mathematics

MDPI

Article

TVGeAN: Tensor Visibility Graph-Enhanced Attention Network for Versatile Multivariant Time Series Learning Tasks

Mohammed Baz

Department of Computer Engineering, College of Computers and Information Technology, Taif University, Taif 21944, Saudi Arabia; mo.baz@tu.edu.sa

Abstract: This paper introduces Tensor Visibility Graph-enhanced Attention Networks (TVGeAN), a novel graph autoencoder model specifically designed for MTS learning tasks. The underlying approach of TVGeAN is to combine the power of complex networks in representing time series as graphs with the strengths of Graph Neural Networks (GNNs) in learning from graph data. TVGeAN consists of two new main components: TVG which extend the capabilities of visibility graph algorithms in representing MTSs by converting them into weighted temporal graphs where both the nodes and the edges are tensors. Each node in the TVG represents the MTS observations at a particular time, while the weights of the edges are defined based on the visibility angle algorithm. The second main component of the proposed model is GeAN, a novel graph attention mechanism developed to seamlessly integrate the temporal interactions represented in the nodes and edges of the graphs into the core learning process. GeAN achieves this by using the outer product to quantify the pairwise interactions of nodes and edges at a fine-grained level and a bilinear model to effectively distil the knowledge interwoven in these representations. From an architectural point of view, TVGeAN builds on the autoencoder approach complemented by sparse and variational learning units. The sparse learning unit is used to promote inductive learning in TVGeAN, and the variational learning unit is used to endow TVGeAN with generative capabilities. The performance of the TVGeAN model is extensively evaluated against four widely cited MTS benchmarks for both supervised and unsupervised learning tasks. The results of these evaluations show the high performance of TVGeAN for various MTS learning tasks. In particular, TVGeAN can achieve an average root mean square error of 6.8 for the C-MPASS dataset (i.e., regression learning tasks) and a precision close to one for the SMD, MSL, and SMAP datasets (i.e., anomaly detection learning tasks), which are better results than most published works.

Keywords: graph neural networks; complex networks; multivariant time series

MSC: 68T07

check for updates

Citation: Baz, M. TVGeAN: Tensor Visibility Graph-Enhanced Attention Network for Versatile Multivariant Time Series Learning Tasks. *Mathematics* **2024**, *12*, 3320. https://doi.org/10.3390/math12213320

Academic Editors: Sheng-Wei Chi and Jonathan Blackledge

Received: 4 September 2024
Revised: 2 October 2024
Accepted: 19 October 2024
Published: 23 October 2024

Copyright: © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Multivariate Time Series (MTS) analysis is one of the most enduring and important practices. It has receive significant attention due to the widespread use of MTS in various fields and the invaluable insights that can be gained from this analysis. For example, determining the severity of diseases based on the evolution of their symptoms over time is based on MTS classification [1,2]. The identification of potential investment opportunities based on future financial market movements is mainly based on the principles of MTS regression [3,4]. Preventing cyber-attacks is mainly based on MTS anomaly detection algorithms [5,6], while optimizing customer experience by understanding latent common behaviours relies heavily on MTS clustering [7,8]. These and other applications have driven the development of numerous analysis and modeling techniques, ranging from classical parametric methods (such as the vector autoregression family [9] and state space models [10]) to non-parametric approaches (such as kernel smoothing [11] and empirical

mode decomposition [12]) and even their hybrid forms (such as generalized additive models [13] and Bayesian structural time series [14]). Although these techniques have proven successful in various fields, the ever-increasing volume and complexity of MTS data combined with rapid advances in computational resources have made the Deep Neural Network (DNN) the predominant method for processing MTS.

Most DNN models developed for processing MTS data have endeavored to avoid the need to apply hand-crafted feature engineering techniques by using architectures known for their ability to automatically extract the temporal dependencies of time series, such as self-attention mechanisms [15], Temporal Convolutional Neural Networks (TCNs) [16], and Recurrent Neural Networks (RNNs) [17] and their variants. Despite the different principles and design goals of these architectures, the fact that they were designed to process univariate sequences entails the use of multiple instances to facilitate the processing of MTS. The main shortcomings of such a multiple-instance configuration are manifold. These include the high computational cost incurred by running multiple instances [18,19], the risk of losing critical information when the information obtained from these instances is fused [20,21], the limitations in simultaneously capturing the evolving patterns and dependencies of MTS due to the restriction of each instance's hypothesis space to its individual inputs [22,23], poor scalability, high intractability, and low interpretability and explainability [24,25].

To address these shortcomings, Graph Neural Networks (GNNs) [26] have recently been used for MTS learning tasks. GNNs are a class of DNNs designed to process graph-structured data in their native formats. GNNs achieve this by specifying the underlying architectures of DNNs as a graph in which each vertex is equipped with a learnable computational unit that receives an entity from the input space, and each edge represents the relationships between these entities. The aggregation process defined by the GNN allows each vertex to learn not only the behavior of the received entity but also the possible interactions that other entities might have. One of the main advantages of using the GNN for processing MTS is its ability to incorporate the information represented by vertices and edges into the learning process. This capability allows the GNN to capture different levels of dependencies of MTS datasets represented as graphs without the need to use multiple instance configurations as required by other DNN models. The ability of GNN to tailor the learning process to a specific part or component of the graphs is another major advantage of using GNN when processing long and complex MTSs. Indeed, the learning process of GNNs can be fine-tuned to extract the features of individual time steps or variables represented by vertices, to capture the dynamic evolution of two MTS instances connected by the edges, or to interpret the patterns of a part of the MTS represented by a subgraph [27]. Another important advantage of using GNNs for MTS learning tasks is that GNNs impose no restrictions on the type of computational units used at each node or edge. Each node in the graph can be assigned its own computational function, ranging from simple aggregation techniques to a full DNN. This flexibility allows GNNs to deal more effectively with MTS datasets with heterogeneous and incomplete input space, as nodes with missing or corrupted information can adapt their computational unit based on the available data, resulting in more robust and reliable performance. Finally, the inherent compatibility of GNNs for inductive learning approaches enables the generalization of knowledge from the training phase to unseen data during the testing phase. This inductive capability is particularly valuable in MTS problems where the structures of the MTS in the testing phase may differ from the training phases, either due to structural changes, such as the emergence of anomalies, or due to the dynamic nature of the MTS itself, where new relationships may appear over time. GNNs can handle these variations seamlessly without the need to retrain the entire model from scratch or significantly adjust the parameters.

Motivated by the importance of developing effective MTS processing models and the lack of a comprehensive framework for constructing such a model, this work proposes a novel architecture called Tensor Visibility Graph-enhanced Attention Network (TVGeAN). The underlying approach of TVGeAN is to combine the strengths of complex networks

in representing time series as graphs with the strengths of GNNs in learning from graph-structured data in their native format. Specifically, the main contributions of this work can be summarized as follows:

1. Developing a new algorithm for converting MTS into graphs, called Tensor Visibility Graph (TVG), in which MTS datasets are mapped to a weighted temporal graph where both the nodes and edges are tensors. Each node in the TVG represents the MTS observations at a particular point in time, while the weights of the edges are defined based on the visibility angle algorithm [28]. This allows the graphs generated by the TVG algorithm to preserve the temporal dependencies within each univariate time series (via tensor edges) and the mutual relationships between different time series at each time point (via tensor nodes).
2. Developing a novel graph attention mechanism called Graph-enhanced Attention Network (GeAN) that seamlessly integrates the temporal interactions represented in the nodes and edges of graphs into the core learning process. GeAN achieves this by using the outer product to quantify the pairwise interactions of nodes and edges at a fine-grained level so that the learning process can obtain a comprehensive representation of all potential relationships and patterns in the given MTS dataset. GeAN also uses the bilinear model [29,30] to distil the knowledge interwoven in these representations by allowing each node and edge to collaboratively modulate each other's contributions through a shared learning space. This in turn leads to a consolidation of the salient behaviors of the MTS dataset into a compact map that ensures the derivation of accurate attention scores.
3. Designing the architecture of TVGeAN based on the autoencoder approach with sparse and variational learning units [31,32]. The main reason for using the sparse learning units is to promote inductive learning in TVGeAN by prioritizing nodes and edges according to their importance for the learning objectives. This in turn allows TVGeAN to learn from different graph structures without the need to implement additional techniques such as random sampling as in GraphSAGE [33] or node masking as in [34]. The variational learning unit, on the other hand, is used to provide TVGeAN with generative capabilities that allow it to condense the given inputs in a low-dimensional space and then reconstruct them in the output layers. Through this regeneration process, TVGeAN can not only capture the intrinsic behavior of MTS but also learn from MTS datasets even if they are contaminated with noisy or inconsistent measurements.
4. Present a comprehensive evaluation of the TVGeAN model using four widely cited MTS benchmarks for supervised and unsupervised learning tasks: C-MPASS [35], SMD [36], MSL, and SMAP [37]. These evaluations were used to assess the core approaches used to develop TVGeAN from different perspectives. These included the impact of the graph generated by the TVG algorithm on the learnability of the model, the impact of the outer product and bilinear models on the generalizability of the GeAN models, and the overall performance of TVGeAN compared to other related work. These evaluations are quantified using standard statistical metrics such as Root Mean Square Error (RMSE), precision, recall, and F1. The results of these evaluations show that TVGeAN is very competent in various MTS learning tasks. In particular, TVGeAN achieves an average RMSE of 6.8 for the C-MPASS dataset and a precision close to one for the SMD, MSL, and SMAP datasets.

The rest of this paper is organized as follows: Section 2 reviews the most pertinent works in the open literature, Section 3 describes the proposed model, Sections 4 and 5 present the results and discussion, respectively, and Section 6 concludes this article.

2. Related Works

The ubiquity of Multivariate Time Series (MTS) in various fields and the critical decisions that can be made based on the results of their analysis have led to the development of a plethora of analytical and modeling approaches and techniques. Indeed, the history of

MTS analysis is long and colorful, ranging from classical statistical methods to the most advanced deep learning models. Several monographs, e.g., refs. [38,39], have devoted their entire content to a detailed discussion of these approaches and techniques. However, this section is dedicated to exploring the main approaches underlying deep learning methods that are closely related to our proposal. The aim here is to highlight the main contributions and limitations of the related work and to point out the merits of the proposed model. Section 2.1 discusses the traditional deep learning models, and Section 2.2 focuses on GNNs.

2.1. RNN Family, TCN, and Transformer Architectures

The ability of Deep Neural Networks (DNNs) to process large amounts of data without the need for a hand-crafted feature extraction technique allows them to play a crucial role in MTS analysis. DNNs [40] attempt to mimic the mammalian nervous system by defining a network with a set of interconnected computational units called artificial neurons, each equipped with a nonlinear activation function and a set of learnable parameters. The parameters of all neurons are adjusted based on the learning task the model is intended to fulfil, such as regression, anomaly detection, and classification. The flexibility that the DNN approach offers in terms of how the internal structure of the neurons is specified and connected to other neurons has led to the development of several variants of DNN architectures. Among these variants, recurrent neural networks (RNNs) [17] are the most commonly used in time series analysis.

The basic idea of RNNs is to provide each neuron with a recurrent connection so that a neuron can process the current input instance, taking into account the information that the neuron has learnt from processing the previous instances. With this configuration, RNNs can easily capture the temporal dependencies in the time series. However, it has been shown that relying solely on the recurrent connections to extract the historical information can lead to some shortcomings. These include the vanishing gradient problem [41], a high sensitivity to noise and irregularities [42], and the slowness of training and inference [43]. Motivated by these shortcomings, some works propose enhancements variants of RNNs, such as Long Short-Term Memory (LSTM) [44] and Gated Recurrent Units (GRUs) [45]. These variants are based on two design approaches: improving the control of information flow through the network by reengineering the activation functions of individual neurons so that they can act as gates to filter out unwanted information, and storing the extracted information in a common data structure (also known as a state vector) that can be updated by all neurons. The advantages of these variants over traditional RNNs have been emphasized in several works, e.g., in [46–48]. However, the finite space of the state vectors of these variants makes it difficult to balance the long- and short-term dependencies in a stable way, especially for long and complex time series [15,49,50].

The remarkable success of Convolutional Neural Networks (CNNs) [50] in the field of computer vision has inspired their adaptation for analyzing time series and has led to the development of Temporal CNNs (TCNs) [16]. The core idea behind TCNs is to convolve the given time series with a set of kernels of predefined length and learnable parameters so that the temporal dependencies of neighboring instances can be captured. The features derived from all kernels are then condensed to derive higher-level representations that encapsulate the temporal dependencies in the time series. Compared to the RNN family, TCNs offer higher computational efficiency due to more efficient weight sharing and sparser connections [16,51]. Moreover, the different types of convolution operators, such as causal, dilated [51], depth-wise [52], and grouped convolution [53], provide a more versatile way to control the granularity of feature extraction. However, the dependence of TCNs on fixed-length kernels limits their receptive fields, which in turn impairs their ability to effectively capture long-term dependencies [54–56].

The transformer architecture [15] was developed to overcome one of the main shortcomings of the TCN and RNN family, namely their reliance on sequential processing frameworks. The transformer architecture achieves this by utilizing the self-attention mechanism as the sole means of extracting features from given inputs. The core idea of

this mechanism is that each input instance can dynamically determine to what extent the contents of other instances are important for the interpretation of its context. This shift towards parallelization facilitates the realization of large language models (LLMs) that can process large amounts of text data in real time. The advantage of the self-attention mechanism in the context of time series analysis is that it provides each input instance with a global view of possible dependencies between it and other instances, regardless of their time lags. However, this global view comes at the expense of local sensitivity, as the self-attention mechanism does not provide information about the chronological order in which the instances occur. This lack of this information hinders the ability of transformer models to prioritize the importance of nearby input instances, which in turn leads to short-term dependencies being diluted or neglected in favor of long-term dependencies [57–59].

Despite the different approaches and design goals used in the development of the RNN family, TCN, and transformer architectures, their adoption for processing MTS data is fundamentally hindered by their inherent design for processing univariate sequences. The construction of MTS processing models using these architectures therefore typically requires the use of multiple instances of these architectures, each of which is used to process a univariate time series. The results of these instances are then fused together and passed to the output layer from which the inference is drawn. Apart from the high computational complexity required to run these instances [18,19], the fusing of the outputs of these instances can lead to a loss of critical information [20,21]. Furthermore, the use of such multiple-instance configurations can severely limit the model's ability to capture the simultaneously evolving patterns and dependencies of MTS, as the hypothesis space of each instance is limited to its individual input. This in turn undermines the holistic understanding required for making accurate predictions [22,23]. Furthermore, the use of multiple instances often leads to poor scalability, high intractability, and low interpretability and explainability [24,25].

2.2. Graph Neural Networks

Graph Neural Networks (GNNs) [26] are a class of DNNs that are specifically designed to overcome one of the major limitations faced by other DNNs when processing graph-structured data, such as molecular structures, traffic management systems, and social networks. Traditionally, such datasets were often converted into simpler representations such as vectors or images that could be interpreted by conventional DNNs such as MLP, RNNs, CNNs, and transformers. However, this conversion process often leads to the loss of crucial information, such as topological dependencies and mutual relationships, contained in the original graph structure. GNN overcomes these challenges by combining the principles of information diffusion mechanisms with the learning processes of DNNs. In this approach, the graph-structured data are mirrored in the underlying architecture of the GNN, and each node within the GNN is equipped with a dedicated processing unit that is connected to other units via information links through which the relationships contained in the data graph are aggregated. In this way, GNNs can process and analyze graph-based data structures in their native format without the need for preprocessing them. The great success that GNNs have achieved in learning sophisticated relations e.g., [60–64] has drawn attention to their application in MTS learning tasks, so several approaches have been developed for this purpose.

However, one of the main challenges in this application is to map the MTS datasets into a graph that can preserve the full range of temporal dependencies inherent in MTSs. To overcome this challenge, most of the existing works use the spatial or spectral GNN approach [65]. Spatial GNN models (e.g., refs. [66–68]) are based on the assumption that the underlying structure of the data sources from which the MTS dataset is generated is known. This allows the GNN to be constructed to resemble this structure, with each vertex representing a single data source (i.e., the univariate time series) and each link denoting the proximity measure (such as correlation, mutual information, or cosine similarity) between the vertices it connects. Spectral GNN models (e.g., refs. [69–71]), on the other hand, relax

this assumption by utilizing the principles of spectral graph theory. In spectral GNNs, the graph Laplacian and spectral decomposition are used to derive the eigenvalues and eigenvectors of the graph structure in the spectral domain, where the complex interactions of the MTS data can be effectively represented. The learning process is then applied in the spectral domain, and the resulting feature is transformed back to the time domain. A comparison between the spectral and spatial GNN shows that the main strength of the former lies in its ability to process MTS without prior knowledge of the underlying structure of the data sources, which makes it more suitable for general applications. However, this advantage is outweighed by the high computational complexity required to process MTS datasets in both the time and frequency domains.

The second major challenge in using GNN for MTS learning tasks lies in selecting the appropriate learning units at the GNN nodes that can effectively capture the temporal dependencies distributed across the graph. Various approaches and techniques have been developed to overcome this challenge. These include the use of DNN architectures known for their ability to interpret the temporal dependencies, such as RNN, LSTM, GRU, or self-attention mechanisms [72–75] at each GNN node. The main advantage of this approach is its simplicity, as each node can independently process the temporal sequence associated with it. However, this comes at the cost of increasing computational complexity and potential scalability issues, since both the temporal and spatial dependencies must be processed simultaneously. Due to these shortcomings, some works propose to segregate the methods for extracting spatial and temporal features. In these works, the structural features of the graphs are firstly processed by spatial feature extraction modules. These features are then passed to the temporal feature extraction modules to derive the overall spatial–temporal features. Although this segregation can provide a more scalable approach for processing MTS with GNN, it has several drawbacks, such as the high susceptibility to the loss of interactions between spatial and temporal features, since each of these feature extraction processes is performed separately. Moreover, using two heterogeneous feature extraction modules in tandem requires careful adjustment of their hyperparameters to achieve an optimal balance between the spatial and temporal learning rates. Failure to achieve this balance can lead to longer training times, greater difficulty in convergence of the model, or even both. Another approach that has been used to apply the GNN to MTS learning tasks is based on dynamic graphs [76,77]. In this approach, the nodes and/or edges of the graphs can be changed over time to allow the GNN to gradually capture the intricate relationships between MTSs. Dynamic GNNs are particularly useful for processing MTS datasets with highly fluctuating interactions or incomplete data points. However, the need to update the graph structure during the learning steps is not only very computationally intensive but can also increase the risk of model instability. In addition to the above approaches, some works propose the use of a hierarchical GNN architecture [78], where different GNN networks are used to learn different levels of MTS granularity. Although hierarchical GNNs can be effective in cases where an MTS has multiple levels of temporal and spatial granularity, the main challenge of this method is to partition the MTS datasets into hierarchical representations that can preserve their critical relationships and interactions.

As can be seen from the review of related work, the ability of GNN networks to learn the interactions and dependencies embedded in MTS makes these networks one of the most promising DNN architectures for processing MTS. However, finding a suitable technique that can effectively map the MTS into a graph is one of the major challenges to realizing the full potential of GNN in MTS learning tasks. Although complex networks are among the robust approaches that have proven their reliability in representing time series as graphs, little attention has been paid to their application in the context of DNN. Most of the work that has appeared in the published literature, e.g., refs. [28,79] has used complex networks as preprocessing techniques. In particular, most of the studies in this area use complex networks to convert raw time series data into graphs and then extract the topological statistics (such as cluster coefficients, path lengths, and node edge distributions)

from the graphs and feed them into the DNN models. Although this may offer some advantages in terms of reducing the dimensionality of the raw data, summarizing the complex relationships inherent in the MTS datasets as a set of topological metrics may result in a loss of critical and nuanced patterns, biasing the models towards the selected metrics, or even a combination of both. Moreover, this review shows that most works using GNNs for MTS learning tasks do not define a cohesion approach to deal simultaneously with the structural and temporal features distributed over the edges and nodes of the graphs.

This work is dedicated to filling the gaps in using GNNs for MTS by taking the graphs generated by the complex networks as input to the GNNs and developing a new graph attention mechanism that can incorporate the temporal and structural features simultaneously in the learning process. As far as the authors are aware, this is the first work to attempt such an endeavor.

3. The Proposed Model

The main contribution of this work is to develop Tensor Visibility Graph-enhanced Attention Network (TVGeAN), a novel graph autoencoder model specifically tailored for MTS learning tasks. TVGeAN combines the power of complex networks for representing time series as graphs with the strengths of GNNs for learning from graph-structured data in their native formats. The proposed model consists of two new main components: TVG and GeAN. TVG extends the capabilities of visibility graph algorithms to model the MTS datasets by representing them as weighted temporal graphs where the nodes and edges are tensors. Each node in the TVG represents the MTS observations at a particular time point, while the weights of the edges are defined based on the visibility angle algorithm. This allows the TVG graphs to preserve the temporal dependencies within each univariate time series (via tensor edges) and the mutual relationships between different time series at each time point (via tensor nodes). The second main component, GeAN, is a new graph attention mechanism that is primarily designed to seamlessly integrate the temporal dependencies of MTS datasets into the model learning process by automatically prioritizing the attention scores of graph nodes and edge features based on their temporal interactions. This elegant approach addresses the shortcomings associated with the inherent time-agnostic nature of GNNs without the need to employ additional ANN architectures such as RNNs and TCNs or to use external position encoding to facilitate the use of self-attention mechanisms. To take advantage of the proposed model, the GeAN utilizes the autoencoder architecture with sparse and variational learning units [31,32]. The main advantage of the sparse unit is to facilitate inductive learning by allowing TVGeAN to capture the knowledge contained in graphs without being constrained by their specific structures or having to implement additional techniques such as random sampling or node masking, which are typically used in other models for this purpose. The variational learning unit, on the other hand, serves to improve the predictability of the TVGeAN model by allowing it to regenerate the given graphs under noise inputs.

Figure 1 illustrates a high level of abstraction of TVGeAN, where the raw MTS datasets are run through the TVG algorithm to transform them into a set of graphs. These graphs are then processed by the first component of GEAN, the encoder, which consists of several GNN layers, each of which is designed to process the TVG graphs using the GeAN algorithm. The main purpose of the encoder is to convert the given graphs into a set of parameters of the variational posterior distribution that best fits the underlying distribution from which the data represented by the graphs were generated. The stochastic layer is used to introduce randomness into the latent spaces so that the model can account for the inherent uncertainty in the data during learning. To this end, the parameters generated by the encoder are combined with random variables drawn from standard normal distributions. The decoder component then uses the parameters generated by the stochastic layer to derive the prior distribution, from which new graphs resembling the original input are generated. The loss function used in GeAN consists of three parts: sparse loss, reconstruction loss, and Kullback–Leibler divergence. The sparse loss encourages the model to learn from

different graph structures, the reconstruction loss is used to allow the model to evaluate the closeness of the reconstructed data to the original inputs, while the KL divergence loss is used to ensure consistency between the distributions of the encoder and decoder layers. A detailed description of TVG, GeAN and the overall architecture of TVGeAN are provided in Sections 3.1–3.3, respectively.

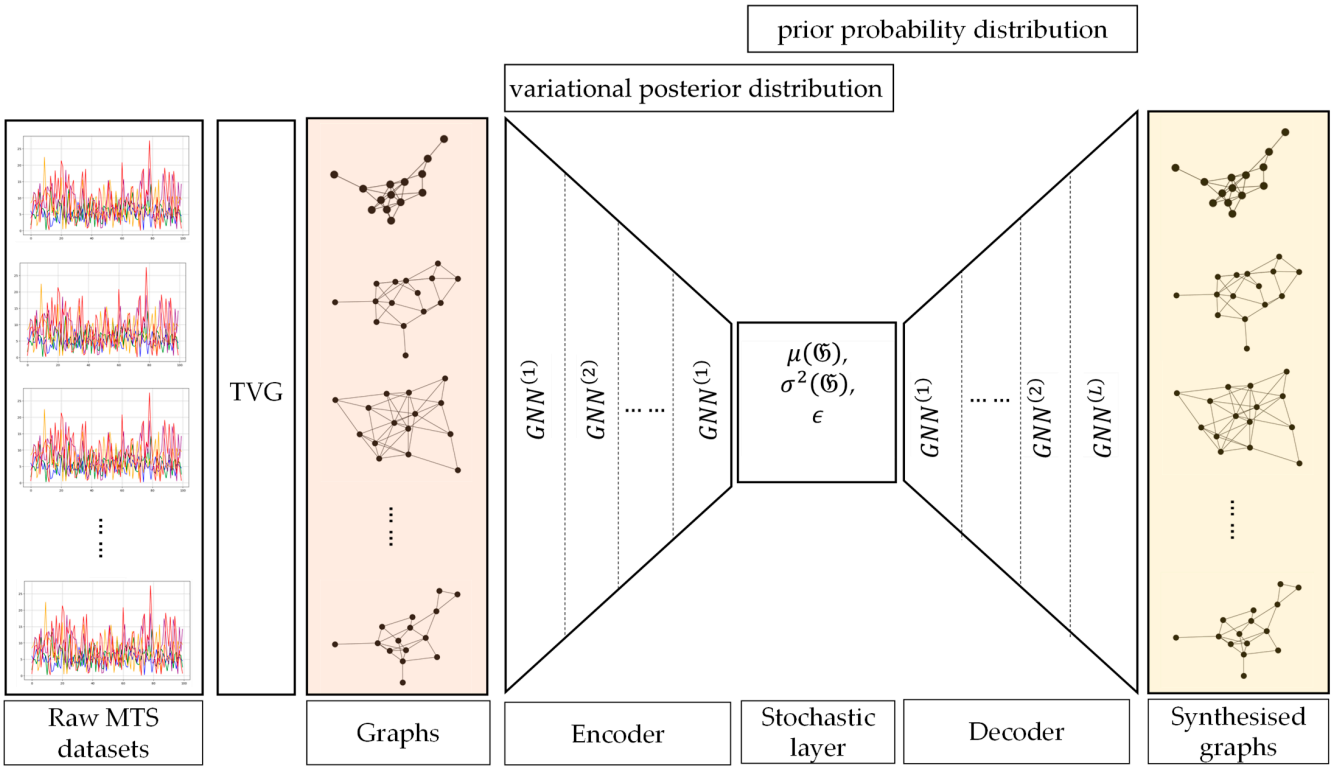


Figure 1. High-level abstraction of the proposed model. The raw MTS datasets are passed to the TVG algorithms to transform them into graphs, which are then processed by three main parts of the proposed model (the encoder, the stochastic layer, and the decoder) to finally produce the synthesized graphs.

3.1. Tensor Visibility Graph

Utilizing the principles of complex networks for modeling and analyzing time series has emerged as a powerful approach in recent decades. The core idea underlying this approach is the representation of time series as graphs so that comprehensive and rigorous graph theories and network science tools can be applied to visualize and quantify the intricate and nuanced patterns of these series. Although there are several methods for representing time series in graphs, such as correlation networks [80], information geometric networks [81], and recurrence networks [82], most of them were originally developed for representing univariate time series.

Extending the capabilities of the above methods to represent MTS has generally been pursued using two main approaches: monoplex networks e.g., refs. [83–87] and multiplexes e.g., refs. [88,89]. Monoplex-based algorithms convert the MTS into a single graph in which each node represents a single univariate time series and the edges between the nodes characterize the dependencies of the series. In contrast, multiplex-based algorithms convert the MTS into several graphs (i.e., layers), each of which represents a univariate time series. In multiplex architectures, there are usually two types of edges: the intra-layer edges, which characterize the relationships between nodes within the same layer, i.e., the relationships within the same univariate time instance, and the inter-layer edges, which define the relationships between different layers, i.e., the relationships between different univariate time instances. The comparison of monoplex and multiplex approaches shows that they

are on opposite sides of the continuum of simplicity and completeness. Monoplex methods emphasize simplicity by summarizing all information in a single diagram. However, this can come at the cost of underrepresenting nuanced dependencies between individual time series. In contrast, creating separate layers for each time series, as in multiplex methods, can preserve the full interactions of MTS, but it often leads to complicated graph structures and labyrinthine relationships.

This work addresses the challenges of representing MTS using complex networks and proposes a new method called the Tensor Visibility Graph (TVG). The TVG is designed to strike a balance between the simplicity and completeness of the MTS representation by defining the features of nodes and edges as tensors. In the TVG algorithm, each vertex represents all MTS observations corresponding to a particular time point. The edges between a pair of vertices are defined as a tensor of visibility angles, as defined in [28], between all observations of the univariate time series represented by the vertices connected by that edge. The use of TVG results in a weighted temporal graph that exactly matches the multidimensional nature of the MTS and preserves the full range of dynamics and relationships inherent in these series through a simple graph structure. It is interesting to note that the use of the visibility angle algorithm to determine the weights of the edges of TVG is due to the appealing properties of this algorithm. In the visibility angle algorithm, the weight of the edges between a pair of observations is defined as the angle at which these observations can see each other, i.e., the inverse of the tangent function of the ratio between their value difference and their temporal distance [28]. Using the visibility angle algorithm leads to a connected graph that is invariant under affine transformations [28,87–89]. This property is important for TVG, as it allows for the representation of MTS dependencies in graphs without isolating nodes. Thus, when the TVG graphs are processed by the GeAN model, the dependencies distributed in the different elements of the graph can be consolidated by following the continuous flow of information through the graph. Furthermore, the invariance of TVG graphs under affine transformations inherited from the use of the visibility angle approach increases the robustness of these graphs in mitigating the effects of variability or confounding factors introduced by the environment from which MTSs are collected. Another compelling reason for applying the visibility angle in TVG is the ability of visibility algorithms to reveal the intrinsic behavior of the underlying processes generating the MTS observations, such as stochasticity, randomness, fractality, and periodicity. This in turn allows TVG to accelerate the learning process by emphasizing the particular patterns of the underlying processes. Figure 2 depicts the underlying concept of TVG.

In order to describe the TVG algorithm precisely, let $\boldsymbol{\mathcal{X}} = \left\{x_{t,k,n}\right\}_{t=1,k=1,n=1}^{t=T,k=K,n=N}$ be a collection of N multivariant time series, each of which consists of K univariant time series, where each univariate time series comprises T observations sampled over the interval $[1, T]$. Thus, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{K\times T\times N}$. Each MTS of $\boldsymbol{\mathcal{X}}$ is passed through the TVG algorithm to convert it to a graph. Let $\boldsymbol{\mathcal{X}_m} = \left\{x_{t,k,n=m}\right\}_{t=1,k=1}^{t=T,k=K}$ be the mth MTS within the $\boldsymbol{\mathcal{X}}, 1 \leq m \leq N$, and the graph generated from the TVG due to this input is denoted by a quadruplet $\mathfrak{G}_{\boldsymbol{m}} = (\boldsymbol{\mathcal{V}}_m, \boldsymbol{\mathcal{E}}_m, \boldsymbol{\mathcal{F}}_m, \boldsymbol{\mathcal{Q}}_m)$, where $\boldsymbol{\mathcal{V}}_m = \{\mathcal{V}_{m,t}\}_{t=1}^{T} = \{t\}_{t=1}^{T}$ is the set of vertices representing the time points of the mth MTS, i.e., an order set from 1 to T. The $\boldsymbol{\mathcal{E}}_m$ is the set of all edges across $\mathfrak{G}_{\boldsymbol{m}}$. An edge is created between a pair of vertices if at least one of the visibility angles between univariant time series at the time point represented by these vertices is non-zero, i.e., $\boldsymbol{\mathcal{E}}_m \subseteq \left\{(\mathcal{V}_{m,t}, \mathcal{V}_{m,r}) \middle| \exists k \in [1, K], \theta(x_{t,k,m}, x_{r,k,m}) \neq 0\right\}$, where $\theta(\cdot)$ is defined as $\theta(x_{t,k,m}, x_{r,k,m}) = \arctan\left(\frac{x_{t,k,m}-x_{r,k,m}}{\mathcal{V}_{m,t}-\mathcal{V}_{m,r}}\right)$, and the edge between two vertices $\mathcal{V}_{m,t}$ and $\mathcal{V}_{m,t}$ is denoted by $\boldsymbol{\mathcal{E}}_{(\mathcal{V}_{m,t},\mathcal{V}_{m,r})}$. $\boldsymbol{\mathcal{F}_m}$ and $\boldsymbol{\mathcal{Q}}_m$ are the set of features of the vertex and edges in $\mathfrak{G}_{\boldsymbol{m}}$, respectively. These sets are used to enrich the learning process by signifying the distinct characteristics of vertex and edges. In this study, the feature of each vertex is defined as all the observations across the K univariate time series of $\boldsymbol{\mathcal{X}_m}$ at a time point represented by that vertex, e.g., the features of node $\mathcal{V}_{m,t}$, denoted by $\boldsymbol{\mathcal{F}}_{m,t}$ are defined as $\boldsymbol{\mathcal{F}}_{m,t} = \left\{x_{t,k,n=m}\right\}_{k=1}^{k=K}$. The features of the edge between a pair of vertices form a vector containing all the values of visibility angles between the vertices

connected by this edge, i.e., the feature of edges between vertices $\mathcal{V}_{m,t}$ and $\mathcal{V}_{m,r}$ is defined as $\mathcal{Q}_{(\mathcal{V}_{m,t},\mathcal{V}_{m,r})} = \{\theta(x_{t,k,m}, x_{r,k,m}) | \forall k \in [1,K]\}$. It is noteworthy that generating graphs for all MTS in $\mathcal{X}$ yields a set of graphs $\mathfrak{G}$, where $\mathfrak{G} = \{\mathfrak{G}_m\}_{m=1}^{N}$ constitutes the input to the encoder of the proposed model.

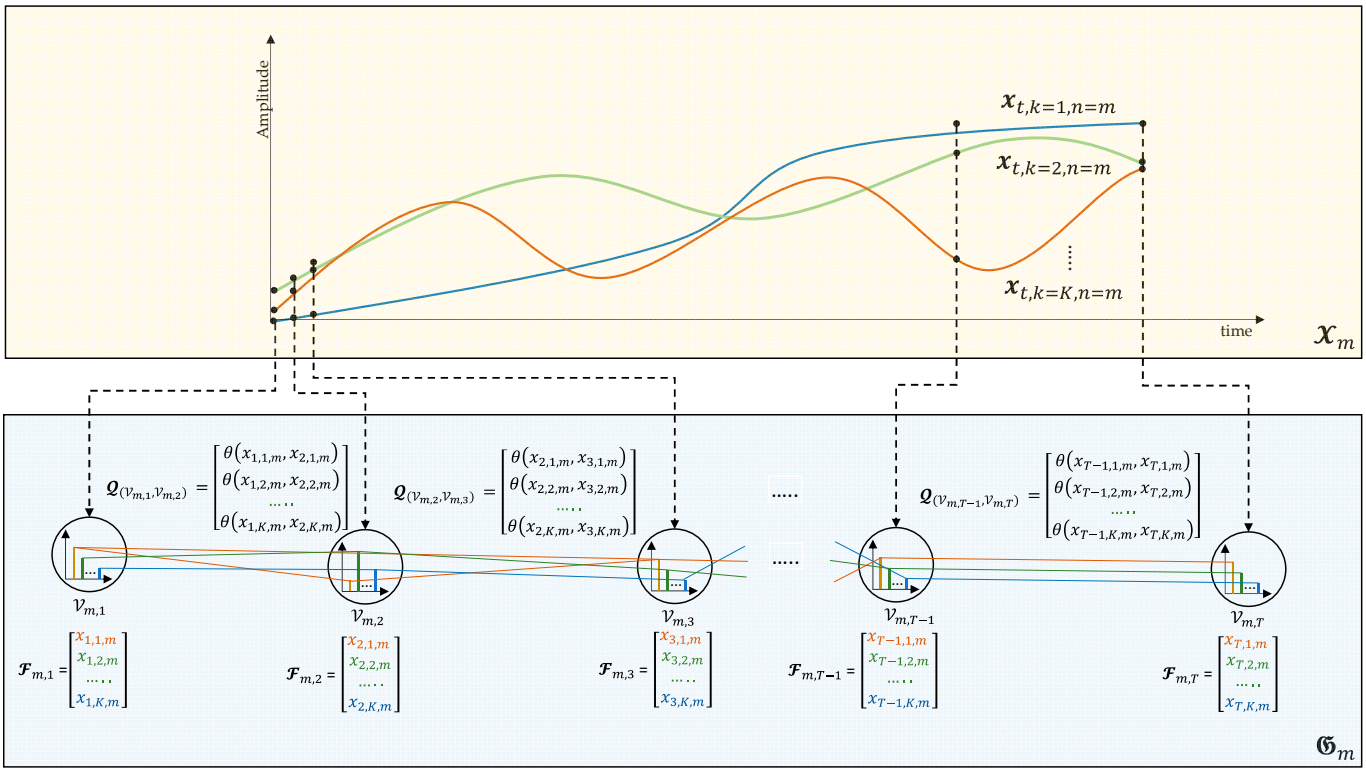


Figure 2. High-level abstraction of the TVG algorithm showing how the raw MTS is mapped into a graph.

3.2. Graph-Enhanced Attention Mehchanism

The core idea underlying Graph Neural Networks (GNNs) is to integrate the principles of information diffusion mechanisms into the learning processes of DNNs so that the complex relationships and interactions present in the original graph-structured data can be effectively captured. This is achieved by introducing aggregation functions that allow each node in the GNN to incorporate the features of its edges and neighbors into its own learning parameters (i.e., the hidden states). Although GNNs impose no constraints on the form of the aggregation function, the superior performance of the Graph Attention Network (GAT) [90] in processing large amounts of data in real-time has led to it becoming one of the most popular methods for designing aggregation functions in most modern GNNs.

Essentially, GAT was developed to overcome the shortcomings that can arise when the features of all neighbors and edges are aggregated without considering their degree of importance, as is the case when using traditional aggregation functions such as averaging, mean/sum aggregation, or max/min pooling. GAT is based on the self-attention mechanism, where the aggregator node uses its own features to query the similarities in the neighbors' features. Specifically, GAT multiplies the features of each GNN node by a learnable weighting matrix to project their features into a higher-dimensional space where their inherent patterns can be easily captured. Each pair of projected features is then concatenated together so that they can be treated in context to each other. The attention coefficient is then calculated by applying the dot product between each concatenated pair and another weighting matrix, and the results are processed using the LeakyReLU function. Since the results of these functions can vary over a wide range of values, they are passed through a monotonic normalization function such as the softmax function to produce

a probability distribution that represents the relative attention that the aggregator node should pay to the features of its neighbors. However, critical evaluations of the performance of GAT show that the "project and then concatenate" approach has some limitations. These include the high susceptibility of GAT to overfitting, which is mainly due to the duplication of model parameters through concatenation, the lack of ability of GAT to capture the full interaction between different features, as each feature is projected individually and independently of the others, and the over-smoothing of GAT's decision boundaries, which is mainly due to the homogenization of node features by concatenating a large number of neighboring features. As a result, several GAT variants have been developed to overcome these limitations using different strategies. Some notable examples are the use of multi-headed attention mechanisms at each node of the GNN [91], the restriction of aggregation to the neighbors with the highest K-attention scores [92], and the reordering of GAT operations [93]. Nevertheless, adapting these attention mechanisms to MTS learning tasks remains a challenge, since most of these attention mechanisms have been developed for time-agnostic graphs, such as bibliometric or social networks, where the edges typically have no temporal features, and the nodes represent categorical data.

Motivated by the limitations of the approaches used to design the attention mechanism of the GNN models developed for MTS, a novel mechanism called Graph-enhanced Attention Network (GeAN) is presented here. GeAN is primarily designed to seamlessly incorporate the full temporal interactions embedded in edges and nodes into the learning process and express them concisely as accurate attention scores. GeAN recognizes the fact that the features of nodes and edges in a graph provide different perspectives on the dynamics of the underlying process it represents. Therefore, the best way to capture the full range of these dynamics is to apply an effective operation that can integrate these features into a coherent yet compact representation. GeAN achieves this integration by using the outer product operation instead of the vector concatenation used in other graph attention mechanisms. The use of the outer product in GeAN is advantageous in several ways. Firstly, the ability of the outer product to represent all pairwise interactions from different data types allows GeAN to capture complex interactions of edge and node features without being affected by their inherent heterogeneous semantics and information content. Secondly, the outer product's ability to keep the representation space compact allows GeAN to process large features represented in dense graphs without compromising processing performance. Finally, the ability of the outer product to provide fine-grained quantification of the interaction allows the GeAN to derive an accurate attention score based on detailed information, which in turn can improve the predictive performance of the overall GeAN network.

Another key design principle of GeAN is to improve the learning process of the attention mechanism by unifying the landscape to which the features are mapped. GeAN achieves this by using bilinear models instead of the unitary models on which most existing graph attention mechanisms are based. The bilinear model [29,30] is a simple but effective approach that primarily aims to capture the interplay between inputs that characterize different aspects of the same learning environment. In bilinear models, all input instances are mapped to a common learning weighting matrix, and the multiplicative interaction approach is used to allow these inputs to modulate each other's contributions. This in turn facilitates the consolidation of mutual associations at nuanced levels. By using the bilinear models, GeAN eliminates the need to project each node and/or edge of the GNN individually, as is the case with other graph attention mechanisms. This can not only reduce the computational effort, but more importantly, it can also increase the representativeness of the learning weight matrix for the feature space. The use of bilinear models in GeAN can also avoid the potential loss of information that can occur when the data are projected into a higher-dimensional space and then projected back into the original space during inference. Furthermore, the use of the bilinear models in GeAN allows for straightforward processing of the MTS data at their full resolution, resulting in better recognition of the intrinsic patterns. This in turn facilitates the preservation of data integrity, improving the

accuracy of attention scores and increasing the overall effectiveness of the learning process. Another important benefit of using bilinear models in GeAN is that the susceptibility to overfitting can be reduced by eliminating the possibility of using an overly dimensional space to project the raw data.

In addition to its main strengths, GeAN also overcomes one of the fundamental shortcomings of using attention mechanisms for time-series learning tasks: the inherent temporal agnosticism of these mechanisms. This shortcoming is typically overcome by using positional encoding networks, which maintain the order of the time series and then feed the attention mechanisms with this information along with the raw input. However, the main challenge of using positional encoding is that it can increase the complexity of the aggregation function and limit the scalability of the overall models. GeAN provides a more elegant solution by integrating the temporal information of the MTS directly into the core learning process of the attention mechanism itself. GeAN obtains the temporal information by simply subtracting the identifiers of the vertices of the graph generated by the proposed Temporal Visibility Graph (TVG) algorithm, thanks to its ability to represent the MTS as a highly informative temporal graph. The result of this subtraction is normalized by dividing it by the maximum number of samples in the dataset to ensure that the temporal information makes a meaningful contribution to the attention score without dominating it. By using signed time differences, GeAN can learn to distinguish between past and future behavior, which in turn improves the understanding of the dynamic evolution of the MTS datasets. Although this may lead to negative learnable weights, the LeakyReLU activation function, commonly used in attention mechanisms, can effectively handle these values by controlling the negative slope.

To describe how the GeAN algorithm works, let $\mathrm{GNN}^{(l)}; 1 \leq l \leq L$ be the lth layer of the encoder component of the proposed model, where L is the total number of encoder layers. The input layer of the encoder is fed by the graph generated by the TVG algorithm; thus, it is defined as $\mathrm{GNN}^{(1)} = \left(\sqsubseteq^{(1)}, \urcorner^{(1)}, \{^{(1)}, \mathbf{\amalg}^{(1)}\right)$, where $\sqsubseteq^{(l)}$, $\urcorner^{(l)}$, $\{^{(l)}$, and $\mathbf{\amalg}^{(l)}$ are the sets of nodes, the edges, the hidden states of the nodes, and the hidden states of the edges of the $\mathrm{GNN}^{(l)}$, respectively. The values of these sets of the model that is trained on the mth MTS are thus $\sqsubseteq^{(1)} = \boldsymbol{\mathcal{V}}_m$, $\urcorner^{(1)} = \boldsymbol{\mathcal{E}}_m$, $\{^{(1)} = \boldsymbol{\mathcal{F}}_m$, and $\mathbf{\amalg}^{(1)} = \boldsymbol{\mathcal{Q}}_m$. Based on the stated description, the attention coefficient between an aggregating node t and its neighbor r in $\mathrm{GNN}^{(1)}$ is calculated based on their features, denoted by $\{_r^{(1)}$ and $\{_t^{(1)}$, respectively, and the features of the edge connecting them, denoted by $\mathbf{\amalg}_{tr}^{(1)}$, as follows:

$$a_{tr}{}^{(1)} = \mathrm{LeakyReLU}\Big((\boldsymbol{W}_1)^{\mathrm{T}}\Big(\Big[\{_r^{(1)} \bigotimes \mathbf{\amalg}_{tr}^{(1)}\Big]\boldsymbol{W}_2\Big[\{_t^{(1)} \bigotimes \Delta_{t\to r}\Big]\Big)\Big) \tag{1}$$

where $\otimes$ denotes the outer product and $\Delta_{t\to r}$ is the time difference between nodes t and r, which is calculated as $\Delta_{t\to r} = \left(\mathcal{V}_{m,t} - \mathcal{V}_{m,r}\right) / \left\|\sqsubseteq^{(1)}\right\|$, where $\left\|\sqsubseteq^{(1)}\right\|$ denotes the total number of vertices, which corresponds to the total number of observations of the given MTS. $W_1 \in \mathbb{R}^{K\times 1}$ and $W_2 \in \mathbb{R}^{K\times K}$ are the learnable weighting matrices that are shared by all layers of the proposed model; hence, their dimensions are set according to the dimension of the features of the nodes and edges as defined by the TVG graphs, i.e., the number of univariate time series forming the MTS. It is worth mentioning that $\boldsymbol{W}_1$ is used to derive the attention coefficient as a scalar value, while the bilinear weighting matrix $\boldsymbol{W}_2$ is used to quantify the temporal interactions contained in the features of nodes and edges. Consequently, the attention score between nodes t and r within the $\mathrm{GNN}^{(1)}$ is computed by applying the softmax function to the attention coefficients:

$$\alpha_{tr}{}^{(1)} = \frac{\exp\left(a_{tr}{}^{(1)}\right)}{\sum_{s\in\mathcal{N}_t}\exp\left(a_{ts}{}^{(1)}\right)} \tag{2}$$

where $\mathcal{N}_t$ is the set of all neighbors of node t. Consequently, the hidden state of node t and the edge connecting it to its neighbor r in the second layer are calculated as follows:

$$\{_t^{(2)} = \mathrm{ReLU}\left(\sum_{s\in\mathcal{N}_t} {\alpha_{ts}}^{(1)}\ \boldsymbol{W}^{(1)}\{_s^{(1)}\right) \tag{3}$$

$$\mathbf{\Pi}_{tr}^{(2)} = \mathrm{ReLU}\left({\alpha_{tr}}^{(1)}\boldsymbol{W}^{(1)}\ \ \mathbf{\Pi}_{tr}^{(1)}\right) \tag{4}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d^{(2)}\times d^{(1)}}; d^{(2)} < d^{(l)}$ is a cross-layer weighting matrix whose main goal is to provide a gradual dimensional reduction of the features space between subsequent layers. Then, the general form for updating the hidden states of the nodes and edges between layers can be expressed by substituting (1) and (2) into (3) and (4):

$$\{_t^{(l+1)} = \mathrm{ReLU}\left(\sum_{s\in\mathcal{N}_t} \frac{\exp\left(\mathrm{LeakyReLU}\left((\boldsymbol{W}_1)^{\mathrm{T}}\left(\left[\{_s^{(l)} \otimes \mathbf{\Pi}_{ts}^{(l)}\right]\boldsymbol{W}_2\left[\{_t^{(l)} \otimes \Delta_{t\to s}\right]\right)\right)\right)}{\sum_{j\in\mathcal{N}_t}\exp\left(\mathrm{LeakyReLU}\left((\boldsymbol{W}_1)^{\mathrm{T}}\left(\left[\{_j^{(l)} \otimes \mathbf{\Pi}_{tj}^{(l)}\right]\boldsymbol{W}_2\left[\{_t^{(l)} \otimes \Delta_{t\to j}\right]\right)\right)\right)}\ \boldsymbol{W}^{(l)}\{_s^{(l)}\right) \tag{5}$$

$$\mathbf{\Pi}_{tr}^{(l+1)} = \mathrm{ReLU}\left(\frac{\exp\left(\mathrm{LeakyReLU}\left((\boldsymbol{W}_1)^{\mathrm{T}}\left(\left[\{_r^{(l)} \otimes \mathbf{\Pi}_{tr}^{(l)}\right]\boldsymbol{W}_2\left[\{_t^{(l)} \otimes \Delta_{t\to r}\right]\right)\right)\right)}{\sum_{s\in\mathcal{N}_t}\exp\left(\mathrm{LeakyReLU}\left((\boldsymbol{W}_1)^{\mathrm{T}}\left(\left[\{_s^{(l)} \otimes \mathrm{\Pi}_{ts}^{(1)}\right]\boldsymbol{W}_2\left[\{_t^{(l)} \otimes \Delta_{t\to s}\right]\right)\right)\right)}\boldsymbol{W}^{(l)}\ \ \mathbf{\Pi}_{tr}^{(l)}\right) \tag{6}$$

3.3. Overall Architecture of the TVGeAN

The proposed model, like other Variational AutoEncoder (VAE) architectures, consists of three main components: the encoder, the reparameterization trick (also known as the stochastic layer), and the decoder. The encoder of the proposed model consists of L layers arranged in a cascaded architecture, where the output of each GNN is fed into the input of the next layer, creating a pipeline through which the set of the input graphs is mapped to the parameters of the latent distribution. Each layer of the encoder is designed to process the graph generated by the tensor visibility graph algorithm, described in Section 3.1, and all nodes and edges within a GNN uses the GeAN mechanism defined in Section 3.2 to update their hidden states.

Following the notations used in these two subsections, the GNN at the lth layer is described as $\mathrm{GNN}^{(l)} = \left(\sqsubseteq^{(l)}, \urcorner^{(l)}, \{^{(l)}, \mathbf{\Pi}^{(l)}\right)$, where $\sqsubseteq^{(l)}$, $\urcorner^{(l)}$, $\{^{(l)}$, and $\mathbf{\Pi}^{(l)}$ are the sets of nodes, the edges, the hidden states of the nodes, and the hidden states of the edges of the $\mathrm{GNN}^{(l)}$, respectively.

Based on the above, the output of the last layer of the encoder due to the samples drawn from the training dataset $\mathfrak{G}$ can be expressed as follows:

$$y = \mathrm{GNN}^{(1)}\left(\{^{(1)}; \mathbf{\Pi}^{(1)}\right) \circ\ \mathrm{GNN}^{(2)}\left(\{^{(2)}; \mathbf{\Pi}^{(2)}\right) \circ \ldots \circ \mathrm{GNN}^{(L)}\left(\{^{(L)}; \mathbf{\Pi}^{(L)}\right) \tag{7}$$

where $\circ$ is the function composition operator. The output of the last layer is passed to two MLP networks to generate the mean and standard deviation of the latent distribution: $\mathrm{MLP}_{\boldsymbol{\mu}}(y) = \boldsymbol{\mu}(\mathfrak{G})$ and $\mathrm{MLP}_{\sigma}(y) = \boldsymbol{\sigma}^2(\mathfrak{G})$. The output of these two layers can be used to generate the variational posterior distribution, which can be expressed as follows:

$$q_{\boldsymbol{\phi}}(y|G) = N\left(y; \boldsymbol{\mu}(\mathfrak{G}), \boldsymbol{\sigma}^2(\mathfrak{G})\boldsymbol{I}\right) \tag{8}$$

where $N(\cdot)$ denotes a normal distribution with a mean of $\boldsymbol{\mu}(\mathfrak{G})$ and covariance matrix $\boldsymbol{\sigma}^2(\mathfrak{G})\boldsymbol{I}$ and is parameterized by weighting and biasing of all layers in the encoder, i.e., $\boldsymbol{\phi} = \left\{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}^{(l)} \middle| 1 \le l \le L\right\}$. The reparameterization trick layer receives $\boldsymbol{\mu}(\mathfrak{G})$

and $\sigma^2(\mathfrak{G})$ and combines them with random variables $\epsilon \sim N(0, I)$ using elementwise multiplication, denoted by $\odot$, to introduce the randomness in the encoder networks.

$$z = \boldsymbol{\mu}(\mathfrak{G}) + \boldsymbol{\sigma}^2(\mathfrak{G}) \odot \epsilon \tag{9}$$

The decoder component of the proposed model has the same architecture as the encoder component, but in reverse order. The first layer of the decoder receives the latent parameter variables generated by the stochastic layer via MLP, while the output layer of the decoder is a GNN that generates a graph ($\hat{\mathfrak{G}}$) that resembles the graphs fed into the input layer of the encoder. Moreover, the decoder network uses the values of these graphs and learning matrix, denoted by $\boldsymbol{\omega}$, to generate the prior probability distribution, i.e., $p_{\boldsymbol{\omega}}(\hat{\mathfrak{G}}|z) = N(\hat{\mathfrak{G}}; \boldsymbol{\mu}(\hat{\mathfrak{G}}), \boldsymbol{\sigma}^2(\hat{\mathfrak{G}})\boldsymbol{I})$.

In order to provide the model with the ability to process graphs with different structures during the training phases and to generalize the knowledge it gains during this phase for unseen samples during the testing phase, a sparse learning unit is employed. The main aim of this unit is to alter the nodes and/edges within and across the layers so that the model does not memorize the graph structure but rather encourage the model to learn the specialized and discriminating features from nodes and/or edges. Another main advantage of employing the sparse learning unit is that can provide the model with an adaptive method for dimensionality reduction by amputating the edges and/or nodes whose contributions to the learning tasks are neglectable. It is of interest to mention that other GNN models employ a fixed length dimension reduction; that is, the dimensionality of the weighing matrix in each layer is predefined and stays fixed during the training and testing phase, regardless of its impact on the prediction performance. This study proposes an adaptive approach based on determining the effective contributions of the individual hidden states of the nodes and edges to the performance of the whole network:

$$\mathcal{L}_{\text{sparse}} = \sum_{l=1}^{L} \sum_{i \in \sqsubseteq^{(l)}} \frac{\left\| \{_i^{(l)} \right\|_p}{\sum_{l=1}^{L} \sum_{j \in \sqsubseteq^{(k)}} \left\| \{_j^{(k)} \right\|_p} + \sum_{l=1}^{L} \sum_{e(i,j) \in \urcorner^{(l)}} \frac{\left\| \mathbf{II}_{ij}^{(l)} \right\|_p}{\sum_{k=1}^{L} \sum_{e(m,n) \in \urcorner^{(k)}} \left\| \mathbf{II}_{mn}^{(k)} \right\|_p} \tag{10}$$

In additional to the sparse loss function, the model employs the reconstruction and Kullback–Leibler (KL) divergence, where the former is used to measure the extent to which the decoder is able to reconstruct the given input based on the latent parameters passed to it from the stochastic layer, which is expressed as $\mathcal{L}_{\text{recon}} = -\mathbb{E}_{q_{\boldsymbol{\phi}}(Y|G)}\left[\log p_{\boldsymbol{\omega}}(\hat{\mathfrak{G}}|z)\right]$. The KL divergence loss is used to regularize the latent and prior distributions to keep them close to each other, i.e., $\mathcal{L}_{\text{KL}} = \frac{1}{2} D_{KL}(q_{\boldsymbol{\phi}}(Y|G)||p(z))$. Then, the total losses of the model can be expressed as follows:

$$\mathcal{L}_{\text{sparse}} = \mathcal{L}_{\text{sparse}} + \beta \mathcal{L}_{\text{KL}} + \gamma \mathcal{L}_{\text{recon}} \tag{11}$$

where β and γ are the hyperparameters that control the contributions of $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{recon}}$, respectively, with respect to the total losses.

4. Results

The proposed model was implemented using Python 3.12.5 and PyTorch 2.4.0 on a single-node machine running Ubuntu 24.04 LTS on a 14th-generation Intel Core processor (up to 6.0 GHz, 8P + 16E cores) with 62 GB RAM and an NVIDIA GeForce RTX 4090 GPU. A symmetric architecture with five GNN layers in both the encoder and decoder was used for supervised and unsupervised learning tasks. The value of the negative slope of the LeakyReLU activation function was set to 0.2 after experimenting with different values. For the supervised learning tasks, the encoder component ended up with three fully connected layers using the ReLU activation function, while for the unsupervised tasks, the full encoder–decoder model was used. The model was trained with backpropagation and the Adam optimizer, and the initial learning rate was set to 0.001. In addition, a

learning rate scheduler was used, where the learning rate was reduced by a factor of 0.1 every 10 epochs if no improvement was observed during the training phases. A batch size of one-tenth the size of the training dataset was chosen to achieve a balance between computational efficiency and generalization performance. The following statistical metrics were used to quantify the performance of the proposed model: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), scores for the supervised learning tasks, and the precision, recall, and F1 score for the unsupervised tasks. These metrics were selected because they were used by the providers of the datasets and are widely used in comparable work, which facilitates comparisons.

4.1. Dataset Description

With the aim of providing a comprehensive evaluation of the proposed model from different perspectives and with different learning tasks, four different datasets are used in this study. The dataset C-MPASS [35] was used for regression, and the datasets SMD [36], MSL, and SMAP [37] were used for anomaly detection. Table 1 summarizes their main features, where the dimension column contains the number of measurements recorded at each time point in the dataset, the number of entities is the number of systems whose behavior was measured, the size is the number of time points of the MTSs, and the anomaly ratio of SMD, MSL, and SMAP is the percentage of MTSs exhibiting anomalies compared to all MTSs. Each of these datasets is divided into two parts, training and testing, as specified by the dataset providers.

Table 1. Characteristics of the datasets used to assess the model.

Learning Task	Dataset	Sub-Dataset	Dimension	Train		Test		Anomality Ratio (%)
				Entities	Size	Entities	Size	
Regression	C-MPASS	FD001	26	100	20,631	100	13,096	
		FD002	26	260	53,759	259	33,991	
		FD003	26	100	24,720	100	16,596	
		FD004	26	249	61,249	248	41,214	
AD	SMD	-	38	28	708,405	28	708,420	4.16
	MSL	-	55	55	58,317	55	73,729	10.72
	SMAP		25	25	135,183	25	427,617	13.13

The Commercial Modular Aero-Propulsion System Simulation (C-MPASS) was developed by the National Aeronautics and Space Administration (NASA) to provide a reliable benchmark for evaluating the Remaining Useful Life (RUL) of various large commercial turbofan engines. The C-MPASS dataset consists of four subsets: FD001, FD002, FD003, and FD004, each of which contains a collection of MTSs representing the sensory measurements of multiple operating cycles of several engines with different configurations, operating conditions, and failure modes. Each of these four subsets is divided into two groups: training and testing. The training datasets contain a set of MTSs of multiple engines operating cycles, starting with unknown initial health states and ending with the cycle in which the engine is declared to have failed. The test dataset, on the other hand, contains subsets of operating cycles of several engines in fully functional and faulty conditions. The learning task of this dataset is then to determine the first cycle in which each engine in the test dataset develops a fault (i.e., RUL), based on the knowledge gained from processing the training dataset. The ground truth of the RUL values of all engines in the test dataset is included in the C-MPASS repository to evaluate the prediction accuracy of the model.

The Server Machine Dataset (SMD) was introduced by the authors of [36] to evaluate the performance of their proposed anomaly detection algorithm and has since become a highly cited dataset in the field. SMD is a set of key performance indices of server machines (e.g., CPU load, network utilization, and memory usage) collected over a period of five weeks from the data centers of an internet company. This dataset is split into two equal

subsets for training and testing purposes, with each instance representing an MTS. The length of the anomalies of each instance and their modes were also provided in the SMD repository. The learning tasks of this dataset are to predict whether a particular MTS in the test dataset is normal and to indicate the expected length.

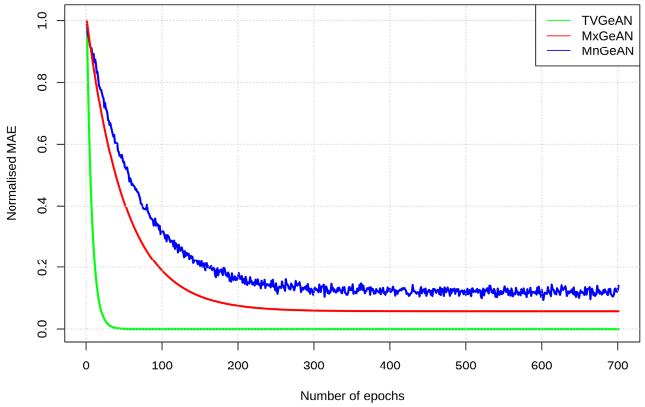
The Soil Moisture Active Passive (SMAP) dataset [37] was introduced by NASA to characterize the soil moisture and freeze–thaw conditions of the Earth's land surface. These data were collected by the SMAP satellite, which combines active radar and passive radiometer instruments to provide high-resolution and accurate measurements. The SMAP dataset contains MTS of various environmental parameters, including soil moisture, temperature, and vegetation conditions. The dataset contains labeled anomalies that represent deviations from typical soil moisture and environmental conditions. The Mars Science Laboratory (MSL) dataset [37] was also collected by NASA to monitor and evaluate the performance of the Curiosity rover, which is part of the Mars Science Laboratory mission. The data were collected from various sensors on the rover and provide valuable insights into the Mars environment and the operational status of the rover, such as the rover's atmospheric conditions and thermal and energetic systems. Similar to SMAP, MSL contains a collection of MTSs in which the anomalies are labelled.

4.2. Effects of TVG on the Learning Performance of the Model

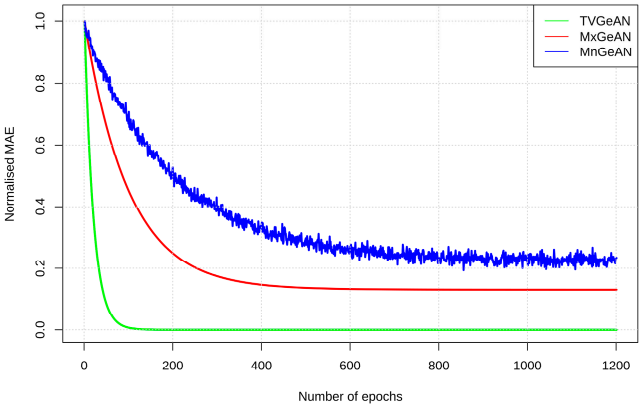
The first evaluation of the proposed model aims to quantify the impact of the tensor visibility graph algorithm on the model's learning ability. To this end, two variants of the proposed model were developed, MnGEAN and MxGEAN, which differ from the proposed TVGeAN only in the algorithms used to convert the MTS into graphs. MnGEAN uses the correlation network algorithm to represent the MTS [94,95]. In this algorithm, each univariate time series is represented by a vertex, while the edges between these vertices are defined as the contemporaneous correlations between the univariate time series represented by the connected vertices. The multiplex visibility graph algorithm, on the other hand, represents each univariate time series as a separate visibility graph and defines the edges between the different layers based on their mutual information [69]. Seven models of MnGEAN, MxGEAN, and TVGeAN were created, each of which was trained on one of the datasets listed in Table 1. The normalized errors of these models at the end of each learning step (epoch) during the training phases were recorded and plotted as shown in Figure 3.

Figure 3 shows that the error curves of MnGeAN, MxGeAN, and TVGeAN follow different characteristics. For example, it can be seen that the error curves of TVGeAN decrease sharply in the early epochs and then plateau at a quasi-negligible value. These characteristics can be attributed to the ability of the TVG algorithm to accurately capture the temporal dependencies inherent in the MTS dataset and represent them concisely as tensor edges and nodes. This simple yet comprehensive representation allows the learning process (i.e., GeAN) to effectively distil the knowledge contained in the different datasets and firmly consolidate it during the learning steps, which in turn drives the convergence of the TVGeAN models towards the zero-error regions. Figure 3 also shows that the error curves of the MnGeAN models exhibit a high degree of fluctuation compared to the other models. This behavior implies that the graphs generated by the monoplex algorithm are not able to represent the important information of the MTS datasets in a way that is useful for the GeAN model. Thus, whenever GeAN attempts to acquire knowledge about the MTSs and their interactions in order to generate the latent features, the poor results of the loss function lead to a repetition of the learning cycles. This repetition leads to a fluctuation of the error curves and prevents MnGeAN from approaching a stable learning curve. The comparison of the error curves of MnGeAN with those of MxGeAN shows that the latter has a slower decay rate with lower error values. These characteristics can be attributed to the fact that the multiplex algorithm provides comprehensive information about the MTS datasets and their interactions via multilayer graphs and the interlayer-mutual information modeling approach. This comprehensive information enriches GeAN's learning process and enables it to gain insightful knowledge about the underlying relationships governing

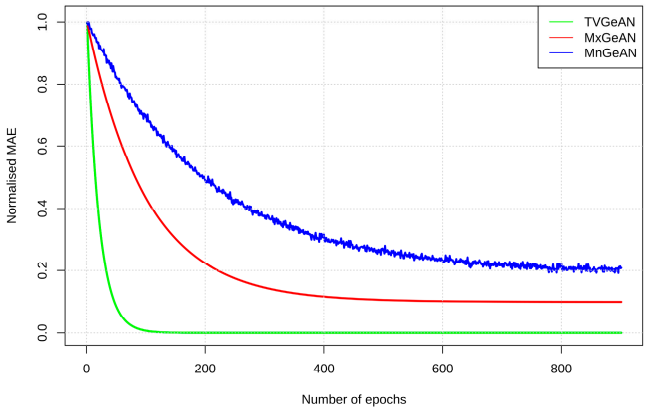
the MTS dataset, which in turn leads to a reduction in training errors over time. However, this comprehensive information and its distribution over multiple graphs imposes great overhead on the GeAN, as it has to simultaneously learn from different inputs and resolve all possible inconsistencies between them in order to generate coherent latent features characterizing the given MTS dataset. This additional overhead is clearly reflected in the low decay rate of the error curves of the MxGEAN models and the level-off of these curves at non-negligible error values.



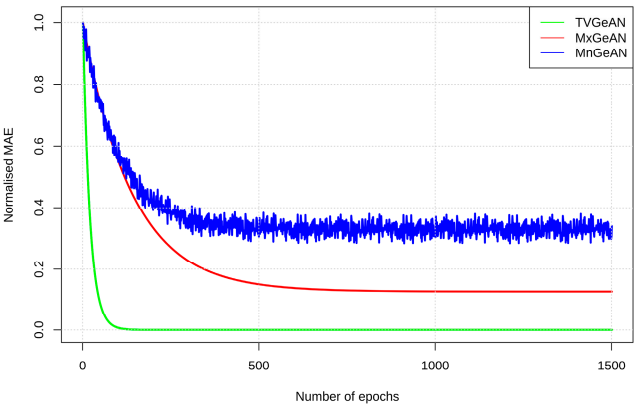
(**a**)



(**b**)

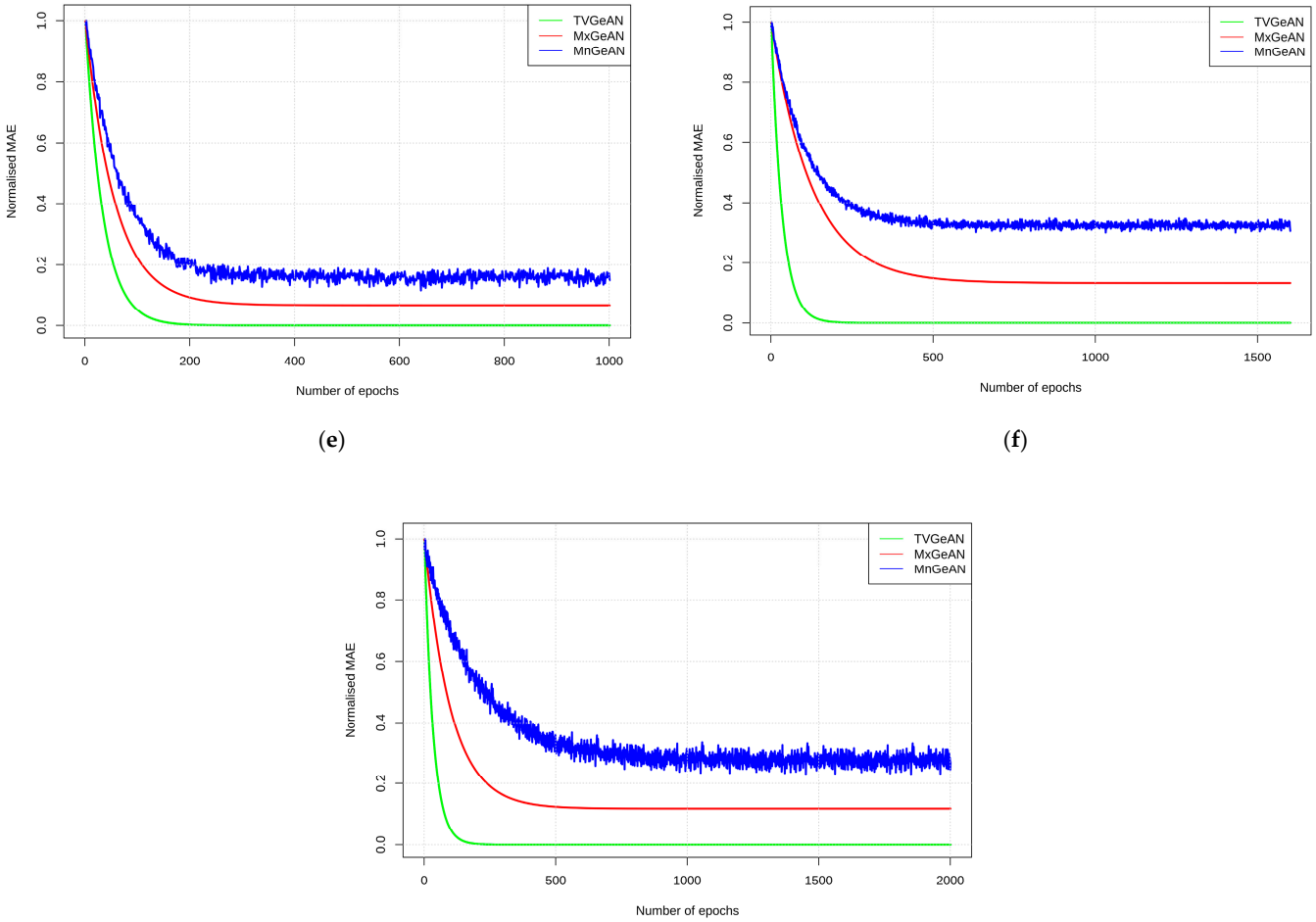


(**c**)



(**d**)

Figure 3. *Cont.*



(**e**) (**f**)

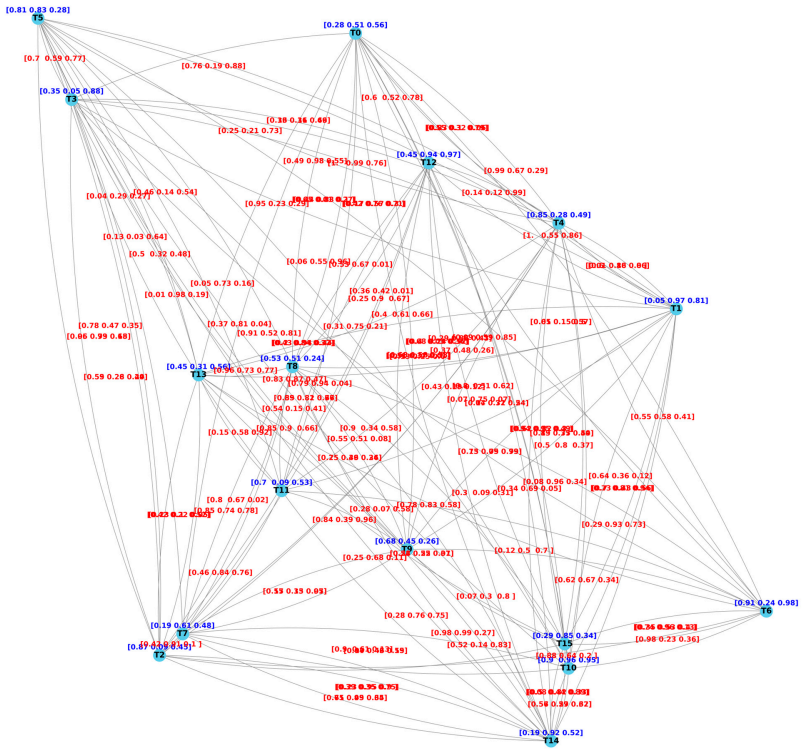
(**g**)

Figure 3. Normalized MAE vs. number of epochs for the seven datasets: (**a**) FD001, (**b**) FD002, (**c**) FD003, (**d**) FD004, (**e**) SMD, (**f**) MSL, (**g**) SMAP.

The comparison of the learning behavior of the TVGeAN, MnGeAN, and MxGeAN models for different datasets reveals some important characteristics of the representation performance of the TVG, monoplex, and multiplex algorithms. For example, it can be seen that the convergence rate of MxGeAN trained on the FD004 dataset is much lower than the convergence rate for the same model trained on the FD001 dataset. It can also be seen that the error curve of the MnGeAN model trained on the FD004 dataset shows stronger fluctuations than its counterpart trained on FD001. These characteristics can be attributed to the fact that FD004 was simulated under six operating conditions and two fault modes, while the FD001 dataset was simulated under a single operating condition and a single fault mode. Therefore, the FD004 dataset contains more dynamic relationships and temporal dependencies than the FD001 dataset. The representation of these sophisticated interactions by the multiplex algorithm leads to complex graphs with large numbers of edges within and between the nodes of the different graphs. This in turn complicates the learning process of MxGeAN and leads to a low convergence speed. On the other hand, since the monoplex algorithm expresses all interactions as contemporaneous correlations between the univariate time series, the representation of the sophisticated interactions of FD004 with this algorithm leads to the creation of a graph whose weighted edges are almost equal (i.e., close to unity). This causes the features of the graph generated by Monoplex to have a high degree of multicollinearity, which in turn causes GeAN to experience difficulty in recognizing the subtle patterns and providing consistent results. By the same argument, we can state that the differences between the error curves of the datasets (e.g., SMD, MSL,

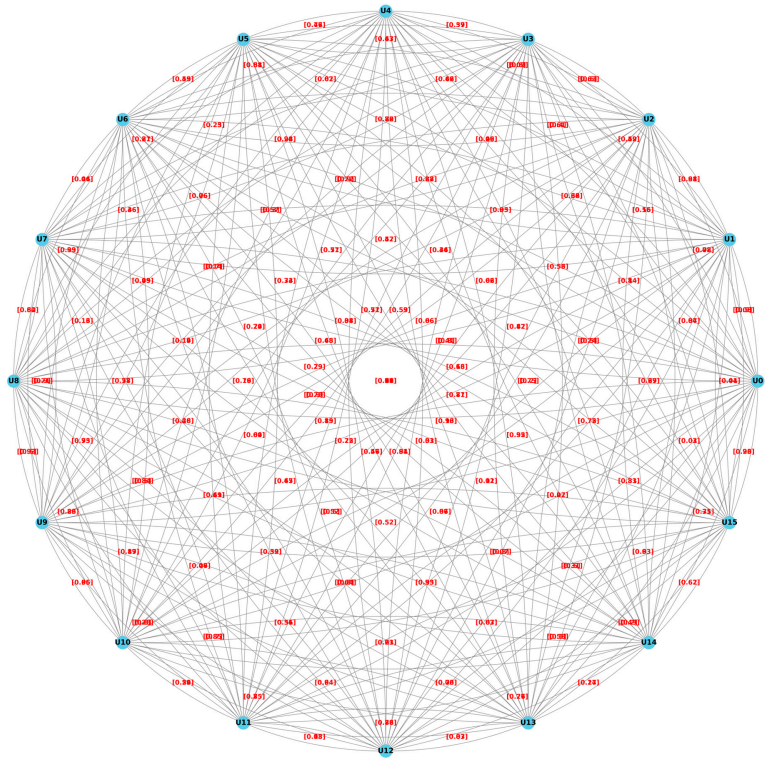
and SMAP) used to evaluate the anomaly detection learning tasks can be interpreted in terms of their anomaly ratio. In the case of anomalies, a higher ratio leads to poorer results, and vice versa. It is worth noting that the high stability of TVGeAN on different datasets is due to the fact that the graphs generated by the TVG algorithm can represent the full spectrum of dynamic interactions in the MTS datasets via the tensor structures. In doing so, the tensor nodes can capture the interactions between the measures of the univariate time series corresponding to the same data points, and the tensor edges can capture the dynamics of the MTS dataset as it evolves over time. This in turn leads to more informative graphs than those generated by monoplex and less complexity compared to multiplex. Another important reason for the better performance of TVGeAN is the adoption of the visibility angle to define the weighting of edges between different nodes. This enables TVG to effectively represent the intrinsic behavior of the underlying processes that generate the MTS datasets.

To better illustrate the graphs generated by the TVG, monoplex, and multiplex algorithms, Figure 4 shows snapshots of the graphs generated by these algorithms from a subset of the SMD dataset. In Figure 4a, the blue and red labels indicate the tenor nodes and the edges generated by the TVG algorithm, respectively. In Figure 4b, the labels associated with the link show the correlation values between the univariate time series calculated by the monoplex algorithm. Finally, in Figure 4c, the different layers generated by the multiplex algorithm are represented by different node colors, and the labels of the edges show their intra- or inter-weighting.

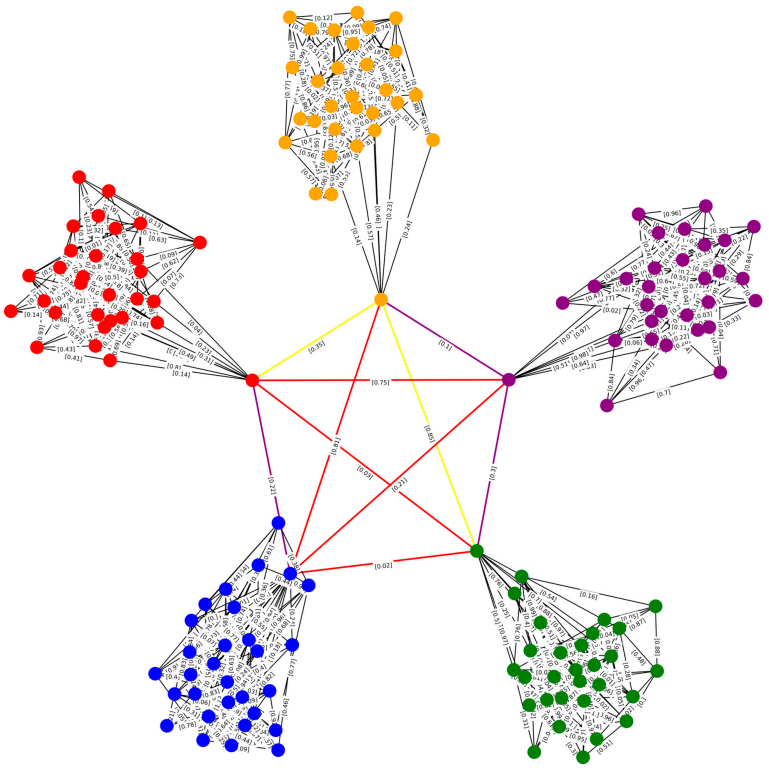


(**a**)

Figure 4. *Cont.*



(**b**)



(**c**)

Figure 4. Snapshots of graphs generated by the (**a**) TVG, (**b**) monoplex, and (**c**) multiplex algorithms for a subset of SDM.

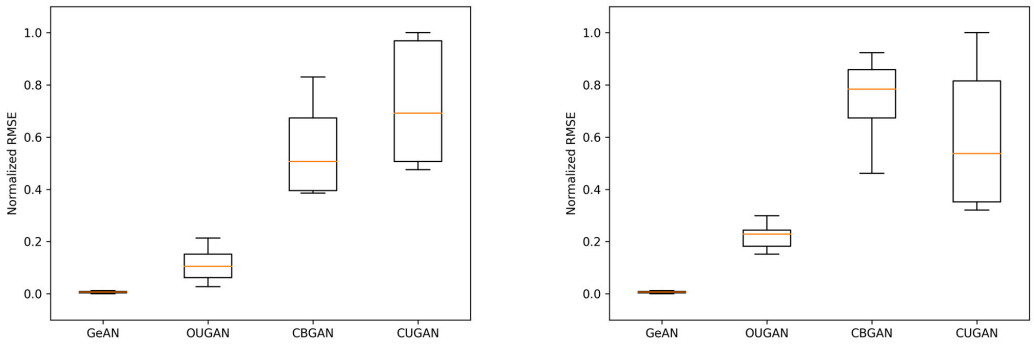
4.3. Performance Evaulations of the GeAN Mechanism

The second evaluation of the proposed model aims to assess the impact of the underlying assumptions made in the development of the Graph-enhanced Attention Network (GeAN) mechanism on its performance. Recall that GeAN is based on the use of the outer product operation to manifest the pairwise features of edges and nodes and on the bilinear model to calculate the attention coefficient. For the purposes of this evaluation, we replace the outer product and the bilinear model with the concatenation operation and the unitary attention mechanism, which are widely used to develop graph attention mechanisms. Therefore, three variants of GeAN were created: (i) OUGAN, in which the unitary attention mechanism is used instead of the bilinear models used by GeAN, while the outer product remains unchanged, (ii) CBGAN, in which the outer product of GeAN is replaced by the concatenation operations, while the bilinear model remains unchanged, and finally, (iii) CUGAN, in which both the outer product and the bilinear models of GeAN are replaced by the concatenation operator and the unitary attention mechanism, respectively. The mathematical descriptions of these attention mechanisms are given in Table 2.

Table 2. Mathematical equations of attention coefficients of the OUGAN, CBGAN, and CUGAN mechanisms.

Mechanism	Attention Coefficient
OUGAN	$a_{tr}{}^{(1)} = \left(\boldsymbol{W}_1{}^{(l)}\right)^{\mathrm{T}} \mathrm{LeakyReLU}\left(\boldsymbol{W}_2{}^{(l)}\{_t^{(l)} \otimes \boldsymbol{W}_2{}^{(l)}\boldsymbol{\Pi}_{tr}^{(l)} \otimes \boldsymbol{W}_2{}^{(l)}\{_r^{(l)}\right);\ \boldsymbol{W}_1{}^{(l)} \in \mathbb{R}^{d^{(l)}\times d^{(l)}\times d^{(l)}};\ \boldsymbol{W}_2{}^{(l)} \in \mathbb{R}^{d^{(l)}\times d^{(l)}}$
CBGAN	$a_{tr}{}^{(1)} = \left(\boldsymbol{W}_1{}^{(l)}\right)^{\mathrm{T}} \mathrm{LeakyReLU}\left([\{_t^{(l)} \Vert \boldsymbol{\Pi}_{tr}^{(l)}] \boldsymbol{W}_2{}^{(l)}\{_r^{(l)}\right);\ \boldsymbol{W}_1{}^{(l)} \in \mathbb{R}^{2d^{(l)}\times 1};\ \boldsymbol{W}_2{}^{(l)} \in \mathbb{R}^{1\times d^{(l)}}$
CUGAN	$a_{tr}{}^{(1)} = \left(\boldsymbol{W}_1{}^{(l)}\right)^{\mathrm{T}} \mathrm{LeakyReLU}\left(\boldsymbol{W}_2{}^{(l)}\{_j^{(l)} \Vert \boldsymbol{W}_2{}^{(l)}\boldsymbol{\Pi}_{ij}^{(l)} \Vert \boldsymbol{W}_2{}^{(l)}\{_i^{(l)}\right);\ \boldsymbol{W}_1{}^{(l)} \in \mathbb{R}^{3d^{(l)}\times 1};\ \boldsymbol{W}_2{}^{(l)} \in \mathbb{R}^{1\times d^{(l)}}$

To ensure a fair comparison, three models were created for each dataset listed in Table 1, each of these models using one of the attention mechanisms listed above. All of these models, as well as the models using the GeAN mechanism described in Section 3.2, were fed with the same graphs generated by the TVG algorithm, and all of these models were created using the same architecture described in Section 3.3. To ensure a comprehensive evaluation of all these models, the 10-fold cross-validation strategy was used. In this approach, each dataset was divided into 10 equally sized sub-datasets, and each of these sub-datasets was used to train the models, while the remaining 9 sub-datasets were used to test the models. To ensure that each sub-dataset was used in both the training and testing phases, a rotation was performed between the training and testing sub-datasets. The normalized accuracy of all the models corresponding to all the datasets was recorded at the end of each testing phases and then visualized in a boxplot as shown in Figure 5.



(a) **(b)**

Figure 5. *Cont.*

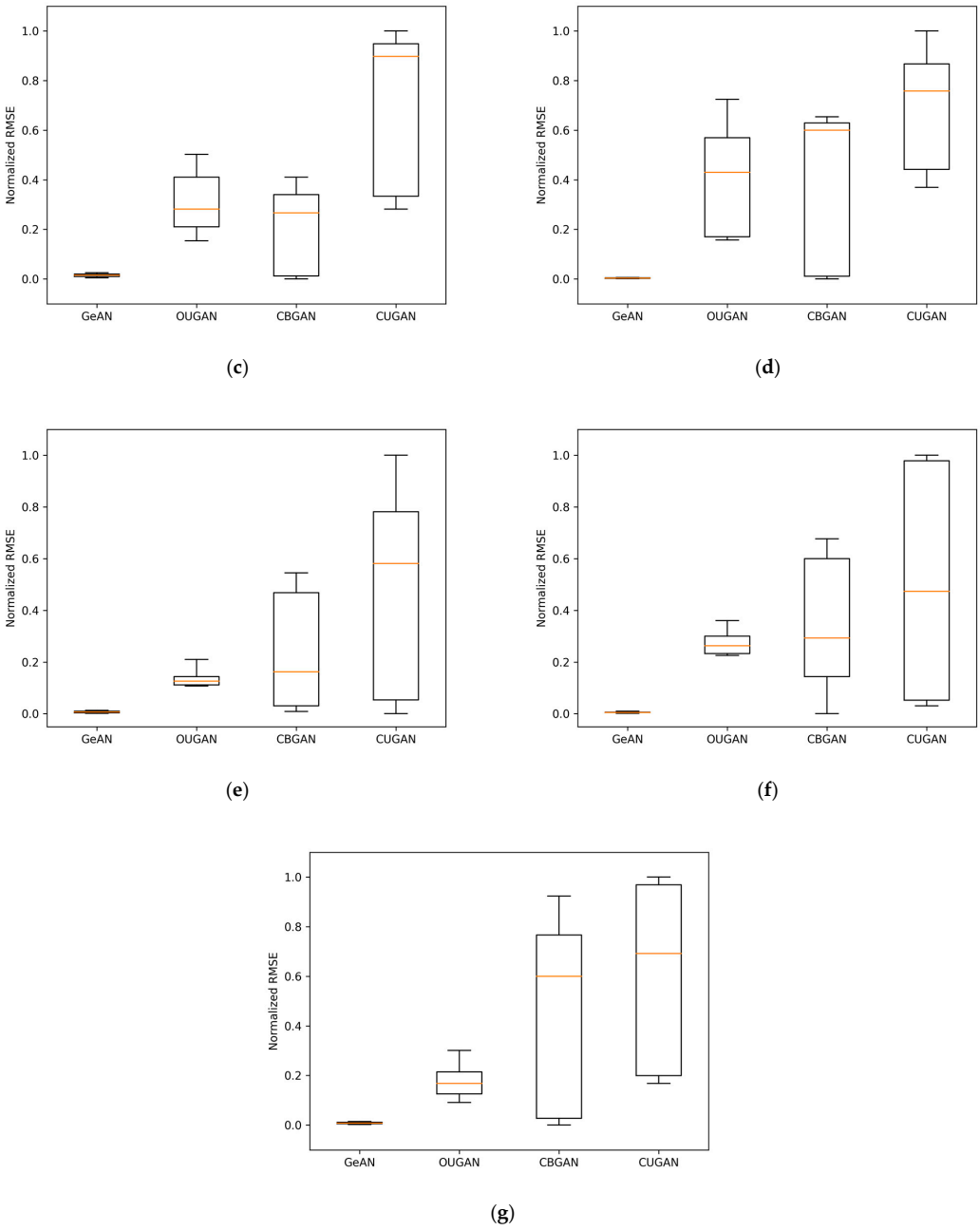


Figure 5. Normalized RMAE vs. number of epochs for the seven datasets: (**a**) FD001, (**b**) FD002, (**c**) FD003, (**d**) FD004, (**e**) SMD, (f) MSL, (**g**) SMAP.

The results presented in Figure 5 show that the generalizability of the GeAN mechanism is more accurate and robust than other mechanisms. In particular, it can be seen that the whisker range of the box plots of the GeAN model is only a fraction of the range of the box plots of the other models, indicating the low variability in the prediction errors generated by the GeAN mechanism across different folds. It can also be seen that the box plots of the GeAN mechanism are much closer to zero compared to the box plots of the other models, indicating the higher prediction accuracy of GeAN. This outstanding performance of the GeAN mechanism corroborates the ability of the design philosophy underlying the GeAN mechanism to endow it with tremendous capabilities to capture the knowledge embedded in different datasets and generalize it effectively to unseen samples. More specifically, applying the outer product operation to the edge and node features leads to comprehensive maps that can represent all possible interactions of these features at a fine-grained level. Using such a map in the learning process allows the GeAN to take into

account the entire behavior of the underlying MTS dataset represented by the graphs in its predictions. This in turn leads to consistent results with low variability represented as small box plots.

Further evidence of the benefit of the outer product can be seen in the box plots of the models that rely on the concatenation operator in the lie of the outer product, namely CBGAN and CUGAN. Figure 5 shows that the interquartile ranges of these two models are much higher than those of the other models, especially in the anomaly detection learning tasks (i.e., SMD, MSL, and SMAP datasets). This can be attributed to the fact that using the concatenation operation to aggregate the features of edges and nodes results in long and complex vectors, as this operator simply appends these features side-by-side without providing information about their interactions. This in turn can lead to the learning algorithm being confronted with a large amount of information that does not reflect the salient patterns of the graph structures or the underlying dataset they represent. Such information overload not only severely limits the achievement of stable and meaningful knowledge distillation during the training phases but also limits the generalization of the knowledge gained for test patterns. As a result, the predictions fluctuate strongly, even with small changes in the inputs, and they show a high variability of the predicted values.

Another strong aspect that leads to the exceptional performance of the GeAN mechanism is the use of the bilinear model to combine the mutual associations of the features generated by the outer product. Indeed, this model provides an interactive learning approach that allows the nodes and edges within a layer not only to modulate their features simultaneously from the perspective of the others but also to consolidate them into concise spaces. Through the learning process, the parameter spaces of the bilinear model can be perfectly fitted to the inherent characteristics of the given MTS datasets, which in turn results in the GeAN making its predictions with zero or negligible errors and providing a box plot centered around the zero-error regions. Figure 5 also shows that the box plots of the mechanisms using the unitary attention approach, namely OUGAN and CUGAN, are significantly far from the desired zero-error values. This poor prediction performance can be mainly attributed to the inherent limitations of the unitary attention mechanism, especially the mismatch between the projected values of the node features and the features of its neighbors and edges. In particular, the unitary attention approach relies on computing the attention scores by projecting each feature individually onto the learning matrix, which results in treating each feature in isolation from the others and does not take into account the possible dependencies and interactions between the features themselves. As a result, the attention score generated may not reflect the actual importance or relevance of each feature in the graphs. This argument has already been pointed out in several papers, e.g., in [91–93], where it is proposed to improve the performance of conventional graph attention mechanisms by projecting the concatenated features once onto the learning matrix.

4.4. Comparsion with State-of-the-Art Models

Sections 4.2 and 4.3 evaluate the performance of the two main components of the proposed models, namely TVG and GeAN, in comparison to different time series-to-graph techniques and attention mechanisms. These evaluations show the superiority of these two components in terms of their ability to capture the knowledge in the datasets and generalize this knowledge to test datasets. This subsection is devoted to a further evaluation of the proposed model by comparing it with several contemporary models developed for the same learning tasks and with the same dataset. The results of this direct comparison are summarized in Table 3, where the first column contains the reference number, the second column contains the year in which the work was published, the third column describes the underlying architectures, and the following columns list the performance values reported in the corresponding references. In this table, the following abbreviations are used to refer to the architecture, FC-STGNN: fully connected spatial–temporal, HierCorrPool: hierarchical correlation pooling, AGCTE: adaptive graph convolutional transformer encoder, MST-GAT: multi-perspective spatial–temporal graph attention, THGNN: temporal and heterogeneous

graph neural networks, STAG: spatio-temporal attention graph neural network, GAT-DAT: deep adaptative transformer enhanced by graph attention network, NSD-TGTN: nonlinear slow-varying dynamics-assisted temporal graph transformer network, CGNN-MHSA-AR: correlative GNN with multi-head self-attention and auto-regression ensemble method, DGT-PF: dynamic graph transformer-based parallel framework, AnomGraphAdv: anomalies graph neural network with adversarial training, DyGraphAD: dynamic graph, SiET: spatial information-enhanced transformer, event-driven bipartite graph.

Table 3. Comparison with related works.

Supervised Learning Tasks									
Ref.	**Year**	**Architcture**	**FD001**		**FD002**		**FD003**		**FD004**	
			RMSE	**Score**	**RMSE**	**Score**	**RMSE**	**Score**	**RMSE**	**Score**
[96]	2024	FC-STGNN	11.43	187	12.91	689	11.33	186	13.37	753
[78]	2023	HierCorrPool	12.02	233	12.38	599	11.26	189	14.08	847
[97]	2024	AGCTE	12.46	259.37	13.7	833.41	12.95	372.44	15.83	1520.05
[98]	2024	MST-GAT	11.26	168.36	13.47	687.85	10.32	197.25	13.35	908.46
[99]	2024	THGNN	13.15	285	13.84	806	12.61	255	14.65	1166
[100]	2024	STAG	11.50	187.2	16.92	1132.2	11.05	196.0	19.33	1443
[101]	2023	GAT-DAT	13.83	318.6	14.51	1163.8	14.85	438.5	16.80	1928.6
[102]	2023	NSD-TGTN	12.13	266	15.87	1477	12.01	220	16.64	1493
Proposed	2024	TVGeAN	7.24	89.78	6.8	330.01	6.17	74.47	7.24	304.6
Unsupervised Learning Tasks										
Ref.	**Year**		**SMD**			**MSL**			**SMAP**	
			Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
[103]	2023	CGNN-MHSA-AR	0.8390	0.8670	0.8530	0.9060	0.9110	0.9250	0.7270	0.9990	0.842
[76]	2024	DGT-PF	0.8959	0.9585	0.9226	0.9884	0.9760	0.9502	0.9433	0.9890	0.9668
[104]	2024	AnomGraphAdv	0.9646	0.9982	0.9811	0.9613	0.9916	0.9762	0.9806	0.9906	0.9855
[77]	2023	DyGraphAD	0.9514	0.9819	0.9655	0.9701	0.9486	0.9592	0.9339	0.9655	0.9494
[105]	2024	SiET	0.8955	0.9970	0.9307	0.9194	0.9681	0.9429	0.9583	0.9932	0.9754
[106]	2021	Event2graph	0.8861	0.8338	0.8493	0.8812	0.8311	0.8555	0.8654	0.9433	0.9027
Proposed	2024	TVGeAN	0.9998	0.9997	0.9998	0.9990	0.9992	0.9981	0.9988	0.9990	0.9997

The diversity of GNN architectures shown in Table 3 emphasizes the high adaptability that GNNs can offer in tackling different MTS learning tasks. However, the high variability in the performance readings of these architectures shows the high sensitivity that the design decisions have on these architectures. For example, the model proposed in [101] shows the lowest performance on the supervised learning tasks compared to the others. To clarify the main reasons for this, we should investigate the architecture of the model in more detail. In this work, the monoplex approach is used for graph modeling, in which the MTS is mapped into a graph, with each node representing a single univariate time series. The edges between these nodes are defined based on the cosine similarity between the univariate time series represented by the nodes. The graphs generated using this method are then processed with Graph Convolutional Networks (GCNs) and graph pooling networks, which use the Graph Attention Network (GAT) to extract the low-level spatial features. These features are then passed through a Deep Adaptative Transformer (DAT), which comprises a set of multi-headed self-attention mechanisms with position coding and temporally correlated computational units, to extract the temporal features of the MTS. One of the possible reasons for the modest performance of this model could be the use of cosine similarity to define the connectivity of the graphs representing the MTS. While this metric can quantify the general shape and direction of time series, its scale invariance and lack of consideration of the temporal relationship between different instances can impose some limitations in representing complex nonlinear relationships and temporal dependencies of MTS at full resolution. This can weaken the representation power of the generated graph, which in turn impairs the ability of the model to capture the underlying characteristics of the MTS represented by the graphs. Another important reason for the low performance of the model

presented in [101] could be the information leakage caused by using the graph pooling network as a dimensionality reduction technique. Indeed, the over-aggregation of node and edge features resulting from the use of graph pooling networks can lead to the loss of the nuanced information inherent in MTS. This has already been pointed out in several articles, e.g., in [33,99]. Therefore, various approaches such as hierarchical graph pooling networks or spectral graphs have been developed to remediate this problem. Above all, the segregation between spatial and temporal feature extraction processes in this model could be one of the main reasons for its modest performance, as this segregation prevents the model from capturing the inter- and intra-dependencies inherent in MTS simultaneously, which in turn can hinder the model's ability to converge to an optimal set of weights and biases. Table 3 also shows that the model presented in [97] occupies the penultimate position in terms of performance. A look at the architectures of this model shows that it follows the design approach of [101] by using GAT to extract the spatial features and multi-head self-attention to extract the temporal features, with the exception that in [97], an adaptive embedding vector is added to each node within the GCN. As can be seen, such addition can lead to some improvement in the prediction performance. However, the moderate performance of these two models suggests that the segregation of the spatial and temporal extraction processes with different networks has a negative impact on the learning ability of the models.

Running the spatial and temporal feature extraction processes in parallel rather than in tandem is another approach that appeared in [102]. In this work, the graphs are generated from MTS using a constrained correlation approach, where edges between pairs of nodes (representing univariate time series) are only created if the correlation between their time series is within a predefined threshold. These graphs are then passed through two parallel feature extraction units: the spatial and the temporal. The former uses a modified version of the GAT mechanism, and the latter employs a self-attention mechanism. However, instead of routing the outputs of these extraction units through a single MLP from which the predicted values are obtained, as is the case in other models, the outputs of these units are routed into two linear regression layers. These layers use a customized kernel that combines a number of indices, including the monotonicity index, the correlation index, and the slowness index, to provide the model with some information about the various dynamic properties of the MTS. Comparing the results of [102] with those of [97,101], it can be seen that the model of [102] shows some improvement compared with these models, which could be due to the parallel approach in this work. However, the results of the model in [97] show that equipping the spatio-temporal GNN networks with an additional optimization layer can compensate for the low prediction performance resulting from the segregation of spatial and temporal feature extraction. In [98], the multi-head ExpSparse self-attention mechanism is used for spatial feature extraction, and TCN is used for temporal feature extraction. The output of TCN is then passed through two fully connected neural networks, which are used to derive the optimistic and pessimistic predictions. These two values are trained together based on the Prediction Interval Coverage Probability (PICP) approach to refine their losses with respect to the ground truth values. Ref. [103] is another example that uses parallel spatial–temporal feature extraction networks, but in the domain of unsupervised learning. This model uses Fast Fourier Transform (FFT) and min–max normalization techniques to denoise the raw MTS dataset. This preprocessed version is then run through TCN networks to extract the sequence features, which are then processed by two parallel GNN networks to detect the spatial and temporal correlations of the extracted features. The results of these two networks are merged and then passed again to a multi-head self-attention mechanism, which is used to extract the context information. This information is then passed to the GRU to generate the predicted value for the next time step. However, since this predicted value does not reflect whether it is a normal or abnormal value, a first-order autoregressive model is applied to the preprocessed data to obtain the second predicted value. The two predicted values of the GNN-based model and the autoregressive model are combined according to a predefined weighting to determine the status of the

next measured readings. This description shows that one of the possible main reasons for the fair performance of this model is the dilution of the intrinsic patterns of the MTS datasets due to the cascading of multiple networks with heterogeneous transformations. The authors of this model have pointed out this problem and tried to overcome it by using first- and second-prediction networks.

In addition to cascading and parallelizing the spatial and temporal extraction networks, the model presented in [96] proposes to combine them in order to develop a fully connected spatio-temporal network. In this model, the MTS is divided into several patches using predefined windows. Each univariate time series within the same patch is processed independently using a dedicated ANN network. To compensate for the loss of chronological order caused by the segmentation, each encoder is additionally equipped with a sinusoidal position code. Since the outputs of these encoders capture the features within the univariate time series but not their mutual interactions, the dot product followed by the softmax function is applied to the representations generated by all the networks. It is claimed that these operations can be used to quantify the similarity index between different encoded univariate time series segments. Based on the value of the similarity index, the graph representing the MTS is defined, where the individual vertices represent the segments, and the edges are defined based on the similarity index. However, since this graph does not contain any temporal information about the patches, a parameterized exponential decay is used to inject the temporal distances into the graph. The manipulated graph is then processed by two moving–pooling GNN networks running in tandem to extract the local and global features, respectively. These features are then stacked and forwarded to an MLP from which the output is obtained. This approach can offer some improvements over other models (see Table 3). However, one of the main drawbacks of this approach is that several parameters, such as the segmentation window, the pooling size, and the decay rate, need to be adjusted manually at the same time. Such adjustments require not only prior knowledge of the detailed behavior of the MTS dataset but also a sophisticated intensification process to find the optimal combination that leads to the best prediction performance. The model proposed in [78] is another example of the challenges associated with applying the hierarchical approach to GNN architecture. In this work, the raw MTS dataset is divided into several segments, each of which is processed by TCN to extract its features. These features are then used to construct multiple graphs, each of which corresponds to a particular segment of the original MTS. However, since it is difficult to learn from multiple graphs, a correlation pooling scheme with a hierarchical mapping matrix is used to retrieve the results of the learning process on different graphs.

It can also be seen from Table 3 that applying preprocessing techniques to the MTS before generating graphs on which GNNs work can bring marginal improvements to the prediction performance. For example, the work proposed in [99] uses GRU networks to convert the temporal features from the MTS into the embedding vectors for each time step. These embedding vectors, together with their pairwise similarities computed based on the Hawkes process, are used to construct the graphs. In this work, several pooling operations are used to generate the feature vectors at the readout layer and then convert them back to a scalar vector. Ref. [100] is another example where the MTSs are clustered based on their operating conditions and the results are then processed using GCN with TCN. The modest improvement observed in these models is likely due to the noise reduction achieved by preprocessing. Reducing the noise in the original MTS dataset can lead to more informative graph structures from which the models can more easily identify the salient features. However, this can be at the expense of diluted or neglected nuanced information contained in the MTS dataset. In [105], a new transformer architecture is developed where the prior association and series association of each time point in the MTS dataset is used to generate its spatio-temporal features. These features are then used to modify the adjacency matrices and then cluster each GNN layer through the diffpooling module. In this work, the autoregression-based MLP model is also used in parallel to improve the learning stability of the model.

Table 3 also shows that some models such as [76,77,104,106] have adopted the principles of the dynamic graph approach to endow the DNN models with the ability to construct the graphs at runtimes based on the dynamics represented in the MTSs. However, this approach usually has some drawbacks, such as the need to use additional modules to maintain the historical patterns extracted from the graphs as they evolve. In addition to the overhead that may be imposed on the learning process by the additional modules, the inconsistency between the predicted values generated by different modules requires the use of ensemble learning strategies. For example, in the model presented in [76], the MTS is fed into two parallel modules. The first consists of multiple GCN networks with diffpooling networks, while the second module consists of autoregressive and MLP networks. The results of these two modules are then passed to another layer based on the weighted sum approach to generate the final anomaly scores of the MTS. In addition to the below-average performance of this model compared to other works, the high computational complexity was mentioned as one of the major weaknesses by the authors of this work. The work proposed in [77] uses the 2D convolutional layers as the second learning module of the main graph learning module. The work proposed in [106] is an example of an attempt to overcome the shortcomings associated with the use of different modules by using a bipartite graph. In this approach, the MTS datasets are represented as two disjoint sets of two nodes: the event nodes and the time series nodes. The event nodes represent the individual time series, while the time series nodes represent the different segmentations of the MTS. The connections between these nodes are created spontaneously based on the dynamic temporal distance between the nodes of the different groups, and the temporal dependencies within these graphs are extracted based on a combination of RNN and attention mechanisms. However, in order to store the state of the interactions between the two subgraphs generated by the bipartite graph approach, this model requires that each node is equipped with a memory unit. The main disadvantage of using external memory units in a DNN network is their negative impact on the prediction delay, since all memory units of all nodes need to be synchronized at each learning step to ensure consistent prediction values. In [104], the adversarial training approach was introduced to circumvent the need for additional learning modules. In this work, autoencoder with two blocks was used, including graph blocks and temporal blocks to enable adversarial learning between these two blocks. While adversarial learning provides a powerful approach, it also comes with some challenges and limitations, such as low generalizability and stability. Another drawback of using dynamic GNN is the long training time required to achieve a stable learning environment [105].

The exceptional results of the proposed model (i.e., TVGeAN) compared to other work confirms the effectiveness and thoroughness of the underlying assumptions used to develop its components. In particular, the use of the TVG algorithm to map the MTS dataset into graphs eliminates the need to censor or truncate some parts of the MTS relationships, as is the case with the constrained correlation approach used in [102]; to rely on simplified metrics such as cosine similarity, as is the case in [101]; or to patch the MTS, as is the case in [96]. Furthermore, the ability of GeAN to learn the temporal interactions embedded in the node and edge features of graphs in their native formats solves the shortcomings associated with using separate feature extraction processes to capture the temporal and spatial features, as is the case in [81,82]. Furthermore, by using sparse and variational autoencoder architectures, TVGeAN can capture the intrinsic knowledge from a small training space and generalize it effectively without using complex approaches, as is the case with the models that use the dynamic graph approach, e.g., refs. [103–106].

4.5. Analysis of the Computational Complexity

The previous subsection has shown that the proposed model can outperform some of the state-of-the-art models in terms of predictive accuracy. This naturally raises the question of whether this superior performance requires a higher consumption of computational resources. To address this question, the computational complexity of the proposed model,

as well as models that have similar architectures to those listed in Table 4, is evaluated for the FD004 and SMAP dataset. This evaluation is carried out using the following three metrics: (i) FLOPs, which measure the FLOating Point Operations per second required to run the model, (ii) parameter count, or the total number of weights and biases used by a model to fit the dataset, and (iii) inference time, which is defined as the time required for the model to generate the output corresponding to a given input in the testing phase.

Table 4. Computational complexity of the proposed model and some related architectures for the FD004 and SMAP datasets.

Dataset	Architecture	FLOPs	Parameter Counts	Inference Time (ms)
FD004	FC-STGNN	856,072	50,125	3.95
	HierCorrPool	939,889	42,630	5.29
	AGCTE	986,076	65,357	6.88
	MST-GAT	878,374	42,545	2.78
	THGNN	696,513	67,389	4.42
	STAG	598,763	55,641	3.99
	GAT-DAT	709,820	41,339	4.45
	NSD-TGTN	962,851	65,169	5.17
	TVGeAN	321,000	25,125	2.21
SMAP	CGNN-MHSA-AR	349,401	43,598	3.78
	DGT-PF	541,652	44,015	4.49
	AnomGraphAdv	635.692	58,885	2.78
	DyGraphAD	465,980	62,712	3.19
	SiET	554,254	45,383	5.24
	Event2graph	324,254	32,803	4.55
	TVGeAN	214,214	19,214	1.05

The results of this evaluation, as presented in Table 4, show that the proposed model, i.e., TVGeAN, has the lowest FLOPs, the fastest inference time, and the least number of parameters compared to the other models. One of the main reasons for the high efficiency of TVGeAN is the incorporation of the sparse learning unit in its autoencoder architectures. This incorporation allows the model to amputate the edges and nodes whose contributions to the learning tasks are negligible. By amputating insignificant components, the model remains compact, which in turn leads to a reduction in computational effort and inference time. Another important reason for the performance of TVGeAN is its adoption for the weight-sharing approach to derive attention scores across the different layers. This approach facilitates the reduction in the total number of parameters by reusing the same set of weighting matrices. Such a reduction in the number of parameters leads to a reduction in the FLOPS requirement of the model and thus to a reduction in inference time. It is interesting to note that the computational complexity of the proposed TVG model can be written as $\mathcal{O}(KT^2)$, where K is the number univariant time series forming MTS, and T is the number of its time points.

5. Discussion

The comprehensive evaluation of the proposed model presented in the previous section has emphasized its superior performance and computational efficiency. Nevertheless, it is of interest to discuss the potential limitations of the model and the possible improvements that can be made in the future to address them. The proportional growth of the diameter and relationships of the graphs generated by the TVG algorithm can be one of the possible limitations, especially when modeling very long and complex MTS datasets. Although this limitation has been overcome in this work by using autoencoder architectures with sparse learning units, our future research focuses on developing algorithms that can model MTS in graphs without being scaled up with their data points. This could include graph clustering algorithms, dimensionality reduction techniques, or a state space representation for the MTS to create more compact graph structures while preserving the

essential temporal relationships. Modelling the MTS with missing and incomplete data points using graphs is another important concern that needs further investigation. Indeed, most real-world MTS datasets are prone to subtle information loss during acquisition or storage, so it is important to recognize the possibility of information loss before processing MTS. While there are several techniques that can be used for this purpose, the ability of the TVG algorithm to visualize MTS at a high level could be further exploited by combining it with graph metrics to enable quantification of lost information. In addition to the aforementioned information, it is of tremendous importance to investigate how to improve the ability of the proposed graph attention mechanisms to jointly process multiple MTS data points. The development of such an improvement cannot only significantly increase the computational complexity of the mechanism but also pave the way for the implementation of this mechanism for real-time applications.

6. Conclusions

This paper proposed TVGeAN, a novel graph auto-encoder model developed specifically for MTS learning tasks. The underlying approach of TVGeAN is to combine the power of complex networks in representing time series as graphs with the strengths of GNNs in learning from graph-structured data in their native formats. The first component of the proposed model, the Tensor Visibility Graph (TVG), extends the capabilities of visibility graph algorithms to model the MTS datasets by representing them as weighted temporal graphs where the nodes and edges are tensors. The second main component, GeAN, is a new graph attention mechanism that is primarily designed to seamlessly integrate the temporal dependencies of MTS datasets into the model's learning process by automatically prioritizing the attention scores of graph nodes and edge features based on their temporal interactions. Furthermore, TVGeAN utilizes the autoencoder architecture with sparse and variational learning units to enable both generative and inductive learning. The performance of the proposed model has been extensively evaluated from different perspectives in both supervised and unsupervised learning domains. The results of this evaluation show that TVGeAN has an excellent learning curve, high generalizability, and superior performance compared to peer works. Apart from the advantages that TVGeAN offers in the field of MTS prediction, this work paves the way for the convergence of complex networks and deep learning domains to solve real-world problems.

Funding: This research was funded by Taif University, Saudi Arabia, Project No. (TU-DSPP-2024-143).

Data Availability Statement: All data used in this study are available from the providers mentioned in this manuscript.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L. Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks. In *International Conference on Web-Age Information Management*; Springer International Publishing: Cham, Switzerland, 2014.
2. Ismail Fawaz, H.; Lucas, B.; Forestier, G.; Pelletier, C.; Schmidt, D.F.; Weber, J.; Webb, G.I.; Idoumghar, L.; Muller, P.A.; Petitjean, F. Inceptiontime: Finding alexnet for time series classification. *Data Min. Knowl. Discov.* **2020**, *34*, 1936–1962. [CrossRef]
3. Masini, R.P.; Medeiros, M.C.; Mendes, E.F. Machine learning advances for time series forecasting. *J. Econ. Surv.* **2023**, *37*, 76–111. [CrossRef]
4. Sezer, O.B.; Gudelek, M.U.; Ozbayoglu, A.M. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Appl. Soft Comput.* **2020**, *90*, 106181. [CrossRef]
5. Baz, M. SEHIDS: Self Evolving Host-Based Intrusion Detection System for IoT Networks. *Sensors* **2022**, *22*, 6505. [CrossRef]
6. Wang, X.; Lu, X. A Host-Based Anomaly Detection Framework Using XGBoost and LSTM for IoT Devices. *Wirel. Commun. Mob. Comput.* **2020**, *2020*, 8838571. [CrossRef]
7. Wu, R.-S.; Chou, P.-H. Customer segmentation of multiple category data in e-commerce using a soft-clustering approach. *Electron. Commer. Res. Appl.* **2011**, *10*, 331–341. [CrossRef]
8. Wang, S.; Sun, L.; Yu, Y. A dynamic customer segmentation approach by combining LRFMS and multivariate time series clustering. *Sci. Rep.* **2024**, *14*, 17491. [CrossRef]

9. Lütkepohl, H. *New Introduction to Multiple Time Series Analysis*; Springer: Berlin/Heidelberg, Germany, 2005.
10. Durbin, J.; Koopman, S.J. *Time Series Analysis by State Space Methods*; OUP Oxford: Oxford, UK, 2012; Volume 38.
11. Silverman, B.W. *Density Estimation for Statistics and Data Analysis*; Routledge: Oxfordshire, UK, 2018.
12. Huang, N.E.; Shen, Z.; Long, S.R.; Wu, M.C.; Shih, H.H.; Zheng, Q.; Yen, N.C.; Tung, C.C.; Liu, H.H. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.* **1998**, *454*, 903–995. [CrossRef]
13. Wood, S.N. *Generalized Additive Models: An Introduction with R*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2017.
14. Scott, S.L.; Varian, H.R. Predicting the present with Bayesian structural time series. *Int. J. Math. Model. Numer. Optim.* **2014**, *5*, 4–23. [CrossRef]
15. Ashish, V. Attention is all you need. *arXiv* **2017**, arXiv:1706.03762.
16. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
17. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [CrossRef]
18. Lipton, Z.C. Learning to diagnose with LSTM recurrent neural networks. *arXiv* **2015**, arXiv:1511.03677.
19. Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.* **2019**, *33*, 917–963. [CrossRef]
20. Yang, J.; Wang, K.; Peng, X.; Qiao, Y. Deep recurrent multi-instance learning with spatio-temporal features for engagement intensity prediction. In Proceedings of the 20th ACM International Conference on Multimodal Interaction, Boulder, CO, USA, 16–20 October 2018.
21. Wang, R.; Pei, X.; Zhu, J.; Zhang, Z.; Huang, X.; Zhai, J.; Zhang, F. Multivariable time series forecasting using model fusion. *Inf. Sci.* **2022**, *585*, 262–274. [CrossRef]
22. Karim, F.; Majumdar, S.; Darabi, H.; Harford, S. Multivariate LSTM-FCNs for time series classification. *Neural Netw.* **2019**, *116*, 237–245. [CrossRef]
23. Graves, A.; Fernández, S.; Schmidhuber, J. Multi-dimensional recurrent neural networks. In Proceedings of the International Conference on Artificial Neural Networks, Porto, Portugal, 9–13 September 2007; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2007.
24. Samek, W.; Wiegand, T.; Müller, K.-R. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv* **2017**, arXiv:1708.08296.
25. Ye, L.; Keogh, E. Time series shapelets: A novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.* **2011**, *22*, 149–182. [CrossRef]
26. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef]
27. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [CrossRef]
28. Supriya, S.; Siuly, S.; Wang, H.; Cao, J.; Zhang, Y. Weighted visibility graph with complex network features in the detection of epilepsy. *IEEE Access* **2016**, *4*, 6554–6566. [CrossRef]
29. Tenenbaum, J.B.; Freeman, W.T. Separating Style and Content with Bilinear Models. *Neural Comput.* **2000**, *12*, 1247–1283. [CrossRef] [PubMed]
30. Kim, J.H.; Jun, J.; Zhang, B.T. Bilinear attention networks. In Proceedings of the Advances in Neural Information Processing Systems 31 (NeurIPS 2018), Montreal, QC, Canda, 3–8 December 2018; pp. 1137–1144.
31. Ranzato, M.A.; Poultney, C.; Chopra, S.; Cun, Y. Efficient learning of sparse representations with an energy-based model. In Proceedings of the Advances in Neural Information Processing Systems 19 (NIPS 2006), Vancouver, BC, Canada, 4–7 December 2006.
32. Kingma Diederik, P.; Welling, M. Auto-encoding variational bayes. *arXiv* **2013**, arXiv:1312.6114.
33. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
34. Mishra, P.; Piktus, A.; Goossen, G.; Silvestri, F. Node masking: Making graph neural networks generalize and scale better. *arXiv* **2020**, arXiv:2001.07524.
35. Saxena, A.; Goebel, K.; Simon, D.; Eklund, N. Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation. In Proceedings of the 1st International Conference on Prognostics and Health Management (PHM08), Denver, CO, USA, 6–9 October 2008.
36. Su, Y.; Zhao, Y.; Niu, C.; Liu, R.; Sun, W.; Pei, D. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019.
37. Hundman, K.; Constantinou, V.; Laporte, C.; Colwell, I.; Soderstrom, T. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018. Multivariate time series analysis and applications.
38. Wei, W.W.S. *Multivariate Time Series Analysis and Applications*; John Wiley & Sons: Hoboken, NJ, USA, 2018.

39. Sanchez, J. *Time Series for Data Scientists: Data Management, Description, Modeling and Forecasting*; Cambridge University Press: Cambridge, UK, 2023.
40. Aggarwal, C.C. *Neural Networks and Deep Learning: A Textbook*, 1st ed.; Springer Nature: Cham, Switzerland, 2018.
41. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
42. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 17–19 June 2013.
43. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on se-quence modeling. *arXiv* **2014**, arXiv:1412.3555.
44. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
45. Zaremba, W.; Sutskever, I.; Vinyals, O. Recurrent neural network regular-ization. *arXiv* **2014**, arXiv:1409.2329.
46. Shewalkar, A.; Nyavanandi, D.; Ludwig, S.A. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. *J. Artif. Intell. Soft Comput. Res.* **2019**, *9*, 235–245. [CrossRef]
47. Tao, F.; Liu, G. Advanced LSTM: A study about better time dependency modeling in emotion recognition. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; IEEE: Piscataway, NJ, USA, 2018.
48. Miller, J.; Hardt, M. Stable recurrent models. *arXiv* **2018**, arXiv:1805.10369.
49. Wen, X.; Li, W. Time series prediction based on LSTM-attention-LSTM model. *IEEE Access* **2023**, *11*, 48322–48331. [CrossRef]
50. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
51. Yu, F.; Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv* **2015**, arXiv:1511.07122.
52. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017.
53. Wu, Y.; He, K. Group normalization. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018.
54. Luo, W.; Li, Y.; Urtasun, R.; Zemel, R. Understanding the effective receptive field in deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
55. Jacobsen, J.H.; Van Gemert, J.; Lou, Z.; Smeulders, A.W. Structured receptive fields in cnns. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
56. Araujo, A.; Norris, W.; Sim, J. Computing receptive fields of convolutional neural networks. *Distill* **2019**, *4*, e21. [CrossRef]
57. Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; Volume 35, pp. 11106–11115.
58. Chen, Z.; Ma, M.; Li, T.; Wang, H.; Li, C. Long sequence time-series forecasting with deep learning: A survey. *Inf. Fusion* **2023**, *97*, 101819. [CrossRef]
59. Zhang, Y.; Yan, J. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In Proceedings of the Eleventh International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
60. Han, J.; Kwon, Y.; Choi, Y.-S.; Kang, S. Improving chemical reaction yield prediction using pre-trained graph neural networks. *J. Cheminform.* **2024**, *16*, 25. [CrossRef]
61. Zhang, J.; Jiang, Z.; Hu, X.; Song, B. A novel graph attention adversarial network for predicting disease-related associations. *Methods* **2020**, *179*, 81–88. [CrossRef]
62. Yang, L.; Liu, Z.; Dou, Y.; Ma, J.; Yu, P.S. Consisrec: Enhancing gnn for social recommendation via consistent neighbor aggregation. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 11–15 July 2021.
63. Liang, K.; Meng, L.; Liu, M.; Liu, Y.; Tu, W.; Wang, S.; Zhou, S.; Liu, X.; Sun, F.; He, K. A survey of knowledge graph reasoning on graph types: Static, dynamic, and multi-modal. *IEEE Trans. Pattern Anal. Mach. Intell.* 2024; 1–20, *Early Access*. [CrossRef]
64. Huang, X.; Han, K.; Yang, Y.; Bao, D.; Tao, Q.; Chai, Z.; Zhu, Q. Can GNN be Good Adapter for LLMs? In Proceedings of the ACM on Web Conference, Singapore, 13–17 May 2024; pp. 893–904.
65. Jin, M.; Koh, H.Y.; Wen, Q.; Zambon, D.; Alippi, C.; Webb, G.I.; King, I.; Pan, S. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *arXiv* **2023**, arXiv:2307.03759. [CrossRef]
66. Bloemheuvel, S.; van den Hoogen, J.; Jozinović, D.; Michelini, A.; Atzmueller, M. Graph neural networks for multivariate time series regression with application to seismic data. *Int. J. Data Sci. Anal.* **2023**, *16*, 317–332. [CrossRef]
67. Ding, J.; Zhang, L.; Cheng, J. Multi-robot Path Planning Based on Spatio-Temporal Information in Large-scale Unknown Environment. In Proceedings of the 2021 27th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Shanghai, China, 26–28 November 2021; pp. 795–800. [CrossRef]
68. Wang, X.; Ma, Y.; Wang, Y.; Jin, W.; Wang, X.; Tang, J.; Jia, C.; Yu, J. Traffic flow prediction via spatial temporal graph neural network. In Proceedings of the Web Conference 2020, Taipei, Taiwan, 20–24 April 2020; 2020; pp. 1082–1092.

69. 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. In Proceedings of the Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Virtual, 6–12 December 2020; pp. 17766–17778.
70. Özden, C.; Bulut, M. Spectral temporal graph neural network for multivariate agricultural price forecasting. *Ciência Rural.* **2023**, *54*, e20220677. [CrossRef]
71. Jin, M.; Shi, G.; Li, Y.F.; Wen, Q.; Xiong, B.; Zhou, T.; Pan, S. How Expressive are Spectral-Temporal Graph Neural Networks for Time Series Forecasting? *arXiv* **2023**, arXiv:2305.06587.
72. Fan, J.; Bai, J.; Li, Z.; Ortiz-Bobea, A.; Gomes, C.P. A GNN-RNN approach for harnessing geospatial and temporal information: Application to crop yield prediction. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 22 February–1 March 2022; Volume 36.
73. Lazcano, A.; Herrera, P.J.; Monge, M. A combined model based on recurrent neural networks and graph convolutional networks for financial time series forecasting. *Mathematics* **2023**, *11*, 224. [CrossRef]
74. Liang, Y.; Jiang, S.; Gao, M.; Jia, F.; Wu, Z.; Lyu, Z. GLSTM-DTA: Application of prediction improvement model based on GNN and LSTM. *J. Phys. Conf. Ser.* **2022**, *2219*, 012008. [CrossRef]
75. Xie, Y.; Xiong, Y.; Zhu, Y. SAST-GNN: A self-attention based spatio-temporal graph neural network for traffic prediction. In Proceedings of the Database Systems for Advanced Applications: 25th International Conference, DASFAA 2020, Jeju, Republic of Korea, 24–27 September 2020; Proceedings, Part I 25; Springer International Publishing: Berlin/Heidelberg, Germany, 2020.
76. He, H.; Li, X.; Chen, P.; Chen, J.; Liu, M.; Wu, L. Efficiently localizing system anomalies for cloud infrastructures: A novel Dynamic Graph Transformer based Parallel Framework. *J. Cloud Comput.* **2024**, *13*, 115. [CrossRef]
77. Chen, K.; Feng, M.; Wirjanto, T.S. Multivariate time series anomaly detection via dynamic graph forecasting. *arXiv* **2023**, arXiv:2302.02051.
78. Wang, Y.; Wu, M.; Li, X.; Xie, L.; Chen, Z. Multivariate Time-Series Representation Learning via Hierarchical Correlation Pooling Boosted Graph Neural Network. *IEEE Trans. Artif. Intell.* **2023**, *5*, 321–333. [CrossRef]
79. Eroglu, D.; Marwan, N.; Stebich, M.; Kurths, J. Multiplex visibility graphs to investigate recurrent neural network dynamics. *Sci. Rep.* **2018**, *7*, 44037.
80. Yang, Y.; Yang, H. Complex network-based time series analysis. *Phys. A Stat. Mech. Its Appl.* **2008**, *387*, 1381–1386. [CrossRef]
81. Sun, J.; Yang, Y.; Liu, Y.; Chen, C.; Rao, W.; Bai, Y. Univariate time series classification using information geometry. *Pattern Recognit.* **2019**, *95*, 24–35. [CrossRef]
82. Donner, R.V.; Small, M.; Donges, J.F.; Marwan, N.; Zou, Y.; Xiang, R.; Kurths, J. Recurrence based time series analysis by means of complex network methods. *Int. J. Bifurc. Chaos* **2011**, *21*, 1019–1046. [CrossRef]
83. Nakamura, T.; Tanizawa, T.; Small, M. Constructing networks from a dynamical system perspective for multivariate nonlinear time series. *Phys. Rev. E* **2016**, *93*, 032323. [CrossRef] [PubMed]
84. Lanne, M.; Nyberg, H. Generalized forecast error variance decomposition for linear and nonlinear multivariate models. *Oxf. Bull. Econ. Stat.* **2016**, *78*, 595–603. [CrossRef]
85. Zhang, J.; Zhou, J.; Tang, M.; Guo, H.; Small, M.; Zou, Y. Constructing ordinal partition transition networks from multivariate time series. *Sci. Rep.* **2017**, *7*, 7795. [CrossRef]
86. Feldhoff, J.H.; Donner, R.V.; Donges, J.F.; Marwan, N.; Kurths, J. Geometric detection of coupling directions by means of inter-system recurrence networks. *Phys. Lett. A* **2012**, *376*, 3504–3513. [CrossRef]
87. Ren, H.; Yuan, Q.; Semba, S.; Weng, T.; Gu, C.; Yang, H. Pattern interdependent network of crosscorrelation in multivariate time series. *Phys. Lett. A* **2020**, *384*, 126781. [CrossRef]
88. Silva, V.F.; Silva, M.E.; Ribeiro, P.; Silva, F. MHVG2MTS: Multilayer Horizontal Visibility Graphs for Multivariate Time Series Analysis. *arXiv* **2023**, arXiv:2301.02333.
89. Lacasa, L.; Nicosia, V.; Latora, V. Network structure of multivariate time series. *Sci. Rep.* **2015**, *5*, 1550. [CrossRef]
90. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
91. Qiu, J.; Tang, J.; Ma, H.; Dong, Y.; Wang, K.; Tang, J. Deepinf: Social influence prediction with deep learning. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018.
92. Wang, G.; Ying, R.; Huang, J.; Leskovec, J. Improving graph attention networks with large margin-based constraints. *arXiv* **2019**, arXiv:1910.11945.
93. Brody, S.; Alon, U.; Yahav, E. How attentive are graph attention networks? *arXiv* **2021**, arXiv:2105.14491.
94. Agrawal, S.; Steinbach, M.; Boley, D.; Chatterjee, S.; Atluri, G.; Dang, A.T.; Liess, S.; Kumar, V. Mining Novel Multivariate Relationships in Time Series Data Using Correlation Networks. *IEEE Trans. Knowl. Data Eng.* **2020**, *32*, 1798–1811. [CrossRef]
95. Masuda, N.; Boyd, Z.M.; Garlaschelli, D.; Mucha, P.J. Correlation networks: Interdisciplinary approaches beyond thresholding. *arXiv* **2023**, arXiv:2311.09536. Available online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8572131/ (accessed on 1 October 2024).
96. Wang, Y.; Xu, Y.; Yang, J.; Wu, M.; Li, X.; Xie, L.; Chen, Z. Fully-Connected Spatial-Temporal Graph for Multivariate Time-Series Data. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2024.
97. Ma, M.; Wang, Z.; Zhong, Z. Transformer Encoder Enhanced by an Adaptive Graph Convolutional Neural Network for Prediction of Aero-Engines' Remaining Useful Life. *Aerospace* **2024**, *11*, 289. [CrossRef]

98. Zhou, L.; Wang, H. MST-GAT: A multi-perspective spatial-temporal graph attention network for multi-sensor equipment remaining useful life prediction. *Inf. Fusion* **2024**, *110*, 102462. [CrossRef]
99. Wen, Z.; Fang, Y.; Wei, P.; Liu, F.; Chen, Z.; Wu, M. Temporal and Heterogeneous Graph Neural Network for Remaining Useful Life Prediction. *arXiv* **2024**, arXiv:2405.04336.
100. Huang, Z.; He, Y.; Sick, B. Spatio-Temporal Attention Graph Neural Network for Remaining Useful Life Prediction. In Proceedings of the 2023 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 13–15 December 2023.
101. Liang, P.; Li, Y.; Wang, B.; Yuan, X.; Zhang, L. Remaining useful life prediction via a deep adaptive transformer framework enhanced by graph attention network. *Int. J. Fatigue* **2023**, *174*, 107722. [CrossRef]
102. Gao, Z.; Jiang, W.; Wu, J.; Dai, T.; Zhu, H. Nonlinear slow-varying dynamics-assisted temporal graph transformer network for remaining useful life prediction. *Reliab. Eng. Syst. Saf.* **2024**, *248*, 110162. [CrossRef]
103. Song, Y.; Xin, R.; Chen, P.; Zhang, R.; Chen, J.; Zhao, Z. Identifying performance anomalies in fluctuating cloud environments: A robust correlative-GNN-based explainable approach. *Future Gener. Comput. Syst.* **2023**, *145*, 77–86. [CrossRef]
104. Bajpai, S.; Murthy, P.K.; Kumar, N. AnomGraphAdv: Enhancing Anomaly and Network Intrusion Detection in Wireless Networks Using Adversarial Training and Temporal Graph Networks. In Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Seoul, Republic of Korea, 27–29 May 2024.
105. Xiong, W.; Wang, P.; Sun, X.; Wang, J. SiET: Spatial information enhanced transformer for multivariate time series anomaly detection. *Knowl.-Based Syst.* **2024**, *296*, 111928. [CrossRef]
106. Wu, Y.; Gu, M.; Wang, L.; Lin, Y.; Wang, F.; Yang, H. Event2graph: Event-driven bipartite graph for multivariate time-series anomaly detection. *arXiv* **2021**, arXiv:2108.06783.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.