Clustering Time Series Data with Gaussian Mixture Embeddings in a Graph Autoencoder Framework

Amirabbas Afzali[1⋆[0009−0006−1965−5367]], Hesam Hosseini[1⋆[0009−0003−6234−7675]], Mohammadamin Mirzai[1[0009−0001−5028−5281]], and Arash Amini[2[0000−0002−7082−9581]]

[1] Independent Researcher,
{amir8afzali, hesam8hosseini, moh.amin.mirzaii}@gmail.com
[2] Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran,
aamini@sharif.edu,

Abstract. Time series data analysis is prevalent across various domains, including finance, healthcare, and environmental monitoring. Traditional time series clustering methods often struggle to capture the complex temporal dependencies inherent in such data. In this paper, we propose the Variational Mixture Graph Autoencoder (VMGAE), a graph-based approach for time series clustering that leverages the structural advantages of graphs to capture enriched data relationships and produces Gaussian mixture embeddings for improved separability. Comparisons with baseline methods are included with experimental results [3], demonstrating that our method significantly outperforms state-of-the-art time-series clustering techniques. We further validate our method on real-world financial data, highlighting its practical applications in finance. By uncovering community structures in stock markets, our method provides deeper insights into stock relationships, benefiting market prediction, portfolio optimization, and risk management.

Keywords: Graph Neural Network, Gaussian mixture embeddings, Time series clustering, Finance.

1 Introduction

Time series is commonly referred to a sequence of data points collected or recorded at successive time instances, usually at uniform intervals. For instance, in finance, time series data might include daily closing prices of a stock over a year [37]. In healthcare, it could be the EEG signal of a person's brain in a specific time interval [39], and in environmental monitoring, it might involve hourly temperature readings [56].

Numerous studies have been conducted on time series analysis, encompassing various tasks such as forecasting [43], classification [17], clustering [1], anomaly detection [38], visualization [10], pattern recognition [27], and trend analysis [31].

Time series clustering is a powerful method for grouping similar time series data points based on their characteristics, especially when there is no prior knowledge of the

⋆ Equal contribution. Author ordering is determined by coin flip.
[3] Code available at: https://github.com/Sam-the-first/VMGAE

data structure [26]. It has diverse applications, such as stock market forecasting, where it is used for feature extraction to predict stock movements, helping investors anticipate market behavior and enhance model predictions [2]. In portfolio optimization, clustering identifies stocks with similar traits, fostering diversification and reducing risks. Additionally, it supports risk management by predicting market volatility using algorithms like Kernel K-Means and Gaussian Mixture Models [4] and contributes to fraud detection by flagging anomalies that deviate from typical cluster patterns [5].

Despite its practical significance, unsupervised time series clustering faces notable challenges. Time series data often vary significantly in their critical properties, features, temporal scales, and dimensionality across different domains. Real-world data further complicate this process by introducing issues such as temporal gaps and high-frequency noise [15]. To address these challenges, researchers have developed methods focusing on three main aspects: 1) time series similarity measures [25, 41], 2) discriminative representation learning [19, 29, 53], and 3) clustering mechanisms [23, 34]. These advancements aim to enhance the reliability and applicability of time series clustering in complex, real-world scenarios.

In this paper, we leverage all these aspects by constructing a graph from time series data using dynamic time wrapping (DTW) [36] to capture the relationships between individual time series. By exploiting a specialized graph autoencoder, we can also learn how to embed each node properly. This embedding not only represents the unique features of each data point but also captures shared features from nodes similar to the data point. To the best of our knowledge, this is the first work that employs graph autoencoder architecture for time series clustering.

The novel contributions of this work can be summarized as follows:

- We propose a new framework for time series clustering. This approach uses a graphical structure to capture more detailed information about data relationships. Turning a time series dataset into graphs can effectively capture both temporal and relational dependencies.
- We introduce a specialized graph autoencoder, named Variational Mixture Graph Autoencoder (VMGAE), that generates a Mixture of Gaussian (MoG) embeddings. This allows the separation of data through a Gaussian Mixture Model (GMM) in the embedding space, enhancing clustering performance and providing a more precise representation of time series data.
- We conducted a comprehensive comparison of our method against strong baselines, demonstrating significant improvements over the state-of-the-art. Additionally, we evaluated the practicality of our approach using real-world financial data.

2 Related Work

Time series data clustering has been a significant area of research for decades, leading to various algorithms. Classical clustering algorithms like k-means and spectral clustering can be executed on raw time series data, while some methods use modified versions of classical methods. K-shape [34], assigns data to clusters based on their distance to centroids and updates the centroids like k-means, but instead uses cross-correlation for

distance measurement. KSC([49]) uses k-means for clustering by adopting a pairwise scaling distance measure and computing the spectral norm of a matrix for centroid computation.

Another approach is to use shapelets to extract discriminative features from time series data, as demonstrated in [44], [53], [23], and [52]. The main challenge in these methods is identifying suitable shapelets for the shapelet transform process, which extracts meaningful features from raw data to perform clustering. R-clustering method [19] employs random convolutional kernels for feature extraction, which are then used for clustering. Additionally, [41] implements a hierarchical clustering algorithm that uses Granger causality [8] as the distance measure, fusing pairs of data to create new time series and continuing the clustering process. STCN [28] uses an RNN-based model to forecast time series data, employs pseudo labels for its classifier, and utilizes the learned features for clustering. Since our method leverages both the autoencoder architecture and a graph-based approach for clustering time series data, we will review autoencoder-based methods and graph-based techniques separately.

2.1 Autoencoder-based methods

Autoencoders have demonstrated empirical success in clustering by using their learned latent features as data representations. For instance, DEC [48] adds a KL-divergence term between two distributions to its loss function, alongside the reconstruction error loss, to make the latent space more suitable for clustering. DTC [32] introduces a new form of distribution for the KL-divergence term, applying this method to trajectory clustering. Another method, DCEC [12], incorporates a convolutional neural network as its autoencoder within the DEC method for image clustering. VaDE [18] adds a KL-divergence term between the Mixture-of-Gaussians prior and the posterior to the reconstruction loss. This is done using the embeddings of data points in the latent space of a variational autoencoder and a prior GMM distribution. In the domain of time series clustering, DTCR [29] trains an autoencoder model with the addition of k-means loss on the latent space and employs fake data generation and a discriminator to classify real and fake data, enhancing the encoder's capabilities. Also, TMRC [22] proposes a representation learning method called temporal multi-features representation learning (TMRL) to capture various temporal patterns embedded in time-series data and ensembles these features for time-series clustering.

2.2 Graph-based methods

Graphs have significantly enhanced the capabilities of deep learning methods in various tasks. Variational Graph Autoencoder (VGAE) [21] utilizes GCN [20] for link prediction and node classification tasks. Specifically, graphs have also found significant applications in the time series domain. Recent works such as [40], [3], and [50] use graph-based methods for time series forecasting, while [51] and [47] apply them for classification. Additionally, [54], [7], and [14] utilize graph-based models for anomaly detection in time series data. Graphs have also been employed for time series data clustering [25].

One of the critical challenges in graph-based methods for the time series domain is constructing the adjacency matrix. Several methods address this issue by introducing metrics to compute the similarity or distance between two time series samples. The Granger causality method [8] leverages the causal effect of a pattern in one time series sample on another to measure similarity between samples. The Dynamic Time Warping (DTW) method [36] minimizes the effects of shifting and distortion in time by allowing the elastic transformation of time series to compute the distance between two samples. There are many extensions of the DTW method, such as ACDTW [24], which uses a penalty function to reduce many-to-one and one-to-many matching, and shapeDTW [55], which represents each temporal point by a shape descriptor that encodes structural information of local subsequences around that point and uses DTW to align two sequences of descriptors.

The similarity between two time series can be used directly as the edge representation, but the distance needs to be processed for use in the adjacency matrix. One method is to apply a threshold on distances to predict whether an edge exists between two nodes in a binary graph [25].

3 Problem Definition and Framework

3.1 Notation

In the following sections, we denote the training dataset as $\mathcal{D} = \{d_1, \ldots, d_n\}$, where d_i represents the i-th time series, and n is the size of the training dataset. The length of the i-th time series is denoted by l_i. Each time series d_i belongs to a cluster c_i, where $c_i \in \{1, \ldots, K\}$, and K is the number of clusters. Furthermore, we define $\mathbf{c} = [c_1, \ldots, c_n]^\top$ as the vector of the corresponding clusters for all time series in $\mathcal{D}$.

Additionally, a graph is represented as $\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$, where $\mathbf{V} = \{v_i\}_{i=1}^n$ is the set of nodes, and each edge $e_{i,j} = \langle v_i, v_j \rangle \in \mathbf{E}$ represents a connection between nodes v_i and v_j. The structure of the graph is described by an adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{i,j} = 1$ if $e_{i,j} \in \mathbf{E}$, and $\mathbf{A}_{i,j} = 0$ otherwise. The feature vector $\mathbf{x}_i \in \mathbf{X}$ corresponds to the content attributes of node v_i, which, in our context, is equivalent to the time series d_i.

Given the graph $\mathbf{G}$, our objective is to map each node $v_i \in \mathbf{V}$ to a low-dimensional vector $\mathbf{z}_i \in \mathbb{R}^h$. This mapping is formally defined as:

$$f : (\mathbf{A}, \mathbf{X}) \mapsto \mathbf{Z},$$

where the i-th row of the matrix $\mathbf{Z} \in \mathbb{R}^{n \times h}$ is denoted by $\mathbf{z}_i^\top$. Here, h is the dimensionality of the embedding space. The matrix $\mathbf{Z}$, which contains these embeddings, is designed to preserve both the structural information of the graph, captured by $\mathbf{A}$ and the content features represented by $\mathbf{X}$.

3.2 Overall Framework

Our goal is to represent a time series dataset as a graph $\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$, where each node corresponds to a time series. We aim to learn a robust embedding for each node in

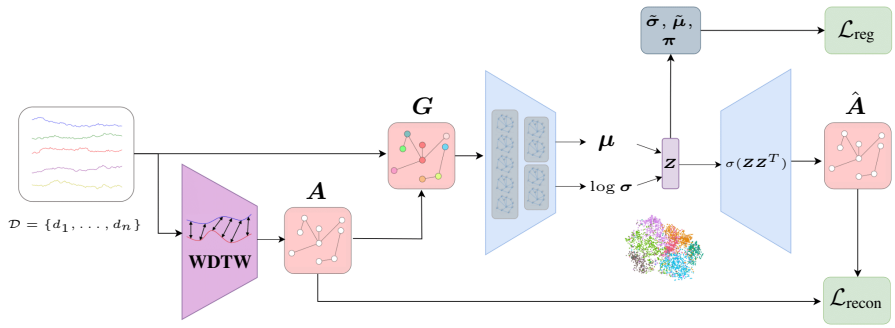

Fig. 1: The general architecture of the Variational Mixture Graph Autoencoder (VMGAE). The dataset $\mathcal{D}$ consists of multiple time series data, and Weighted Dynamic Time Warping (WDTW) is used to compute distances that form the adjacency matrix $\boldsymbol{A}$, representing connections in the graph $\boldsymbol{G}$. The mean $\boldsymbol{\mu}$ and log standard deviation $\log \boldsymbol{\sigma}$ are computed for the variational latent space, creating node embeddings $\boldsymbol{Z}$. These embeddings undergo transformation to reconstruct the adjacency matrix $\hat{\boldsymbol{A}}$, with the reconstruction loss $\mathcal{L}_{\text{recon}}$ enforcing fidelity to $\boldsymbol{A}$. The regularization loss $\mathcal{L}_{\text{reg}}$ applies to the mixture model parameters $\tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\mu}}, \boldsymbol{\pi}$, enhancing the latent space structure.

this graph to perform clustering. To achieve this, we first construct the graph. Next, we apply an unsupervised graph representation learning approach within an autoencoder framework, enhanced for clustering, to process the entire graph and learn effective node embeddings $\mathbf{Z}$.

Graph Construction. Each time series is represented as a node in the graph construction phase. The distance matrix $\mathbf{S}$, calculated using the Dynamic Time Warping (DTW) method, captures the alignment between time series of varying lengths. We then apply our novel transformation to convert these distances into similarity scores, which determine the graph's structure $\mathbf{A}$. This approach ensures that the graph reflects the true underlying relationships in the data, preserving important temporal patterns.

Learning Representation via a Graph Structure. After constructing the graph, we leverage a specialized autoencoder framework to learn embeddings for each node. This unsupervised method compresses the graph's information into a lower-dimensional representation $\mathbf{Z}$. Our approach ensures that the resulting embeddings are informative, generalizable, and discriminative, making them particularly effective for clustering tasks. Figure 1 provides a comprehensive overview of our method.

4 Methodology

This section describes the steps taken to represent a time series dataset as a graph and how we use this graph structure to learn meaningful embeddings for clustering.

4.1 Graph Construction

For graph construction, we use a variant of DTW called Weighted Dynamic Time Warping (WDTW) and a constraint to limit the window size of the wrapping path. The distance between two sequences $X = (x_1, x_2, \ldots, x_N)$ and $Y = (y_1, y_2, \ldots, y_M)$ with a weight funtion w and a window size W is computed as follows:

$$WDTW(X,Y) = \min_{\pi} \left(\sum_{(i,j)\in\pi} w[|i-j|] \cdot d_{\text{inner}}(x_i, y_j) \right), \tag{1}$$

subject to the constraint:

$$|i-j| \leq W, \tag{2}$$

where π is a warping path that aligns the sequences X and Y, $d_{\text{inner}}(x_i, y_j)$ is the distance between elements x_i and y_j. This could be any customized distance. For simplicity, we use Euclidean distance. $w[|i-j|]$ is a weight function that depends on the absolute difference between indices i and j, and W is the window size that limits the maximum allowable shift between indices.

The weight function $w[n]$ should be a monotonic function of n, as it penalizes alignments where the indices are farther apart, favoring closer alignments. For simplicity, we set $w[n] = \gamma \cdot n$, where γ is a positive hyperparameter.

Given the training dataset $\mathcal{D} = \{d_1, \ldots, d_n\}$, we construct a distance matrix S, where S_{ij} represents $WDTW(d_i, d_j)$ with fixed parameters γ and W. Next, we propose a novel transformation approach to convert the distance matrix S into a similarity matrix. By fixing the density rate $\alpha = \frac{\#\{A_{ij}=1\}}{n^2}$, we compute a threshold δ to construct an adjacency matrix A, where $A_{ij} = 1$ if $S_{ij} < \delta$ and $A_{ij} = 0$ otherwise. The key difference compared to previous work [25] is that, instead of fixing δ, we fix α and use it to compute the corresponding δ for each dataset. This is important because the optimal threshold δ may vary across datasets, while the optimal α is much more stable. Figure 2 presents a sample graph constructed using this method. While the current representation demonstrates good separation, further refinement can be achieved with the use of *VMGAE*.

4.2 Learning Representation via a Graph Structure

Graph Convolutional Autoencoder. In our unsupervised setting, we utilize a graph convolutional autoencoder architecture to embed a graph $\mathbf{G} = \{\mathbf{V}, \mathbf{E}, \mathbf{X}\}$ into a low-dimensional space. Specifically, we derive an embedding $\mathbf{z_i} \in \mathbf{Z}$ for the i-th node of the graph. This approach presents two key challenges: 1) How can both the graph structure $\mathbf{A}$ and node features $\mathbf{X}$ be effectively integrated within the encoder? 2) What specific information should the decoder reconstruct?

Graph Convolutional Layer. To effectively capture both the structural information $\mathbf{A}$ and node features $\mathbf{X}$ in a unified framework, we employ graph convolutional network (GCN) [20]. Graph convolutional operator extends the convolution operation to graph data in the spectral domain and applies a layer-wise transformation using a specialized

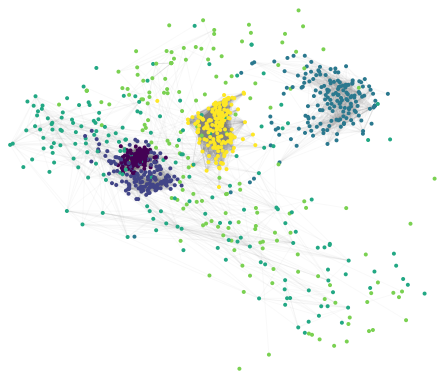

Fig. 2: Graph visualizations of the Symbols dataset, illustrating effective data separation. Different colors correspond to distinct labels.

convolutional function. Each layer of the graph convolutional network can be expressed as follows:

$$\mathbf{Z}^{(l+1)} = \phi\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{(l)}\mathbf{W}^{(l)}\right), \tag{3}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. Here, $\mathbf{I}$ represents the identity matrix and ϕ is an activation function like ReLU $\phi(x) = \max(0, x)$. Also, $\mathbf{Z}^{(l)}$ denotes the input at the l-th layer, and $\mathbf{Z}^{(l+1)}$ is the output after the convolution operation. Initially, $\mathbf{Z}^0 = \mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{n\times m}$ represents the input features of the graph with n nodes and m features. The matrix $\mathbf{W}^{(l)}$ contains the parameters to be learned. Additionally, in this work, we denote each convolutional layer with activation function ϕ as $f_\phi(\mathbf{Z}^{(l)}, \mathbf{A} \mid \mathbf{W}^{(l)})$.

In addition to this convolutional layer, several other variants suitable for node-level tasks have been proposed [6, 9, 13, 45]. In Appendix F.2, we compare the effects of different convolutional layers on the performance of our method.

Encoder Model $\mathcal{G}(\mathbf{X}, \mathbf{A})$. The encoder of *VMGAE* is defined by an inference model:

$$q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod_{i=1}^{n} q(\mathbf{z}_i|\mathbf{X}, \mathbf{A}), \tag{4}$$

$$q(\mathbf{z}_i|\mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)). \tag{5}$$

Here, $\boldsymbol{\mu}_i$ and $\log \boldsymbol{\sigma}_i$ are constructed using a two-layer convolutional network, where the weights $\mathbf{W}^{(0)}$ in the first layer are shared:

$$\mathbf{Z}^{(1)} = f_{\text{Relu}}(\mathbf{X}, \mathbf{A}|\mathbf{W}^{(0)}), \tag{6}$$

$$\begin{cases} \boldsymbol{\mu} = f_{\text{linear}}(\mathbf{Z}^{(1)}, \mathbf{A}|\mathbf{W}_\mu{}^{(1)}), \\ \log \boldsymbol{\sigma} = f_{\text{linear}}(\mathbf{Z}^{(1)}, \mathbf{A}|\mathbf{W}_\sigma{}^{(1)}). \end{cases} \tag{7}$$

The encoder model $\mathcal{G}(\mathbf{X}, \mathbf{A})$ encodes both graph structure and node features into a latent representation $\mathbf{Z} = q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$. According to the reparameterization trick, $\mathbf{z}_i$ is obtained by:

$$\mathbf{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \circ \boldsymbol{\epsilon}, \tag{8}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$, $\circ$ is element-wise multiplication.

Decoder Model $\mathcal{D}(\mathbf{Z}, \mathbf{A})$. Decoder model is given by an inner product between latent variables:

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j), \tag{9}$$

and the conditional probability is usually modeled as:

$$p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j), \tag{10}$$

where $\sigma(\cdot)$ is the logistic sigmoid function.

Thus, the embedding $\mathbf{Z}$ and the reconstructed graph $\hat{\mathbf{A}}$ can be presented as follows:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^\top), \quad \text{here } \mathbf{Z} = q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \tag{9}$$

Learning Algorithm. In *VMGAE*, our objective is to maximize the log-likelihood of the data points, $\log p(\mathbf{A})$. Based on the decoder model The joint probability $p(\mathbf{A}, \mathbf{Z}, \mathbf{c})$ can be factorized as:

$$p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) = p(\mathbf{A} \mid \mathbf{Z}) p(\mathbf{Z} \mid \mathbf{c}) p(\mathbf{c}). \tag{11}$$

The log-likelihood can be expressed as:

$$\begin{aligned} \log p(\mathbf{A}) &= \log \int_{\mathbf{Z}} \sum_{\mathbf{c}} p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) \, d\mathbf{Z} \\ &\geq \mathbb{E}_{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})} \left[\log \frac{p(\mathbf{A}, \mathbf{Z}, \mathbf{c})}{q(\mathbf{Z}, \mathbf{c} \mid \mathbf{X}, \mathbf{A})} \right] \qquad (12) \\ &= \mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}). \qquad (13) \end{aligned}$$

The inequality in Equation 12 is derived from Jensen's inequality. Instead of maximizing the log-likelihood directly, we aim to maximize its Evidence Lower Bound (ELBO), and using the factorization in Equation 11, it can be rewritten as follows:

$$\begin{aligned} &\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}) = E_{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})}[\log p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) - \log q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})] \\ &= E_{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})} [\log p(\mathbf{A}|\mathbf{Z}) + \log p(\mathbf{Z}|\mathbf{c}) + \log p(\mathbf{c})] \\ &\quad - E_{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})} [\log q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) + \log q(\mathbf{c}|\mathbf{X}, \mathbf{A})] \,, \end{aligned} \tag{14}$$

where the last line is obtained under the assumption of a mean-field distribution for $q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})$.

Similar to the approach in [18], a mixture of Gaussian latent variables is used to learn the following distributions:

$$p(c_i) = \mathrm{Cat}(c_i \mid \boldsymbol{\pi}) \tag{15}$$

$$p(\mathbf{z_i} \mid c_i). = \mathcal{N}\left(\mathbf{z_i} \mid \tilde{\boldsymbol{\mu}}_{c_i}, \tilde{\boldsymbol{\sigma}}^2_{c_i}\mathbf{I}\right). \tag{16}$$

By assuming a mean-field distribution, the joint probability $p(\mathbf{c})$ and $p(\mathbf{Z} \mid \mathbf{c})$ can be factorized as:

$$p(\mathbf{c}) = \prod_{i=1}^{n} \mathrm{Cat}(c_i \mid \boldsymbol{\pi}), \tag{17}$$

$$p(\mathbf{Z} \mid \mathbf{c}) = \prod_{i=1}^{n} \mathcal{N}\left(\mathbf{z_i} \mid \tilde{\boldsymbol{\mu}}_{c_i}, \tilde{\boldsymbol{\sigma}}^2_{c_i}\mathbf{I}\right), \tag{18}$$

where $\boldsymbol{\pi}_k$ is the prior distribution of cluster k hence $\sum_k \boldsymbol{\pi}_k = 1$, $\mathrm{Cat}(\cdot \mid \boldsymbol{\pi})$ is the categorical distribution parametrized by $\boldsymbol{\pi}$. Also $\boldsymbol{\mu}_{c_i}$ Moreover, $\boldsymbol{\sigma}^2_{c_i}$ are the mean and the variance of the Gaussian distribution corresponding to cluster c_i, $\mathbf{I}$ is an identity matrix.

Using Monte Carlo estimation for the expected value calculation in full-batch mode and substituting the assumptions from Equations 17 and 18 into Equation 14, the objective function can be expressed as:

$$\begin{aligned}
&\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}) = \\
&\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{n}\left(\sum_{j=1}^{n} A_{ij}\log\hat{A}_{ij} + (1 - A_{ij})\log(1 - \hat{A}_{ij})\right)\right. \\
&- \frac{1}{2}\sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A})\sum_{h=1}^{H}\left(\log\tilde{\boldsymbol{\sigma}}^2_{c_i h} + \frac{\boldsymbol{\sigma}^2_h}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}} + \frac{\left(\boldsymbol{\mu}_h - \tilde{\boldsymbol{\mu}}_{c_i h}\right)^2}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}}\right) \\
&\left. + \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A})\log\frac{\boldsymbol{\pi}_{c_i}}{q(c_i \mid \mathbf{X}, \mathbf{A})} + \frac{1}{2}\sum_{h=1}^{H}\left(1 + \log\boldsymbol{\sigma}^2_h\right)\right].
\end{aligned} \tag{19}$$

The first term represents the standard reconstruction loss, while the second and third terms act as regularizers, encouraging the model to generate a Gaussian mixture embedding. A detailed derivation of Equation 19 is provided in Appendix A.

The next question is how to compute $q(c_i \mid \mathbf{X}, \mathbf{A})$. According to the derivations in Appendix B, the ELBO can be rewritten as:

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}) &= E_{q(\mathbf{Z}, \mathbf{c} \mid \mathbf{X}, \mathbf{A})}\left[\log\frac{p(\mathbf{A}, \mathbf{Z}, \mathbf{c})}{q(\mathbf{Z}, \mathbf{c} \mid \mathbf{X}, \mathbf{A})}\right] \\
&= E_{q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A})}\left[\log\frac{p(\mathbf{A} \mid \mathbf{Z})p(\mathbf{Z})}{q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A})}\right] \\
&- \sum_{i=1}^{n}\int_{\mathbf{z_i}} q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) D_{\mathrm{KL}}(q(c_i \mid \mathbf{X}, \mathbf{A}) \| p(c_i \mid \mathbf{z_i})) d\mathbf{z_i}.
\end{aligned} \tag{20}$$

In the equation above, the first term is independent of $\mathbf{c}$, and the second term is non-negative. Therefore, similar to the approach in [18], to maximize $\mathcal{L}_{\mathrm{ELBO}}(\mathbf{X}, \mathbf{A})$, we assume $D_{\mathrm{KL}}(q(c_i|\mathbf{X}, \mathbf{A}) \| p(c_i|\mathbf{z_i}))$ to be zero. Consequently, the following equation can be used to compute $q(c_i|\mathbf{X}, \mathbf{A})$ in *VMGAE*:

$$q(c_i \mid \mathbf{X}, \mathbf{A}) = p(c_i \mid \mathbf{z_i}) \equiv \frac{p(c_i)p(\mathbf{z_i} \mid c_i)}{\sum_{c'=1}^{K} p\left(c'\right) p\left(\mathbf{z_i} \mid c'\right)}. \tag{21}$$

While the learned distribution can be directly used for clustering, we have empirically found that refitting a GMM on the learned representation $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$ significantly improves clustering performance.

Additionally, in our experiments, we introduce a weight parameter λ for the second component of the loss function, allowing us to balance the contribution of each term in the final loss function:

$$\mathcal{L}_{\mathrm{VMGAE}}(\mathbf{X}, \mathbf{A}) = \mathcal{L}_{\text{recon}}(\mathbf{A}) + \lambda \cdot \mathcal{L}_{\text{reg}}(\mathbf{X}, \mathbf{A}), \tag{22}$$

where $\mathcal{L}_{\text{recon}}(\mathbf{A})$ represents the reconstruction loss on the adjacency matrix $\mathbf{A}$:

$$\mathcal{L}_{\text{recon}}(\mathbf{A}) = -\frac{1}{n^2} \sum_{i,j=1}^{n} \Big(A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log(1 - \hat{A}_{ij}) \Big), \tag{23}$$

and the regularizer term $\mathcal{L}_{\text{reg}}(\mathbf{X}, \mathbf{A})$ is defined as follows:

$$\begin{aligned} \mathcal{L}_{\text{reg}}(\mathbf{X}, \mathbf{A}) = \frac{1}{n} \sum_{i=1}^{n} \Bigg[\frac{1}{2} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \sum_{h=1}^{H} \Bigg(\log \tilde{\boldsymbol{\sigma}}^2_{c_i h} + \frac{\boldsymbol{\sigma}_h^2}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}} \\ + \frac{\left(\boldsymbol{\mu}_h - \tilde{\boldsymbol{\mu}}_{c_i h}\right)^2}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}} \Bigg) - \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \log \frac{\boldsymbol{\pi}_{c_i}}{q(c_i \mid \mathbf{X}, \mathbf{A})} \\ + \frac{1}{2} \sum_{h=1}^{H} \left(1 + \log \boldsymbol{\sigma}_h^2\right) \Bigg) \Bigg]. \end{aligned} \tag{24}$$

Compared to vanilla GAE and VGAE, our method introduces only a few additional parameters $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}},$ and $\boldsymbol{\pi}$, which need to be learned. However, this does not significantly increase the computational overhead. Initializing these parameters using a GMM proves effective. In practice, performing a few epochs of pretraining with GAE—e.g., using only the reconstruction loss—followed by fitting a GMM on the latent embeddings is sufficient to achieve a strong initialization.

Finally, we summarize the complete set of steps involved in our proposed method in Algorithm 1.

5 Experiments

5.1 Experimental Setup and Datasets

We employed 19 datasets from the UCR time series classification archive [16] for our clustering experiments, with specific details provided in Table 1 and Table 2. Our networks were implemented and tested using PyTorch [35], Torch_Geometric [11], and

Algorithm 1: VMGAE Training Procedure

Input: Time series dataset $\mathcal{D}$
Parameters: Hyperparameters $\{W, \gamma, \lambda, \alpha\}$, Pre-training iterations T_{pre}, Training iterations T
Output: Clustering results

```
Compute the distance matrix S using WDTW (Eq. (1)).
Convert the distance matrix S into an adjacency matrix A.
Initialize GAE with random weights.
for t = 1 to T_pre do
    Pre-train the GAE by minimizing L_recon(A) (Eq. (23)).
end for
Fit a GMM to the latent representations Z from the GAE.
Initialize parameters μ̃, σ̃, and π using the fitted GMM.
for t = 1 to T do
    Train VMGAE by minimizing L_VMGAE(X, A) (Eq. (22)).
end for
Fit a final GMM on the learned latent representations Z.
return Clustering results based on the final GMM.
```

executed on an A100 GPU (40G). *VMGAE* was trained with a learning rate of $1e^{-4}$ for 500 epochs in full-batch mode, using the Adam optimizer for optimization. Dropout with $p = 0.01$ was applied to prevent overfitting. A significant advantage of our method is that we can leverage the latent distribution to tune hyperparameters (as illustrated in Figure 4 in Appendix E) . The hyperparameters γ, λ, W, and α were tuned by visualizing the latent distribution of the training set for each dataset separately. During the testing phase, these hyperparameters were fixed, and the final results were evaluated. The details of the datasets used, the sensitivity of hyperparameters, and the evaluation metrics are provided in Appendices C, F.1, and D, respectively.

5.2 Quantitative Analysis

The performance of *VMGAE* was benchmarked against several time series clustering methods to evaluate its clustering capabilities thoroughly. The results presented in Tables 1 and 2 are sourced from the original papers, except R-Clustering [19], where results were obtained by running the authors' publicly available code. Both tables highlight the best result for each dataset in bold.

As shown in Table 1, *VMGAE* delivers superior performance, achieving the lowest average rank of 3.1579, the highest average NMI score of 0.6553, and surpassing state-of-the-art (SOTA) methods on seven datasets. Similarly, Table 2 highlights *VMGAE*'s strong results based on the Rand Index (RI) metric, with the lowest average rank of 2.6842, the highest average RI of 0.8605, and outperforming SOTA across seven datasets. Notably, on specific datasets such as *TwoPatterns*, *SonyAIBORobotSurface*, and *TwoLeadECG*, the SOTA results were significantly exceeded, with NMI improvements of 0.5089, 0.2212, and 0.1815, and RI improvements of 0.1471, 0.0991, and 0.0169, respectively.

Further extensive qualitative analysis of our method is provided in Appendix E.

Dataset	K-means	SC	KSC	K-shape	u-shaplet	USSL	DTCR	STCN	R-clust	TMRC	VMGAE
Beef	0.2925	0.4063	0.3828	0.3338	0.3413	0.3338	0.5473	0.5432	0.2475	**0.7424**	0.5237
Car	0.2540	0.3349	0.2719	0.3771	0.3655	0.4650	0.5021	0.5701	0.5390	0.3917	**0.6193**
DiatomSizeReduction	0.9300	0.8387	**1.0000**	**1.0000**	0.4849	**1.0000**	0.9418	**1.0000**	0.6154	0.6324	0.8882
Dist.Phal.Outl.AgeGroup	0.1880	0.3474	0.3331	0.2911	0.2577	0.3846	0.4553	**0.5037**	0.4343	0.3298	0.4400
ECG200	0.1403	0.1350	0.1403	0.3682	0.1323	0.3776	0.3691	**0.4316**	0.1561	0.3763	0.3643
ECGFiveDays	0.0002	0.0005	0.0682	0.0002	0.1498	0.6502	0.8056	0.3582	0.0173	0.2758	**0.8378**
Meat	0.2510	0.2732	0.2846	0.2254	0.2716	0.9085	0.9653	0.9393	0.6420	0.7980	**1.0000**
Mid.Phal.TW	0.4134	0.4952	0.4486	0.5229	0.4065	**0.9202**	0.5503	0.6169	0.4138	0.4802	0.4409
OSULeaf	0.0208	0.0814	0.0421	0.0126	0.0203	0.3353	0.2599	0.3544	**0.4453**	0.3012	0.3739
Plane	0.8598	0.9295	0.9218	0.9642	**1.0000**	**1.0000**	0.9296	0.9615	0.9892	0.8917	0.9678
Prox.Phal.Outl.AgeGroup	0.0635	0.4222	0.0682	0.0110	0.0332	**0.6813**	0.5581	0.6317	0.5665	0.5731	0.5639
SonyAIBORobotSurface	0.6112	0.2564	0.6129	0.7107	0.5803	0.5597	0.6634	0.6112	0.6620	0.2300	**0.9319**
SwedishLeaf	0.0168	0.0698	0.0073	0.1041	0.3456	**0.9186**	0.6663	0.6106	0.7151	0.5099	0.5886
Symbols	0.7780	0.7855	0.8264	0.6366	0.8691	0.8821	0.8989	0.8940	0.8775	0.8159	**0.8996**
ToeSegmentation1	0.0022	0.0353	0.0202	0.3073	0.3073	0.3351	0.3115	0.3671	0.0179	**1.0000**	0.3081
TwoPatterns	0.4696	0.4622	0.4705	0.3949	0.2979	0.4911	0.4713	0.4110	0.3181	0.1347	**1.0000**
TwoLeadECG	0.0000	0.0031	0.0011	0.0000	0.0529	0.5471	0.4614	0.6911	0.4966	0.0287	**0.8726**
Wafer	0.0010	0.0010	0.0010	0.0010	0.0010	0.0492	0.0228	0.2089	0.0000	**0.5019**	0.2136
WordSynonyms	0.5435	0.4236	0.4874	0.4154	0.3933	0.4984	0.5448	0.3947	**0.8885**	0.4210	0.5812
AVG Rank	8.7894	7.6316	7.1053	7.4739	8.1053	3.4736	3.7368	3.4210	5.8947	6.0000	**3.1579**
AVG NMI	0.3071	0.3316	0.3362	0.3513	0.3321	0.5967	0.5749	0.5841	0.4759	0.4965	**0.6553**
Best	0	0	1	1	1	5	0	3	2	3	**7**

Table 1: Normalized Mutual Information (NMI) comparisons on 19 time series datasets

Dataset	K-means	SC	KSC	K-shape	u-shaplet	USSL	DTCR	STCN	R-Clust	TMRC	VMGAE
Beef	0.6713	0.6206	0.7057	0.5402	0.6966	0.6966	0.8046	0.7471	0.6703	**0.8229**	0.7862
Car	0.6345	0.6621	0.6898	0.7028	0.6418	0.7345	0.7501	0.7372	0.7507	0.7322	**0.8045**
DiatomSizeReduction	0.9583	0.9254	**1.0000**	**1.0000**	0.7083	**1.0000**	0.9682	0.9921	0.8140	0.8539	0.9719
Dist.Phal.Outl.AgeGroup	0.6171	0.7278	0.6535	0.6020	0.6273	0.6650	**0.7825**	**0.7825**	0.7425	0.6477	0.6827
ECG200	0.6315	0.5078	0.6315	0.7018	0.5758	0.7285	0.6648	0.7018	0.6206	0.7424	**0.7862**
ECGFiveDays	0.4783	0.4994	0.5257	0.5020	0.5968	0.8340	**0.9638**	0.6504	0.0173	0.6492	0.9523
Meat	0.6595	0.7197	0.6723	0.6575	0.6742	0.7740	0.9763	0.9186	0.8341	0.8847	**1.0000**
Mid.Phal.TW	0.0983	0.8052	0.8187	0.6213	0.7920	0.7920	**0.8638**	0.8625	0.7915	0.6850	0.8132
OSULeaf	0.5615	0.7314	0.5714	0.5538	0.5525	0.6551	0.7739	0.7615	**0.8067**	0.7644	0.7798
Plane	0.9081	0.9333	0.9603	0.9901	**1.0000**	**1.0000**	0.9549	0.9663	0.9973	0.9472	0.9868
Prox.Phal.Outl.AgeGroup	0.5288	0.7791	0.5305	0.5617	0.5206	0.7939	0.8091	**0.8379**	0.8021	0.8189	0.8147
SonyAIBORobotSurface	0.7721	0.5082	0.7726	0.8084	0.7639	0.8105	0.8769	0.7356	0.8843	0.6529	**0.9834**
SwedishLeaf	0.4987	0.6897	0.4923	0.5333	0.6154	0.8547	0.9223	0.8872	**0.9302**	0.8537	0.8825
Symbols	0.8810	0.8959	0.8982	0.8373	0.9603	0.9200	0.9168	0.9088	**0.9821**	0.9088	0.9677
ToeSegmentation1	0.4873	0.4996	0.5000	0.6143	0.5873	0.6718	0.5659	0.8177	0.5112	**1.0000**	0.6712
TwoPatterns	0.8529	0.6297	0.8585	0.8046	0.7757	0.8318	0.6984	0.7619	0.7273	0.6295	**1.0000**
TwoLeadECG	0.5476	0.5018	0.5464	0.8246	0.5404	0.8628	0.7114	0.9486	0.7984	0.5873	**0.9655**
Wafer	0.4925	0.5336	0.4925	0.4925	0.4925	0.8246	0.7338	0.8433	0.5349	**0.9082**	0.5853
WordSynonyms	0.8775	0.8647	0.8727	0.7844	0.8230	0.8540	0.8984	0.8748	0.8995	0.8875	**0.9168**
AVG Rank	8.7368	8.3684	6.8947	7.5263	7.7895	4.4739	4.0000	3.9474	5.2105	5.5263	**2.6842**
AVG RI	0.6398	0.6860	0.6943	0.6911	0.6812	0.8054	0.8229	0.8282	0.7428	0.7882	**0.8605**
Best	0	0	1	1	1	2	3	2	3	3	**7**

Table 2: Rand Index (RI) comparisons on 19 time series datasets

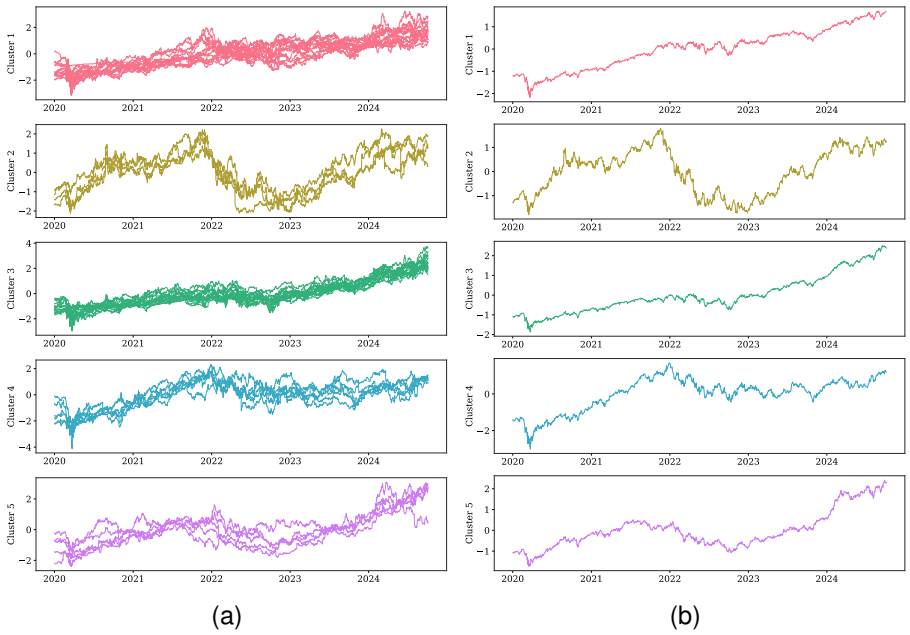

Fig. 3: (a) Clustering results of the normalized closing prices for the top 50 U.S. stocks, grouped into five clusters. (b) The average normalized closing price for each cluster shows distinct patterns across the clusters.

5.3 Application in Finance

Understanding stock market dynamics in finance is essential for making informed investment decisions. Detecting patterns and communities within this complex network of stocks helps gain insights into market behavior and make better investment choices.

In this section, we demonstrate the effectiveness of our approach by applying it to real-world stock market data and evaluating the quality of the resulting clusters. We selected the top 50 publicly traded U.S. stocks listed on NASDAQ, NYSE, and NYSE American, ranked by market capitalization. The input time series for our model consists of daily normalized closing prices from January 1, 2020, to October 4, 2024. We set the number of clusters to 5 based on the Elbow Method [42]. The results are displayed in Figure 3a, with the average for each cluster shown in Figure 3b, highlighting distinct discriminative patterns.

6 Conclusion

In this work, we introduce a novel method for clustering time series data by leveraging graph structures, achieving strong performance across various datasets. Our approach transforms time series data into graph representations using Weighted Dynamic Time

Warping, enabling the capture of temporal dependencies and structural relationships. We then apply the proposed Variational Mixture Graph Autoencoder (VMGAE) to generate a Gaussian mixture latent space, improving data separation and clustering accuracy. Extensive experiments demonstrate the effectiveness of our method, including sensitivity analysis on hyperparameters and the evaluation of different convolutional layer architectures. Furthermore, we applied our method to real-world financial data, uncovering community structures in stock markets and showcasing its potential benefits for market prediction, portfolio optimization, and risk management. These findings highlight the versatility and practical applications of VMGAE in addressing time series clustering challenges.

Appendix

A. Derivation of ELBO for VMGAE

The Evidence Lower Bound (ELBO) for VMGAE is defined as follows:

$$\begin{aligned}\log p(\mathbf{A}) &= \log \int_{\mathbf{Z}} \sum_{\mathbf{c}} p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) \\ &= \log \int_{\mathbf{Z}} \sum_{\mathbf{c}} \left[p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) \frac{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})}{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})} \right] \\ &\geq E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log \frac{p(\mathbf{A}, \mathbf{Z}, \mathbf{c})}{q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})}] \\ &= \mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}),\end{aligned} \tag{25}$$

where $\mathbf{X}$ refers to the feature matrix (or time series matrix in our case), and $\mathbf{A}$ represents the adjacency matrix. Jensen's inequality is applied to arrive at this bound.

The expanded form of $\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A})$ using 11 is given by :

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}) &= E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}, \mathbf{Z}, \mathbf{c}) - \log q(\mathbf{Z}, \mathbf{c}|\mathbf{X}, \mathbf{A})] \\ &= E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\underbrace{\log p(\mathbf{A}|\mathbf{Z})}_{(I)} + \underbrace{\log p(\mathbf{Z}|\mathbf{c})}_{(II)} + \underbrace{\log p(\mathbf{c})}_{(III)}] \\ &- E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\underbrace{\log q(\mathbf{Z}|\mathbf{X}, \mathbf{A})}_{(IV)} + \underbrace{\log q(\mathbf{c}|\mathbf{X}, \mathbf{A})}_{(V)}].\end{aligned} \tag{26}$$

Next, we compute the expectations over the various terms in the ELBO.

Term (I):

$$\begin{aligned}E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] &= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \log p(A_{ij}|\mathbf{z_i}, \mathbf{z_j}) \\ &= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log(1 - \hat{A}_{ij}).\end{aligned}$$

Term (II):

$$
\begin{aligned}
&E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{Z} \mid \mathbf{c})] \\
&= \sum_{i=1}^{n} \int_{\mathbf{z_i}} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) \log p(\mathbf{z_i} \mid c_i) d\mathbf{z_i} \\
&= \sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \int_{\mathbf{z_i}} \mathcal{N}\left(\mathbf{z_i} \mid \boldsymbol{\mu_i}, \boldsymbol{\sigma_i}^2 \mathbf{I}\right) \log \mathcal{N}\left(\mathbf{z_i} \mid \tilde{\boldsymbol{\mu}}_{c_i}, \tilde{\boldsymbol{\sigma}}_{c_i}^2 \mathbf{I}\right) d\mathbf{z_i},
\end{aligned} \tag{27}
$$

According to appendix B [18], we have:

$$
\begin{aligned}
&E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{Z} \mid \mathbf{c})] \\
&= -\sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \Bigg[\frac{H}{2} \log(2\pi) \\
&+ \frac{1}{2} \left(\sum_{h=1}^{H} \log \tilde{\boldsymbol{\sigma}}_{c_i h}^2 + \sum_{h=1}^{H} \frac{\boldsymbol{\sigma}_h^2}{\tilde{\boldsymbol{\sigma}}_{c_i h}^2} + \sum_{h=1}^{H} \frac{\left(\boldsymbol{\mu}_h - \tilde{\boldsymbol{\mu}}_{c_i h}\right)^2}{\tilde{\boldsymbol{\sigma}}_{c_i h}^2} \right) \Bigg].
\end{aligned} \tag{28}
$$

Term (III):

$$
\begin{aligned}
&E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{c})] = \\
&= \sum_{i=1}^{n} \int_{\mathbf{z_i}} q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \log \boldsymbol{\pi}_{c_i} d\mathbf{z_i} \\
&= \sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) \log \boldsymbol{\pi}_{c_i}.
\end{aligned} \tag{29}
$$

Term (IV):

$$
\begin{aligned}
&E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A})] \\
&= \sum_{i=1}^{n} \int_{\mathbf{z_i}} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X}, \mathbf{A}) q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) \log q(\mathbf{z_i} \mid \mathbf{X}, \mathbf{A}) d\mathbf{z_i} \\
&= \int_{\mathbf{z}} \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I}\right) \log \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I}\right) d\mathbf{z} \\
&= -\frac{H}{2} \log(2\boldsymbol{\pi}) - \frac{1}{2} \sum_{h=1}^{H} \left(1 + \log \boldsymbol{\sigma}_h^2\right).
\end{aligned} \tag{30}
$$

Term (V):

$$
\begin{aligned}
&E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})}[\log q(\mathbf{c} \mid \mathbf{X},\mathbf{A})] \\
&= \sum_{i=1}^{n} \int_{\mathbf{z_i}} \sum_{c_i=1}^{K} q(\mathbf{z_i} \mid \mathbf{X},\mathbf{A}) q(c_i \mid \mathbf{X},\mathbf{A}) \log q(c_i \mid \mathbf{X},\mathbf{A}) d\mathbf{z_i} \\
&= \sum_{i=1}^{n} \int_{\mathbf{z_i}} q(\mathbf{z_i} \mid \mathbf{X},\mathbf{A}) \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X},\mathbf{A}) \log q(c_i \mid \mathbf{X}) d\mathbf{z_i} \\
&= \sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X},\mathbf{A}) \log q(c_i \mid \mathbf{X},\mathbf{A}).
\end{aligned} \tag{31}
$$

By putting all terms together, we will have:

$$
\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\mathbf{X},\mathbf{A}) &= \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log(1 - \hat{A}_{ij}) \\
&- \frac{1}{2} \sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X},\mathbf{A}) \sum_{h=1}^{H} \Bigg(\log \tilde{\boldsymbol{\sigma}}^2_{c_i h} + \frac{\boldsymbol{\sigma}^2_h}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}} + \\
&\frac{\left(\boldsymbol{\mu}_h - \tilde{\boldsymbol{\mu}}_{c_i h}\right)^2}{\tilde{\boldsymbol{\sigma}}^2_{c_i h}} \Bigg) + \sum_{i=1}^{n} \sum_{c_i=1}^{K} q(c_i \mid \mathbf{X},\mathbf{A}) \log \frac{\boldsymbol{\pi}_{c_i}}{q(c_i \mid \mathbf{X},\mathbf{A})} \\
&+ \frac{1}{2} \sum_{h=1}^{H} \left(1 + \log \boldsymbol{\sigma}^2_h\right).
\end{aligned} \tag{32}
$$

B. Derivation of $q(c_i|\mathbf{X}, \mathbf{A})$

An important point is how to calculate $q(c_i \mid \mathbf{X},\mathbf{A})$. We can reformat the ELBO into the following form:

$$
\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\mathbf{X},\mathbf{A}) &= E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})} \left[\log \frac{p(\mathbf{A},\mathbf{Z},\mathbf{c})}{q(\mathbf{Z},\mathbf{c} \mid \mathbf{X},\mathbf{A})} \right] \\
&= E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})} \left[\log \frac{p(\mathbf{A} \mid \mathbf{Z}) p(\mathbf{c} \mid \mathbf{Z}) p(\mathbf{Z})}{q(\mathbf{Z} \mid \mathbf{X},\mathbf{A}) q(\mathbf{c} \mid \mathbf{X},\mathbf{A})} \right] \\
&= E_{q(\mathbf{Z},\mathbf{c}|\mathbf{X},\mathbf{A})} \left[\log \frac{p(\mathbf{A} \mid \mathbf{Z}) p(\mathbf{Z})}{q(\mathbf{Z} \mid \mathbf{X},\mathbf{A})} + \log \frac{p(\mathbf{c} \mid \mathbf{Z})}{q(\mathbf{c} \mid \mathbf{X},\mathbf{A})} \right] \\
&= E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})} \left[\log \frac{p(\mathbf{A} \mid \mathbf{Z}) p(\mathbf{Z})}{q(\mathbf{Z} \mid \mathbf{X},\mathbf{A})} \right] \\
&- \sum_{i=1}^{n} \int_{\mathbf{z_i}} q(\mathbf{z_i} \mid \mathbf{X},\mathbf{A}) \sum_{c_i} q(c_i \mid \mathbf{X},\mathbf{A}) \log \frac{q(c_i \mid \mathbf{X},\mathbf{A})}{p(c_i \mid \mathbf{z_i})} d\mathbf{z_i} \\
&= E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})} \left[\log \frac{p(\mathbf{A} \mid \mathbf{Z}) p(\mathbf{Z})}{q(\mathbf{Z} \mid \mathbf{X},\mathbf{A})} \right] \\
&- \sum_{i=1}^{n} \int_{\mathbf{z_i}} q(\mathbf{z_i} \mid \mathbf{X},\mathbf{A}) D_{\text{KL}}(q(c_i \mid \mathbf{X},\mathbf{A}) \| p(c_i \mid \mathbf{z_i})) d\mathbf{z_i}.
\end{aligned} \tag{33}
$$

In the Equation above, the first term is not dependent on $\mathbf{c}$ and the second is non-negative. Hence, to maximize $\mathcal{L}_{\text{ELBO}}(\mathbf{X}, \mathbf{A}), D_{KL}(q(c_i \mid \mathbf{X}, \mathbf{A}) \| p(c_i \mid \mathbf{z_i})) \equiv 0$ should be satisfied. As a result, we use the following Equation to compute $q(c_i \mid \mathbf{X}, \mathbf{A})$ in *VMGAE*:

$$q(c_i \mid \mathbf{X}, \mathbf{A}) = p(c_i \mid \mathbf{z_i}) \equiv \frac{p(c_i) p(\mathbf{z_i} \mid c_i)}{\sum_{c'=1}^{K} p(c') \, p(\mathbf{z_i} \mid c')}.$$

C. Datasets

We conducted our clustering experiments using 19 datasets from the UCR Time Series Classification Archive [16], a widely recognized benchmark for time series analysis. The details of these datasets are presented in Table 3.

No.	Name	Train/Test	Length	Classes
1	Beef	30/30	470	5
2	Car	60/60	577	4
3	DiatomSizeReduction	16/306	345	4
4	Dist.Phal.Outl.AgeGroup	400/139	80	3
5	ECG200	100/100	96	2
6	ECGFiveDays	23/861	136	2
7	Meat	60/60	448	3
8	Mid.Phal.TW	399/154	80	6
9	OSULeaf	200/242	427	6
10	Plane	105/105	144	7
11	Prox.Phal.Outl.AgeGroup	400/205	80	3
12	SonyAIBORobotSurface	20/601	70	2
13	SwedishLeaf	500/625	128	15
14	Symbols	25/995	398	6
15	ToeSegmentation1	40/228	277	2
16	TwoPatterns	1000/4000	128	4
17	TwoLeadECG	23/1139	82	2
18	Wafer	1000/6164	152	2
19	WordSynonyms	267/638	270	25

Table 3: Statistics of the 19 datasets from the UCR benchmark used in our experiments.

D. Evaluation Metrics

We evaluate the clustering performance in our analysis using two well-established metrics: the Rand Index (RI) and Normalized Mutual Information (NMI). The Rand Index, which quantifies the agreement between the predicted and actual clustering assignments, is computed as follows:

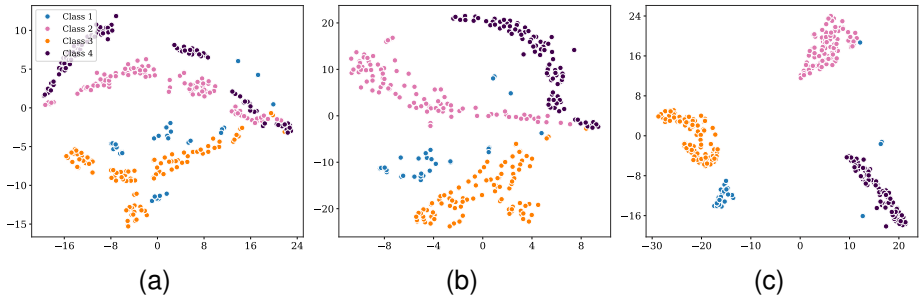

Fig. 4: The visualizations with t-SNE on the dataset *DiatomSizeReduction*. The colors of the points indicate the actual labels. (a) epoch 0, (b) epoch 10, (c) epoch 100.

$$RI = \frac{TP + TN}{TP + FP + FN + TN}. \tag{34}$$

In this expression, TP (True Positive) denotes the number of pairs of time series correctly classified into the same cluster, while TN (True Negative) refers to the number of pairs accurately assigned to different clusters. Conversely, FP (False Positive) captures the number of pairs incorrectly grouped into the same cluster, and FN (False Negative) accounts for pairs that should be clustered together but are mistakenly separated.

The NMI score is defined as:

$$NMI = \frac{\sum_{i=1}^{K} \sum_{j=1}^{K} N_{ij} \log\left(\frac{n \cdot N_{ij}}{|G_i| \cdot |P_j|}\right)}{\sqrt{\left(\sum_{i=1}^{K} |G_i| \log\left(\frac{|G_i|}{n}\right)\right) \cdot \left(\sum_{j=1}^{M} |P_j| \log\left(\frac{|P_j|}{n}\right)\right)}}, \tag{35}$$

where N_{ij} represents the number of time series that are common between the i-th ground truth cluster G_i and the j-th predicted cluster P_j. $|\cdot|$ is the number of time series in the cluster. The variables K and n in Equations 34 and 35 are defined as previously explained in the section Notation.

E. Qualitative Analysis

We further present visualizations of the evolving clusters during training on the DiatomSizeReduction in Figure 4. These clusters are mapped from the latent space representations $\mathbf{Z}$ to a 2D space using t-SNE [30]. The t-SNE plots illustrate how the latent representations become increasingly well-separated as training progresses, reflecting *VMGAE*'s capacity to learn distinct clusters from the time series data.

Dataset	Car				Meat				SonyAIBORobotSurface1			
α	0.025	0.05	0.075	0.1	0.025	0.05	0.075	0.1	0.025	0.05	0.075	0.1
NMI	0.4906	0.5634	**0.6193**	0.4884	**1.0000**	**1.0000**	**1.0000**	0.8996	0.8524	0.9182	**0.9319**	0.9089
RI	0.7322	0.7813	**0.8045**	0.7559	**1.0000**	**1.0000**	**1.0000**	0.9570	0.9544	0.9801	**0.9834**	0.9769

(a) Impact of hyperparameter α

Dataset	Car				Meat				SonyAIBORobotSurface1			
λ	0.1	0.01	0.001	0.0001	0.1	0.01	0.001	0.0001	0.1	0.01	0.001	0.0001
NMI	0.4421	0.6088	**0.6193**	**0.6193**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9298	**0.9319**	0.9298	0.9298
RI	0.7384	0.7920	**0.8045**	**0.8045**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9834**	**0.9834**	**0.9834**	**0.9834**

(b) Impact of hyperparameter λ

Dataset	Car				Meat				SonyAIBORobotSurface1			
γ	0.2	0.4	0.7	1.0	0.2	0.4	0.7	1.0	0.2	0.4	0.7	1.0
NMI	0.5308	**0.6193**	0.5483	0.4895	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9319	0.9190	**0.9427**	**0.9427**
RI	0.7751	**0.8045**	0.7853	0.7661	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9834	0.9801	**0.9867**	**0.9867**

(c) Impact of hyperparameter γ

Table 4: The tables show the impact of different hyperparameters on the NMI and RI metrics for 3 different datasets.

F. Ablation Study

F.1. Hyperparameter Sensitivity Analysis

In this section, we analyze the impact and sensitivity of the hyperparameters γ, λ, and α on our method. To assess the sensitivity of each hyperparameter, the other hyperparameters were kept fixed at their optimal values, as shown in Table 4. The hyperparameter values $\gamma = 0.7$ and $\gamma = 1.0$ yield better metric results for the *SonyAIBORobotSurface1* dataset compared to $\gamma = 0.2$, which was used to report the results in Tables 1 and 2. This improvement was not evident through the visualization process. As shown in the table, for some datasets like *Meat*, the model is not sensitive to the hyperparameter values, whereas for other datasets, such as *Car*, the model shows some sensitivity to the hyperparameter values.

F.2. Impact of Convolutional Layer Variants

Several advanced graph convolutional layers have been developed to enhance information propagation in graph neural networks, each with distinct methods and advantages. One well-known type of convolutional layer is the Graph Attention Network (GAT) [45]. GAT layers introduce attention mechanisms to graph convolutions, enabling the model to assign different importance to neighboring nodes rather than treating them uniformly. Specifically, the GAT layer computes attention coefficients α_{ij} based on node features, which are then used to aggregate information from neighboring

nodes. The process of each GAT layer is expressed as follows:

$$\mathbf{Z}_i^{(l+1)} = \phi\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}^{(l)}\mathbf{Z}_j^{(l)}\right), \tag{36}$$

where $\mathcal{N}(i)$ denotes the neighbors of node i, and ϕ is an activation function. The attention mechanism allows GAT layers to dynamically adjust the influence of neighboring nodes, leading to more flexible and potentially more accurate embeddings.

Another variant is SAGEConv [13], which stands for Sample and Aggregation Convolution. This layer generalizes GCNs by allowing for aggregating features from a sampled set of neighbors instead of using all neighbors. Various aggregation operators like mean aggregator, LSTM aggregator, and polling aggregator can perform the aggregation process. The final formula is given by :

$$\mathbf{Z}_i^{(l+1)} = \phi\Big(\mathbf{W}_1^{(l)}\mathbf{Z}_i^{(l)} + \mathbf{W}_2^{(l)}\text{AGGREGATE}(\{\mathbf{Z}_j^{(l)} : j \in \mathcal{N}(i)\})\Big), \tag{37}$$

where AGGREGATE is a function that combines the features of the neighbors.

ChebConv [6] is another robust convolutional layer that utilizes a recursive process to produce Z_i^j's and aggregate them by some learnable parameters. The ChebConv whole operation is given by:

$$\begin{aligned}
\mathbf{Z_i^{(1)}} &= \mathbf{X_i} \\
\mathbf{Z_i^{(2)}} &= \tilde{\mathbf{L}}.\mathbf{X_i} \\
\mathbf{Z_i^{(k)}} &= \mathbf{2}\tilde{\mathbf{L}}.\mathbf{Z_i^{k-1}} - \mathbf{Z_i^{k-2}} \\
\mathbf{\tilde{X}_i} &= \sum_{k=0}^{K} \mathbf{\Theta_k}\mathbf{Z_i^k},
\end{aligned} \tag{38}$$

where $\mathbf{T}_k(\mathbf{L})$ denotes the Chebyshev polynomial of order k, and $\mathbf{L}$ is the graph Laplacian.

Similarly, SGConv [46], or Simplifying Graph Convolution, provides an efficient alternative that simplifies the graph convolution operation while maintaining good performance. The operation can be expressed as:

$$\mathbf{Z_i} = \text{Softmax}\Big(\mathbf{S^k x_i \Theta}\Big), \tag{39}$$

where S is the normalized adjacency matrix and k is a fixed number and Θ is the laearnable parameter matrix.

Finally, TAGConv [9], or Adaptive Graph Convolution, adapts the convolution operation based on the local graph structure. It computes the convolution by taking into account the varying degrees of nodes:

$$\mathbf{Z}_i = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}_i W_k, \tag{40}$$

where A is the normalized adjacency matrix and W's are learnable parameters.

we examines how different convolutional layers affect the model's ability to learn node embeddings and perform clustering. In the main results shown in Tables 1 and 2, we used Graph Convolutional Network (GCN) layers. Here, we test other types of convolutional layers and compare their effects on the model's performance across different datasets. The results of these comparisons are shown in Table 5.

Conv. Layer	Beef		Car		Dist. Age Group	
	NMI	RI	NMI	RI	NMI	RI
GCN	**0.5237**	**0.7862**	**0.6193**	**0.8045**	0.4400	0.6827
GAT	0.4926	0.7463	0.4165	0.6548	0.4322	0.7426
SAGEConv	0.4907	0.7429	0.4499	0.7119	**0.4637**	**0.7492**
ChebConv	0.2789	0.7152	0.1683	0.6534	0.3243	0.5933
SGConv	0.4673	0.7418	0.4900	0.7402	0.4363	0.7405
TAGConv	0.4907	0.7429	0.4304	0.7122	0.3876	0.7218

Table 5: Performance comparison of different convolutional layers on clustering across datasets (Beef, Car, Distinct Age Group), evaluated using Normalized Mutual Information (NMI) and Rand Index (RI).

	VMGAE	ARGA	VGAE	GAE
NMI	**0.459**	0.450	0.436	0.429

Table 6: NMI Comparisons on cora data-set

F.3. Versatility of VMGAE: Application to Graph Datasets

While our primary contribution focuses on applying VMGAE to time series data transformed into graph representations, it is important to highlight the versatility of our method, which can be effectively applied to any graph input. The architecture is designed to learn meaningful latent representations across diverse graph datasets.

To demonstrate this, we employed the *Cora* dataset, a benchmark graph dataset comprising scientific publications grouped into distinct categories, with citation relationships forming the edges between nodes. Each node corresponds to a publication, and the edges represent citation links. This dataset is commonly used in graph-based machine-learning tasks due to its structured graph topology and rich node features.

Our experiments on the *Cora* dataset further validate the flexibility of our VMGAE architecture. For this evaluation, the learning rate was set to $1e^{-5}$ the λ parameter was

set to 0.001, the model was trained for 500 epochs, and dropout was applied with a rate of 0.01. Table 6 provides a comparison of NMI scores between VMGAE and other recognized graph-based methods such as GAE, VGAE [21], and ARGA [33].

References

1. Aghabozorgi, S., Shirkhorshidi, A.S., Wah, T.Y.: Time-series clustering–a decade review. Information systems **53**, 16–38 (2015)
2. Babu, M.S., Geethanjali, N., Satyanarayana, B.: Clustering approach to stock market prediction. International Journal of Advanced Networking and Applications **3**(4), 1281 (2012)
3. Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al.: Spectral temporal graph neural network for multivariate time-series forecasting. Advances in neural information processing systems **33**, 17766–17778 (2020)
4. Chaudhuri, T.D., Ghosh, I.: Using clustering method to understand indian stock market volatility. arXiv preprint arXiv:1604.05015 (2016)
5. Close, L., Kashef, R.: Combining artificial immune system and clustering analysis: A stock market anomaly detection model. Journal of Intelligent Learning Systems and Applications **12**(04), 83–108 (2020)
6. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering (2017), https://arxiv.org/abs/1606.09375
7. Deng, A., Hooi, B.: Graph neural network-based anomaly detection in multivariate time series. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 4027–4035 (2021)
8. Ding, M., Chen, Y., Bressler, S.L.: Granger causality: basic theory and application to neuroscience. Handbook of time series analysis: recent theoretical developments and applications pp. 437–460 (2006)
9. Du, J., Zhang, S., Wu, G., Moura, J.M.F., Kar, S.: Topology adaptive graph convolutional networks (2018), https://arxiv.org/abs/1710.10370
10. Fang, Y., Xu, H., Jiang, J.: A survey of time series data visualization research. In: IOP Conference Series: Materials Science and Engineering. vol. 782, p. 022013. IOP Publishing (2020)
11. Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
12. Guo, X., Liu, X., Zhu, E., Yin, J.: Deep clustering with convolutional autoencoders. In: Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24. pp. 373–382. Springer (2017)
13. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs (2018), https://arxiv.org/abs/1706.02216
14. Han, S., Woo, S.S.: Learning sparse latent graph representations for anomaly detection in multivariate time series. In: Proceedings of the 28th ACM SIGKDD Conference on knowledge discovery and data mining. pp. 2977–2986 (2022)
15. Hird, J.N., McDermid, G.J.: Noise reduction of ndvi time series: An empirical comparison of selected techniques. Remote Sensing of Environment **113**(1), 248–258 (2009)
16. Huang, X., Ye, Y., Xiong, L., Lau, R.Y., Jiang, N., Wang, S.: Time series k-means: A new k-means type smooth subspace clustering for time series data. Information Sciences **367-368**, 1–13 (2016). https://doi.org/https://doi.org/10.1016/j.ins.2016.05.040, https://www.sciencedirect.com/science/article/pii/S0020025516303796
17. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data mining and knowledge discovery **33**(4), 917–963 (2019)

18. Jiang, Z., Zheng, Y., Tan, H., Tang, B., Zhou, H.: Variational deep embedding: An unsupervised and generative approach to clustering. arXiv preprint arXiv:1611.05148 (2016)
19. Jorge, M.B., Rubén, C.: Time series clustering with random convolutional kernels. Data Mining and Knowledge Discovery pp. 1–27 (2024)
20. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
21. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016)
22. Lee, J., Kim, D., Sim, S.: Temporal multi-features representation learning-based clustering for time-series data. IEEE Access (2024)
23. Li, G., Choi, B., Xu, J., Bhowmick, S.S., Mah, D.N.y., Wong, G.L.H.: Autoshape: An autoencoder-shapelet approach for time series clustering. arXiv preprint arXiv:2208.04313 (2022)
24. Li, H., Liu, J., Yang, Z., Liu, R.W., Wu, K., Wan, Y.: Adaptively constrained dynamic time warping for time series classification and clustering. Information Sciences **534**, 97–116 (2020)
25. Li, P., Boubrahimi, S.F., Hamdi, S.M.: Graph-based clustering for time series data. In: 2021 IEEE International Conference on Big Data (Big Data). pp. 4464–4467. IEEE (2021)
26. Liao, T.W.: Clustering of time series data—a survey. Pattern recognition **38**(11), 1857–1874 (2005)
27. Lin, J., Williamson, S., Borne, K., DeBarr, D.: Pattern recognition in time series. Advances in machine learning and data mining for astronomy **1**(617-645), 3 (2012)
28. Ma, Q., Li, S., Zhuang, W., Wang, J., Zeng, D.: Self-supervised time series clustering with model-based dynamics. IEEE Transactions on Neural Networks and Learning Systems **32**(9), 3942–3955 (2020)
29. Ma, Q., Zheng, J., Li, S., Cottrell, G.W.: Learning representations for time series clustering. Advances in neural information processing systems **32** (2019)
30. van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research **9**(86), 2579–2605 (2008), http://jmlr.org/papers/v9/vandermaaten08a.html
31. Mudelsee, M.: Trend analysis of climate time series: A review of methods. Earth-science reviews **190**, 310–322 (2019)
32. Olive, X., Basora, L., Viry, B., Alligier, R.: Deep trajectory clustering with autoencoders. In: ICRAT 2020, 9th International Conference for Research in Air Transportation (2020)
33. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding (2019), https://arxiv.org/abs/1802.04407
34. Paparrizos, J., Gravano, L.: k-shape: Efficient and accurate clustering of time series. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. pp. 1855–1870 (2015)
35. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems **32**, 8024–8035 (2019)
36. Sakoe, H.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing **26**, 159–165 (1978), https://api.semanticscholar.org/CorpusID:17900407
37. Shah, D., Isah, H., Zulkernine, F.: Stock market analysis: A review and taxonomy of prediction techniques. International Journal of Financial Studies **7**(2), 26 (2019)
38. Shaukat, K., Alam, T.M., Luo, S., Shabbir, S., Hameed, I.A., Li, J., Abbas, S.K., Javed, U.: A review of time-series anomaly detection techniques: A step to future perspectives. In: Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 1. pp. 865–877. Springer (2021)

39. Siuly, S., Li, Y., Zhang, Y.: Eeg signal analysis and classification. IEEE Trans Neural Syst Rehabilit Eng **11**, 141–144 (2016)
40. Song, C., Lin, Y., Guo, S., Wan, H.: Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34, pp. 914–921 (2020)
41. Tan, S.Y., Saha, H., Jacoby, M., Henze, G., Sarkar, S.: Granger causality based hierarchical time series clustering for state estimation. IFAC-PapersOnLine **53**(2), 524–529 (2020)
42. Thorndike, R.L.: Who belongs in the family? Psychometrika **18**(4), 267–276 (1953)
43. Torres, J.F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., Troncoso, A.: Deep learning for time series forecasting: a survey. Big Data **9**(1), 3–21 (2021)
44. Ulanova, L., Begum, N., Keogh, E.: Scalable clustering of time series with u-shapelets. In: Proceedings of the 2015 SIAM international conference on data mining. pp. 900–908. SIAM (2015)
45. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2018), https://arxiv.org/abs/1710.10903
46. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. In: Proceedings of the 36th International Conference on Machine Learning. pp. 6861–6871 (2019)
47. Xi, W., Jain, A., Zhang, L., Lin, J.: Lb-simtsc: An efficient similarity-aware graph neural network for semi-supervised time series classification. arXiv preprint arXiv:2301.04838 (2023)
48. Xie, J., Girshick, R., Farhadi, A.: Unsupervised deep embedding for clustering analysis. In: International conference on machine learning. pp. 478–487. PMLR (2016)
49. Yang, J., Leskovec, J.: Patterns of temporal variation in online media. In: Proceedings of the fourth ACM international conference on Web search and data mining. pp. 177–186 (2011)
50. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875 (2017)
51. Zha, D., Lai, K.H., Zhou, K., Hu, X.: Towards similarity-aware time-series classification. In: Proceedings of the 2022 SIAM International Conference on Data Mining (SDM). pp. 199–207. SIAM (2022)
52. Zhang, Q., Wu, J., Yang, H., Tian, Y., Zhang, C.: Unsupervised feature learning from time series. In: IJCAI. pp. 2322–2328. New York, USA (2016)
53. Zhang, Q., Wu, J., Zhang, P., Long, G., Zhang, C.: Salient subsequence learning for time series clustering. IEEE transactions on pattern analysis and machine intelligence **41**(9), 2193–2207 (2018)
54. Zhao, H., Wang, Y., Duan, J., Huang, C., Cao, D., Tong, Y., Xu, B., Bai, J., Tong, J., Zhang, Q.: Multivariate time-series anomaly detection via graph attention network. In: 2020 IEEE international conference on data mining (ICDM). pp. 841–850. IEEE (2020)
55. Zhao, J., Itti, L.: shapedtw: Shape dynamic time warping. Pattern Recognition **74**, 171–184 (2018)
56. Zhao, J., Zhu, N., Lu, S.: Productivity model in hot and humid environment based on heat tolerance time analysis. Building and environment **44**(11), 2202–2207 (2009)