Deep Time Series Sketching and Its Application on Industrial Time Series Clustering

Hao Huang*, Tapan Shah* and Shinjae Yoo†
* *GE Global Research*, San Ramon, CA
haohuangcssbu@gmail.com
* *GE Global Research*, San Ramon, CA
Tapan.Shah@ge.com
† *Brookhaven National Laboratory*, Upton, NY
sjyoo@bnl.gov

2022 IEEE International Conference on Big Data (Big Data) | 978-1-6654-8045-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/BigData55660.2022.10020260

***Abstract*—Today, voluminous multivariate time series data collected from sensors provides tremendous benefit for understanding of modern industrial systems such as power plants, wind turbines and aircrafts. However, the dynamic and complex nature of these systems, as well as the lack of prior knowledge impose challenges in perceiving different system behaviors from the time series data. To handle these issues, time series clustering has become one of the key analysis techniques. Nevertheless, the data nonlinearity, varying lengths and high dimensions of industrial time series could hinder the quality of clustering. To deal with these challenges, we propose Deep Time Series Sketching (DTSS) model. This model is a representation learning model based on temporal convolutional networks that perform on a sliding window basis along time series to learn the windows' embeddings. The sequence of embeddings is then fed into embedding sketching to obtain its sketch. Such sketch is a descriptor of the whole time series and will be fed into K-means for clustering. Our model is a novel end-to-end hybrid model that incorporates both local and global contextual features. It is able to project multivariate time series with varying lengths into the same latent space. Moreover, we show that our model is able to perform early clustering as it can assign real-time label without seeing the whole time series. We test our model on both benchmark and real world industrial datasets, and experiments show that our proposed method outperforms popular time series clustering baselines.**

***Index Terms*—multivariate time series, time series clustering.**

I. Introduction

Nowadays, large volume of time series data are collected by various sensors in industrial systems such as power plants, aircrafts and wind turbines. These sensors measure different variables of the systems in high frequency over time, resulting in a vast mass of data mostly impractical for a manual analysis. Machine Learning based time series analysis are therefore becoming increasingly prevalent in industrial applications because of their capability to automatically learn the system behaviors from data.

Among all these techniques, the development of effective unsupervised clustering is extremely crucial in practical scenarios, where labeling enough data to cover all classes for supervised learning may be too expensive, if not impossible. Moreover, clustering allows to discover unknown types of system behavior that go beyond the prior knowledge, serving as tool to support subsequent decision, analysis processes and system maintenance.

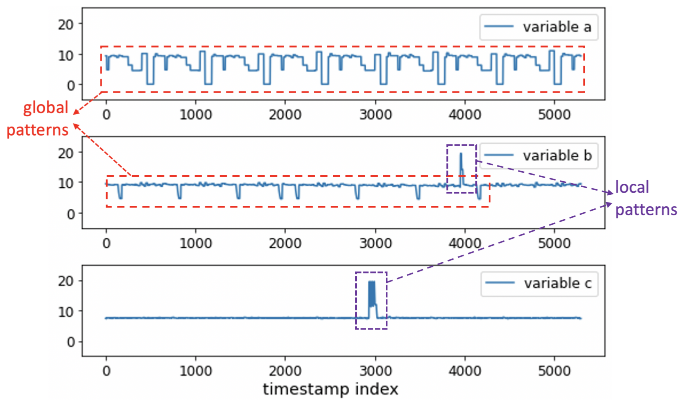


Fig. 1. Example of local and global patterns in industrial time series.

Learning a brief but comprehensive representation of the whole time series is of considerable importance to time series clustering. In industrial multivariate time series, there are usually local and global patterns, as shown in Figure 1. Global patterns usually refer to those patterns that appear repetitively or continuously in a long range, while local patterns are infrequent patterns that only appear in few short term windows. In the context of industrial time series clustering, global patterns are usually more critical for discovering the patterns of interest. However, some salient local patterns (e.g. outliers with extreme large or small values) are also important for revealing certain specific system behaviors. Therefore, we need to design a representation of time series that captures both global and salient local characteristics, while filtering out noise and patterns with limited predictive value.

Conventional time series clustering focus on learning time series representation or similarity metric in raw variable space [25], [8], [28]. However, they do not scale well when dealing with large time series. Moreover, they are very sensitive to noise and outliers [2]. Recent deep learning based sequential models have been popularly used in time series clustering [14], [11], [36]. However, they either fail to balance global and local pattern learning, or cannot deal with extremely long time series

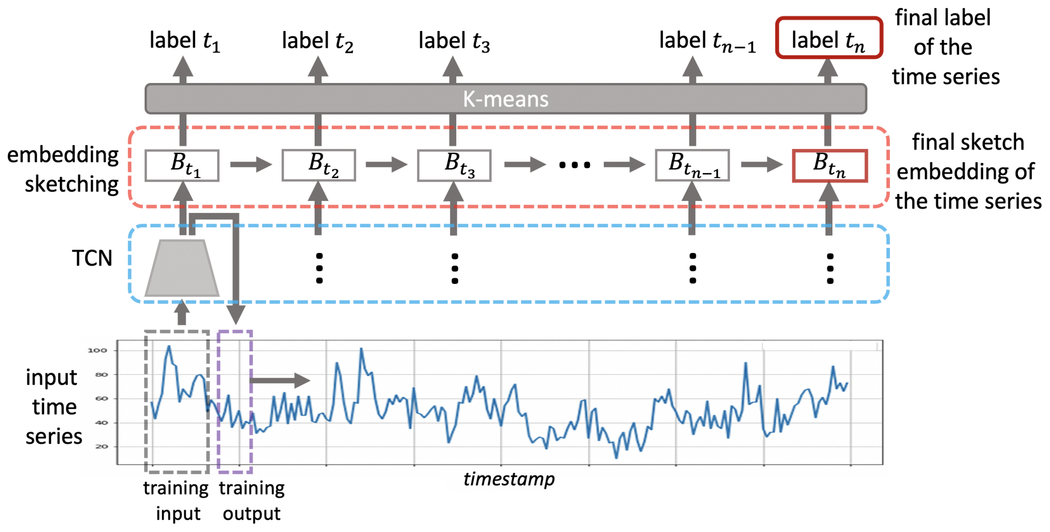


Fig. 2. Our *DTSS* framework. It takes input from time series in a sliding window fashion, extracts embedding vector from each window and updates the sketch of embedding sequence. At any time the current sketch can be fed into K-means to output cluster label of the time series. *DTSS* comprises two parts: temporal convolutional networks (*TCN*) and embedding sketching.

or time series with varying lengths [18], [5].

In this paper, we propose a deep learning and embedding sketching based model, ***Deep Time Series Sketching*** (*DTSS*), for clustering industrial multivariate time series with varying lengths. It is designed as a representation learning method that takes input from time series in a sliding window manner. Nonlinear features of each window is extracted and used to update the time series sketch as the window slides from the beginning to the end of the time series. As shown in Figure 2, our *DTSS* consists of two parts: temporal convolutional networks and embedding sketching. While the encoding form of temporal convolutional networks pays more attention to the local characteristics in each window, the embedding sketching part considers and synthesizes the complete set of patterns along all windows, and outputs a balanced and concise representation that contains both global and salient local characteristics. Furthermore, our *DTSS* can be used in a real time streaming, meaning that, unlike the other algorithms, *DTSS* doesn't require to see the whole time series to assign labels. Instead, it is capable to output the up-to-date representation and label of the time series in real time.

II. Related Works

Conventional time series clustering mainly rely on designing similarity metric or feature extraction. The popular similarity metrics include euclidean distance [35], [7], dynamic time warping [25], [8], correlation [39], [34] and cross-correlation [1], [31]. Among them, dynamic time warping (DTW) is widely used in various applications due to its capability to handle non-linear distortions. The idea is to align (warp) the time series before computing the distance. However, its limitation is on its global operation: DTW considers the shape of the whole time series. Therefore its performance will be highly affected by the noisy or informative part in time series, especially if only a small portion is crucial for prediction [3]. Moreover, even though DTW has some efficient versions like [8], it still does not scale well when dealing with large time series data [2]. In [28], the authors developed k-Shape whereby the shapes of the time-series are considered by applying cross-correlation measures. However, all these methods are usually sensitive to noise and impractical for long time series because all time points are considered in measuring distance between time series [2].

Many aforementioned works measure similarity or distance on raw input space. However, due to the high-dimensionality in both temporal and spatio (variable) space, feature extraction methods have been proposed to project time series into reduced space. They include but not limit to Principal Component Analysis [38], [32], Independent Component Analysis [13], Multidimensional Scaling [16], [12], K-grams [35], Fourier Transform [15] and Discrete Wavelet Transform [29], [19]. Nevertheless, these methods are either linear extraction that cannot capture nonlinear information, or based on certain distribution assumption that data may not follow.

Since conventional methods face challenges in capturing the unknown and complex diversity in high dimensionality and temporal scales in time series data [2], deep learning based methods are gaining more and more popularity due to its capability to perform automatic high-level feature extraction without explicit assumption on data distribution, and its high prediction accuracy when trained with huge amount of data. Recurrent Neural Network (RNN) is one of the most widely used deep learning techniques in time series learning [23], [37], [41], [9]. One recent example is [14] where a bidirectional Gated Recurrent Unit autoencoder is used to produce low dimensional embeddings from the hidden state vectors, followed by a clustering refinement procedure to stretch the embedding manifolds toward different clusters. Nevertheless, it is also well-known that RNN suffers from gradient vanishing problems and cannot capture patterns exist in long sequence[5]. Meanwhile, multi-level temporal convolutional network (TCN) [42], [21], [37] is frequently applied in applications with long sequential data. In particular, the work in [11] constructs an encoder-only architecture using TCN with triple loss and negative sampling to generate representation embeddings. Recently, the authors in [36] proposed a deep-learning based time series clustering method IT-TSC to explore variable associations in each time series and perform clustering subsequently.

It is worth noting that many algorithms require the lengths of all input time series to be the same [28], [30], [27]. For delivering compatible representations while allowing the raw time series to have unequal lengths, these methods usually require either stretching or padding of the shorter sequences, or shrinking of the longer sequences [28] as necessary pre-processing step. However, padding requires to know the time series length in advanced which is impossible in real time clustering when we cannot see the whole time series yet. On the other hand, stretching or shrinking may cause undesirable effect especially for industrial time series. In Figure 3 we illustrate one such case: time series a and b are with different lengths but belong to the same cluster as they both carry similar outlier pattern. But once resampling and interpolation is used to time series a to stretch its length, the original 'sharp-peak' outlier becomes too much smooth and the similarity between time series a and b will be lost.

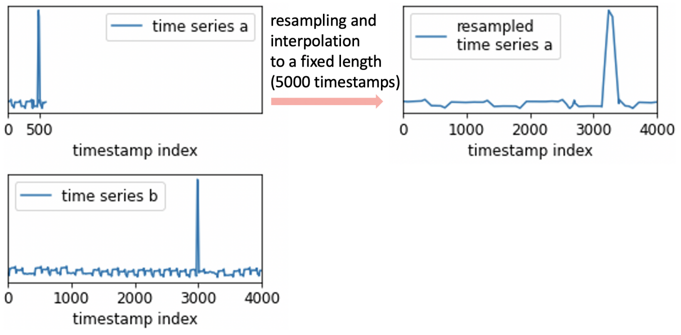


Fig. 3. Example of resampling that may cause undesirable effect. The raw time series a and b should belong to the same cluster. But after resampling, a is no longer similar to b.

In this work, **we propose a deep learning based time series clustering that can handle input with varying lengths without resampling the input data.** By using TCN structure, our method first automatically explores the nonlinear features in a sliding window fashion along time series with any length. Then the sequence of embeddings obtained from TCN bottleneck layer is fed into embedding sketching to learn the sketch of embeddings. The sketch captures the important structure by considering both frequently appearing embeddings and certain salient embeddings regardless of time series length. Experiment in Section V shows that our model is more robust and capable of distilling contextual information from multivariate time series for unsupervised clustering.

III. High Level Description and Notations

In various industrial applications, multivariate time series are collected from sensors with high frequency. A time series with m variables and n timestamps can be noted as $X \in \mathbb{R}^{m\times n}$ [1]. We use $x_t \in \mathbb{R}^{m\times 1}$ to denote the values at timestamp t. In this work, a sliding window is a unit we use to scan each time series. Here $x_{t-\ell+1:t}$, or simply $\vec{x}_{(\ell)t} \in \mathbb{R}^{m\times \ell}$ is used to denote the window that consists of the latest ℓ timestamps before time t.

Our problem setting in this paper is that: given a set of time series $X^1, X^2, ..., X^N$ (N is the total number of time series) with the same number of variables m but possibly different lengths, we aim to propose a clustering algorithm that assigns one cluster label to each time series $label^1, label^2, ..., label^N$.

As shown in Figure 2, our *DTSS* model takes input from each time series in a sliding window fashion, and output cluster label of the time series at any time. *DTSS* comprises two parts: temporal convolutional networks (*TCN*) and embedding sketching. The first part is to learn the nonlinear embedding of each window. This process is formulated as a *regression problem*: given $\vec{x}_{(\ell)t}$, we apply a multi-level *TCN* to predict the next timestamp $x_{t+1} \in \mathbb{R}^{m\times 1}$. The embedding $v_t \in \mathbb{R}^{d\times 1}$ is intermediate output during the regression process, with d the dimensions of latent space.

The second part of our *DTSS* is embedding sketching, of which objective is to learn a small but comprehensive representation from all the acquired embeddings $v_{t_1}, v_{t_2}, ..., v_{t_n}$ of each time series. To this end, each v_t is used to update the current sketch $B_{t-1} \in \mathbb{R}^{d\times k}$ from the previous window and obtain the up-to-date $B_t \in \mathbb{R}^{d\times k}$, where k is the rank of embeddings that we want to maintain in the sketch. We use B^i to denote the final sketch of time series X^i. The flattened version of B^i is noted as $\bar{B}^i \in \mathbb{R}^{1\times dk}$. For all the N time series we can collect a set of $\|_{i=1}^{N} \bar{B}^i \in \mathbb{R}^{N\times dk}$ (where $\|$ indicates matrix concatenation) and perform clustering analysis using K-means.

It is worth emphasizing that our model can perform early time series clustering. That is, since embedding sketching is an incrementally update process, it can output up-to-date sketch for a time series as long as the model has seen a small number of windows. Therefore we can run K-means on all the currently available sketches and obtain their labels in real time. We will test the clustering performance on both intermediate and final sketches in Section V.

Table I lists the key notations used in this paper.

IV. Detailed Description of our DTSS Model

As described in Section III, our *DTSS* model consists of two modules: learning embeddings from sliding window and embedding sketching. We will describe these two modules in details in the following subsections.

[1]Our model can deal with time series of varying lengths. For notation convenience, we use n to denote the final timestamp of all time series.

TABLE I
LIST OF KEY NOTATIONS USED IN THIS PAPER.

m, n, N	number of variables / timestamps / time series
$X \in \mathbb{R}^{m\times n}$	an observed multivariate time series
$x_t \in \mathbb{R}^{m\times 1}$	the values at timestamp t
$\vec{x}_{(\ell)t} \in \mathbb{R}^{m\times \ell}$	a window consists of the latest ℓ timestamps before t
d	latent dimension of embeddings
$v_t \in \mathbb{R}^{d\times 1}$	the embedding of $\vec{x}_{(\ell)t}$
$A \in \mathbb{R}^{d\times n}$	the sequence of embeddings in a time series
k	the rank of sketch embedding
$B \in \mathbb{R}^{d\times k}$	sketch of A
$\widetilde{B} \in \mathbb{R}^{d\times k}$	normalized sketch
$\bar{B} \in \mathbb{R}^{1\times dk}$	flattened sketch
$\|\bar{B} \in \mathbb{R}^{N\times dk}$	the concatenated sketches of all time series

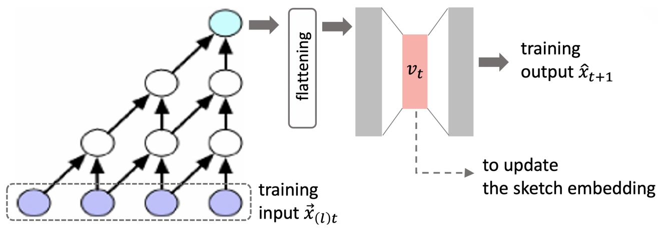

flattening
v_t
training output $\hat{x}_{t+1}$
to update the sketch embedding
training input $\vec{x}_{(l)t}$

Fig. 4. Module 1 is to learn the embedding v_t from each window during the regression from $\vec{x}_{(\ell)t}$ to $\hat{x}_{t+1}$. It consists of a temporal convolutional network (*TCN*) and a fully connected network (*FCN*).

A. Module 1: TCN to Learn Embeddings from Sliding Window

The objective of module 1 is to acquire the embeddings (nonlinearly transformed features in latent space) from each window. The learning is designed in a regression setting: to predict the next timestamp x_{t+1} using the input window $\vec{x}_{(\ell)t}$. The embeddings $v_t \in \mathbb{R}^{d\times 1}$ are intermediate output during the regression process, with d the dimensions of latent space.

As shown in Figure 4, module 1 includes two parts: a temporal convolutional network (*TCN*) followed by a fully connected network (*FCN*).

We first apply *TCN* on the input window to acquire its underlying nonlinear features. *TCN* is originally proposed in [26] and popularly used in various sequence modeling tasks (e.g. [5], [11]). Different from RNN, it is not a recursive structure therefore suffers less from gradient vanishing issues [5]. To be capable to learn from longer window with more nonlinearity, *TCN* usually consists of multiple levels. As shown in the left part of Figure 4, we apply multi-levels *TCN* here. The output of *TCN* is nonlinear features that represent the input window.

Given the (flattened) output from *TCN*, a sequence of fully connected layers are used to regress to predict $\hat{x}_{t+1}$, illustrated in the right part of Figure 4. The embedding v_t of the input window is obtained from the bottleneck layer in this sequence that provides low dimensional temporal embedding. The same module 1 is applied to all windows in all time series to learn their embeddings in the same latent space.

In a detailed manner, the first level of *TCN* takes raw readings from window $\vec{x}_{(\ell)t}$. At each level, several 1D convolutions are applied on the activation (Hyperbolic Tangent function, or *tanh*, is used in our design) of the previous level. We denote the flattened output from the last *TCN* layer as $g_t \in \mathbb{R}^{p\times 1}$. Then g_t is fed into the chain of fully connected layers to predict the next timestamp x_{t+1}. A *tanh* activation function is applied to all the layers' output except for the last layer's. Among all these fully connected layers, there is a bottleneck layer that is a lower dimensional layer where the embedding v_t is produced. The bottleneck layer outputs the embedding v_t with a lower number of nodes and this number of nodes (d) also gives the dimension of the embeddings. The last layer of the chain outputs the predicted value $\hat{x}_{t+1}$. The residual is calculated by mean squared error (*MSE*) between $\hat{x}_{t+1}$ and the regression target x_{t+1}. The training process uses this residual to update the model with back-propagation. Please note that the neural network training in module 1 is **self-contained**, that is: it doesn't take any feedback from module 2 or K-means.

For each time series, module 1 is applied to all the windows and output a sequence of embeddings $v_{t_1}, v_{t_2}, ..., v_{t_n}$. We use $A \in \mathbb{R}^{d\times n}$ [2] to denote the concatenated version of this sequence (along columns).

B. Module 2: Embedding Sketching

The objective of module 2 is to obtain the sketch of the concatenated embeddings A from module 1. A sketch of matrix is an approximation of the original matrix but much smaller. That is, we want to reduce $A \in \mathbb{R}^{d\times n}$ to its sketch $B \in \mathbb{R}^{d\times k}$ where $k \ll n$. One key requirement of sketching is that the variations of the sketch attributes' variance with each other should be nearly the same as those in the original matrix. In other words, the difference of their covariance matrix $\|AA' - BB'\|_2$ should be very small where $(*)'$ denotes matrix transpose and $\| * \|_2$ denotes ℓ_2 norm.

In our work there are three key motivations to obtain sketch: 1) The number of n is varying across different time series due to their unequal lengths. Therefore it is crucial to reduce them to a constant size k in order to perform further comparison among them. 2) The number of n can be extremely large because some industrial time series are very long. 3) Furthermore, obtaining sketch from a sequence of window embeddings can connect the local view of each window to the global view of the whole time series by filtering out the insignificant embeddings and meanwhile preserving the frequently seen local patterns or some salient local patterns (e.g. outliers with immensely small/negative or large/positive values).

Below we will discuss two ways of embedding sketching we used in the paper.

[2]The actual size of A is $\mathbb{R}^{d\times(n-\ell+1)}$. Here we simplify it for notational convenience.

1) Low-Rank Singular Vector Decomposition: One straightforward way to obtain sketch is through a rank-k approximation using truncated singular vector decomposition (SVD_k). We denote the singular values, left and right singular vectors as $S_{(k)} \in \mathbb{R}^{1\times k}$, $U_{(k)} \in \mathbb{R}^{d\times k}$ and $V_{(k)} \in \mathbb{R}^{n\times k}$. Then, we can get the sketch B from A by the following equations:

$$\begin{aligned} &U_{(k)}, S_{(k)}, V_{(k)} \leftarrow SVD_k(A), \\ &B \leftarrow U_{(k)}\,\mathrm{diag}(S_{(k)}), \end{aligned} \tag{1}$$

where diag denotes the diagonal operator that converts a vector to a diagonal square matrix. In this case, it is easy to prove that $\|AA' - BB'\|_2$ is bounded by $\|\,\mathrm{diag}(S_{k+1:n})\|_2$ that is the ℓ_2 norm of the diagonal matrix that consists of all but the first k singular values.

2) Frequent-Directions: Although the first way can provide sketch, its drawbacks are obvious: SVD_k is very time consuming if the input matrix A is extremely large (i.e. n is large). Moreover, it is not practical if we want to run early clustering.

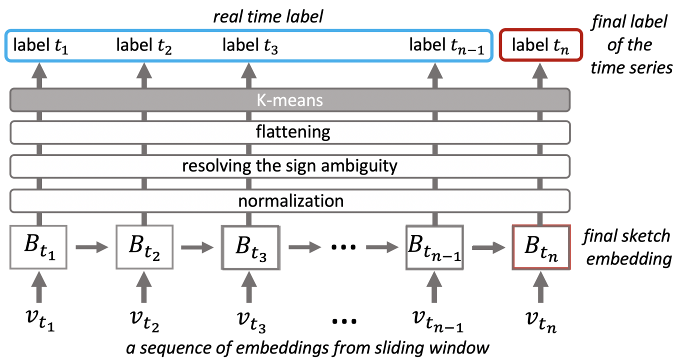


Fig. 5. Module 2 learns the sketch from the sequence of embeddings along a time series, which will be fed into K-means to obtain the time series's label.

To solve these problems, we resort to *frequent-directions* [22], [43]. *Frequent-directions* is a real time data compression technique that is characterized by the property of preserving most of the information that are presented in a streaming fashion [10]. Different from SVD_k, it doesn't require to see the whole matrix. Instead, this method maintains an up-to-date sketch and updates it whenever new sample arrives. Particularly, it takes v_t to update the current $B_t \in \mathbb{R}^{d\times k}$ to $B_{t+1} \in \mathbb{R}^{d\times k}$, as shown in Figure 5. It is proved in [22] that $\|AA' - BB'\|_2 \leq 2\|A\|_f^2/k$ by using this method where $\|*\|_f$ denotes Frobenius norm. In this work, we apply *frequent-directions* to obtain the sketch of the concatenated embeddings A from each time series. Every time it takes input from a column of A. The computation complexity is $O(dk^2)$ that is much less than $O(dnk)$ in SVD_k.

The sketching procedure we used is described in Algorithm 1. The key part is from line 5 to 7. By setting the smaller singular values to be zero and reweighing the larger singular values, it absorbs the informative part from new embeddings into the sketch, and compresses the current sketch to make space for the coming embeddings.

It is worth emphasizing that we can use the intermediate B_t at any time t ($1 \leq t < n$) for early clustering. We will further show this in our experiment section.

Algorithm 1 Embedding Sketching (using *frequent-directions*)

```
Input: $k$, a sequence of embeddings $v_{t_1}, v_{t_2}, ..., v_{t_n}$ from the
    same time series.
Output: sketch embedding $B$
    Initialization : $B \leftarrow$ all zeros matrix $\in \mathbb{R}^{m\times k}$
 1: for $i = 1$ to $n$ do
 2:     insert $v_{t_i}$ into a zero valued column of $B$
 3:     if $B$ has no zero valued columns then
 4:         $[U, S, V] \leftarrow \mathrm{SVD}(B)$
 5:         $\delta \leftarrow S^2_{\lfloor k/2 \rfloor}$
 6:         $\check{S} \leftarrow \sqrt{\max(S^2 - I_k\delta, 0)}$
 7:         $B \leftarrow U\,\mathrm{diag}(\check{S})$
 8:     end if
 9: end for
10: return $B$
```

After getting the sketch, there are three necessary processing steps we need to apply to B before feeding them into K-means, as shown in the upper part of Figure 5:

Normalization. In both Equation 1 and line 7 of Algorithm 1, the scales of S are proportional to the value of n (the length of the time series). This will significantly increase the distance between shorter and longer time series in the final K-means even they should belong to the same cluster. To handle this issue, we perform a $l2$-normalization to S. The normalized sketch is then obtained by:

$$\begin{aligned} &\widetilde{B} \leftarrow U_{(k)}\,\mathrm{diag}(S_{(k)}/\|S_{(k)}\|_2), \quad \text{if using } SVD_k, \\ &\widetilde{B} \leftarrow U\,\mathrm{diag}(\check{S}/\|\check{S}\|_2), \quad \text{if using Algorithm 1.} \end{aligned} \tag{2}$$

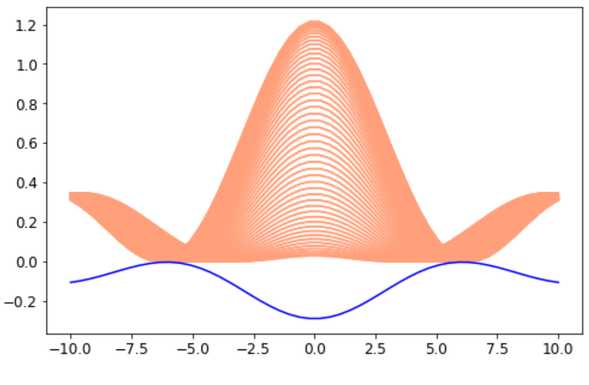


Fig. 6. Example of *SVD*'s sign ambiguity. Each red line represents one column in a 50 × 100 matrix and the solid blue line the associated first left singular vector of this matrix. Apparently this singular vector has the wrong sign as it points in the opposite direction of the vectors it represents.

Resolving Sign Ambiguity. Although both SVD_k and frequent-directions (which is also based on *SVD*) have been used in many applications, it is not popularly mentioned that there is an intrinsic sign indeterminacy that can significantly impact the conclusions drawn from their results [6]. Precisely, the singular vector decomposition is only unique up to a reflection of each set of singular components, but inside each component it holds that:

$$U_i S_i V_i' = (-U_i) S_i (-V_i'), \tag{3}$$

where U_i, S_i and V_i' indicate the i-th set of singular components. In other words, the *SVD* itself provides no means for determining the sign of each singular vector. And this indeterminacy is inherited so that the individual singular vectors have an 'arbitrary' sign [6]. An example of *SVD*'s sign ambiguity is shown in Figure 6. This problem will lead to large Euclidean distance between two sketches with similar shapes but different signs in the final K-means.

To solve this issue, we resort to the technique proposed in [6], and document the solution in Algorithm 2.

Algorithm 2 Resolving Sign Ambiguity

```
Input: $\widetilde{B} \in \mathbb{R}^{d\times k}$, $A \in \mathbb{R}^{d\times n}$
Output: $\widetilde{B}$ with corrected signs
1: $[U, S, V] \leftarrow \mathrm{SVD}(\widetilde{B})$        ▷ V is just identity matrix
2: for $i = 1$ to $k$ do
3:     $w \leftarrow U_i' A$
4:     $w \leftarrow sign(w) \circ (w \circ w)$        ▷ $\circ$ is Hadamard product
5:     if $\sum w < 0$ then
6:         $U_i \leftarrow (-1) U_i$
7:     end if
8: end for
9: $\widetilde{B} \leftarrow U\,\mathrm{diag}(S)$
10: return $\widetilde{B}$
```

The logic behind Algorithm 2 is that, the sign of a singular vector should be similar to the sign of the majority of vectors it represents. Geometrically, it should point in the same direction as the majority of vectors it represents. Specifically, line 3 in Algorithm 2 is to calculate the dot product of each singular vector and each vector in A, which tells their Euclidean magnitudes and the cosine of the angle between them. In line 5 if $\sum w < 0$, it means that the sign of the singular vector is different from the sign of the majority of vectors in A, so its direction need to be corrected by line 6.

However, line 3 could be impractical when the size of A is large. In this case we can use reservoir sampling [33], [20] that is a uniform sampling method of data streams (since our embeddings come in a streaming fashion with sliding window). By doing so we can maintain a real time down-sampled version of A and use it in line 3.

By applying Algorithm 2 and Equation 2, we make sure that similar sketches are close to each other in the Euclidean space.

Flattening. The last step before feeding sketches into K-means is flattening all sketches by using:

$$\begin{aligned} &\bar{B}^i \leftarrow flattening(\widetilde{B}^i), \\ &\|_{i=1}^{N} \bar{B}^i \leftarrow concatenate([\bar{B}^1, \bar{B}^2, ..., \bar{B}^N]), \end{aligned} \tag{4}$$

where $\bar{B}^i \in \mathbb{R}^{1\times dk}$ and $\|_{i=1}^{N} \bar{B}^i \in \mathbb{R}^{N\times dk}$. The final K-means takes $\|_{i=1}^{N} \bar{B}^i$ as input and performs row-wise clustering. The label of each row is the label of the corresponding time series.

V. Experiments

This section demonstrates the superior capability of our *DTSS* in time series clustering. We show two versions of our *DTSS*: *DTSS-SVD*$_k$ and *DTSS-FP*. The main difference is how to obtain the sketch of embeddings: *DTSS-SVD*$_k$ uses rank-k *SVD* (Section IV-B1) while *DTSS-FP* uses frequent-directions (*FP*) (Section IV-B2). If not specified, both of them will use the final sketch of time series to perform clustering. But in Section V-C we will show experiment result on early clustering using intermediate sketch by *DTSS-FP*.

A. Experiment Setup

Datasets. Here we highlight an industrial time series dataset 'TurbineTrips' collected from the power-generation turbines sensors. This dataset includes **over five million timestamp samples** from more than 600 turbines across 30 different site locations. Each time series corresponds to a turbine trip record that is an emergency shutdown of a power-generation turbine due to unexpected event. Each trip record has 10ms resolution for about 60 to 90 seconds before and after the trip. Conventionally, all these signals need to be scanned manually to identify different failure modes. The objective here is to see if our algorithm can successfully identify five different failure modes in these trips. This dataset depicts real-world scenarios in one of the target applications, and the time series are already reviewed and labelled by experts from turbine operation team. Therefore it can provide an accurate industrial benchmark for our performance evaluation.

Besides, we also include six public benchmark datasets, of which statistics are summarized in Table II. The balance ratio is defined as the ratio between the number of time series of the smallest and largest cluster. The datasets include three activity datasets: HAR dataset [17] was recorded with smartphones sensors for human activity recognition; Epilepsy dataset [4] was generated with healthy participants simulating the class activities; NATOPS dataset [17] reveals the various Naval Air Training and Operating Procedures Standardization motions that used to control plane movements. Besides, CK dataset [17] is for action unit and emotion-specified facial expression learning. RemSensor dataset [14] is a remote sensing dataset for geoscience learning. TEP dataset [40] is from a realistic simulation of industrial processes that has been widely used in process control studies.

Baselines. We compare our proposed methods against the following baselines for time series clustering, which are also discussed in Section II:

1) *DTW* [25] uses the *dynamic time warping measure* to obtain the distance matrix.
2) *softDTW* [8] is a differentiable smoothed distance measure extended from *DTW*.
3) *k-shape* [28] is an iterative clustering algorithm with normalized cross correlation (*NCC*) as distance measure.

TABLE II
DATASET STATISTICS AND DESCRIPTION.

Dataset	source	# timestamps	# time series	# variables	min/max length	# clusters	balance ratio	description
HAR	[17]	76,800	600	9	128/128	6	1	activity sensor
Epilepsy	[4]	56,650	275	3	206/206	4	0.81	activity sensor
NATOPS	[17]	18,360	360	24	51/51	6	1	activity sensor
CK	[17]	5,876	327	136	6/ 71	7	0.22	facial expression
RemSensor	[14]	61,901	1673	16	37/37	7	0.28	geo-sensor
TEP	[40]	450,000	900	52	300/300	3	1	industrial processes
TurbineTrips	business dataset	>5,000,000	650	5	5326/8000	5	0.003	turbine trips

4) *DeTSEC* [14] first builds a bidirectional *GRU* autoencoder with attention and gating mechanisms to produce low dimensional embeddings from the hidden state vectors. Then, it applies a clustering refinement procedure to stretch the embedding manifolds toward different clusters.
5) *TCN* [11] constructs an encoder-only architecture using temporal convolutional networks with triple loss and negative sampling to generate representation embeddings.
6) *IT-TSC* [36] is a deep-learning based time series clustering method that explores variable associations in different clusters.

The last three algorithms are all recently proposed deep learning based methods. The first three algorithms are conventional algorithms for measuring similarity between temporal sequences. Their resulting distance matrices, as well as the flatten sketches by our *DTSS-SVD*$_k$ and *DTSS-FP* are fed into *K-means* in python *sklearn* package to generate cluster labels.

Metrics. We use two popular metrics to evaluate the clustering performance: normalized mutual information (*NMI*) and adjusted rand index (*ARI*) [24]. *NMI* and *ARI* definition are as follows:

$$NMI = \frac{2 \times I(Y;C)}{H(Y) + H(C)}, \tag{5}$$

$$ARI = \frac{RI - Expected\_RI}{max(RI) - Expected\_RI}, \tag{6}$$

where Y are ground truth and C are predicted labels, $H(*)$ measures entropy and $I(*,*)$ measures mutual information, RI represents rand index. Both metrics range between 0 and 1 and reach 1 when the clustering partition completely matches the ground truth (up to permutation).

B. Experiments on Regular Clustering

Table III and IV show the performance comparison on time series clustering. We list the result by six baselines, as well as our two *DTSS* versions: *DTSS-SVD*$_k$ and *DTSS-FP*. For every dataset, we run each algorithm 15 times and document the result averages and standard deviations. At the end, we also compare the average result across all datasets and measure the p-value of our two methods against each baseline.

We have the following observations:

- K-shape has the worst performance among all algorithms, especially on CK, RemSensor, TEP and TurbineTrips. It may be because that k-shape suffers from high dimensionality. Moreover, k-shape has trouble in clustering data where data is noisy or clusters are of varying sizes and density [2].
- Generally speaking, deep learning based methods, including DeTSEC, TCN, IT-TSC and our two *DTSS*, outperform the three conventional baselines. It shows that deep neural networks are more capable to capture the nonlinear features in both temporal and spatio dimensions of multivariate time series in unsupervised setting.
- Overall, the average result of our proposed *DTSS* is over 42% better in NMI (56% better in ARI) than the average performance of all baselines. Particularly *DTSS* outperforms the second best algorithm over 11% in NMI and 17% in ARI. Besides, both of *DTSS-SVD*$_k$ and *DTSS-FP* obtain very small p-values across all the baselines. It proves that by projecting time series windows into embeddings and learning their sketches, we can obtain more informative representation of multivariate time series in the context of time series clustering.
- *DTSS-SVD*$_k$ performs better than *DTSS-FP* especially on the datasets with short time series (such as RemSensor and CK). But both of them reach very similar performance on those with long time series (such as TEP and TurbineTrips). This may be because frequent-directions approximates top components better for longer sequence than for shorter ones. Although *DTSS-SVD*$_k$ provides slightly better result, one advantage of *DTSS-FP* is that it can provide early clustering result even before seeing the whole time series. We will show this in the next subsection.

C. Experiments on Early Clustering

In this subsection, we test the performance of our *DTSS-FP* in early time series clustering. The experiments are run on our industrial TurbineTrips dataset because it has long time series.

We uniformly split all the 650 time series into five batches: each batch contains 130 time series. The samples in each time series come as a continuous streams in timestamp order. But the arrival time of batch are different from each other. As shown in Figure 7, the five batches' arrival time are at time index 0, 4000, 8000, 12000 and 16000 respectively. Our *DTSS-FP*'s early clustering starts at index 1000 when the model collects the first 1000 timestamps from batch1. At each timestamp the model is trained for 10 epochs, and outputs labels of all the currently seen time series (even just partially) till index 24000 when all the samples arrived. We evaluate the

TABLE III
NMI SCORE OF TIME SERIES CLUSTERING (AVERAGE AND STANDARD DEVIATION). OUR *DTSS* USING *SVD$_k$* OBTAINS THE BEST RESULT ON ALL THE DATASETS, WHILE THE FREQUENT-DIRECTIONS VERSION (*DTSS-FP*) RANKS THE SECOND-BEST.

	k-shape	softDTW	DTW	DeTSEC	TCN	IT-TSC	**DTSS-SVD$_k$**	**DTSS-FP**
HAR	.393 (.033)	.732 (.033)	.754 (.030)	.622 (.024)	.769 (.069)	.806 (.081)	**.907** (.063)	.902 (.071)
Epilepsy	.249 (.020)	.347 (.032)	.351 (.064)	.252 (.033)	.342 (.099)	.406 (.076)	**.491** (.063)	.489 (.061)
NATOPS	.111 (.022)	.643 (.001)	.683 (.001)	.621 (.028)	.481 (.091)	.709 (.003)	**.772** (.021)	.732 (.030)
CK	.173 (.053)	.481 (.013)	.460 (.016)	.613 (.012)	.311 (.057)	.700 (.020)	**.747** (.025)	.712 (.033)
TEP	.160 (.057)	.750 (.005)	.781 (.005)	.851 (.112)	.969 (.005)	.902 (.142)	**.993** (.005)	.991 (.006)
RemSensor	.184 (.033)	.432 (.027)	.523 (.024)	.601 (.011)	.573 (.031)	.581 (.021)	**.655** (.031)	.610 (.047)
TurbineTrips	.380 (.029)	.428 (.035)	.453 (.033)	.793 (.046)	.843 (.051)	.801 (.048)	**.904** (.043)	.888 (.048)
average	.236	.546	.574	.613	.613	.701	**.781**	.761
p-value (*DTSS-SVD$_k$*)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01		
p-value (*DTSS-FP*)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01		

TABLE IV
ARI SCORE OF TIME SERIES CLUSTERING (AVERAGE AND STANDARD DEVIATION). OUR *DTSS* USING *SVD$_k$* OBTAINS THE BEST RESULT ON ALL THE DATASETS, WHILE THE FREQUENT-DIRECTIONS VERSION (*DTSS-FP*) RANKS THE SECOND-BEST.

	k-shape	softDTW	DTW	DeTSEC	TCN	IT-TSC	**DTSS-SVD$_k$**	**DTSS-FP**
HAR	.252 (.030)	.573 (.015)	.618 (.040)	.520 (.032)	.661 (.106)	.680 (.134)	**.858** (.094)	.850 (.102)
Epilepsy	.106 (.010)	.214 (.032)	.218 (.047)	.164 (.030)	.244 (.099)	.291 (.070)	**.379** (.035)	.372 (.039)
NATOPS	.049 (.014)	.475 (.002)	.532 (.002)	.450 (.035)	.340 (.087)	.558 (.053)	**.632** (.042)	.587 (.057)
CK	.124 (.043)	.469 (.016)	.449 (.020)	.542 (.080)	.222 (.065)	.663 (.037)	**.775** (.039)	.704 (.043)
TEP	.112 (.065)	.757 (.004)	.789 (.004)	.805 (.168)	.977 (.008)	.871 (.213)	**.997** (.005)	.991 (.004)
RemSensor	.117 (.030)	.313 (.026)	.441 (.028)	.493 (.018)	.458 (.049)	.471 (.022)	**.564** (.029)	.512 (.041)
TurbineTrips	.323 (.055)	.412 (.062)	.432 (.058)	.831 (.082)	.891 (.079)	.851 (.073)	**.955** (.082)	.934 (.076)
average	.155	.459	.497	.542	.542	.626	**.737**	.707
p-value (*DTSS-SVD$_k$*)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01		
p-value (*DTSS-FP*)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01		

NMI score at an interval of 1000 indexes, and plot the result with blue curve in Figure 7.

We have the following observations:

- Our early clustering reach high performance even in the middle of data collection, i.e. NMI > 0.85 at index 12000 when half of the data are still unseen. It proves that our *DTSS-FP* is both efficient and effective in early time series clustering.
- There are three obvious score boost-up (shown as red circles in the NMI curve). The first one appears at index 6000, which is the same time when the significant temporal signatures of turbine trips come into sight (marked with red rectangle in batch1). Similarly, when the informative part of batch2 and batch3 arrive, the model refines itself with more useful information in data so the NMI score is raised.
- Last but not least, we can see that our *DTSS-FP* reaches its top performance very fast, without waiting for the last two batches' arrival.

D. Sensitivity Study

Fig 8 shows the performance robustness of our *DTSS* method under four key parameter tuning, with the red surrounded cells indicate the performance with our default parameter setting (listed in Section V-F). We can observe that even with sub-optimal parameter configuration, our *DTSS* still performs reasonably and stably well. However, it is worth noting that 1) dimensions of embedding has to be large enough (e.g. $\geq$ 100 in this example) to capture complete and informative patterns; 2) input windows and TCN level shouldn't be too large given limited amount of parameters; 3) the rank of sketch cannot be too small nor too large, because too small k may lead to missing important components, and on the other hand too large k may bring in noise with ineffectual dimensions.

E. Ablation Study

Our *DTSS* framework includes several different but essential parts. To understand each part's contribution, we conduct an ablation study and evaluate the performance on TurbineTrips dataset (because to show the comparison between with or without singular value normalization we need time series with varying lengths). Figure 9 shows the NMI scores by seven versions of our *DTSS*, among which five of them are ablated versions by removing certain part at a time, and the rest two are full versions of *DTSS*. When ablating any part of neural network, we tried to maintain the number of parameters (i.e. weights/bias) in the whole model to be the same.

In short, the two full versions achieve the best performance, which shows that all parts are integrated systematically and each plays an indispensable role. Clearly, the version without TCN performs the worst, which shows that module 1 especially TCN is crucial for learning temporal-spatio features from data. Besides, the second worst result comes from the version of which embedding sketching part is replaced by a random sampling on embeddings (that means, we perform random sampling on the columns of A to obtain B). It shows that both *SVD$_k$* and *frequent-directions* are effective in learning sketches than simple random sampling. Furthermore,

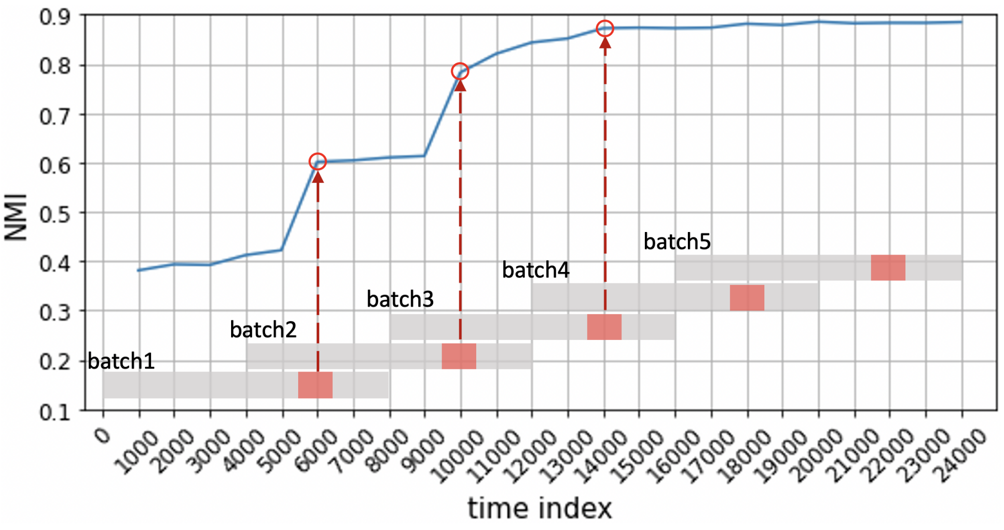
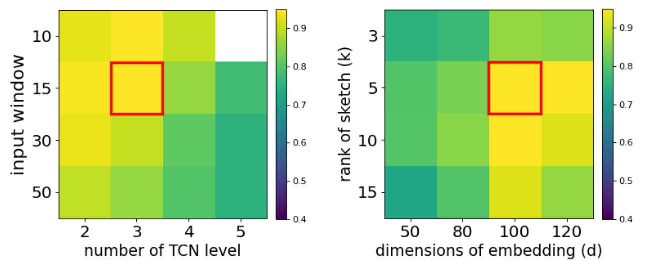


Fig. 7. Early time series clustering result by *DTSS-FP* on TurbineTrips dataset.



(a) (b)

Fig. 8. Parameter sensitivity analysis of *DTSS* on *TEP* dataset. The red surrounded cells indicate the performance with our default parameter setting. White cell denotes that the corresponding setting cannot be applied.

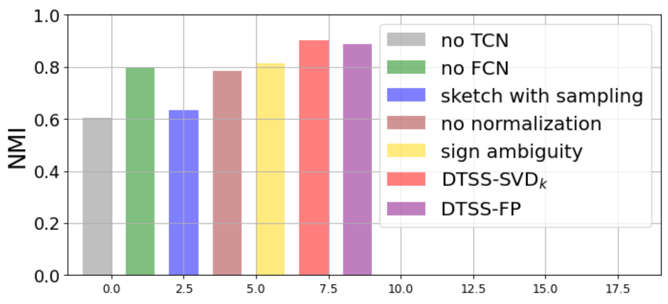


Fig. 9. Ablation study of our model.

we can see normalization and removing sign ambiguity all play important roles in gaining higher clustering accuracy, since versions without either of them return lower NMI score.

F. Implementation Details

Our model is implemented in *python*. Particully, module 1 is implemented in *pytorch*, and K-means we used in our model is from *sklearn* package. Our default parameter configurations are as follows: 1) For datasets with short time series (< 10 timestamps), window size $\ell = 3$ and kernel size is set to 2. TCN has 2 levels with 100 output channels on each level. The fully connected network has three layers. The first two layers have output channel size 80 and 100 (80 is the bottleneck dimensions d). The last layer projects to the original feature space. The rank of embeddings k is set to 5. 2) For the other datasets, we set window size $\ell = 15$ and kernel size is set to 5. TCN has 3 levels with 100 output channels on each level. The fully connected network consist of three layers. The first two layers have output channel size 100 and 120, where 100 is the bottleneck dimensions d. The last layer projects to the original feature space. The rank of embeddings k is set to 5.

We set a training batch size as 50 and use *adam* optimizer with learning rate $1e$-4 to train our model.

VI. Conclusion

In this paper we introduce *DTSS*, a deep learning and embedding sketching based approach to cluster industrial multivariate time series with varying lengths. It is designed as a representation learning method that takes input from time series in a sliding window manner. From each window it extracts nonlinear embedding that is used to update the time series sketch as the window slides from the beginning to the end. With such design, our model is capable of considering and synthesizing the complete set of patterns along all windows, and providing a balanced and concise representation that contains both global and salient local characteristics. Furthermore, our *DTSS* can perform early clustering without seeing the whole time series. Experiments show that our model has superior performance over popular baselines in time series clustering on public benchmark and real industrial datasets.

References

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information systems*, 53:16–38, 2015.

[2] Ali Alqahtani, Mohammed Ali, Xianghua Xie, and Mark W Jones. Deep time-series clustering: A review. *Electronics*, 10(23):3001, 2021.

[3] Mourtadha Badiane and Pádraig Cunningham. An empirical evaluation of kernels for time series. *Artificial Intelligence Review*, 55(3):1803–1820, 2022.

[4] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

[5] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[6] Rasmus Bro, Evrim Acar, and Tamara G Kolda. Resolving the sign ambiguity in the singular value decomposition. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 22(2):135–140, 2008.

[7] Paolo Buono, Aleks Aris, Catherine Plaisant, Amir Khella, and Ben Shneiderman. Interactive pattern search in time series. *Visualization and Data Analysis 2005*, 5669:175–186, 2005.

[8] Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. *arXiv preprint arXiv:1703.01541*, 2017.

[9] Erdem Doğan. Lstm training set analysis and clustering model development for short-term traffic flow prediction. *Neural Computing and Applications*, 33(17):11175–11188, 2021.

[10] Roberta Falcone, Laura Anderlucci, and Angela Montanari. Matrix sketching for supervised classification with imbalanced classes. *Data Mining and Knowledge Discovery*, 36(1):174–208, 2022.

[11] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *arXiv preprint arXiv:1901.10738*, 2019.

[12] Takanori Fujiwara, Jianping Kelvin Li, Misbah Mubarak, Caitlin Ross, Christopher D Carothers, Robert B Ross, and Kwan-Liu Ma. A visual analytics system for optimizing the performance of large-scale networks in supercomputing systems. *Visual Informatics*, 2(1):98–110, 2018.

[13] Chonghui Guo, Hongfeng Jia, and Na Zhang. Time series clustering based on ica for stock data analysis. In *2008 4th international conference on wireless communications, networking and mobile computing*, pages 1–4. IEEE, 2008.

[14] Dino Ienco and Roberto Interdonato. Deep multivariate time series embedding clustering via attentive-gated autoencoder. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 318–329. Springer, 2020.

[15] Gareth J Janacek, Anthony J Bagnall, and Michael Powell. A likelihood ratio distance measure for the similarity between the fourier transform of time series. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 737–743. Springer, 2005.

[16] Dong Hyun Jeong, Alireza Darvish, Kayvan Najarian, Jing Yang, and William Ribarsky. Interactive visual analysis of time-series microarray data. *The Visual Computer*, 24(12):1053–1066, 2008.

[17] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 2019.

[18] Xuan-Hien Le, Hung Viet Ho, Giha Lee, and Sungho Jung. Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, 11(7):1387, 2019.

[19] Daoyuan Li, Tegawendé François D Assise Bissyande, Jacques Klein, and Yves Le Traon. Time series classification with discrete wavelet transformed data: Insights from an empirical study. In *The 28th international conference on software engineering and knowledge engineering (SEKE 2016)*, 2016.

[20] Kim-Hung Li. Reservoir-sampling algorithms of time complexity o (n (1+ log (n/n))). *ACM Transactions on Mathematical Software (TOMS)*, 20(4):481–493, 1994.

[21] Li Li and Hirokazu Kameoka. Deep clustering with gated convolutional networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 16–20. IEEE, 2018.

[22] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588, 2013.

[23] Qingjian Lin, Ruiqing Yin, Ming Li, Hervé Bredin, and Claude Barras. Lstm based similarity measurement with spectral clustering for speaker diarization. *arXiv preprint arXiv:1907.10393*, 2019.

[24] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[25] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, 2007.

[26] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[27] John Paparrizos and Michael J Franklin. Grail: efficient time-series representation learning. *Proceedings of the VLDB Endowment*, 12(11):1762–1777, 2019.

[28] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *ACM International Conference on Management of Data*, 2015.

[29] Ivan Popivanov and Renee J Miller. Similarity search over time-series data using wavelets. In *Proceedings 18th international conference on data engineering*, pages 212–221. IEEE, 2002.

[30] Guillaume Richard, Benoît Grossin, Guillaume Germaine, Georges Hébrail, and Anne de Moliner. Autoencoder-based time series clustering with energy applications. *arXiv preprint arXiv:2002.03624*, 2020.

[31] Steven Smith. *Digital signal processing: a practical guide for engineers and scientists*. Elsevier, 2013.

[32] Aurea Soriano-Vargas, Bruno C Vani, Milton H Shimabukuro, Joao FG Monico, Maria Cristina F Oliveira, and Bernd Hamann. Visual analytics of time-varying multivariate ionospheric scintillation data. *Computers & Graphics*, 68:96–107, 2017.

[33] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[34] James S Walker, Mark W Jones, Robert S Laramee, Owen R Bidder, Hannah J Williams, Rebecca Scott, Emily LC Shepard, and Rory P Wilson. Timeclassifier: a visual analytic system for the classification of multi-dimensional time series data. *The Visual Computer*, 31(6):1067–1078, 2015.

[35] Li Wei and Eamonn Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753, 2006.

[36] Chenxiao Xu, Hao Huang, and Shinjae Yoo. A deep neural network for multivariate time series clustering with result interpretation. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[37] Rui Yan, Jiaqiang Liao, Jie Yang, Wei Sun, Mingyue Nong, and Feipeng Li. Multi-hour and multi-site air quality index forecasting in beijing using cnn, lstm, cnn-lstm, and spatiotemporal clustering. *Expert Systems with Applications*, 169:114513, 2021.

[38] Kiyoung Yang and Cyrus Shahabi. A pca-based similarity measure for multivariate time series. In *Proceedings of the 2nd ACM international workshop on Multimedia databases*, pages 65–74, 2004.

[39] Jiaqi Ye, Chengwei Xiao, Rui Máximo Esteves, and Chunming Rong. Time series similarity evaluation based on spearman's correlation coefficients and distance measures. In *Second International Conference on Cloud Computing and Big Data in Asia*, pages 319–331. Springer, 2015.

[40] Shen Yin, Steven X Ding, Adel Haghani, Haiyang Hao, and Ping Zhang. A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process. *Journal of process control*, 22, 2012.

[41] Manzil Zaheer, Amr Ahmed, and Alexander J Smola. Latent lstm allocation: Joint clustering and non-linear dynamic modeling of sequence data. In *International Conference on Machine Learning*, pages 3967–3976. PMLR, 2017.

[42] Geng Zhang, Adam R Singer, and Nickolas Vlahopoulos. Temporal clustering network for self-diagnosing faults from vibration measurements. *arXiv preprint arXiv:2006.09505*, 2020.

[43] Mengjiao Zhang and Shusen Wang. Matrix sketching for secure collaborative machine learning. In *International Conference on Machine Learning*, pages 12589–12599. PMLR, 2021.