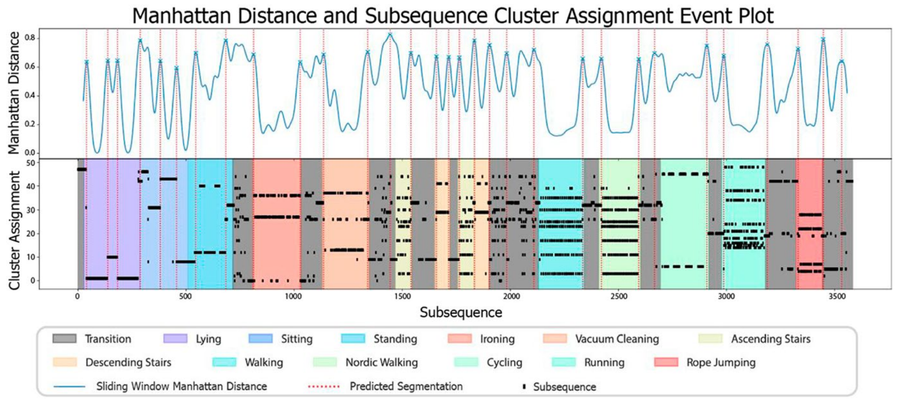


Fig. 7 The top half displays the sliding window result of the clustered subsequences. The bottom half displays an event plot of the clustered subsequence assignments over time. Each black line represents a subsequence, and its position on the y-axis indicates the cluster assignment. The background color shows the ground truth of the dataset and the red vertical dotted lines show where our method predicts a changepoint

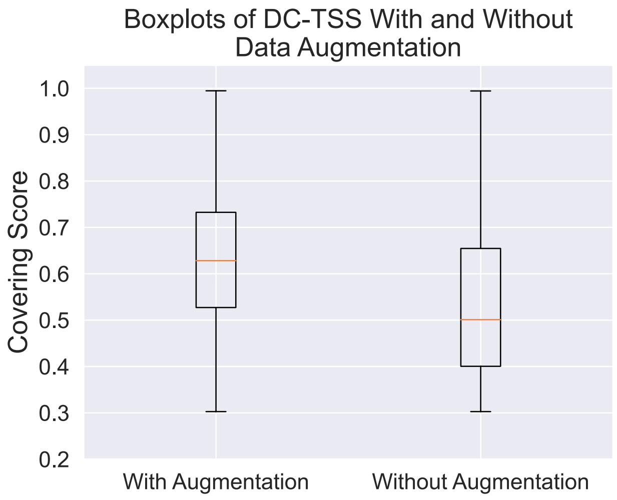


Fig. 8 Boxplots showing DC-TSS coverings scores with and without data augmentation

stant high distance seen in regimes like the second transition (between standing and ironing) suggest a chaotic period in the TS with lots of unique movement from the person. The bottom event plot helps us see the relationship between segments based on their cluster assignments.

For example, walking and Nordic walking share many cluster labels, suggesting these segments are almost identical. Periods of transition also share some cluster labels with walking, which makes sense since the person would need to walk to different locations around the house between activities. We may also see if some activities are subsumed by others, meaning they share most of their cluster labels but one falls into more clusters than the other. Metrics like the Jaccard Index can easily quantify the similarities between segments. The insights DC-TSS provides are helpful for other TS analytical tasks, such as pattern recognition, and are typically not found in other TSS algorithms.

4.8 Data augmentation analysis

To evaluate the effectiveness of our data augmentation strategy, we test DC-TSS both with and without data augmentation on the 98 univariate TS dataset. We select this dataset because it contains many short regimes, which are more likely to be underrepresented during training and thus stand to benefit most from our augmentation technique. Figure 8 presents boxplots of the covering scores under both conditions. The results clearly show that DC-TSS achieves significantly higher covering scores when data augmentation is applied, indicating improved detection of short and sparse regimes.

5 Conclusions

This work introduces DC-TSS, a time series segmentation method for complex large-scale TS datasets. DC-TSS features an efficient architecture for learning MTS representations, a novel component for identifying change points from tokenized time series segments, a data augmentation strategy to address underrepresented states, and the use of computationally efficient operators to improve runtime performance. Our method also provides valuable insights into the underlying properties of the TS.

We test five variations of our method against seven competitive baselines. We test DC-TSS on 27 MTS datasets that are much more complex and larger than other datasets used in current TSS research. In addition to these MTS datasets, we test DC-TSS on a more traditional set of 98 simpler univariate TS datasets. Our results show that DC-TSS outperforms the current state-of-the-art methods by a wide margin on the larger and more complex datasets, while the simpler datasets reveal its limitations. Our results indicate a need for more research on TSS for complex and large-scale datasets and highlights the differences between these datasets and the more widely used simpler datasets from other TSS research.

Acknowledgements This work was supported by Grant Nos. NSF #1757207, #1914635, and GAANN DOE Grant No. P200A180005.

Author contributions Each author has participated sufficiently in the work to take public responsibility for the content.

Data availability Data is publicly available and cited throughout the manuscript. We also provide the preprocessed versions of the data used in our research. Our preprocessed data is available online: https://drive.google.com/drive/folders/1ECwHJetl8EPRkQSMD-rLuWimJ5kMp8qW

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Adams RP, MacKay DJ (2007) Bayesian online changepoint detection. arXiv:0710.3742

Altun K, Barshan B, Tunçel O (2010) Comparative study on classifying human activities with miniature inertial and magnetic sensors. Patt Recogn 43(10):3605–3620. https://doi.org/10.1016/j.patcog.2010.04.016

Aminikhanghahi S, Cook DJ (2017) A survey of methods for time series change point detection. Knowl Inf Syst 51(2):339–367

Burg GJ, Williams CK (2020) An evaluation of change point detection algorithms. arXiv:2003.06222

Chen H, Chu L (2023) Graph-based change-point analysis. Ann Rev Stat Appl 10(1):475–499

De Ryck T, De Vos M, Bertrand A (2021) Change point detection in time series data using autoencoders with a time-invariant representation. IEEE Trans Signal Process 69:3513–3524

Defferrard M, Benzi K, Vandergheynst P, Bresson X (2016) Fma: A dataset for music analysis. arXiv:1612.01840

Draayer E, Huiping C, Gong Q (2025) Supporting website for this paper. https://github.com/ecdraayer/DC-TSS

Ermshaus A, Schäfer P, Leser U (2023) Clasp: parameter-free time series segmentation. Data Min Knowl Disc 37(3):1262–1300

Ermshaus A, Schäfer P, Bagnall A, Guyet T, Ifrim G, Lemaire V, Leser U, Leverger C, Malinowski S (2023) Human activity segmentation challenge@ ecml/pkdd'23. In: International workshop on advanced analytics and learning on temporal data. Springer, pp. 3–13

Everingham M, Van Gool L, Williams CK, Winn J, Zisserman A (2010) The pascal visual object classes (voc) challenge. Int J Comput Vision 88:303–338

Gharghabi S, Yeh C-CM, Ding Y, Ding W, Hibbing P, LaMunion S, Kaplan A, Crouter SE, Keogh E (2019) Domain agnostic online semantic segmentation for multi-dimensional time series. Data Min Knowl Disc 33:96–130

Gharghabi S, Ding Y, Yeh C-CM, Kamgar K, Ulanova L, Keogh E (2017) Matrix profile viii: domain agnostic online semantic segmentation at superhuman performance levels. In: 2017 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 117–126

Harańczyk G (2023) Change points detection in multivariate signal applied to human activity segmentation. In: International Workshop on Advanced Analytics and Learning on Temporal Data. Springer, pp. 14–24

Khalifa Y, Mandic D, Sejdić E (2021) A review of hidden Markov models and recurrent neural networks for event detection and localization in biomedical signals. Inform Fusion 69:52–72

Killick R, Fearnhead P, Eckley I (2012) Optimal detection of changepoints with a linear computational cost. J Am Stat Assoc 107(500):1590–1598

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980

Le QV (2013) Building high-level features using large scale unsupervised learning. In: 2013 IEEE International conference on acoustics, speech and signal processing. IEEE, pp. 8595–8598

Lee W-H, Ortiz J, Ko B, Lee R (2018) Time series segmentation through automatic feature learning. arXiv preprint arXiv:1801.05394

Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–137

Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9(11):2579–2605

Masci J, Meier U, Cireşan D, Schmidhuber J (2011) Stacked convolutional auto-encoders for hierarchical feature extraction. In: Artificial Neural Networks and Machine Learning–ICANN. Springer, pp. 52–59

Matsubara Y, Sakurai Y, Faloutsos C (2014) Autoplait: automatic mining of co-evolving time sequences. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 193–204

McInnes L, Healy J, Melville J (2018) Umap: Uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426

Murtagh F, Legendre P (2014) Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? J Classif 31:274–295

Nagano M, Nakamura T, Nagai T, Mochihashi D, Kobayashi I, Takano W (2019) Hvgh: unsupervised segmentation for high-dimensional time series using deep neural compression and statistical generative model. Front Robot AI 6:115

Ng A, Jordan M, Weiss Y (2001) On spectral clustering: Analysis and an algorithm. Advances in neural information processing systems 14

Podder P, Khan TZ, Khan MH, Rahman MM (2014) Comparative performance analysis of hamming, hanning and blackman window. Int J Comput Appl 96(18):1–7

Reiss A, Stricker D (2012) Introducing a new benchmarked dataset for activity monitoring. In: Proceedings of the 16th IEEE International Symposium on Wearable Computers (ISWC)

Reiss A, Stricker D (2012) Pamap2 physical activity monitoring data set. UCI ML Repository

Rijsbergen CJ (1979) Information Retrieval. Butterworth-Heinemann, Oxford

Schäfer P, Ermshaus A, Leser U (2021) Clasp-time series segmentation. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 1578–1587

Schalk G, McFarland DJ, Hinterberger T, Birbaumer N, Wolpaw JR (2004) Bci 2000: a general-purpose brain-computer interface (BCI) system. IEEE Trans Biomed Eng 51(6):1034–1043

Schmidt P, Reiss A, Duerichen R, Marberger C, Van Laerhoven K (2018) Introducing wesad, a multimodal dataset for wearable stress and affect detection. In: Proceedings of the 20th ACM International Conference on Multimodal Interaction, pp. 400–408

Schubert E, Sander J, Ester M, Kriegel HP, Xu X (2017) Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Trans Database Syst (TODS) 42(3):1–21

Strommen KJ, Tørresen J, Côté-Allard U (2023) Latent space unsupervised semantic segmentation. Front Physiol 14:1151312

Truong C, Oudre L, Vayatis N (2020) Selective review of offline change point detection methods. Signal Process 167:107299

Van Der Maaten L (2009) Learning a parametric embedding by preserving local structure. In: Artificial Intelligence and Statistics. PMLR, pp. 384–391

Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P-A, Bottou L (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11(12):3371–3408

Wang C, Wu K, Zhou T, Cai Z (2023) Time2state: an unsupervised framework for inferring the latent states in time series data. Proceed ACM Manag Data 1(1):1–18

Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: International Conference on Machine Learning. PMLR, pp. 478–487

Xu B, Wang N, Chen T, Li M (2015) Empirical evaluation of rectified activations in convolutional network. arXiv:1505.00853

Zhao J, Itti L (2016) Decomposing time series with application to temporal segmentation. In: 2016 IEEE Winter conference on applications of computer vision (WACV). IEEE, pp. 1–9

Authors and Affiliations

Erick Draayer[1] · Huiping Cao[1] · Qixu Gong[1]

✉ Erick Draayer
edraayer@nmsu.edu

Huiping Cao
hcao@nmsu.edu

Qixu Gong
qixugong@nmsu.edu

[1] Department of Computer Science, New Mexico State University, Las Cruces, NM 88011, USA