Graph-based Time-series Forecasting in Deep Learning

Hongjie Chen

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Hoda Eldardiry, Chair
Clifford A. Shaffer
Layne T. Watson
Feng Guo
Ryan A. Rossi

March 6th, 2024
Blacksburg, Virginia

Keywords: graph time-series modeling, time-series forecasting, hypergraph modeling

Graph-based Time-series Forecasting in Deep Learning

Hongjie Chen

(ABSTRACT)

Time-series forecasting has long been studied and remains an important research task. In scenarios where multiple time series need to be forecast, approaches that exploit the mutual impact between time series results in more accurate forecasts. This has been demonstrated in various applications, including demand forecasting and traffic forecasting, among others. Hence, this dissertation focuses on graph-based models, which leverage the internode relations to forecast more efficiently and effectively by associating time series with nodes.

This dissertation begins by introducing the notion of *graph time-series models* in a comprehensive survey of related models. The main contributions of this survey are: (1) A novel categorization is proposed to thoroughly analyze over 20 representative graph time-series models from various perspectives, including temporal components, propagation procedures, and graph construction methods, among others. (2) Similarities and differences among models are discussed to provide a fundamental understanding of decisive factors in graph time-series models. Model challenges and future directions are also discussed.

Following the survey, this dissertation develops graph time-series models that utilize complex time-series interactions to yield context-aware, real-time, and probabilistic forecasting. The first method, Context Integrated Graph Neural Network (CIGNN), targets resource forecasting with contextual data. Previous solutions either neglect contextual data or only leverage static features, which fail to exploit contextual information. Its main contributions include: (1) Integrating multiple contextual graphs; and (2) Introducing and incorporating temporal, spatial, relational, and contextual dependencies; The second method, Evolving Super Graph Neural Network (ESGNN), targets large-scale time-series datasets through training on super graphs. Most graph time-series models let each node associate with a time series, potentially resulting in a high time cost. Its main contributions include: (1) Generating multiple super graphs to reflect node dynamics at different periods; and (2) Proposing an efficient super graph construction method based on K-Means and LSH; The third method, Probabilistic Hypergraph Recurrent Neural Network (PHRNN), targets datasets under the assumption that nodes interact in a simultaneous broadcasting manner. Previous hypergraph approaches leverage a static weight hypergraph, which fails to capture the interaction dynamics among nodes. Its main contributions include: (1) Learning a probabilistic hypergraph structure from the time series; and (2) Proposing the use of a KNN hypergraph for hypergraph initialization and regularization. The last method, Graph Deep Factors (GraphDF), aims at efficient and effective probabilistic forecasting. Previous probabilistic approaches neglect the interrelations between time series. Its main contributions include: (1) Proposing a framework that consists of a relational global component and a relational local component; (2) Conducting analysis in terms of accuracy, efficiency, scalability, and simulation with opportunistic scheduling. (3) Designing an algorithm for incremental online learning.

Graph-based Time-series Forecasting in Deep Learning

Hongjie Chen

(GENERAL AUDIENCE ABSTRACT)

Time-series forecasting has long been studied due to its usefulness in numerous applications, including demand forecasting, traffic forecasting, and workload forecasting, among others. In scenarios where multiple time series need to be forecast, approaches that exploit the mutual impact between time series results in more accurate forecasts. Hence, this dissertation focuses on a specific area of deep learning: graph time-series models. These models associate time series with a graph structure for more efficient and effective forecasting.

This dissertation introduces the notion of *graph time series* through a comprehensive survey and analyzes representative graph time-series models to help readers gain a fundamental understanding of graph time series. Following the survey, this dissertation develops graph time-series models that utilize complex time-series interactions to yield context-aware, real-time, and probabilistic forecasting. The first method, Context Integrated Graph Neural Network (CIGNN), incorporates multiple contextual graph time series for resource time-series forecasting. The second method, Evolving Super Graph Neural Network (ESGNN), constructs dynamic super graphs for large-scale time-series forecasting. The third method, Probabilistic Hypergraph Recurrent Neural Network (PHRNN), designs a probabilistic hypergraph model that learns the interactions between nodes as distributions in a hypergraph structure. The last method, Graph Deep Factors (GraphDF), targets probabilistic time-series forecasting with a relational global component and a relational local model.

These methods collectively covers various data characteristics and model structures, including graphs, super graph, and hypergraphs; a single graph, dual graphs, and multiple graphs; point forecasting and probabilistic forecasting; offline learning and online learning; and both small and large-scale datasets. This dissertation also highlights the similarities and differences between these methods. In the end, future directions in the area of graph time series are also provided.

Dedication

To my Grandmother in heaven, who always believed in my dreams and inspired me to reach for the stars. Your love and wisdom continue to guide me on this academic journey.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Hoda Eldardiry, for her inspiring guidance during my Ph.D. life. I greatly appreciate her encouragement and support during my stressful times. She not only teaches me how to do research, but also guides me through difficult situations. I believe she is a continuing source of inspiration to me and to all of us. I am extremely fortunate to be advised by Dr. Eldardiry, and I believe all these years under her supervision will remain a precious memory to me.

I extend my thanks to my committee members for their constructive feedback and suggestions. Dr. Cliff Shaffer provided tremendous support for me in reaching departmental milestones. Dr. Layne Watson and Dr. Feng Guo offered valuable advice during my preliminary proposal and research defense, which significantly contributing to the improvement of my research work. Special thanks to Dr. Ryan Rossi, with whom I closely collaborated for several years, for his guidance and insightful advice on Ph.D. life. I am deeply grateful for their feedback during the final defense, which inspires me to explore new research directions.

I am grateful to my colleagues in the machine learning laboratory, Jiaying Gong and Vasanth Baddam, for their unwavering help and support. Jiaying and Vasanth gave me many suggestions and motivated me to pursue better research work during my Ph.D. study. Additionally, I would like to express my appreciation to my research collaborators, Dr. Sungchul Kim, Dr. Kanak Mahadik, and Dr. Meizhu Liu. Their advice and collaborative efforts have significantly broadened my research scope. I want to thank them for their patience and guidance when I am new to a research area, especially to Dr. Sungchul Kim for inspiring me with many research directions.

Finally, I want to thank my family and friends for their trust and wholehearted support. The completion of my doctoral study would be impossible without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

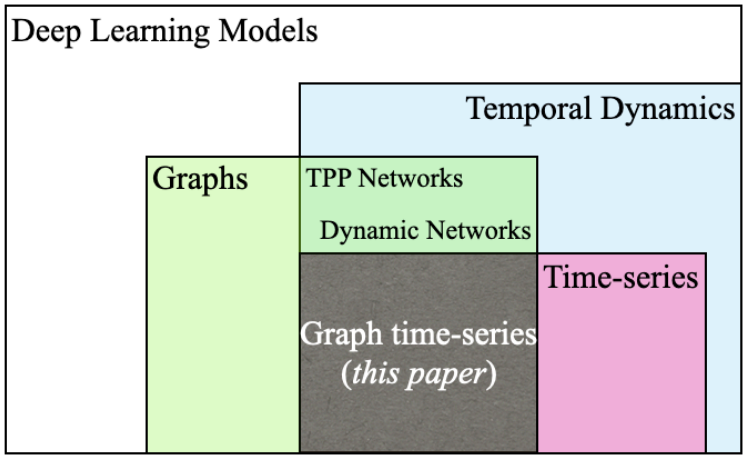


Figure 1.1: This dissertation bridges the gap of graph modeling and time-series modeling.

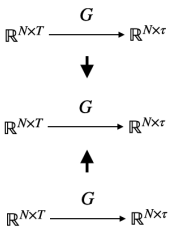
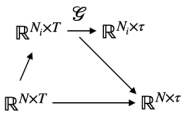
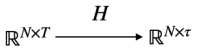
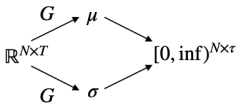
Time-series modeling and graph modeling are both important research areas that have gained increasing attention due to their wide applications. Generally, a time-series model studies time series data, while a graph model renders a graph structure. Recent efforts have focused on integrating these two domains, resulting in the emergence of graph time-series modeling. The position of graph time-series research within deep learning is depicted in Fig. 1.1. This dissertation centers around this concept and proposes several novel models to address various research problems.

1.1 Research Problems

This dissertation aims to answer following research problems with graph time-series models:

- What model components are effective for graph time-series models? Specifically, we focus on graph time-series forecasting accuracy and efficiency.
- How to address different model requirements and data characteristics, such as the incorporation of contextual information, generating probabilistic forecasting, and hypergraph modeling?

Table 1.1: Primary model characteristics and use scenarios

Models	Concept Diagrams	Use Scenarios
CIGNN		This model is designed for datasets that containing one target graph time series and multiple contextual graph time series. For example, *CIGNN* is apt for forecasting bike supply series across multiple locations, utilizing temperature series and humidity series.
ESGNN		This model is designed for large-scale datasets containing tens of thousands of time series or more. While *ESGNN* may not achieve the highest accuracy, it excels in terms of speed and efficiency.
PHRNN		*PHRNN* targets situations where time series interacts in a beyond-pairwise manner, such as in cloud systems where nodes communicate via simultaneous broadcasting.
GRAPHDF		*GraphDF* is a probabilistic forecasting method that generates distributions as predictions instead of real values. An extension named *IOGraphDF*, which adopts incremental learning, is also proposed.

1.2 Contributions

This dissertation develops graph time-series models that utilize complex time-series interactions to yield context-aware, real-time, and probabilistic forecasting. Table 1.1 highlights the primary characteristics of the proposed models as well as their use scenarios to assist readers in selecting suitable models. The main contributions are summarized below.

1.2.1 A Comprehensive Review of Graph Time-series Models

To the best of our knowledge, this is the first literature review that unifies the research areas of time-series modeling and graph modeling in deep learning. It covers a range of tasks, including regression, classification, and anomaly detection. We compile a comprehensive list of related models and detail over twenty representative models. In order to advance the understanding of graph time-series models, we categorize models from various perspectives, highlight the similarities and differences between models, and have in-depth model discussions. Specifically, we propose the categorization of Graph Recurrent/Convolutional Neural Networks, where models are further grouped into Recurrent Neural Network (RNN)-based

or Convolutional Neural Network (CNN)-based from the time-series modeling perspective, and Self-derived graph-based and Evolving graph-based from the graph modeling perspective. Furthermore, we propose the categorization of Graph Attention Neural Networks and discuss different attention types. We provide insightful directions for future research and applications of graph time series models.

1.2.2 Context Integrated Graph Neural Network

We propose a novel graph model, Context Integrated Graph Neural Network (CIGNN), that jointly models various dependencies, including temporal, relational, spatial, and dynamic contextual dependencies, in contrast to existing methods that do not capture dynamic contextual information. CIGNN consistently outperforms previous methods in terms of MAE and RMSE on two real-world datasets.

1.2.3 Evolving Super Graph Neural Network

We propose another novel graph time-series prediction model, namely, Evolving Super Graph Neural Network (ESGNN), that has a significantly lower time and space complexity compared to previous state-of-the-art baselines, while achieving comparable accuracy. ESGNN leverages K-Means and Locality-Sensitive Hashing (LSH) to quickly construct evolving super graphs, which allows the model to learn from the most recent graph connections.

1.2.4 Probabilistic Hypergraph Recurrent Neural Network

We introduce a novel hypergraph time-series forecasting model, Probabilistic Hypergraph Recurrent Neural Network (PHRNN), which leverages the interrelationships among time series by learning a probabilistic hypergraph. The probabilistic hypergraph approach benefits our model with the advantages of both probabilistic modeling and hypergraph modeling. To the best of our knowledge, PHRNN is the first probabilistic hypergraph model for time-series forecasting. The novel probabilistic hypergraph modeling within our proposed PHRNN serves as an adaptable spatial component and is readily integrated with other models. We conduct extensive experiments with closely related baselines on multiple datasets. Our experimental results show that PHRNN outperforms the baselines in terms of forecasting accuracy. We investigate the effectiveness of PHRNN through an hyperparameter study, including prior knowledge hypergraph constructions and regularization coefficients.

1.2.5 Graph Deep Factors for Cloud Resource Forecasting

We propose a general and extensible deep hybrid graph-based probabilistic forecasting framework called *Graph Deep Factors (GraphDF)* that is capable of learning complex nonlinear time-series patterns globally using the graph time-series data to improve both computational efficiency and forecasting accuracy while learning individual probabilistic models for individual time series based on their own time series and the collection of time series from the immediate neighborhood of the node in the graph. The GraphDF framework is data-driven, fast, scalable for real-time demand forecasting, and highly data efficient. In addition, considering the time-series streaming nature where newly incoming values arrive at each time step, we further propose an *incremental online GraphDF (IOGraphDF)* model that advances GraphDF model tremendously with respect to training runtime.

1.3 Organization of the Dissertation

Chapter 2 gives a comprehensive literature review of graph time-series models. Chapter 3-Chapter 6 introduce four proposed graph time-series models for time-series forecasting tasks. Specifically, Chapter 3 introduces a novel model named CIGNN that aims to leverage contextual graphs to gain more accurate forecasting for the target graph time series. Chapter 4 introduces a model named ESGNN that leverages super graph construction to reduce time and space complexity. Chapter 5 introduces a probabilistic hypergraph model named PHRNN that models interrelations based on a hypergraph structure. Chapter 6 introduces a model named GraphDF and its extension IOGraphDF for streaming values. Both models forecast future values as a probabilistic distribution. In the end, Chapter 7 summarizes all involved models, and presents the contributions of each work in detail. Future research directions are also given.

Chapter 2

A Comprehensive Review of Deep Graph Time-series Models

This chapter introduces and defines the terminology of *Graph Time Series*. By analyzing the latest related works on the topic of graph time series, this chapter explains the fundamentals of graph time series. Moreover, this chapter provides a comprehensive categorization of graph time-series models.

2.1 Introduction

Deep learning (DL) research has seen rapid progress in time-series analysis and graph modeling. **Time-series analysis** is essential in many areas such as signal processing, statistics, and data mining [58, 104, 124]. While there exists many different tasks that involve time-series analysis [78, 141], this survey specifically focuses on time-series tasks, including time-series classification, time-series anomaly detection, and time-series forecasting. We define deep learning models for time-series analysis as a category of models that leverage neural networks to handle time-series data for the aforementioned tasks. Deep learning models for time-series classification typically learn an embedding to classify time series [118, 209]; Time-series forecasting methods leverage deep learning for various problems such as univariate forecasting [145, 172], multivariate forecasting [23, 204], and forecasting model interpretability [40, 57]; Time-series anomaly detection methods leverage deep learning for anomaly detection at time-series level [111], time-period level [95, 144], and time-tick level [51, 228]. Regardless of their specific tasks, these models can be classified into two groups: models that focus on individual time series and models that simultaneously consider multiple time series. While the former group typically needs fewer parameters and is easier to interpret, it overlooks the interrelations between time series that potentially enhances model performance. In contrast, models in the latter group either implicitly or explicitly harness the interrelations between time series, thereby enabling each time series to leverage knowledge from others. Graph time-series models, as discussed in this survey, exemplify the latter group. **Graph models**, on the other hand, are no less investigated for their universal power to model topological relations between graph node entities [115, 227]. Deep graph learning research has recently seen significant progress in node classification [106, 182], node embeddings [74], and link prediction [222]. In this survey, we define graph models as those that leverage a

graph structure, where entities of interest are represented as nodes, and their relationships are represented as edges. Graph time-series models fall under the category of graph models where nodes are associated with time series.

Both time-series models and graph models are actively surveyed on their own. Many existing papers survey the topic of either time series [14, 18, 97] or graphs [227, 235], however, they only enable a partial understanding of models that interweave the two subjects. While a recent paper surveys both topics jointly, it only investigates a specific task of anomaly detection [89]. By contrast, this survey aims to fill the gap in available survey papers and provides a timely summary of the joint topic of time-series modeling and graph modeling in deep learning. We refer to related models as deep graph time-series models or **graph time-series models** for short. Graph time-series models aim to address time-series tasks using two indispensable components, a time-series component that captures the intraseries dependency, and a graph component that captures the interseries dependency. This survey gives an overview of graph time-series models and provides insights for researchers on what may contribute to model performance.

In the research area of deep learning, we define graph time-series models as models comprising one or more graphs, where the graph(s) is linked to time series through their node representations. In a typical graph time-series model, each node of the utilized graph(s) is paired with a time series. This association extends beyond time-series values and also encompasses the static or dynamic contextual information pertaining to the time series, if available. Each edge of the graph connects two nodes and indicates the strength of their connection. For instance, in a graph time-series model designed for stock price prediction, a graph can be employed where each node corresponds to a price series and a financial status of a company. Intuitively, stocks in the same sector (e.g., aviation) are more likely to correlate with each other than those in different sectors. This correlation leads to more accurate forecasting when closely related stock series are considered. Therefore, each edge can be derived from the correlation between its connecting nodes to indicate their connection strength.

Our survey is outlined as follows: we compare our survey with related survey papers, including those that only survey neural time-series models, those that only survey deep graph models, and other related survey papers (Sec. 2.2). We provide a high-level picture of how time series and graphs are modeled on their own in deep learning (Sec. 2.3), and then give the preliminary and more technical details on fundamental computations (Sec. 2.4). We follow by categorizing representative graph time-series models into two main categories (Sec. 2.5), namely, Graph Recurrent/Convolutional Neural Networks (GRCNN), and Graph Attention Neural Networks (GANN). In the GRCNN category, we further categorize models with respect to time-series modeling and graph modeling. In the GANN category, we further categorize models with respect to their targeted tasks. We discuss and analyze the use of representational components in graph time-series models, such as gated mechanism and skip connections, as well as the model interpretability (Sec. 2.6). We summarize real-world applications and commonly used datasets. We also discuss how graph time-series models

adapt to the irregularity from either time series or graphs (Sec. 2.7). At the same time, we show the performance of selected models on two time-series tasks, namely, regression and anomaly detection, to observe what models perform the best and analyze the possible reasons. Finally, we discuss open issues and future directions (Sec. 2.8).

Our survey aims to provide researchers insight on state-of-the-art deep graph time-series models. For this purpose, we select and discuss representative models on the topic. Our main contributions are summarized below:

- To the best of our knowledge, this is the first literature review that unifies the research areas of time-series modeling and graph modeling in deep learning to cover a range of tasks, including regression, classification, and anomaly detection. We compile a comprehensive list of related models and detail over twenty representative models.
- In order to advance the understanding of graph time-series models, we categorize models from various perspectives, highlight the similarities and differences between models, and have in-depth model discussions.
- We present the performance of selected models on commonly used datasets and analyze possible contributing factors for performance improvement. We provide insightful directions for future research and applications of graph time series models.

2.2 Related Work

Our survey is closely related to surveys that review either deep time-series models or deep graph models, as well as those that review temporal dynamics of graphs from perspectives of temporal point processes or dynamic networks.

Time series in deep learning are extensively researched and many models have been proposed for various tasks. For example, Ismail Fawaz et al. [97] survey classification models and groups them into generative and discriminative models. Forecasting models [14, 126] are discussed with respect to various task types, such as point forecasting versus probabilistic forecasting, single-step forecasting versus multistep forecasting, and so on. Blázquez-García et al. [18] provide a taxonomy of anomaly detection models for time-series level, period level, and time-point level outlier detection. Meanwhile, a considerable amount of deep graph models are also studied. For example, Wu et al. [203] provide an overview of representative Graph Neural Networks (GNNs). Another parallel survey paper also summarized different types of GNNs [227]. Moreover, Zhou et al. [235] discuss GNNs in terms of different propagation methods, sampling methods, and pooling methods. Although existing surveys on time series do not include graph-based models, and the aforementioned surveys on graphs rarely discuss time-series models, these papers and their surveyed models establish a solid foundation for researchers to propose and advance graph time-series models. Recent efforts

have aimed at integrating both topics as well. For example, Ho et al. [89] explore graph-based models for time-series anomaly detection. Unlike the exclusive focus on anomaly detection in that work, models surveyed in this chapter covers a broad spectrum of various tasks, including regression, classification, and anomaly detection. This survey serves to bridge the existing gap in the combined domain of time-series modeling and graph modeling in deep learning.

Similar to graph time-series models, temporal point processes in networks and dynamic networks are two types of models that capture the temporal dynamics embedded in a graph structure. Temporal Point Processes (TPP) are random processes that have the realization as a list of discrete events at different time points [63, 109]. TPP is commonly used to model time series of discrete events in continuous time. For example, TPP-based network models predict the number of link creations in social networks by formulating each link creation as a discrete event, and the time interval between events as a random variable. By contrast, graph time-series models use datasets that aggregate link count to derive time-series of a regular time interval. TPP-based models are more refined for temporal modeling, however, they require expert knowledge to select the probabilistic distributions behind random variables, which can be difficult for users without a statistical background. Besides, they typically take quadratic time complexity, which is computationally expensive for large-scale data. Dynamic network models commonly form a time-evolving graph to capture the interactions of nodes for node classification, link prediction, and other tasks [12, 170]. For example, Gupta and Bedathur [77] survey time-dependent graph representation learning and generative modeling. These models do not consider time series and therefore fail to solve time-series tasks. Different from existing survey papers, our survey primarily addresses two key research questions: RQ1. How do graph time-series models integrate graph models and time-series models, earning the advantages from both? RQ2. How do graph time-series models distinctively incorporate diverse structural designs, such as attention and residual connections, among more complicated ones? We navigate these research questions throughout the chapter via in-depth discussion of numerous graph time-series models, which helps readers understand the strength of graph time-series models on various time-series tasks. Moreover, this understanding contributes to further advancements in related research domains.

2.2.1 Related Work on Time-series Forecasting

Most work detailed in this chapter targets time-series forecasting. Hence, in this section we briefly introduce related work on the topic of time-series forecasting. Specifically, we introduce classical forecasting models and DL-based models that are not based on graphs.

2.2.1.1 Classical Time-series Forecasting Models

Classical time-series models such as AutoRegressive Integrated Moving Average (ARIMA) and exponential smoothing have demonstrated huge successes in univariate time-series prediction [68, 78, 91, 138, 181], however, they are mainly statistical and often fall short of incorporating input features [25]. Their main focus is on forecasting individual time series, which limits the model scalability and fails to extract the nonlinear relationships across time series. By contrast, multivariate time-series prediction takes the advantage of modeling the interdependencies across time series to improve prediction accuracy [17, 161, 176]. One example of multivariate time-series models is Vector AutoRegression (VAR) [173], which is considered as a generalization of autoregressive model. VAR treats the relationships across time series equivalently without difference, which is unrealistic. Deb et al. [48] summarized nine classical methods for forecasting energy usage, including artificial neural networks, support vector machine, among others.

2.2.1.2 DL-based Time-series Forecasting Models

Recent advance in deep learning and increase of available datasets have led to substantial improvement on time-series prediction [52, 72, 92, 99, 127, 162, 191, 192, 226], among which Recurrent Neural Networks (RNN) have received tremendous popularity for its accuracy and flexibility in modeling nonlinear complex temporal dependency [7, 13, 20, 23, 82, 126, 145, 219]. Long Short-Term Memory units (LSTM) [10, 90] and Gated Recurrent Units (GRU) [36, 37, 38] are two RNN models that are broadly adopted for their competence to overcome the vanishing gradient problem. Based on the LSTM and GRU architectures, many sequence-to-sequence models have been developed to allow prediction for a modest number of horizon [8, 59, 125, 175].

RNN models are also adopted to share information across variates and hence improve the forecast accuracy [175, 201]. For instance, Qin et al. [152] proposed a dual stage attention-based RNN model. Huang et al. [93] introduced a dual-attention mechanism for periodic or nonperiodic multivariate time-series forecasting.

2.3 Time series and Graphs in Deep Learning: Individual Modeling

In this section, we discuss how time series and graphs are separately modeled in deep learning. Our discussion mainly focuses on a high-level generalization of embedding learning for time series and nodes in graphs.

2.3.1 Time-series Encoding in Deep Learning

Two types of neural networks are mainly used to encode time series: Convolutional Neural Networks (**CNN**) and Recurrent Neural Networks (**RNN**). Through convolutional filters, CNN models capture local relationships between time-series values within ranges of convolutional filters. RNN models with their recurrent digesting characteristics, instead, aim at learning a hidden state that embeds knowledge from all earlier observations in the time series, after which the hidden state is used for downstream tasks such as classification or forecasting. Using a multivariate time series, denoted by $\mathbf{X} \in \mathbb{R}^{T \times d}$ of T time steps and d feature dimensions, as an input example, an RNN structure outputs a hidden state $\mathbf{h}^t \in \mathbb{R}^{d_h}$ of a user-defined d_h dimensions for each time step t. A general RNN takes the form below,

$$\mathbf{h}^t = f_\theta \left(\mathbf{h}^{t-1}, \mathbf{x}^t \right) \tag{2.1}$$

where θ generalizes the structure and parameters of the neural network and $\mathbf{x}^t \in \mathbb{R}^d$ is the observation value vector at time step t. The starting hidden state $\mathbf{h}^0$ is typically initialized as $\mathbf{0}$ when there is no prior knowledge. The model is extended to generalize the case where there are N multivariate time series as $X \in \mathbb{R}^{N \times T \times d}$. The hidden state $\mathbf{H}^t \in \mathbb{R}^{N \times d}$ at time step t is hence computed by the following equation,

$$\mathbf{H}^t = f_\theta \left(\mathbf{H}^{t-1}, \mathbf{X}^t \right) \tag{2.2}$$

with $\mathbf{H}^0$ initialized as $\mathbf{0}$ and $\mathbf{X}^t \in \mathbb{R}^{N \times d}$ the observation matrix at time step t. By having t moving forward along the time dimension, the hidden state $\mathbf{H}^T$ at the last time step $t = T$ learns from data values at all previous time steps. Later in Sec. 2.4, we will detail two popular RNNs, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU).

2.3.2 Graph Modeling in Deep Learning

A great many DL-based graph models have recently been proposed, addressing various graph problems and data natures. In this section, we primarily focus on node embedding models with **Graph Neural Networks (GNN)**. Effective node embeddings can then be used in downstream tasks such as node classification and link prediction.

2.3.2.1 Node Embedding

Learning informative node representations remains one of the most important tasks in deep graph learning. Effective node embeddings can be used in many tasks [51, 94]. Earlier models leverage random walk [74, 94], while recent models advance task performance with neural layers that efficiently aggregate neighboring node information and model the nonlinear interdependency between nodes to improve representation learning [106, 135, 183].

Given a graph $G = (V, E)$ with a node set V and an edge set E, and node features $\mathbf{X} \in \mathbb{R}^{N \times d}$ of $N = |V|$ nodes and d dimensions, the goal of node embedding is to learn a hidden state $\mathbf{h}_v \in \mathbb{R}^{d_h}$ of a selected dimension d_h for each node $v \in V$. A generalized node embedding graph model takes the form below,

$$\mathbf{h}_v = f_{\theta,G}\big(\mathbf{x}_v, \{\mathbf{h}_u \mid u \in V \wedge u \neq v\}\big) \tag{2.3}$$

where θ denotes the GNN structure and parameters. The equation implies that the embedding of node v relies on both its input features and the embeddings of other nodes. In practice, the range of other nodes is limited to a neighborhood set of v, denoted by $\Gamma(v)$, hence resulting in $u \in \Gamma(v)$.

2.3.2.2 Representative Graph Neural Networks

GNNs constitute the majority of recent graph-based deep learning models [203, 227]. Here, we briefly introduce how graph structures are rendered in two widely used GNN models.

Graph Convolutional Networks (GCN) operate directly on graph structures with graph convolutional layers, which is considered a counterpart of CNN which operates on grid-structure data [106, 223]. For each node in the graph, GCN aggregates information from other nodes based on their connection strength, i.e., nodes in proximity have more impact than distant nodes. A graph convolutional layer allows the aggregation of one-hop neighbor nodes. Multihop neighborhood aggregation can be achieved by stacking multiple graph convolutional layers. Essentially, GCN can be seen as a node embedding method and it was initially applied to a node classification task.

Graph Attention Networks (GAT) integrate the attention mechanism which consists of self-attentional layers. These layers learn pairwise relations between nodes based on node embeddings [183]. Although the standard GAT takes the graph connectivity information as input, GAT can assume a complete graph when such connectivity information is missing. Node embeddings are used to compute attention scores, which consequently serve as edge weights in the graph. This procedure has twofold advantages. Firstly, the attention scores offer a plausibly intuitive interpretation of mutual impacts between nodes. Secondly, the node embeddings and attention scores can be iteratively computed in turn, providing a self-contained and relatively simple model regarding parameter complexity. External graph structure knowledge, if exists, can be utilized to select attention scores to compute. Similarly, the GAT model can be seen as a node embedding method that learns a hidden state for each node by incorporating their neighborhood information.

Table 2.1: A notation table of frequently used symbols

Notations	Descriptions	Notations	Descriptions
X	a tensor	c	number of channels
$\mathbf{X}, \mathbf{W}$	matrices	d	number of features or variates
$\boldsymbol{\theta}, \mathbf{a}, \mathbf{b}$	parameter vectors	α	attention scores
θ, λ	parameter scalar values	$\mathcal{G}$	a series of graphs
T	length of time series	$G = (V, E)$	a graph, its node set and edge set
w	lookback window size	$N = \lvert V \rvert$	number of nodes
τ	future horizon	$\Gamma(v)$	neighbor node set of v
$\mathbf{h}, \mathbf{H}$	hidden states	$\mathbf{A}$	adjacency matrix
$\mathbf{I}$	identity matrix	$\mathbf{D}$	degree matrix
$\mathbf{L}$	Laplacian matrix	l	number of layers
$\Vert$ or $\oplus$	concatenation operator	$\lvert \cdot \rvert$	cardinality operator
$\odot$	Hadamard product	$f_{FC}, f_{EMB}, f_{CONV}$	neural network functions
f_{LSTM}, f_{GRU}	neural network modules	$\sigma(\cdot), \tanh(\cdot)$	sigmoid and tanh activation functions

2.4 Preliminaries and Definitions

This section defines most frequently used symbols and functions for convenience. Additionally, we define commonly used neural modules and formulate time-series tasks. The following typeset conventions are adopted for readability. We use a plain lowercase Latin letter to denote a scalar value or a function (e.g., c, d as scalar values, $f(\cdot), g(\cdot)$ as functions), a plain lowercase Greek letter to denote a hyperparameter or a parameter in the neural network (e.g., θ, λ as parameters), a plain uppercase Latin letter to denote a tensor (e.g., $X \in \mathbb{R}^{N \times T \times d}$), a bold lowercase Latin letter to denote a vector (e.g., $\boldsymbol{\theta}, \mathbf{b}$ as vectors), a bold uppercase Latin letter to denote a matrix (e.g., $\mathbf{X}, \mathbf{W}$ as matrices), a calligraphic uppercase Latin letter to denote a set (e.g., $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$). Subscripts and superscripts are also used to distinguish symbols when they share the same letters. A summary of notations is provided in Table 2.1. The definitions of *time series*, *graph*, and *graph time-series* are given below.

Definition 1 (Time Series)**.** A time series is defined as a sequence of data values either obtained from a continuous space by sampling or recorded in a discrete space. We let $\mathbf{x} = [x_1 x_2 \ldots x_T]^\mathsf{T} \in \mathbb{R}^T$ containing T observed scalar values denote **a univariate time series** of length T. When there are two or more time series (e.g., d time series) associated with an entity, **a multivariate time series** is usually created and we let $\mathbf{X} \in \mathbb{R}^{T \times d}$ denote a multivariate time series of length T and d variates. Some papers use d to denote the number of time features or hidden features. We overload the notation d to denote both the number of variates and the number of features, as the context makes it clear which definition is being

used. In time-series forecasting tasks, a lookback window containing some most recent values is often constructed to simplify model complexity. We let w denote the window size, and τ the prediction horizon.

Definition 2 (Graph)**.** Let $G = (V, E)$ denote a graph with V and E denoting its node set and edge set. The size of the node set is denoted by $N = |V|$, where $|\cdot|$ denotes the set cardinality operator. The adjacency matrix, denoted by $\mathbf{A} \in \mathbb{R}^{N \times N}$, is derived from the edge set, i.e., the element A_{ij} represents the edge weight from node i to node j. In the case where the graph is a connectivity graph, $\mathbf{A}$ is instead a binary matrix such that $\mathbf{A} \in \{0, 1\}^{N \times N}$. For each node $v \in V$, we let $\Gamma(v) \subset V$ denote the set of its neighbor nodes. The graph can contain node features, in which case we let $\mathbf{X} \in \mathbb{R}^{N \times d}$ denote the d dimensional node features. For convenience, we let $\mathbf{L}$ and $\mathbf{D}$ respectively denote the Laplacian matrix and the degree matrix of the adjacency matrix where $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$.

Definition 3 (Graph Time Series)**.** We define graph time series as time series X whose mutual relationships between time series are described in a graph $G(V, E, X)$. Depending on the context, graph time series can be univariate or multivariate. In graph-based time-series models, each node $v \in V$ in the graph is associated with a time series, hence, there are in total N time series in the dataset that forms node feature data $X \in \mathbb{R}^{N \times T \times d}$ of length T and d dimensions. This association is of one-on-one mappings, i.e., v_1 is associated with $\mathbf{X}_1$, v_2 is associated with $\mathbf{X}_2$, ..., and v_N is associated with $\mathbf{X}_N$. Without loss of generality, the dimension can be set $d = 1$ to include univariate time-series cases. Unless otherwise mentioned, we use a superscript to denote the time index, i.e., X^t denotes all values at the moment t. We use a subscript to denote the node and the multivariate index, i.e., $\mathbf{X}_i$ denote all values associated with node i.

2.4.1 Fundamental Computations in Neural Networks

For convenience, we formulate commonly used functions, layers, and modules in neural networks. We let $f_*(\cdot)$ denote them and distinguish them by a subscript, e.g., $f_{FC}(\cdot)$ stands for a fully connected layer. For the sake of conciseness, activation functions, vectorization, and cornerstone functions such as fully connected layers are depicted in the appendix (Sec. A). As defined in the appendix, we let $f_{FC}(\cdot), f_{softmax}(\cdot), f_{EMB}(\cdot), f_{CONV}(\cdot)$ denote a fully connected layer, a softmax layer, an embedding layer, and a convolutional layer, respectively. Interchangeably, we also use $\mathrm{FC}(\cdot)$ to denote the fully connected layer, and $\mathrm{Conv}(\cdot)$ to denote the convolutional layer, respectively.

Long Short-Term Memory (LSTM) is one of the most popular RNN variants. The input data to LSTM is a time series of T time step and d dimensions, denoted by $\mathbf{X} \in \mathbb{R}^{T \times d}$. The time series is fed to LSTM for each time step $t \in \{1, 2, \ldots, T\}$. At any time point t,

LSTM digests the input vector $\mathbf{x}^t \in d$ with equations below,

$$\mathbf{f}^t = \sigma\left(\mathbf{x}^t\mathbf{W}_{f1} + \mathbf{h}^{t-1}\mathbf{W}_{f2} + \mathbf{b}_f\right) \tag{2.4}$$

$$\mathbf{i}^t = \sigma\left(\mathbf{x}^t\mathbf{W}_{i1} + \mathbf{h}^{t-1}\mathbf{W}_{i2} + \mathbf{b}_i\right) \tag{2.5}$$

$$\mathbf{o}^t = \sigma\left(\mathbf{x}^t\mathbf{W}_{o1} + \mathbf{h}^{t-1}\mathbf{W}_{o2} + \mathbf{b}_o\right) \tag{2.6}$$

$$\mathbf{c}^t = \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \tanh\left(\mathbf{x}^t\mathbf{W}_{c1} + \mathbf{h}^{t-1}\mathbf{W}_{c2} + \mathbf{b}_c\right) \tag{2.7}$$

$$\mathbf{h}^t = \mathbf{o}^t \odot \mathbf{c}^t \tag{2.8}$$

where symbols σ, tanh, and $\odot$ denote the sigmoid function, the hyperbolic tangent function, and the Hadamard product, respectively. The dimension $c_{in} = d$ is transformed to the dimension c_{out}. The dimensions of the variables are $\mathbf{f}^t, \mathbf{i}^t, \mathbf{o}^t, \mathbf{c}^t, \mathbf{h}^t \in \mathbb{R}^{c_{out}}$, $\mathbf{W}_{*1} \in \mathbb{R}^{c_{in} \times c_{out}}$, $\mathbf{W}_{*2} \in \mathbb{R}^{c_{out} \times c_{out}}$, and $\mathbf{b}_* \in \mathbb{R}^{c_{out}}$. Variables $\mathbf{f}^0, \mathbf{i}^0, \mathbf{o}^0, \mathbf{c}^0, \mathbf{h}^0$ are initialized with user-selected values (e.g., $\mathbf{0}$). Thus, we let $f_{LSTM}(\mathbf{X}) = \left[h^1 h^2 \cdots h^T\right]^\intercal \in \mathbb{R}^{T \times c_{out}}$ denote LSTM.

Gated Recurrent Units (GRU) is another popular RNN variant. With the same input $\mathbf{X} \in \mathbb{R}^{T \times d}$ as that to LSTM, GRU is described by following equations,

$$\mathbf{r}^t = \sigma\left(\mathbf{x}^t\mathbf{W}_{r1} + \mathbf{h}^{t-1}\mathbf{W}_{r2} + \mathbf{b}_r\right) \qquad \mathbf{u}^t = \sigma\left(\mathbf{x}^t\mathbf{W}_{u1} + \mathbf{h}^{t-1}\mathbf{W}_{u2} + \mathbf{b}_u\right) \tag{2.9}$$

$$\mathbf{c}^t = \tanh\left(\mathbf{x}^t\mathbf{W}_1 + \left(\mathbf{r}^t \odot \mathbf{h}^{t-1}\right)\mathbf{W}_2 + \mathbf{b}\right) \qquad \mathbf{h}^t = \mathbf{u}^t \odot \mathbf{h}^{t-1} + \left(1 - \mathbf{u}^t\right) \odot \mathbf{c}^t \tag{2.10}$$

where the dimensions of variables are $\mathbf{r}^t, \mathbf{u}^t, \mathbf{c}^t, \mathbf{h}^t \in \mathbb{R}^{c_{out}}$, $\mathbf{W}_{*1} \in \mathbb{R}^{c_{in} \times c_{out}}$, $\mathbf{W}_{*2} \in \mathbb{R}^{c_{out} \times c_{out}}$, and $\mathbf{b}_* \in \mathbb{R}^{c_{out}}$. Thus, we let $f_{GRU}(\mathbf{X}) = \left[h^1 h^2 \cdots h^T\right]^\intercal \in \mathbb{R}^{T \times c_{out}}$ denote the GRU function.

Graph Convolutional Neural Networks (GCN) is a fundamental category of GNN models that transform node features with a graph structure. Given a graph $G = (V, E, \mathbf{A})$ and its node features $\mathbf{X} \in \mathbb{R}^{N \times d}$, a GCN model with l graph convolutional layers learns node embeddings by

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I} \qquad \hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{ij} \qquad \tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}} \tag{2.11}$$

$$\mathbf{H}^0 = \mathbf{X} \qquad \mathbf{H}^l = f_{GCN}^l(X, \mathbf{A}) = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{l-1}\mathbf{W}^{l-1}) \tag{2.12}$$

where $\mathbf{A}, \mathbf{I}, \mathbf{D}, \hat{\mathbf{A}}, \hat{\mathbf{D}} \in \mathbb{R}^{N \times N}$ are adjacency matrix, identity matrix, diagonal matrix, normalized adjacency matrix, and normalized degree matrix. $\mathbf{W}^* \in \mathbb{R}^{c_{in,*} \times c_{out,*}}$ with $c_{in,0} = d$, and $c_{in,l} = c_{out,l-1}$. When the context is clear, we let $f_{GCN}(X)$ denote $f_{GCN}(X, \mathbf{A})$.

Graph Attention Network (GAT) is another powerful GNN. GAT relies on attention scores instead of explicit edge weights for node message aggregation. The attention coefficients are computed with:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\intercal\left[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k \in \Gamma_i \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\intercal\left[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_k\right]\right)\right)} \tag{2.13}$$

$$\mathbf{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \Gamma_i \cup \{i\}} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j\right) \tag{2.14}$$

where α_{ij} is the attention score from node j in the neighboring node set Γ_i to node i, which is calculated by obtaining scores for all neighbor nodes of i, with a trainable parameter matrix $\mathbf{W}$ and the concatenated hidden states of node i and j, denoted by $\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_j$. A neural network layer LeakyReLU $\left(\mathbf{a}^{\mathsf{T}}(\cdot)\right)$ is applied with a trainable parameter vector $\mathbf{a}$ and a softmax normalization, as described by Eq. 2.13.

The attention scores can be extended to K channels, therefore, a new hidden state $\mathbf{h}'_i$ for node i is calculated as the average of multihead attention, and each channel is calculated with attention scores α^k_{ij}, a trainable parameter matrix $\mathbf{W}^k$, and hidden states $\mathbf{h}_j$ where $j \in \Gamma_i \cup \{i\}$, in the local neighborhood of node i.

2.4.2 Problem Definitions

Graph time-series models essentially target various time-series tasks. This section formulates some of the most important ones, namely, time-series classification [116, 147, 148, 220], time-series forecasting [23, 34, 117, 146, 163, 167, 180, 190, 205], and anomaly detection on time-series [19, 51, 228]. Without the loss of generality, we assume the input to all these tasks is associated with a graph G, where each time series is associated with a node of G.

Graph Time-series Classification aims to predict labels of time-series. Assuming that the input is time series $X \in \mathbb{R}^{N\times T\times d}$ and there is a label set $\mathcal{Y} = \{1, 2, \ldots, C\}$ of C different classes, a classification model learns a mapping from each time series to its label $f_{\text{classification}} : (G, X) \mapsto \mathbf{y}$ where $\mathbf{y} \in \mathcal{Y}^N$. This task is useful in many areas, especially in the field of robotics, where gesture classification is predicted based on motion series. A graph structure is utilized to model the relationships between motions, aiming to improve classification accuracy.

Graph Time-series Forecasting aims to predict values in the future horizon given historical observations. Assuming that the input is time series $X \in \mathbb{R}^{N\times T\times d}$ and the target prediction horizon is τ, a forecasting model learns a mapping from the time series to predicted future values $f_{\text{forecasting}} : \left(G, X^{1:T}\right) \mapsto \hat{X}^{T+1:T+\tau}$. This task is widely studied for various applications, including traffic speed forecasting, crime forecasting, and epidemiological forecasting. In these applications, the time series are associated with geographic locations and exhibit mutual interactions between each other. In graph time-series models, these interactions can be effectively modeled within the graph structure.

Graph Time-series Anomaly Detection can be categorized into three levels, i.e., time-series level, period level, and time point level. Time-series level anomaly detection tasks are essentially binary classification tasks where a time series is classified as normal or not normal. Period-level anomaly detection aims to find out irregular periods of time series. Given time series $\mathbf{X} \in \mathbb{R}^{N\times T}$, a period-level anomaly detection model learns a mapping from time series to a set of periods $f_{detection} : \mathbf{X} \mapsto \{(s_p, t_p) \mid p = 1, 2, \ldots, P\}$, where P denotes the number of detected anomalous periods, with s_p and t_p marking the starting and ending index of each

period. Time point level anomaly detection can be seen as a special case of the period level where the period has a length of 1. These fundamental definitions can be modified without difficulty to agree with the actual formulation such as dynamic graphs or multivariate time series. Applications of this task include passenger flow anomaly detection, soil moisture anomaly detection, among others [102].

Table 2.2: Selected representative graph time-series models

Years	Graph Recurrent/Convolutional Neural Networks	Graph Attention Neural Networks
2018 or before	GCRN [166], DCRNN [121], STGCN [217]	GaAN [220]
2019	Graph WaveNet [202], T-GCN [229], LRGCN [116]	ASTGCN [75]
2020	MTGNN [204], DGSL [167]	MTAD-GAT [228], STAG-GCN [133] STGNN [190], Cola-GNN [53], ST-GRAT [149]
2021	FC-GAGA [146], Radflow [180] STFGNN [117], Z-GCNET [34], TStream [33]	GDN [51], StemGNN [23]
2022	VGCRN [32], GRIN [39] RGSL [218], GANF [45], ESG [212]	GReLeN [224] FuSAGNet [80], THGNN [206]

2.5 Deep Graph Time-series Modeling

Recent graph time-series models are primarily divided into two categories: **Graph Recurrent/Convolutional Neural Networks (GRCNN)** and **Graph Attention Neural Networks (GANN)**. GRCNN models leverage autoregressive or convolutional layers to learn the temporal dependency between time values (Sec. 2.5.1.1 and Sec. 2.5.1.2). We also discuss GRCNN models regarding the use of self-derived graphs (Sec. 2.5.1.3) and evolving graphs (Sec. 2.5.1.4). GANN models integrate attention mechanisms to capture various dependencies. Different attention mechanisms provide possible interpretations of mutual impact between nodes and between values across timestamps. We discuss different attention types in detail throughout Sec. 2.5.2. Note that our categories are not mutually exclusive, and a model may fall into different categories. In this situation, we discuss the model in the most appropriate category. Throughout the discussion, we summarize and highlight key functions or modules of the introduced models, with detailed model equations included in the appendix for reference (Sec. B). A timeline of surveyed models is provided in Table 2.2. Details of related experimental setups such as time-series normalization and graph construction are also given in the appendix (Sec. C).

2.5.1 Graph Recurrent/Convolutional Neural Networks

An intuitive way of modeling graphs and time series is to combine a graph model and a time-series model. In deep learning, RNN and CNN are commonly used for time-series modeling,

Table 2.3: An attribute comparison of selected GRCNN models.

Categories	Models	Time series			Propagation			Graph Types			Evolving Graphs
		RNN	CNN	Gates	GCN	Diffusion	Gates	Spatial	Temporal	Semantic	
RNN-based Graph Time-series Modeling (Sec. 2.5.1.1)	GCRN [166]	✓			✓			✓			
	DCRNN [121]	✓				✓		✓			
	T-GCN [229]	✓			✓			✓			
	DGSL [167]	✓				✓				✓	
	VGCRN [32]	✓			✓					✓	
	GANF [45]	✓			✓			✓			
	GRIN [39]	✓			✓			✓			
CNN-based Graph Time-series Modeling (Sec. 2.5.1.2)	STGCN [217]		✓		✓	✓					
	G-WaveNet [202]		✓		✓			✓		✓	
	MTGNN [204]		✓		✓				✓		
Models with Self-derived Graphs (Sec. 2.5.1.3)	FC-GAGA [146]			✓			✓		✓	✓	
	STFGNN [117]	✓				✓		✓	✓		
	RGSL [218]	✓			✓			✓		✓	
Models with Evolving Graphs (Sec. 2.5.1.4)	Radflow [180]	✓						✓			✓
	TStream [33]	✓			✓			✓			✓
	LRGCN [116]	✓			✓			✓			✓
	Z-GCNET [34]		✓		✓			✓			✓
	ESG [212]	✓		✓	✓			✓	✓		✓

while GNN is used for graph modeling. Table 2.3 provides a comparison of Graph Recurrent/Convolutional Neural Networks (GRCNN) with respect to their time-series modeling and graph modeling. Most GRCNN models build a graph where each node is associated with a time series, either univariate or multivariate. Unless otherwise mentioned and without loss of generality, we assume the associated time-series are all multivariate and share the same feature dimension size in all nodes. In this section, we first discuss models from the perspective of time-series modeling, as in Sec. 2.5.1.1 and Sec. 2.5.1.2, then we discuss models from the perspective of graph modeling, as in Sec. 2.5.1.3 and Sec. 2.5.1.4. We summarize various model attributes in Table 2.3.

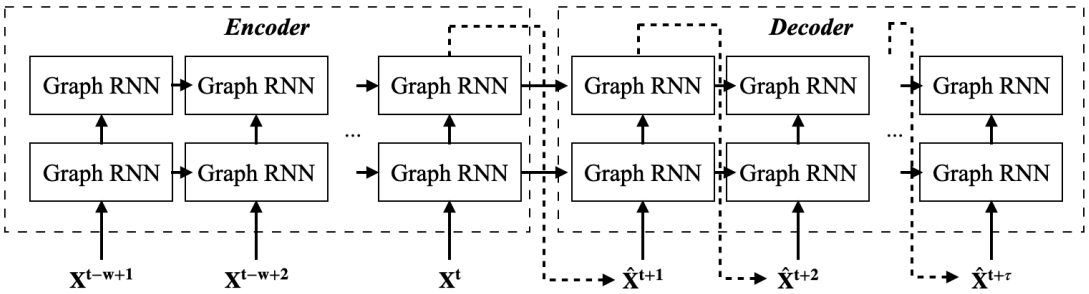


Figure 2.1: A diagram of Graph RNN models. The graph component (e.g., GCN or diffusion) is nested within RNN cells.

2.5.1.1 RNN-based Graph Time-series Modeling

RNN models, as described in Eq. 2.4-Eq. 2.10, can perform node message propagation by substituting their weight matrix multiplication with graph convolution or graph diffusion. Hence, the modified RNN models not only model the temporal dependency through its recursive nature but also capture the nonlinear mutual impact among nodes. In this category, we cover GCRN, DCRNN, and DGSL, all of which nest a GNN model within an RNN cell, as depicted in Fig. 2.1.

GCRN [166] substitutes fully connected layers in LSTM with graph convolutional layers. Specifically, the weight matrix multiplication for all gates in LSTM is replaced with GCN, resulting in an LSTM variant, denoted by $\textsc{lstm}^*$,

$$\mathbf{H}^t = \textsc{lstm}^*_{f_{GCN},G}\left(\mathbf{X}^t, \mathbf{H}^{t-1}\right) \tag{2.15}$$

GCRN leverages the graph structure for training which allows each node to efficiently aggregate neighborhood information, in contrast to a raw RNN (in this case, the LSTM model) that models the dependency between nodes by fully connected layers. The GCRN model takes a similar form as an LSTM model, therefore, the derived hidden state in Eq. 12 is used in time-series forecasting tasks in the same way as how the hidden state from LSTM is used. See Sec. B.1 for detailed equations.

DCRNN [121] uses diffusion convolution to replace the weight matrix multiplication in a different RNN variant, the Gated Recurrent Units (GRU). The graph diffusion operator, denoted by $g_{\boldsymbol{\theta}}(\cdot)$ with respect to parameters $\boldsymbol{\theta}$ and an adjacency matrix $\mathbf{A}$, is defined as

$$g_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{l=0}^{L-1}\left(\theta_{l,1}\left(\mathbf{D}_O^{-1}\mathbf{A}\right)^l + \theta_{l,2}\left(\mathbf{D}_I^{-1}\mathbf{A}^\mathsf{T}\right)^l\right)\mathbf{X} \tag{2.16}$$

where L denotes the diffusion depth. $\mathbf{D}_O$ and $\mathbf{D}_I$ denote the in-degree matrix and the out-degree matrix, respectively. DCRNN utilizes the random walk matrix $\mathbf{D}_O^{-1}\mathbf{A}$ and $\mathbf{D}_I^{-1}\mathbf{A}^\mathsf{T}$ for information propagation. Regarding the temporal component, GRU is chosen instead of LSTM, which brings the benefit of lighter computing workload due to the simpler structure of GRU model. The complete $\textsc{dcrnn}\left(\mathbf{X}^t, G\right)$ model can be described by

$$\mathbf{R}_t = \sigma\left(g_R\left[\mathbf{X}^t,\ \mathbf{H}^{t-1}\right] + \mathbf{b}_R\right) \tag{2.17}$$

$$\mathbf{U}_t = \sigma\left(g_U\left[\mathbf{X}^t,\ \mathbf{H}^{t-1}\right] + \mathbf{b}_U\right) \tag{2.18}$$

$$\mathbf{C}_t = \tanh\left(g_C\left[\mathbf{Y}_t,\ \left(\mathbf{R}^t \odot \mathbf{H}^{t-1}\right)\right] + \mathbf{b}_C\right) \tag{2.19}$$

$$\textsc{dcrnn}\left(\mathbf{X}^t, G\right) \coloneqq \mathbf{H}_t = \mathbf{U}_t \odot \mathbf{H}^{t-1} + \left(1 - \mathbf{U}^t\right) \odot \mathbf{C}^t \tag{2.20}$$

DCRNN inspires many models, including DGSL [167], T-GCN [229], and GraphDF [27].

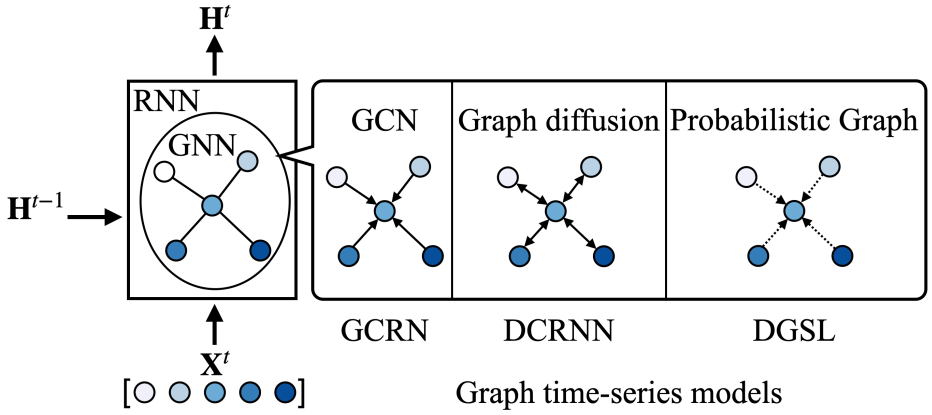


Figure 2.2: A detailed diagram of a Graph RNN cell. $\mathbf{X}^t$ denotes all time-series values at time t. Each node is associated with an individual time-series. The interrelations between time series are captured through GNN, which is differently employed in different selected graph time-series models. In GCRN, each node learns from its neighbors through GCN. In DCRNN, its diffusion learning process considers both the in-degree and out-degree of each node, as represented by the dual-headed arrows. In DGSL, the utilized graph structure is probabilistic, as indicated by the dashed arrows.

DGSL [167] can be seen as a DCRNN variant with a probabilistic graph, where the graph structure is learned from time-series data. Given time-series data $X \in \mathbb{R}^{N\times T\times d}$, a graph $\mathbf{A} \in \{0,1\}^{N\times N}$ is parameterized and sampled by the following equations,

$$\mathbf{z}_i = f_{FC,z}\left(\mathrm{Vec}\left(f_{conv,T}(\mathbf{X}_i)\right)\right) \qquad \theta_{ij} = \sigma\left(f_{FCs}\left(\mathbf{z}_i \,\|\, \mathbf{z}_j\right)\right) \tag{2.21}$$

$$\mathbf{A}_{ij} = \sigma\left(\frac{\log\frac{\theta_{ij}}{1-\theta_{ij}} + \left(g_{ij}^1 - g_{ij}^2\right)}{s}\right), \quad i,j \in \{1,2,\ldots,N\} \tag{2.22}$$

where $g_{ij}^1, g_{ij}^2 \sim \mathrm{Gumbel}(0,1)$ (Defined in Sec. C.5). The parameter θ_{ij} for each node pair is learned from a link prediction method that takes node embeddings as input, as shown in Eq. 2.21. A binary graph is constructed by sampling edges from distributions in Eq. 2.22, which leverages the Gumbel reparameterization method [98], with s a selected temperature parameter. With the sampled graph, DGSL renders the DCRNN model for sequence-to-sequence training and forecasting. Since the graph is parameterized with the time-series data, DGSL optimizes trainable parameters for both time series and graph learning simultaneously.

More recent methods construct graphs in various ways for tasks other than time-series forecasting. For instance, VGCRN [32] takes a probabilistic approach and builds a deep variational network for time-series anomaly detection. GANF [45] builds a graph-augmented flow model with a Bayesian network that models the causal relationships between times series. Moreover, GRIN [39] targets time-series imputation with GNN and RNN.

Model Comparison. For graph modeling, GCRN uses GCN that operates on undirected

graphs, while DCRNN and DGSL use graph diffusion which has the advantage of modeling directed graphs. Further, GCRN and DCRNN require external graph structures, while DGSL takes a probabilistic approach and learns graph structures from node embeddings, without using a predefined graph. A diagram of Graph RNN cell is shown in Fig. 2.2, which highlights the distinctions and commonalities among the discussed models.

2.5.1.2 CNN-based Graph Time-series Modeling

Another line of work models time series through CNN instead of RNN.

STGCN [217] models graph structure and temporal dependency individually with separate layers, instead of nesting graphs in an RNN structure. STGCN introduces a spatio-temporal convolutional block that consists of temporal-spatial-temporal layers. The temporal layer is built upon a 1D convolutional layer with gated linear units, which needs less time compared to RNN models. The spatial layer, on the other hand, is a GCN layer. Therefore, STGCN inspires a separate and individual view and usage of temporal and spatial dimensions.

Graph WaveNet [202] points out that the explicitly given graph structure may not be able to represent relations between nodes due to missing node connections. To mitigate the missing information and the ineffectiveness of the RNN component, Graph WaveNet develops an adaptive adjacency matrix. Moreover, it replaces RNN models with stacked dilated convolutional layers. With respect to the graph modeling, Graph WaveNet adapts the graph diffusion layer from DCRNN by adding a third term of an adaptive adjacency $\mathbf{A}_{adapt}$ to the diffusion matrix in Eq. 2.16:

$$\mathbf{A}_{adapt} = f_{softmax}\left(\text{ReLU}\left(\mathbf{H}_{emb1}\mathbf{H}_{emb2}^{\mathsf{T}}\right)\right) \tag{2.23}$$

where the adaptive matrix is learned through the embeddings $\mathbf{H}_{emb1}$ and $\mathbf{H}_{emb2}$, which are independent of the given graph structure. Both $\mathbf{H}_{emb1}$ and $\mathbf{H}_{emb2}$ are randomly initialized.

MTGNN [204] also builds a graph through node embeddings and requires no prior graph structure knowledge. MTGNN models time series and graphs separately and subsequently incorporates them in a pipeline order. For the sake of conciseness, we focus on its graph modeling and let $g_{cnn}(\cdot)$ denote the temporal component, which renders dilated CNN layers to drastically reduce the temporal dimension of time series. Given time-series data $\mathbf{X} \in \mathbb{R}^{N\times T\times D}$, node embeddings $\mathbf{H}_{emb_1}, \mathbf{H}_{emb_2} \in \mathbb{R}^{N\times d_{emb}}$, an adjacency matrix is computed by,

$$\mathbf{Z}_1 = \tanh\left(\alpha f_{FC,W_1}\left(\mathbf{H}_{emb_1}\right)\right) \qquad \mathbf{Z}_2 = \tanh\left(\alpha f_{FC,W_2}\left(\mathbf{H}_{emb_2}\right)\right) \tag{2.24}$$

$$\mathbf{A}_0 = \text{ReLU}\left(\tanh\left(\alpha\left(\mathbf{Z}_1\mathbf{Z}_2^{\mathsf{T}} - \mathbf{Z}_2\mathbf{Z}_1^{\mathsf{T}}\right)\right)\right) \qquad \mathbf{A} = \underset{row}{\text{topk}}(\mathbf{A}_0) \tag{2.25}$$

where two individually calculated node embeddings are transformed by fully connected layers and a $\tanh(\cdot)$ activation with a selected hyperparameter α, as shown in Eq. 2.24. Note that $\mathbf{A}$ is guaranteed to be asymmetric, and row-wise top K selections are used to further sparsify connections, as formulated in Eq. 2.25.

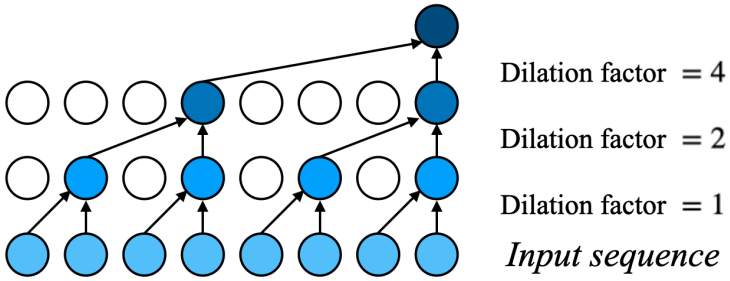


Figure 2.3: A model with three dilated CNN layers (dilation factor as $[1, 2, 4]$) can digest 8 time values, which is generally faster than an RNN structure unfolding 8 times.

To preserve the initial graph structure denoted by $\mathbf{A}$, MTGNN leverages a random walk with restart method to derive multilevel hidden states for each node and thereafter concatenates all hidden states. Let $\mathbf{H}^0 = g_{cnn}(\mathbf{X})$ denote the initial hidden state from the temporal component, the multilevel hidden states are computed and aggregated with the transformed graph structure $\tilde{\mathbf{A}}$ from Eq. 2.11, resulting in the hidden states at l layer as,

$$\mathbf{H}^l = \beta\mathbf{H}^0 + (1-\beta)\tilde{\mathbf{A}}\mathbf{H}^{l-1} \tag{2.26}$$

$$\mathbf{H} = \sum_l f_{FC,W_l}(\mathbf{H}^l) \tag{2.27}$$

Model Comparison. STGCN, Graph WaveNet, and MTGNN use different CNN variants. STGCN uses a gated causal CNN and Graph WaveNet uses a dilated CNN. MTGNN combines several dilated CNNs to derive its inception dilated CNN, which allows it to model time-series values regarding various granularities. We also notice that STGCN uses a GCN layer to model the provided spatial graph, while Graph WaveNet proposes an adaptive graph that combines both the spatial graph and the self-derived semantic graph. MTGNN only relies on a self-derived semantic graph and requires no explicit graph from datasets.

Model Comparison on Time-series Components: RNN versus CNN. To model a time-series in deep learning, one of the most intuitive ways is to apply a fully connected layer or a series of fully connected layers to the input time-series data and update network parameters by fitting predicted values with actual ground truth [14]. Each neuron in the input layer holds a value for each time step, and neurons in the following layer summarize values at all time steps by network parameters, however, this leads to an explosive growth of parameters and inefficiency of optimization as the number of layers increases. To address these issues, convolutional layers are preferred, which drastically reduce model complexity with fewer neural connections between layers. Further, a dilated CNN layer (shown in Fig. 2.3) models very long sequences with low model complexity, as it downsamples the time-series input with a preselected frequency. CNN-based models are also called direct methods since they predict future values in one shot [53], on the contrary, RNN-based models capture the temporal dependency by recurrently consuming the time values to derive

a final embedding. Due to this time-series modeling nature, RNN is most widely used in temporal modeling, as explained in Sec. 2.3.1. RNN-based graph time-series models nest graph modeling within RNN cells, and are therefore relatively more constrained regarding the model design, compared to CNN-based models where graph and time series are separately modeled with different layers, which allows diverse combinations, such as the temporal-spatial-temporal structure in STGCN.

2.5.1.3 Models with Self-derived Graphs

Most related datasets such as traffic data and electricity workload data provide an external graph that primarily describes the geographical connectivity of nodes, however, external graph structures are not always available, nor do they necessarily reflect the actual connections between nodes. In these situations, some models propose using self-derived graphs, i.e., the graph structure is derived either from time series or from node embeddings. This section describes models from the perspective of graph modeling, specifically on how different models leverage self-derived graphs to capture the interrelations between nodes. We discuss three models in this section, FC-GAGA [146], STFGNN [117], and RGSL [218]. Other models such as Graph WaveNet and MTGNN that are discussed in the previous section, also belong to this category. The graph types in different models are summarized in Table 2.3. At the end of this section, we compare models in terms of their graph modeling.

FC-GAGA [146], similar to MTGNN, requires no prior knowledge of graph structure. Note that FC-GAGA uses time-series gates instead of RNN or CNN for time-series modeling. The time-series gates can be seen as a special type of CNN as they are used to weight time covariate features. Let $\mathbf{X} \in \mathbb{R}^{N \times T}$ denote the node time series, and let $\mathbf{H}_{emb} = f_{EMB}(\mathbf{X}) \in \mathbb{R}^{N \times d_{emb}}$ denote the node embeddings. FC-GAGA is described by following equations,

$$\mathbf{A} = \exp\left(\epsilon \mathbf{H}_{emb} \mathbf{H}_{emb}^{\mathsf{T}}\right) \qquad \tilde{\mathbf{x}}_i = \max_j \mathbf{X}_{ij} \qquad \mathbf{G}_{i,j+k} = \text{ReLU}\left[\frac{\mathbf{A}_{ij}\mathbf{X}_{jk} - \tilde{\mathbf{x}}_i}{\tilde{\mathbf{x}}_i}\right] \tag{2.28}$$

$$\mathbf{Z} = \left[\mathbf{H}_{emb} \,\|\, \frac{\mathbf{X}}{\tilde{\mathbf{x}}} \,\|\, \mathbf{G}\right]^{\mathsf{T}} \qquad \hat{\mathbf{X}} = f_{res}(\mathbf{Z}) \tag{2.29}$$

where the graph structure $\mathbf{A}$ is learned and optimized from the node embeddings and is used together with transformed time-series data $\tilde{\mathbf{x}}_i$ in a gated mechanism to shut off connections of irrelevant node pairs as in Eq. 2.28, with ϵ as a selected constant hyperparameter. A hidden state $\mathbf{Z}$ is constructed from a concatenation of node embedding $\mathbf{H}_{emb}$, scaled time-series data $\mathbf{X}/\tilde{\mathbf{x}}$, and the gated states $\mathbf{G}$. Subsequently, $\mathbf{Z}$ serves as the initial input for a residual module, denoted by f_{res}, which generates the time-series prediction $\hat{\mathbf{X}}$. For the purpose of conciseness, we include the details of the residual module f_{res} in Sec. B.2. FC-GAGA benefits from the freedom of automatically deriving graph structures, instead of relying on the Markov model-based or distance-based graph topological information. However, the graph construction from node embedding costs a time complexity of $\mathcal{O}(N^2)$ which limits the scalability of FC-GAGA.

STFGNN [117], unlike many models that only use one graph, integrates three graphs with a fusion layer, where each graph encodes one type of information. STFGNN defines a temporal graph $\mathbf{A}_t$ that encodes temporal similarity relationships between graph time series with a Dynamic Time Warping (DTW) variant (defined in Sec. C.3), a spatial graph $\mathbf{A}_s$ that is derived from the geographical distance between nodes, and a connectivity graph $\mathbf{A}_c$ that indicates the connections of nodes between two adjacent time steps. The three matrices are then arranged into a spatial-temporal fusion graph $\mathbf{A} \in \mathbb{R}^{KN \times KN}$ with a user-selected slice size K for hidden state learning.

RGSL [218] also combines external graph structures and semantic graphs with the aid of the Gumbel softmax function. RGSL incorporates domain knowledge in its semantic graphs.

Graph Modeling: Goals and Limitations. The objective of graph modeling in graph time-series models is similar to that in ordinary deep graph models, i.e., to effectively and efficiently utilize the neighbor information for each node. Great efforts are made to increase scalability in order to address the limitation of time complexity. As a consequence, many GNNs only require $\mathcal{O}(|E|)$ [106, 121, 166, 183, 205, 217, 220, 228] instead of $\mathcal{O}(N^2)$ [146, 190, 204], hence, graph sparsification on edges significantly reduces time complexity.

In distinction to ordinary deep graph models, graph structures in graph time-series models are strongly related to the time-series. Researchers should be aware of their constraints, some

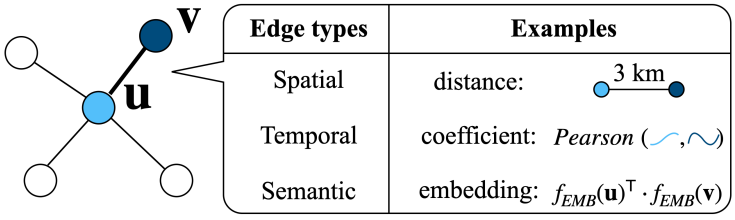


Figure 2.4: A diagram of three methods of graph constructions.

of which are listed as follows, (a) the graph density can limit the computation efficiency. Although many GNN models reduce the time complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(|E|)$, when the graph is dense (a complete graph in the worst case), the number of edges will be close to the square of the number of nodes, i.e., $|E| \approx N^2$, which causes the model to degrade to the worst time complexity. This is a severe limitation for tasks that require a timely response. (b) Moreover, although deep learning models whose modeling power generally increases as the depth of the model or the number of layers increases, this does not hold true for deep graph models by simply adding graph neural layers. When stacking many graph neural layers, included distant nodes cause the problem of oversmoothing, where the locality information is not well utilized due to message propagation from a large neighborhood set [130, 132, 203, 230]. Graph time-series models having a nested GNN within an RNN structure, are more susceptible to oversmoothing when modeling long sequences, due to the occurrence of message propagation at each RNN unfolding step. (c) Another problem is oversquashing which is due to the excessive neighbor node information while the learned output is a fixed-length vector, therefore, the loss of information is unavoidable [2].

Model Comparison on Graph Components: Graph Construction methods. Many graph time-series datasets contain a predefined graph. For example, Wikipedia data have a graph that represents linking relations between sites [180]. In biological networks, a graph is provided to represent links between bio-entities [53]. However, for other datasets that do not explicitly provide a graph, some metrics are needed to derive one. We describe three ways to derive a graph, namely, the constructions of a spatial graph, a temporal graph, and a semantic graph, respectively. Fig. 2.4 illustrates these graph construction approaches, each accompanied with examples. (a) **Spatial Graphs** are based on distance information [121]. Directly using distance as edge weights of graphs will cause closer nodes to have smaller edge weights, due to the shorter distance between them. However, most GNN models require a closeness graph where greater edge weights represent stronger connections. One common way to convert a distance graph to a closeness graph is through the radial basis function (RBF) (defined in C.2). A special case is *Connectivity Graphs* which are binary, i.e., an edge exists and has the edge weight as 1 between two nodes if and only if they are connected. Under the assumption of the first law of geography, *Everything is related to everything else, but near things are more related than distant things*, spatial graphs and connectivity graphs are the intuitive choices if they are available in the datasets,

as they are used in most models. (b) **Temporal Graphs** are constructed based on the temporal similarity between node time series [133]. By formatting each node time series as a sequence, many sequence similarity metrics can be used to derive a temporal graph, including cosine similarity, coefficient metrics, and Dynamic Time Warping (DTW), among many others. In the aforementioned GRCNN models, for instance, STFGNN [117] uses DTW to derive temporal graphs. Temporal graphs are effective if connections and the mutual impact between nodes are highly correlated to their time-series patterns. Besides, temporal graphs go beyond the first law of geography as they may connect two highly correlated node time-series that are geographically distant. (c) **Semantic Graphs** are constructed based on node embeddings to connect nodes that are semantically close to each other, that these nodes share some similar hidden features between them [146]. In semantic graphs, spatially distant nodes may share similar features and are closely connected [117]. The level of similarity can be measured by an embedding similarity metric, such as the correlation coefficient. Additionally, edge embeddings can be used as parameters of distribution functions, from which the edge weights are sampled [167]. It is also possible to integrate more than one semantic type of graph [117, 133]. Semantic graphs have the advantage of being independent from external graph structures or time-series patterns. In practice, a combination of various graph types is commonly adopted, through cascade processing [146], fusion [117], or using one graph as a mask for another graph [190].

2.5.1.4 Models with Evolving Graphs

Temporal dynamics are not only limited to lie in the node level time series but can also exist in evolving graph structures, which induces another level of modeling complexity. In contrast to a static graph, a series of graphs are used in evolving graph models, denoted by $\boldsymbol{\mathcal{G}} = (G_1, G_2, \ldots, G_T)$ where a graph snapshot $G_t = (V_t, E_t)$ exists for each time step t. The number of nodes is dynamically dependent on the time step, as $N_t = |V_t|$. Since the dynamics in evolving graphs are much more complicated to model, this section only briefly describes representative models.

Radflow [180] independently models a spatial component and a temporal component, and subsequently performs a component combination by summing them up. The input of Radflow is a series of graphs $\mathcal{G}$ and node multivariate time series $X \in \mathbb{R}^{N \times T \times d}$. A connectivity graph is derived from the dataset for each time step, hence, there is a series of adjacency matrices, denoted by $A \in \mathbb{R}^{N \times N \times T}$. Radflow consists of following equations,

$$\hat{\mathbf{Q}}^t = f_{res,Q} \cdot f_{LSTM}(X) \qquad \hat{\mathbf{U}}^t = f_{attn} \cdot f_{res,U} \cdot f_{LSTM}(X) \tag{2.30}$$

$$\hat{\mathbf{X}}^t = f_{FC,recurrent}\left(\hat{\mathbf{Q}}^t\right) + f_{FC,graph}\left(\hat{\mathbf{U}}^t\right) \tag{2.31}$$

At time step t, the prediction $\hat{\mathbf{X}}^t$ is the summation of two variables $\hat{\mathbf{Q}}^t$ and $\hat{\mathbf{U}}^t$, which are respectively calculated from a recurrent component and a graph component, as illustrated

in Eq. 2.31. Both components utilize a residual module and a LSTM model. The graph component uses an extra attention module that is based on the graph attention mechanism and the general attention mechanism. Detailed equations can be found in Sec. B.3.

Radflow learns node embeddings that are dependent on time steps and use them for interpretable prediction and imputation. The graph connectivity information is used to select neighbor nodes for attention computation of each node, without creating trainable parameters over graphs, which makes it relatively lightweight, scalable, and fast compared to GCN-based methods.

TStream [33] adapts GNN models in a continual learning manner to quickly learn expansive evolving network patterns. Under the assumption $\Delta N_t \ll N_t$, TStream leverages Jensen-Shannon divergence (JSD) to measure the similarity between the distributions of two derived hidden states in adjacent time steps. Those nodes with high JSD scores are considered as having drastic changes and are updated together with newly added nodes. An information replay method and a parameter smoothing method are used to avoid historical observations from being forgotten.

LRGCN [116] targets a path classification problem in time-evolving graphs. The model predicts possible path failures in the real world, such as those in traffic networks or telecommunication networks. The prediction is based on path embedding which is an aggregation (by LSTM and attention layers) of all node embeddings along the paths. **Z-GCNET** [34] proposes a time-aware persistent homology representation learning method to track topological features and use them in traffic forecasting and cryptocurrency price forecasting. **ESG** [212] claims that evolving graph structures vary depending on the time scale of interests. Its proposed model learns a different graph for each time scale. In summary, evolving graph models are more complicated than previous categories due to their requirement of modeling the dynamic graph structures as well as the dynamics of time series and node relations. A series of graphs are typically given or derived from datasets and one key challenge is to handle these graphs without an explosion of parameters.

2.5.2 Graph Attention Neural Networks

Attention mechanisms are used in graph time-series models, which help interdependency modeling between nodes and between time steps. We refer to graph time-series models that use any attention mechanism as Graph Attention Neural Networks (GANN). We discuss three attention types, spatial/graph, temporal and general. A majority of GANN models leverage graph attention, which can be seen as a special combination of a spatial graph and a semantic graph. Due to the significance of attention mechanisms, we discuss GANN models in this section instead of putting related models under previous sections. We further categorize GANN models to those for forecasting and those for anomaly detection. An attribute comparison of GANN models is presented in Table 2.4.

Table 2.4: An attribute comparison of selected GANN models

Categories	Models	Attention types			Tasks		
		Spatial/Graph	Temporal	General	Classification	Regression	Anomaly Detection
Attention for Forecasting (Sec. 2.5.2.1)	GaAN [220]	✓			✓	✓	
	Cola-GNN [53]	✓				✓	
	ASTGCN [75]	✓	✓			✓	
	STAG-GCN [133]	✓	✓	✓		✓	
	STGNN [190]	✓		✓		✓	
	StemGNN [23]	✓		✓		✓	
	Radflow [180]	✓				✓	
	ST-GRAT [149]	✓	✓	✓		✓	
	THGNN [206]		✓	✓	✓		
Attention for Anomaly Detection (Sec. 2.5.2.2)	MTAD-GAT [228]	✓					✓
	GDN [51]	✓					✓
	GReLeN [224]			✓			✓
	FuSAGNet [80]	✓					✓

2.5.2.1 Attention for Forecasting

We discuss six models that use graph attention for time-series forecasting, namely, GaAN [220], Cola-GNN [53], ASTGCN [75], STAG-GCN [133], STGNN [190], and StemGNN [23].

GaAN [220] utilizes a convolutional subnetwork to control the importance of each attention head. GaAN adds a graph aggregator with a gated characteristic, which is different from the GAT model (Eq. 2.14),

$$\mathbf{H}'_i = f_{FC}\left(\mathbf{H}_i \,\|\, \left[\Big\|_{k=1}^{K} w_i^k \sum_{j\in\Gamma_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{H}_j\right]\right) \tag{2.32}$$

$$\mathbf{w}_i = \left[w_i^1 w_i^2 \dots w_i^K\right] = g_{pool}(\mathbf{H}_i, \mathbf{H}_{\Gamma_i}) \tag{2.33}$$

with α_{ij}^k calculated by Eq. 2.13, and $\mathbf{w}$ the gate vector. g_{pool} denotes a designed convolutional subnetwork that learns gate values from node i and its neighbor nodes. For example, GaAN can leverage max pooling and average pooling in g_{pool}. The model is used for both classification and forecasting. When used in time-series forecasting, GaAN can be seen as a GCRN variant with the graph component replaced by the GaAN structure, while the LSTM structure is preserved.

Cola-GNN [53] leverages the graph attention mechanism in a pairwise manner for influenza-like illness forecasting. **ASTGCN** [75] fuses various blocks of graph attention layers and temporal attention layers. **STAG-GCN** [133] is an adaptive gated GCN method for traffic forecasting. It leverages a distance-based spatial graph, and a semantic graph that is based on DTW. STAG-GCN also adopts the general attention mechanism (Eq. 2.40) for temporal modeling. Similarly, **STGNN** [190] and **StemGNN** [23] adopt the general attention mechanism with incorporated states in their spatial component. Most of these forecasting models target regression tasks, however, there are also classification models. For example, **THGNN** [206] forecasts the direction of stock prediction movement by employing graph structure learning through graph attention and general attention.

2.5.2.2 Attention for Anomaly Detection

MTAD-GAT [228], GDN [51], and GReLeN [224] target time-series anomaly detection.

MTAD-GAT [228] utilizes GAT models to decide whether there is a time point level anomaly. Given time-series data $\mathbf{X} \in \mathbb{R}^{T \times d}$, MTAD-GAT is described by following equations,

$$\mathbf{H}_d = f_{GAT,d}\left(\mathbf{X}\right) \qquad \mathbf{H}_T = f_{GAT,T}\left(\mathbf{X}\right) \qquad \tilde{\mathbf{X}} = \underset{min-max}{\text{Norm}}\left(\mathbf{X}\right) \tag{2.34}$$

$$\mathbf{H} = f_{GRU}\left(\left[\mathbf{H}_d \,\|\, \mathbf{H}_T \,\|\, \hat{\mathbf{X}}\right]\right) \tag{2.35}$$

MTAD-GAT is a self-supervised method that leverages GAT from two perspectives: (a) by treating each feature dimension as a node, a GAT model is used to extract relations between features and derives $\mathbf{H}_d$, and (b) by treating each time step as a node, another GAT model is used to capture relations between time steps and derives $\mathbf{H}_T$. The feature-oriented GAT and the time-oriented GAT are fused with the normalized data, as shown in Eq. 2.34. A GRU structure is used to capture the long-term temporal dependency and derive the final hidden state $\mathbf{H} \in \mathbb{R}^{T \times d_{hid}}$, as shown in Eq. 2.35. Finally, $\mathbf{H}$ is fed to a fully connected network and a VAE network to derive a forecasting loss and a reconstruction loss, respectively.

GDN [51] is also a point level multivariate time-series anomaly detection model, which renders a GAT model to aggregate information from top-k neighbors for each node. Given time-series data $X \in \mathbb{R}^{N \times T \times d}$, we let $\mathbf{H}_{emb} \in \mathbb{R}^{N \times d_{emb}}$ denote the initial node embeddings. GDN is described by following equations,

$$\mathbf{A} = \text{ReLU}\left(\text{sgn}\left(\text{topk}\left(\mathbf{H}_{emb}\mathbf{H}_{emb}^{\mathsf{T}}\right)\right)\right) \tag{2.36}$$

$$\mathbf{h}_i^t = \text{ReLU}\left(\sum_{j \in \Gamma(i) \cup \{i\}} \alpha_{ij} f_{FC}\left(\mathbf{x}_j^t\right)\right) \tag{2.37}$$

$$\mathbf{X}^t = f_{FCs}\left(\|_i \, \mathbf{h}_{emb,i} \odot \mathbf{h}_i^t\right) \tag{2.38}$$

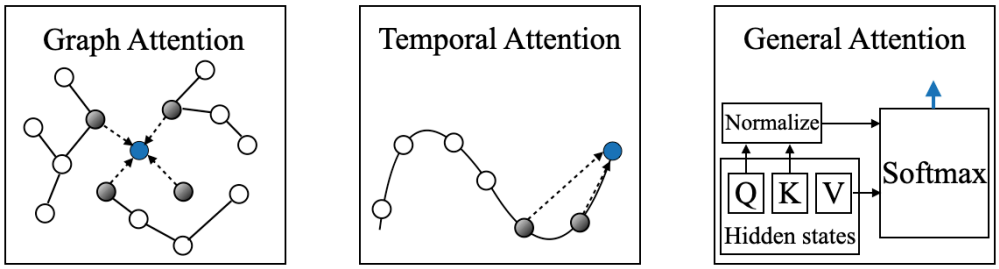


Figure 2.5: A diagram of different attention mechanisms: graph attention, temporal attention, and general attention.

where a binary graph is constructed from the node embeddings and is sparsified by selecting top-k neighbors, as shown in Eq. 2.36. sgn denotes the sign function and is defined in Sec. C.5. The hidden state $\mathbf{h}_i^t$ of node i at time step t is calculated by an attention-based aggregation of node embeddings in the neighborhood, as shown in Eq. 2.37, where pairwise attention scores α_{ij} are calculated with Eq. 2.13 by taking $[\mathbf{h}_{emb,i} \parallel f_{FC}(\mathbf{x}_i^t)]$ as input vectors. Finally, an elementwise multiplication is calculated between the embeddings $\mathbf{H}_{emb}$ and the hidden states $\mathbf{H}$, after which the result is concatenated and fed through a fully connected network, to derive the binary prediction, as shown in Eq. 2.38.

The node embeddings in GDN have threefold functions: (a) they are used to construct a binary connectivity graph, and (b) they are used to compute pairwise attention scores which essentially represent semantic proximity between nodes, and (c) they are used in final hidden states for forecasting. Without a recurrent component, GDN has the advantage of being more lightweight than other RNN-based graph models. **FuSAGNet** [80] further enhances anomaly detection performance compared to GDN by uniquely modeling the temporal process for each node embedding, thereby enhancing the model's robustness.

GReLeN [224] utilizes Variational AutoEncoder (VAE) to model internode dependency for anomaly detection. GReLeN encodes temporal features with general attention and uses VAE to generate a probabilistic graph based on Gumbel-softmax categorical reparametrization.

Model Comparison on Graph Components: Graph Attention. *Graph attention mechanism* is proposed to compute the pairwise attention scores between nodes [183]. It is not necessary for GAT to have a predefined graph structure, since the graph structure is essentially learned through graph attention, GAT becomes a standardized cornerstone module for many later models. For example, MTAD-GAT leverages GAT to construct a feature-oriented graph and a time-oriented graph for time-series anomaly detection [228]. GDN likewise utilizes a GAT to aggregate node features from neighboring nodes for time-series anomaly detection [51]. Some GAT variants were also proposed, for instance, Cola-GNN builds a GAT-like structure based on the known connectivity information for influenza-like illness forecasting [53]. AGNN [178] uses cosine operation instead of concatenation when calculating attention scores.

Temporal Attention and General Attention. In addition to graph attention, temporal attention, and general attention are commonly used in graph time-series modeling. A diagram of these attention mechanisms is depicted in Fig. 2.5.

When using RNN models in sequence-to-sequence forecasting tasks, the hidden state prior to the last time step from the RNN model, i.e., $\mathbf{h}^{t-1}, \mathbf{h}^{t-2}, \ldots$, may contribute to the forecasting performance. Attention mechanism is initially proposed to leverage both the latest hidden state and earlier hidden ones by multiplying state values with attention scores [6]. We call this attention type for sequence modeling as *a temporal attention mechanism*, which takes the form:

$$\mathbf{z}^t = f_{FC}\left(\tanh\left(f_{FC}\left(\mathbf{h}^t\right)\right)\right) \qquad \alpha^t = f_{softmax}\left(\mathbf{z}^t\right) \qquad \mathbf{h}'^t = \sum_t \alpha^t \mathbf{z}^t \tag{2.39}$$

where the new hidden state $\mathbf{h}'^t$ incorporates a sequence of hidden states instead of only the last one. As an example, LSTM-RGCN [119] adopts temporal attention for stock selection.

Since most graph time-series models consist of a spatial component and a temporal component, researchers naturally take advantage of graph attention for graph modeling and temporal attention for time-series modeling. For instance, ASTGCN [75] proposes a spatial-temporal block that combines the two attention mechanisms. GMAN [233] also renders them and uses a gated fusion to incorporate their resulting states. STHAN-SR [163] uses a temporal Hawkes attention and hypergraph attention for stock selection.

With the success of the temporal attention mechanism, *general attention*, also called *self-attention*, was proposed to learn effective representations. It describes attention as a function that maps a query and a set of key-value pairs to output [182]. In this line of work, there are three variables derived from the attention embedding $\mathbf{H}$ to represent query, key, and value states, and an aggregation method is applied to them. The general attention takes the following form:

$$\mathbf{Q} = f_{FC,Q}\left(\mathbf{H}\right) \qquad \mathbf{K} = f_{FC,K}\left(\mathbf{H}\right) \qquad \mathbf{V} = f_{FC,V}\left(\mathbf{H}\right) \qquad \text{Attn} = f_{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.40}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are of dimension $\mathbb{R}^{N\times d_q}$, $\mathbb{R}^{N\times d_k}$, and $\mathbb{R}^{N\times d_v}$, respectively. Radflow adapts the general attention mechanism with a different aggregation on the query, key, and value states, as described in Eq. 19 [180]. STGNN leverages both graph attention and general attention for traffic forecasting [190]. ST-GRAT [149] takes one step further and includes all three attention types. Attention mechanisms are also used in multimodal learning. For example, Ekambaram et al. [57] fuse attention scores from images, text, and historical sales for product sale forecasting. In addition, there are many other types of attention mechanisms using graphs, such as self-attention [116] and gated attention [220], among others [115].

2.6 Representational Components

Apart from the perspective of time-series modeling (temporal components) and graph modeling (spatial components), we notice that some computational methods are widely used in graph time-series models. We call these methods representational components, which practically transform intermediate hidden states in the neural networks. We discuss and analyze representational components, including gated mechanisms and skip connections. Furthermore, we discuss model interpretability in the context of graph time-series models.

2.6.1 Gated Mechanisms

Gated neurons serve as activation or selection variables to help nonlinear dependency modeling. From the perspective of data flow, gates can be seen as data transformation [117] or data fusion [218, 233]. For the purpose of transformation, given a hidden state $\mathbf{h}$, a gate variable $\mathbf{g}$ whose element sits in the range of $[0, 1]$, can be derived from itself and then multiplied by it elementwise to derive a new hidden state $\mathbf{h}'$ for use:

$$\mathbf{g} = g(\mathbf{h}) \qquad \mathbf{h}' = \mathbf{g} \odot \mathbf{h} \tag{2.41}$$

where the function g can be arbitrarily selected from neural structures such as fully connected layers or convolutional layers [117, 217]. While for the purpose of fusion, assume that we are trying to fuse two hidden states $\mathbf{h}_1, \mathbf{h}_2$, one straightforward fusion method is to derive a gate variable $\mathbf{g}$ from one hidden state (e.g., $\mathbf{h}_1$) and use it to derive a new hidden state $\mathbf{h}'$ by balancing the weight between the two hidden states:

$$\mathbf{g} = g(\mathbf{h}_1) \qquad \mathbf{h}' = \mathbf{g} \odot \mathbf{h}_1 + (\mathbf{1} - \mathbf{g}) \odot \mathbf{h}_2 \tag{2.42}$$

where the fusion gates can be extended to encapsulate more than two hidden states as input [220]. Both transformation and fusion techniques are widely used in gated models, among which LSTM (Eq. 2.4-Eq. 2.7) and GRU (Eq. 2.9-Eq. 2.10) are two examples. Since LSTM and GRU are gated models, any model based on them also leverages the gated mechanism, and we call this gated mechanism a *temporal gated mechanism* since the gates control the temporal dependency [123, 229].

Many graph time-series models also use a *graph gated mechanism* where gate variables are derived to select edges to activate. For example, STAG-GCN [133] incorporates adaptive graphs by fusion gates. Similarly, Cola-GNN [53] fuses a connectivity graph with an attention graph. FC-GAGA [146] leverages gate mechanisms for both time and graph modeling. Graph gated mechanisms are further adapted with attention mechanisms [123, 220].

2.6.2 Skip Connections

Skip connections have been increasingly used in deep neural networks [92, 196, 212]. A skip connection is a shortcut that connects the input of a shallow layer to that of a deep layer and results in a residual module. This connection has proved effective in preserving useful features and usually leads to better performance [83]. Skip connections are also widely used in graph time-series models. For instance, ST-GRAT [149] and STFGNN [117] use skip connections to sum the transformed states after each spatio-temporal layer. ASTGCN [75] and Radflow [180] stack multiple spatio-temporal blocks and uses skip connections to connect features before and after each block. Graph WaveNet [202] and MTGNN [204] utilize skip connections for both aforementioned purposes. StemGNN [23] investigates the effectiveness of skip connections and finds that they help improve prediction accuracy. ESG [212] also connects its learned graphs from each module through skip connections.

2.6.3 Model Interpretability

Investigation of model interpretability is important to understand the model result, especially for applications such as clinical analysis [122, 145]. For this purpose, different approaches are proposed, including mask methods [40], saliency-based methods [96], and discrete representation learning on time series [66]. The prediction may also be explained by the learned substructure of graph [216]. In some models, node embeddings and attention scores between nodes are learned by neural networks and interpretation is possible through visualization of learned embeddings [51, 75, 116, 133, 149]. Moreover, explainability is plausible through ablation study [57, 180, 220]. For example, the contribution of a temporal component can be evaluated by dropping the temporal component and then measuring how much the performance is decreased. In GaAN [220], the effectiveness of attention and gate mechanisms are investigated by comparing experiments with and without corresponding components.

2.7 Applications and Datasets

Applications of graph time-series models cover a broad spectrum, including urban traffic planning [31, 33, 117, 133, 146, 167, 171, 190, 204, 220], crime forecasting [186, 205], motion detection [148], crowd flow prediction [174], bike demand prediction [128], weather prediction [194], website view forecasting [180], and epidemiological modeling [53], among many other real-world applications [9, 24, 26, 29, 34, 65, 88, 100, 101, 119, 139, 159, 195, 228]. In this section, we first compare selected regression models in terms of their forecasting performance on two widely used datasets, METR-LA and PEMS-BAY [121]. Furthermore, we conduct a comparison of selected anomaly detection models using two water treatment datasets, SWAT and WADI [51]. We then discuss how related models handle unique data irregularities that are embedded in time series or graph structures.

Table 2.5: A comparison of graph time-series models on traffic speed forecasting. The forecasting performance of 3, 6, and 12 steps ahead of various graph time-series models regarding metrics MAE, RMSE, and MAPE. The best result is highlighted in **bold**, and the second best result is highlighted in *italics*.

Data	Models	3 step ahead			6 steps ahead			12 steps ahead		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	STGCN [217]	2.88	5.74	7.6%	3.47	7.24	9.6%	4.59	9.40	12.7%
	DCRNN [121]	2.77	5.38	7.3%	3.15	6.45	8.8%	3.60	7.59	10.5%
	FC-GAGA [146]	2.75	5.34	7.3%	3.10	6.30	8.6%	3.51	7.31	10.1%
	GaAN [220]	2.71	5.25	7.0%	3.12	6.36	8.6%	3.64	7.65	10.6%
	G-WaveNet [202]	2.69	5.15	6.9%	3.07	6.22	8.4%	3.53	7.37	10.0%
	MTGNN [204]	2.69	5.18	6.9%	3.05	6.17	8.2%	3.49	7.23	9.9%
	ST-GRAT [149]	2.60	5.07	6.6%	3.01	6.21	8.2%	3.49	7.42	10.0%
	StemGNN [23]	*2.56*	5.06	*6.5%*	3.01	6.03	8.2%	*3.43*	7.23	*9.6%*
	STGNN [190]	2.62	*4.99*	6.6%	*2.98*	*5.88*	*7.8%*	3.49	*6.94*	9.7%
	DGSL (best) [167]	**2.39**	**4.41**	**6.0%**	**2.65**	**5.06**	**7.0%**	**2.99**	**5.85**	**8.3%**
PEMS-BAY	DCRNN [121]	1.38	2.95	2.9%	1.74	3.97	3.9%	2.07	4.74	4.9%
	STGCN [217]	1.36	2.96	2.9%	1.81	4.27	4.2%	2.49	5.69	5.8%
	FC-GAGA [146]	1.36	2.86	2.9%	1.68	3.80	3.8%	1.97	4.52	4.7%
	MTGNN [204]	1.32	2.79	2.8%	1.65	3.74	3.7%	1.94	4.49	4.5%
	G-WaveNet [202]	1.30	2.74	*2.7%*	1.63	3.70	3.7%	1.95	4.52	4.6%
	ST-GRAT [149]	*1.29*	2.71	*2.7%*	*1.61*	3.69	*3.6%*	1.95	4.54	4.6%
	DGSL [167]	1.32	*2.62*	2.8%	1.64	*3.41*	*3.6%*	*1.91*	**3.97**	*4.4%*
	STGNN (best) [190]	**1.17**	**2.43**	**2.3%**	**1.46**	**3.27**	**3.1%**	**1.83**	*4.20*	**4.2%**

2.7.1 Regression Model Performance on Traffic Forecasting

Transportation optimization is an ongoing pursuit for traffic policymakers and researchers. One of the key points to the success of a more efficient traffic scheme is to precisely know in advance the traffic situations on roads, including but not limited to the volume size of the traffic flow, moving speeds, and possibly scheduled activities. The forecasting is then used to make downstream decisions such as recommending faster routes or helping urban construction planners understand traffic bottlenecks and hence ameliorate them. Sequential values (e.g., traffic volumes, speeds) over time are collected in multiple locations to provide time-series data, meanwhile, the geographical distribution of locations provides distance information for graph construction.

METR-LA and *PEMS-BAY* [121] are two widely used traffic datasets for forecasting performance evaluation. Their data statistics are listed in appendix D. Table 2.5 shows forecasting performance for a horizon of 3, 6, and 12 steps. Various graph time-series models are reported regarding metrics MAE, RMSE, and MAPE, in an approximate order from the worst model to the best model. The formulas of these metrics are provided in appendix C.4. From the table, we observe that DGSL has the best performance on all forecasting horizons for the METR-LA dataset, and STGNN has the best performance for the PEMS-BAY dataset. Note that among models with the best performance, MTGNN [204], STGNN [190], and DGSL [167] use the provided graph structure to select edges, instead of using geographic distance values in their model. This shows that exploiting temporal graphs and semantic

Table 2.6: A comparison of graph time-series models on anomaly detection in water treatment. The performance is reported regarding precision, recall, and F1 metrics. The best result is highlighted in **bold**, and the second best result is highlighted in *italics*.

Data	Models	Precision	Recall	F1
SWAT	MTAD-GAT [228]	21.0	64.5	31.7
	GDN [51]	**99.4**	68.1	80.8
	FuSAGNet [80]	*98.8*	*72.6*	*83.7*
	GReLeN [224]	95.6	**83.5**	**89.1**
WADI	MTAD-GAT [228]	11.7	30.6	17.0
	GDN [51]	**97.5**	40.2	57.0
	FuSAGNet [80]	*83.0*	*47.9*	*60.7*
	GReLeN [224]	77.3	**61.3**	**68.2**

graphs contributes to performance improvement. Moreover, we also observe that the model performance slightly differs between datasets, suggesting researchers to select and test models based on their datasets.

2.7.2 Anomaly Detection Performance on Water Treatment

The ability to detect anomalies in water treatment is critical for ensuring water quality control. Numerous graph time-series models have been proposed for accurate anomaly detection and have been evaluated on two publicly available datasets, namely *SWAT* and *WADI* [51]. Both datasets consist of time-series data from water treatment sensors, and anomalies are labeled in the testing set. Model performance with respects to precision, recall, and F1 is reported in Table 2.6. It is evident that different models prioritize different metrics. For example, GDN and FuSAGNet achieve the highest and second highest precision across both datasets, while GReLeN achieves the best recall performance. This difference might be attributed to the graph relational learning of GReLeN in predicting negative samples as positive samples, thus leading to an increase in false positive predictions. We suggest readers test related models on their own datasets to evaluate model performance, since the performance of SWAT alone may not be convincing [199].

2.7.3 Data Challenges

Data characteristics impose several challenges for modeling. For example, irregular time series can be sampled at uneven interval time gaps and with anomaly spikes or dips [164, 169, 225]. Other examples include data that have sparse categorical features or dense numerical features [103]. Furthermore, the irregularity can also lie in the graph structure.

2.7.3.1 Irregular Time Series

Time series built on web view count or video view count have values that cover a large span of values from zero to multiple millions, in contrast to traffic speed values that are typically within a small range. In order to forecast time series of different scales, scaling is used in data preprocessing and postprocessing stages [180]. Min-Max scaling and Z-score scaling are the two most common scaling methods. The formulas for Min-Max scaling and Z-score scaling are provided in Sec. C.1.

Heterogeneous types of time series impose another modeling challenge. For example, different types of crimes, such as theft or robbery have different time-series patterns and evolving interdependency. Attention mechanisms or gate mechanisms can be used to build type-sensitive models and capture the interrelations between types. Xia et al. [205] proposed a type-aware embedding layer to learn the mutual influence between types of crimes.

2.7.3.2 Irregular Graph Structures

In applications such as ride-hailing forecasting and crime forecasting [205], location data are collected based on geographic coordinates instead of preselected locations. In this case, grid mapping strategies such as partition and aggregation may be applied to construct a graph from these data. The gird-structure data can be modeled with CNN to capture the spatial correlations [221], or further aggregated for graph models. Moreover, graphs in the real world can be dynamic, where interactions between nodes change over time. There are several strategies to create a new graph or update an existing graph to reflect the dynamics. For example, STAG-GCN [133] utilizes an adaptive graph that has dynamic edge weights.

2.7.4 Other Applications and Datasets

Related models are also used to conduct household electricity usage profiling, which classifies households as different types and allows power companies to optimize pricing policies. Some forecasting models can be easily extended to serve as anomaly detection models by measuring the deviations of the difference between predicted values and actual values. For example, traffic network forecasting models can be converted to detect anomaly traffic flows which may be consequences of road accidents. The same task can also be approached in different ways. For instance, Li et al. [119] target overnight stock price movement prediction and formulate it as a binary classification problem, i.e., whether a stock rises or falls overnight. Another spatio-temporal attention model [163], by contrast, formulates the stock selection task as a time-series prediction problem.

2.8 Future Directions

Despite the rapid progress of graph time-series models, there remain challenges and open issues. This section discusses some future directions of graph time-series models.

Scalability. The existence of large-scale data necessitates the scalability of graph time-series models. When dealing with a big graph, strategies include sampling and pooling can be used to preserve essential structure knowledge and eliminate less useful ones. Similar actions are also applicable for long sequential data where models should select impactful time values instead of digesting all values to boost computational efficiency.

Adoption of Distinctive Considerations. Future work also needs to adapt to distinctive situations for different datasets or objectives. Some considerations include (a) modeling the impact of periodicity and seasonality in time series, and (b) modeling the impact of long-term and short-term time-series patterns, and (c) modeling the impact of local and global neighbor nodes in graphs, and (d) modeling outlier data such as time-series bursts and isolated nodes.

Adoption of Advanced Components. The development of time-series models and deep graph models contribute to the advance of graph time-series models. For example, Transformer [182] has been increasingly used in series modeling, and is integrated into graph time-series models [190, 207]. Similarly, recent advanced GNNs may be considered [203, 235].

Broader Applications to More Tasks. As shown in this survey as well as other surveys [188], the majority of graph time-series models target forecasting problems. However, the power of graph time-series models is not limited to this specific task. The research area of graph time-series has rich potential for future work on classification, recommendation, representation learning, and many other objectives.

2.9 Conclusion

In this survey, we comprehensively summarize recent work on the unified topic of graph time-series modeling. We categorize graph time-series models into, graph recurrent/convolutional neural networks (GRCNN) and graph attention neural networks (GANN), and further categorize them with respect to various components. We thoroughly present representative models in each category. We discuss and analyze these models and their components from the perspectives of temporal, spatial, and representational modeling. This includes an explanation of temporal modeling using RNN versus CNN, various graph construction methods, goals and limitations of graph models, and the widely adopted attention and gated mechanisms. We also discuss real-world applications and datasets. Moreover, we present performance results on forecasting and anomaly detection tasks of selected graph time-series models. In the end, we extend our discussion to include data challenges and future directions in related research areas.

Chapter 3

Context Integrated Graph Neural Network for Resource Forecasting

3.1 Introduction

This chapter introduces a novel graph time-series model that leverages contextual graphs for resource forecasting. Resource (demand and supply) forecasting in spatio-temporal data is widely studied in various fields, including transportation [112, 143], construction [43, 48] and communication [189]. Resource time-series values are typically recorded at various locations and exhibit dependency across, and based on, locations. This requires the modeling of relational and spatial dependencies. Moreover, contextual or environmental factors (e.g., weather) also impact resource values and should be considered. For instance, ride-hailing demand is sensitive to precipitation.

Related work that leverages contextual features includes TensorCast [47], which forecasts author collaboration by incorporating features into tensors. Another work utilizes economic features to forecast housing prices [69]. The time-evolving and dynamic nature of context requires a modeling of dynamic contextual dependency, however, existing approaches assume static contexts [71, 158] or require feature selection [211].

Hence, this chapter introduces a novel model, Context Integrated Graph Neural Network (CIGNN), which learns and incorporates temporal, relational, spatial, and dynamic contextual dependencies for resource forecasting. CIGNN considers a resource network of values recorded at locations, and models it as a graph, where nodes represent locations and the corresponding resource time series. CIGNN also constructs separate graphs as contextual graphs to represent each context type, where nodes represent context-recording locations and their associated context time series. Assuming that various contexts have dynamic impact on resources, our proposed CIGNN model employs a novel fusion mechanism that jointly learns from multiple contextual time series. To the best of our knowledge, CIGNN is the first graph time-series model that integrates multiple sources of dynamic contextual information for resource forecasting.

Table 3.1: A qualitative comparison of CIGNN with other models. CIGNN is the only model that incorporates temporal, relational, spatial, and contextual dependencies.

Modeled Dependencies	ARIMA	VAR	LSTM	STGCN	DCRNN	G WaveNet	**CIGNN**
TEMPORAL	✓	✓	✓	✓	✓	✓	✓
SPATIAL				✓	✓	✓	✓
RELATIONAL				✓	✓	✓	✓
DYNAMIC CONTEXTUAL							✓

Our main contributions are summarized below:

- **Joint Modeling of Various Dependencies**: CIGNN performs resource forecasting by leveraging temporal, relational, spatial, and dynamic contextual dependencies, in contrast to existing methods that do not capture dynamic contextual information. Table 3.1 shows a qualitative comparison of CIGNN against state-of-the-art methods.
- **Unified Multisource Context Learning**: In contrast to existing work that manually conducts feature engineering [211], treats context as static features [71], or ignores contexts [31, 121], CIGNN integrates multiple contextual features in a unified way.
- **Effectiveness**: CIGNN consistently outperforms previous methods in terms of MAE and RMSE on two real-world datasets. CIGNN has an average improvement of 5.7% (MAE) and 9.4% (RMSE) on *CallMi*, and 4.4% (MAE) and 2.3% (RMSE) on *BikeBay*.

3.2 Context Integrated Graph Neural Network

Our proposed Context Integrated Graph Neural Network (CIGNN) leverages temporal, relational, spatial, and contextual dependencies for resource forecasting. CIGNN represents a resource network as a graph, where nodes represent locations and their associated resource time series. CIGNN also represents each context type as a separate graph, where nodes represent locations and their associated contextual time series. We introduce CIGNN using a bike-sharing system as a running example. Given historical bike supply records at various stations, CIGNN predicts future supply by considering four types of dependencies, as illustrated in Fig. 3.1.

Definition 3.1 (Temporal dependency)**.** Given time series $\mathbf{x} = [x_1 x_2 \cdots x_T]$, temporal dependency is defined as a mapping from values prior to x_t (e.g., previous w values) to x_t,

$$[x_{t-w} \cdots x_{t-2} x_{t-1}] \rightarrow x_t \tag{3.1}$$

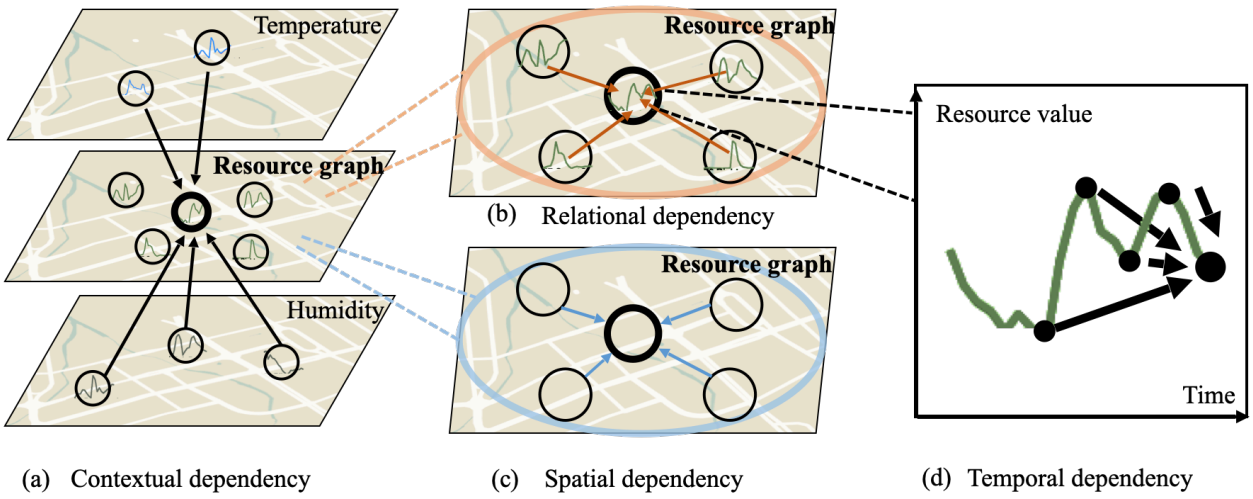


Figure 3.1: A simplified illustration of spatial, temporal, relational, and contextual dependencies for one node. (a) Contextual dependency: contextual (temperature and humidity) impact on resource graph. (b) Relational dependency: correlation between nodes within a graph. (c) Spatial dependency: spatial accessibility between nodes within a graph. (d) Temporal dependency: historical impact on future values within a time series.

Modeling temporal dependency allows models to learn from earlier time-series patterns. Next, we define relational and spatial dependencies to model mutual influence between supply at various locations.

Definition 3.2 (Relational dependency)**.** We let $G = (V, E)$ denote a graph, where each node represents a location with an associated resource (demand or supply) time series. We then define edges to represent relational dependency:

$$E = \{(i, j) \mid \forall (i, j) \in |V| \times |V| \wedge \psi(\mathbf{x}_i, \mathbf{x}_j) \geq \lambda_r\} \tag{3.2}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ denote the associated time series of node i and j, respectively. ψ denotes a correlation metric and λ_r denotes a threshold value to filter out small correlation values.

We build pairwise edges with $\psi(\mathbf{x}_i, \mathbf{x}_j)$ to represent the relational dependency, where edges are removed if their weights are smaller than a predefined threshold λ_r. The insight behind relational dependency is to capture the mutual impact between similar time series.

On the other hand, the spatial dependency is constructed on the first law of geography [179], i.e., *Near things are more related than distant things.*

Definition 3.3 (Spatial dependency)**.** Given a graph $G = (V, E, \mathbf{A})$, where its adjacency matrix $\mathbf{A}$ embodies distance information between nodes, an edge set representing spatial dependency is defined as:

$$E = \{(i, j) \mid \forall (i, j) \in |V| \times |V| \wedge \mathbf{A}_{ij} \geq \lambda_s\} \tag{3.3}$$

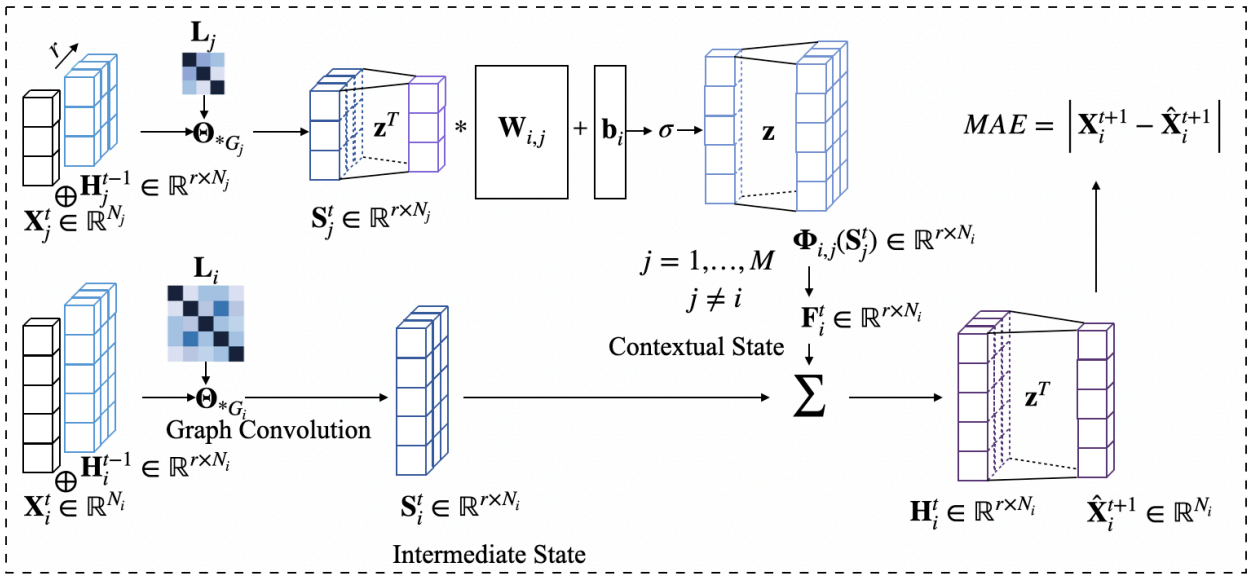


Figure 3.2: An overview of our CIGNN unit. For the purpose of brevity, we depict two graphs in the figure and the number of node features are assumed as one, $d_i = d_j = 1$. We let G_i denote a resource graph and G_j a contextual graph. The graph convolution is denoted by $\mathbf{\Theta}$ and is used to capture the relational and spatial dependencies. The intermediate state $\mathbf{S}$ incorporates the temporal dependency. The contextual state $\mathbf{F}$ is aggregated for contextual dependency modeling. In practice, CIGNN learns $\mathbf{\Phi}_{i,j}$ for $\{(i,j) \mid i,j \in 1,2,\ldots,S \wedge i \neq j\}$.

Each element $\mathbf{A}_{ij}$ is derived from distance information such that a short distance results in a large weight value. Edges with a large weight indicate strong spatial dependency when its weight is greater than a threshold value λ_s.

Definition 3.4 (Contextual dependency)**.** Let $G_s = (V_s, E_s)$ denote a resource graph with a set of nodes V_s representing the locations of interests, connected by edges in E_s. In addition, let $G_c = (V_c, E_c)$ be a contextual graph with a set of nodes V_c representing the locations where contextual features are recorded (e.g., weather stations recording humidity), and connected by edges E_c. Note that E_s and E_c are edges derived either from Definition 3.2 or Definition 3.3. We denote E_{sc} as edges connecting nodes in V_s and nodes in V_c.

$$E_{sc} = \{(i,j) \mid \forall\,(i,j) \in |V_s| \times |V_c|\} \tag{3.4}$$

To account for more than one contextual type (e.g., if there are n contextual types), the definition is extended to include all contextual types: $V_c = V_{c1} \cup V_{c2} \cup \ldots \cup V_{cn}$, $E_c = E_{c1} \cup E_{c2} \cup \ldots \cup E_{cn}$, for contextual types $c1, \ldots, cn$.

In a bike-sharing system, the bike supply is sensitive to weather conditions such as temperature and precipitation, which are recorded in weather stations. In this case, we build one contextual graph for temperature and another contextual graph for precipitation.

3.2.1 Problem Formulation

We formulate resource forecasting as a time-series prediction task, which aims to learn a function f that predicts future resource values. We define a series of graphs, denoted by $\mathcal{G} = \{G_1, G_2, \ldots, G_S\}$, where the first graph G_1 represents a resource graph, and the rest represents contextual graphs. For instance, in order to forecast the bike supply, we derive a bike supply graph, a contextual graph for temperature information and a contextual graph for humidity information. The ith graph is defined as $G_i = (V_i, E_i, \mathbf{A}_i)$, where V_i denotes the node set and E_i denotes the edge set in G_i. Nodes represent locations with associated time series and edges imply the dependency between nodes. $\mathbf{A}_i$ is the adjacency matrix that encodes node connections, capturing spatial or relational dependencies.

For convenience, we refer to the group of time series in a graph as a *graph signal* [67], which contains all time series from the same graph (type). Hence for graph G_i, the graph signal is denoted as $X_i \in \mathbb{R}^{T \times N_i \times d_i}$, where T denotes the time span of interest, $N_i = |V_i|$ is the number of nodes in the ith graph, d_i is its corresponding node feature dimension. We use a subscript to denote which graph is referred, and a superscript to denote a time index. For example, $\mathbf{X}_i^t \in \mathbb{R}^{N_i \times d_i}$ denotes the graph signal of G_i at t.

Given graphs and graph signals, a multistep ahead time-series prediction task for a horizon of τ is formulated as:

$$\left[\mathbf{X}_i^{1:T}, G_i\right] \xrightarrow{f(\cdot)} \hat{\mathbf{X}}_i^{T+1:T+\tau}, \quad i \in [1, \ldots, S]$$

Our model architecture is depicted in Fig. 3.2. We introduce our components for the modeling of relational, spatial, temporal and contextual dependency in the following sections.

3.2.2 Relational Dependency Modeling using Graph Convolution

The relational dependency implies connections between locations. We leverage the graph convolution to model it. Given a graph $G(V, E, \mathbf{A})$ (where $\mathbf{A} \in \mathbb{R}^{N \times N}$ complies with the Definition 3.2) and its graph signal at a moment $\mathbf{Y} \in \mathbb{R}^{N \times d}$ with $N = |V|$ and d the feature dimension, we define a graph convolution layer $\mathbf{\Theta}$ with respect to G as:

$$\mathbf{\Theta} *_G \mathbf{Y} = \mathbf{\Theta}(\mathbf{L})\,\mathbf{Y} = \mathbf{\Theta}\left(\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\mathsf{T}\right)\mathbf{Y} = \mathbf{Q}\mathbf{\Theta}(\mathbf{\Lambda})\,\mathbf{Q}^\mathsf{T}\mathbf{Y} \tag{3.5}$$

where $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\,\mathbf{D}^{-\frac{1}{2}}$ is the normalized Laplacian matrix of adjacency matrix $\mathbf{A}$, with $\mathbf{D}$ its diagonal degree matrix. $\mathbf{L} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\mathsf{T}$ is the Eigen-decomposition of $\mathbf{L}$, where $\mathbf{Q}$ is composed of eigenvectors. $\mathbf{\Lambda}$ is a diagonal matrix where each element is an eigenvalue of the corresponding eigenvector.

Strengthen Locality with Chebyshev Approximation. Chebyshev Approximation is applied to strengthen locality and boost computational efficiency. The intuition behind

leveraging graph convolution is that it is efficient on aggregating information from neighbor nodes. Furthermore, we enhance local connections by extending the graph kernel $\mathbf{\Theta}(\mathbf{\Lambda})$ to a series of bases as:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \tag{3.6}$$

where $\boldsymbol{\theta} = [\theta_1 \theta_2 \dots \theta_k]^\mathsf{T} \in \mathbb{R}^K$ is a trainable coefficient vector and K is a hyperparameter to regulate the locality radius of graph convolution. We set a small number for K to reinforce local impacts. Hence, Eq. 3.6 is modified with the truncated Chebyshev polynomial expansion [49, 177]:

$$\mathbf{\Theta}_{*G}\mathbf{Y} = \mathbf{\Theta}(\mathbf{L})\,\mathbf{Y} \approx \sum_{k=0}^{K-1} \theta_k \,\mathrm{T}_k\left(\tilde{\mathbf{L}}\right)\mathbf{Y} \tag{3.7}$$

$$\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I} = \mathbf{L} - \mathbf{I}, \quad \text{assuming } \lambda_{max} = 2 \tag{3.8}$$

where $\mathrm{T}_k\left(\tilde{\mathbf{L}}\right)$ is the kth Chebyshev polynomial at the scaled Laplacian $\tilde{\mathbf{L}}$, and $\mathbf{I} \in \mathbb{R}^{N\times N}$ is an identity matrix. λ_{max} is the dominant eigenvalue of $\mathbf{L}$, which is assumed as 2 since parameters can adapt to the change in scale during training [106]. The Chebyshev polynomial has a low time complexity as $\mathcal{O}(K|\mathcal{E}|)$, where $\mathcal{E}$ is the number of nonzero edges. Recall that we formulate a resource graph G_i and multiple contextual graphs (S graphs in total), the graph convolution is applied on each graph with separate parameters,

$$\mathbf{\Theta}_{*G_i}(\mathbf{X}_i) = \sum_{k=0}^{K-1} \theta_{k,i} \,\mathrm{T}_k\left(\tilde{\mathbf{L}}\right)\mathbf{X}_i, \quad i = 1, 2, \dots, S \tag{3.9}$$

3.2.3 Spatial Dependency Modeling using Graph Convolution

Alternatively, the mutual impact between locations can be modeled with spatial dependency, as defined in Definition 3.3. In this case, the adjacency matrix in the graph describes the geographic proximity between nodes. The graph convolution is then used upon these graphs.

3.2.4 Temporal Dependency Modeling using Gated Recurrent Units

Inspired by the recent success of RNN structure in time-series prediction and its power of capturing temporal dependency, we leverage a modified Gated Recurrent Units (GRU) [38] variant in our approach. The temporal component in CIGNN is composed of units that take current observations at time t from each graph $\mathbf{X}_i^t \in \mathbb{R}^{N_i \times d_i}$ and previous hidden states $\mathbf{H}_i^{t-1} \in \mathbb{R}^{r\times(N_i * d_i)}$ as input ($i \in [1, \dots, S]$), and yield new hidden states $\mathbf{H}_i^t \in \mathbb{R}^{r\times(N_i * d_i)}$ as

output, where r is the number of neurons. Our GRU-like structure is formulated as follows:

$$\mathbf{r}_i^t = \sigma\left(\underset{r}{\mathrm{FC}}\left(\mathbf{\Theta}_{r*G_i}\left[\mathbf{X}_i^t \oplus \mathbf{H}_i^{t-1}\right]\right)\right) \qquad \mathbf{u}_i^t = \sigma\left(\underset{u}{\mathrm{FC}}\left(\mathbf{\Theta}_{u*G_i}\left[\mathbf{X}_i^t \oplus \mathbf{H}_i^{t-1}\right]\right)\right) \tag{3.10}$$

$$\mathbf{C}_i^t = \tanh\left(\underset{C}{\mathrm{FC}}\left(\mathbf{\Theta}_{C*G_i}\left[\mathbf{X}_i^t \oplus \left(\mathbf{r}_i^t \odot \mathbf{H}_i^{t-1}\right)\right]\right)\right) \qquad \mathbf{S}_i^t = \mathbf{u}_i^t \odot \mathbf{H}_i^{t-1} + \left(1 - \mathbf{u}_i^t\right) \odot \mathbf{C}_i^t \tag{3.11}$$

where $\mathbf{r}$ and $\mathbf{u}$ denote a reset gate and an update gate, respectively. σ and tanh denote the sigmoid and hyperbolic tangent activation functions, respectively. $\mathbf{\Theta}$ denotes the graph convolution layer as introduced in Sec. 3.2.2, which captures either relational dependency or spatial dependency, based on the selected adjacency matrix. Hence, the intermediate hidden state $\mathbf{S}_i$ incorporates temporal, relational or spatial dependency in the ith graph.

3.2.5 Contextual Dependency Modeling with Fusion Layers

With the learned intermediate states from the previous section, we further propose a novel fusion layer, denoted by $\mathbf{\Phi}$ to capture the contextual dependency across graphs, which is defined in Definition 3.4:

$$\mathbf{\Phi}_{i,j}\left(\mathbf{S}_j^t\right) = \sigma\left(\mathbf{z}\left[\mathbf{W}_{i,j}^{\mathsf{T}}\mathbf{z}^{\mathsf{T}}\mathbf{S}_j^t + \mathbf{b}_{i,j}\right]\right) \tag{3.12}$$

$$\mathbf{F}_i^t = \sum_{j=1,\, j\neq i}^{S} \mathbf{\Phi}_{i,j}(\mathbf{S}_j^t) \tag{3.13}$$

where the subscript of $\mathbf{W}$ and $\mathbf{b}$ denotes different sets of parameters. The trainable parameters $\mathbf{W}_{i,j} \in \mathbb{R}^{(N_j*d_j)\times(d_i*N_i)}, \mathbf{b}_{i,j} \in \mathbb{R}^{N_i*d_i}$ are regarded as learning relations between time-series across graphs, i.e., the contextual dependency. Note that $\mathbf{W}$ has a similar form as E_{sc} defined in Eq. 3.4 when we consider a resource graph (e.g., a bike supply network) and its contextual graphs (e.g., a temperature graph and a humidity graph). The parameter $\mathbf{z} \in \mathbb{R}^r$ is a mapping trainable vector that serve to aggregate the dimension.

Finally, our final hidden states $\mathbf{H}_i^t$ combine the intermediate hidden state $\mathbf{S}_i^t$ and the fused hidden state $\mathbf{F}_i^t$:

$$\mathbf{H}_i^t = \mathbf{S}_i^t + \mathbf{F}_i^t \tag{3.14}$$

In contrast to existing work, our fusion layer does not require manual feature engineering.

3.2.6 Prediction and Objective

With the hidden state $\mathbf{H}_i^t$, the forecasting is calculated below:

$$\hat{\mathbf{X}}_i^{t+1} = \mathbf{z}^{\mathsf{T}}\mathbf{H}_i^t \tag{3.15}$$

Table 3.2: A table of dataset descriptions: a graph is constructed for a demand or a supply, and for each contextual type.

Dataset	**CallMi**	**BikeBay**
Data meaning	Mobile call records	Bike supply amount
Data location	Milan city	The Bay Area
Number of nodes	162	70
Time span	Nov. 2013-Dec. 2013	Aug. 2013- Aug. 2015
Time interval	1 hour	2 hour
Contextual types (number of nodes)	temperature(5) humidity (4)	temperature (3), humidity(3),dew(3) sea level(3) and wind speed(3)

Forecasts of further values are recursively calculated. Mean Absolute Error (MAE) is used to minimize training loss across all graphs and all time horizons.

$$\text{Loss} = \frac{1}{|\Omega|}\frac{1}{S}\frac{1}{\tau}\sum_{t\in\Omega}\sum_{i=1}^{S}\sum_{j=1}^{\tau}\left|\hat{\mathbf{X}}_i^{t+j} - \mathbf{X}_i^{t+j}\right| \tag{3.16}$$

where Ω denotes the set of timestamp index in the training set.

3.3 Experiments

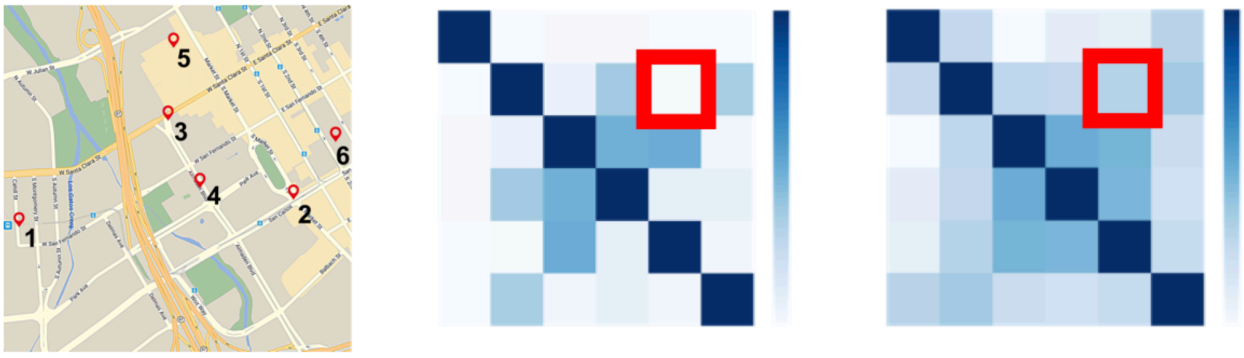
In this section, we introduce our experimental setup, including graph construction methods and hyperparameters. To verify the effectiveness of CIGNN, we compare our model against previous state-of-the-art methods on two real-world datasets.

3.3.1 Graph Construction

We leverage two different methods to construct a graph. Specifically, we are interested in deriving an adjacent matrix $\mathbf{A}$ for CIGNN. One method is based on geographical information described in Definition 3.3 to model the spatial dependency, the other is based on the implicit correlation between time series, as described in Definition 3.2.

Distance-based Gaussian Kernel Matrix. A truncated Gaussian kernel can be used to derive the adjacent matrix, where an element $\mathbf{A}_{ij}$ has a higher value if the distance between node i and node j is smaller [121]:

$$\mathbf{A}_{ij} = \begin{cases} exp(-\frac{\text{dist}(i,j)^2}{\sigma^2}), & \text{if dist}\,(i,j) \leq \kappa \\ 0, & \text{otherwise} \end{cases} \tag{3.17}$$



(a) (b) (c)

Figure 3.3: A comparison between spatial matrix and relational matrix. Deeper color indicates higher correlations. (a) Selected bike dock locations in San Jose. (b) The Gaussian kernel-based spatial matrix. (c) The correlation-based relational matrix. The value between station 2 and 5 (marked by the red squares) is higher.

where $\text{dist}(i, j)$ denotes the geographical distance between node i and node j. σ denotes the standard deviation of distances and κ is a threshold value to control the matrix sparsity. Note that $\lambda_s = exp\left(-\frac{\kappa^2}{\sigma^2}\right)$ as in Definition 3.3.

Relational Matrix based on Correlation Coefficients. We observe in empirical study that Gaussian kernel matrix can fail to capture the hidden relational correlations between distant nodes. For example, in Fig. 3.3a, station 2 and station 5 are far from each other. Distance-based Gaussian Kernel matrix (Fig. 3.3b) suggests that they have low correlations. However, they are in fact highly correlated (Fig. 3.3c) as travellers frequently commute between them along the straight road. To reflect this fact, we utilize the Detrended Cross-Correlation Analysis coefficient (DCCA coefficient) [107, 208] to construct the relational correlation matrix.

DCCA coefficient is a correlation metric on series data, which combines detrended cross-correlation analysis (DCCA) and detrended fluctuation analysis (DFA). Given two time series $\mathbf{x}$, $\mathbf{y}$ of length T and sliding windows of length l, DCCA coefficient is calculated by

$$\rho_{DCCA}(\mathbf{x}, \mathbf{y}, l) = \frac{F^2_{DCCA}(\mathbf{x}, \mathbf{y}, l)}{F_{DFA}(\mathbf{x}, l) F_{DFA}(\mathbf{y}, l)} \tag{3.18}$$

where the numerator and the denominator are the average covariance and variance of the $T - l + 1$ windows (partial sums), respectively:

$$F^2_{DCCA}(\mathbf{x}, \mathbf{y}, l) = \frac{\sum_{s=1}^{T-l+1} f^2_{DCCA}(\mathbf{x}, \mathbf{y}, s)}{T - l + 1} \qquad F^2_{DFA}(\mathbf{x}, l) = \frac{\sum_{s=1}^{T-l+1} f^2_{DFA}(\mathbf{x}, s)}{T - l + 1} \tag{3.19}$$

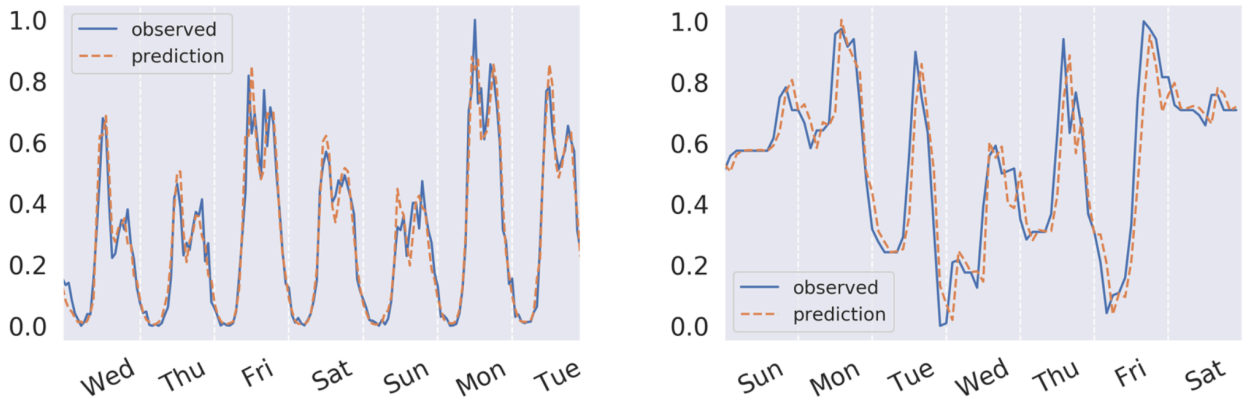


Figure 3.4: A comparison of time-series patterns from the two datasets in use. (a) *CallMi*: Call demand in Milan during Dec. 25-31. (b) *BikeBay*: Bike supply in San Francisco during Aug. 25-31. Values are normalized. Note that *CallMi* shows more periodicity.

The partial sums are calculated with sliding windows across $\mathbf{x}$ and $\mathbf{y}$:

$$f^2_{DCCA}(\mathbf{x}, \mathbf{y}, s) = \frac{\sum_{t=s}^{s+l-1} (\mathbf{x}^t - \bar{\mathbf{x}}_s)(\mathbf{y}^t - \bar{\mathbf{y}}_s)}{l} \qquad f^2_{DFA}(\mathbf{x}, s) = \frac{\sum_{t=s}^{s+l-1} (\mathbf{x}^t - \bar{\mathbf{x}}_s)^2}{l} \tag{3.20}$$

where $\bar{\mathbf{x}}_s = \frac{1}{l}(x_s + x_{s+1} + \ldots + x_{s+l-1})$ is the average value in the window starting at s. The matrix is constructed with pairwise correlations:

$$\mathbf{A}_{ij} = \begin{cases} \rho_{DCCA}(\mathbf{x}, \mathbf{y}, l), & \text{if } \rho_{DCCA}(\mathbf{x}, \mathbf{y}, l) \geq \lambda_r \\ 0, & \text{otherwise} \end{cases} \tag{3.21}$$

3.3.2 Experimental Setup

We conduct experiments on two public real-world datasets, *CallMi* and *BikeBay* (see Table 3.2 for a summary) to show the performance of CIGNN. Both datasets contain dynamic contextual features. *CallMi* [11] contains call demand data in Milan from Nov. 2013 to Dec. 2013. The dataset contains temperature and humidity information as contextual features. The city is partitioned into grids in the raw data, however, some grids have very few records. Hence, we cluster grids into 162 nodes that record mobile call demand. There are 5 temperature nodes and 4 humidity nodes. Each contextual node corresponds to a weather station. The time interval is an hour. *BikeBay* [4] contains bike supply data in 70 dock stations in the Bay Area. The dataset was recorded from Aug. 2013 to Aug. 2015. It contains weather conditions as contextual data. There are 3 nodes for temperature, humidity, dew, sea level, and wind speed, respectively. The time interval is two hours.

Table 3.3: Multistep ahead forecasting performance in terms of MAE and RMSE on *CallMi* and *BikeBay*. The best performance is highlighted in **bold** text.

Data	τ	Metrics	HA	ARIMA	VAR	LSTM	STGCN	DCRNN	G WaveNet	**CIGNN**
CallMi	1	MAE	17.15	14.42	18.54	13.51	11.35	10.41	9.48	**8.89**
		RMSE	38.80	24.26	31.30	25.04	20.46	19.44	18.18	**16.82**
	2	MAE	–	26.78	27.01	17.05	20.48	16.59	12.72	**11.72**
		RMSE	–	44.27	47.14	30.10	35.63	33.59	26.23	**22.84**
	3	MAE	–	38.12	34.49	19.02	33.14	22.60	15.38	**14.86**
		RMSE	–	61.43	60.13	35.04	40.01	55.03	32.09	**29.59**
Bikebay	1	MAE	22.70	7.62	8.04	19.91	6.80	6.55	7.00	**6.37**
		RMSE	28.92	13.35	18.72	25.34	11.98	12.37	13.33	**11.75**
	2	MAE	–	11.70	12.06	20.83	10.76	10.13	11.01	**9.68**
		RMSE	–	18.62	29.42	26.67	16.98	17.65	19.56	**16.60**
	3	MAE	–	14.12	14.34	21.29	13.41	12.56	13.69	**11.90**
		RMSE	–	21.19	35.05	27.73	19.71	20.52	22.25	**19.18**

Diversity of Datasets regarding Periodicity. The two selected datasets demonstrate different pattern characteristics. The call demand time series exhibit periodicity as shown in Fig. 3.4a, while the bike supply time series show more irregularities, as shown in Fig. 3.4b. We choose these two datasets to demonstrate the capability of our CIGNN in accurately predicting time series, whether they exhibit periodicity or not.

We split the time-series data chronologically into training, validation and testing sets with a ratio of $0.7 : 0.1 : 0.2$. Values are normalized using Z-score normalization before training. Predicted values are inversely transformed to calculate the error. The number of lags is set as $w = 6$ and horizons is set $\tau = 3$. The number of neurons is set as $r = 32$. We conducted experiments with both the Gaussian kernel matrix and the DCCA coefficient relational matrix. When using the Gaussian kernel matrix, the distance threshold κ in Eq. 3.17 is set as 0.1, while the window length in Eq. 3.18 is set $l = 4$ when using the relational matrix. A previous study [160] suggests that a small step of graph convolution is more effective in strengthening the local impact, hence the graph convolution step in Eq. 3.9 is set as $K = 1$. We adopt following hyperparameters during training: The learning rate is set as 0.1. The learning rate decay ratio is set as 0.1 for every 10 epochs. We train for a maximum of 100 epochs with the Adam optimizer [105] and use an early stop strategy if the validation loss does not decrease for 10 consecutive epochs. All experiments are implemented using Python Tensorflow (v1.14) and run on Ubuntu 16.04 OS with 8 CPU cores and a memory of 32G. MAE in Eq. 32 is used as a loss measure to update parameters for all horizons in the training set. In evaluation, both MAE and RMSE are calculated, as defined in Eq. 33.

We compare CIGNN with both statistical models and DL-based models, including *HA* His-

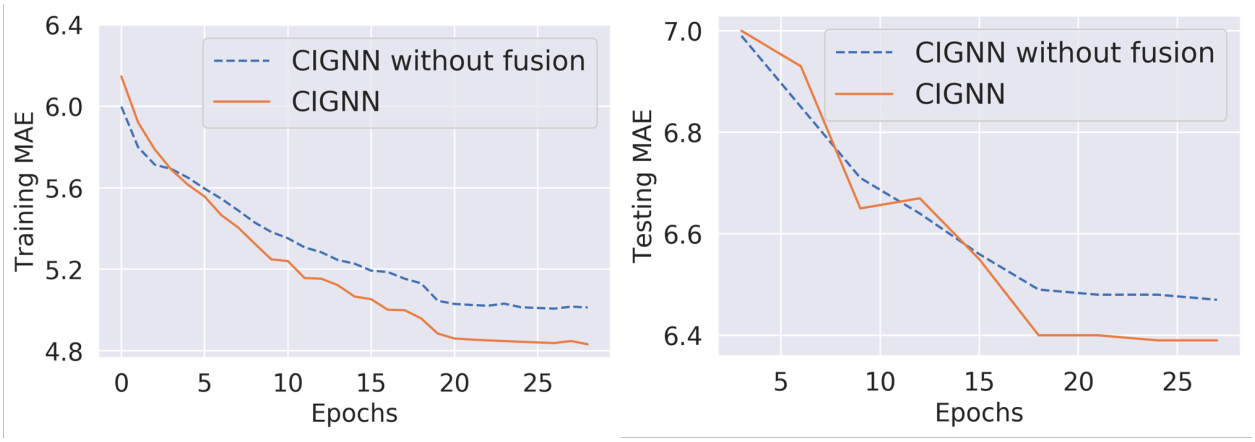


Figure 3.5: An illustration of the effectiveness of the fusion mechanism. CIGNN with fusion outperforms the model without fusion (dashed), in both training and testing.

torical Average gives prediction for a given timestamp as the average of values at the same time of the day over the past four weeks; *ARIMA* AutoRegressive Integrated Moving Average fits each time series and predicts each time series independently; We use the same parameter order $(3, 0, 1)$ as used in [121]. *VAR* Vector AutoRegressive is a multivariate model that generalizes ARIMA to have multiple evolving variables; *MM-LSTM* Multistep Multivariate LSTM. We set the number of neurons as 64. *DCRNN* [121] Diffusion Convolution Recurrent Neural Network is a graph time-series model that utilizes a graph diffusion to capture the spatial dependency. Its details are covered in Sec. 2.5.1.1; *STGCN* [217] Spatio-Temporal Graph Convolutional Network is a graph time-series model that learns both the spatial and temporal dependencies with a gated mechanism to make predictions; *Graph WaveNet* (or *G WaveNet*) [202] Graph WaveNet is a graph times-series model that leverages a self-adaptive adjacency matrix to capture the spatial dependency.

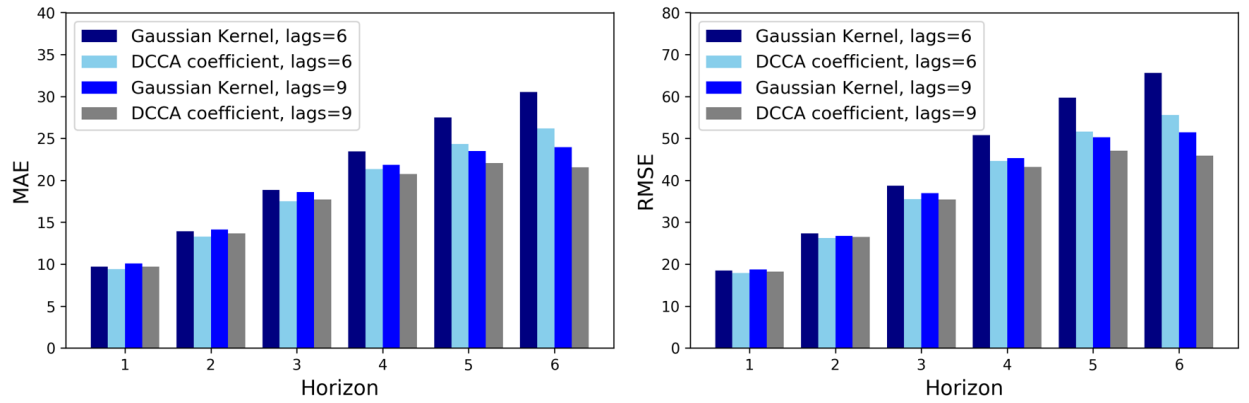


Figure 3.6: A performance comparison of models with different matrix constructions and different lags. DCCA coefficient-based model achieves better performance for both lags.

3.3.3 Results

Multistep Ahead Prediction. We conduct experiments with $w = 6$ to predict values at a horizon up to $\tau = 3$. We report MAE and RMSE results for each horizon in Table 3.3. From the table we note that graph time-series methods generally outperform statistical methods (HA, ARIMA and VAR) as HA can only predict one step ahead, and ARIMA fails to exploit interactions across time series. Among graph time-series models, Graph WaveNet outperforms DCRNN on CallMi but not on BikeBay, while CIGNN consistently outperforms STGCN, DCRNN and Graph WaveNet. CIGNN also outperforms baselines when more observations and a larger horizon are considered. CIGNN outperforms previous state-of-the-art models on both dataset. Specifically, CIGNN is better than Graph WaveNet on CallMi by 5.7% MAE and 9.4% RMSE, and better than DCRNN on BikeBay by 4.4% MAE and 2.3% RMSE.

Effectiveness of Fusion. To assess the effectiveness of our proposed fusion mechanism, we compared our approach to a CIGNN variant that has the fusion layer removed. We ran experiments on the BikeBay dataset and compared the two models on both training and testing losses, as shown in Fig. 3.5. Although CIGNN starts with a higher loss, it converges faster than the variant without fusion layer. This shows that the fusion mechanism is effective in utilizing contextual factors for forecasting.

Adjacency Matrix Analysis. To analyze and compare the effectiveness of the Gaussian Kernel matrix and the DCCA relational coefficients matrix, we ran experiments on the CallMi dataset using lags as 6 and 9. Fig. 3.6 shows MAE and RMSE results for a horizon of 6. We note that predictions based on the DCCA coefficient adjacency matrix are consistently more accurate than predictions based on the Gaussian kernel matrix. Moreover, using 9 lags results in significantly better prediction results than that using 6 lags, and the lag number has an impact on prediction accuracy for a long horizon.

3.4 Conclusion

To model nonlinear temporal, relational, spatial, and contextual dependency in time-series prediction, we propose a novel graph time-series approach for data with dynamic contextual information. Our model employs a novel fusion mechanism to capture the dynamic contextual impact on resource values. The capability of processing multiple graphs at the same time empowers our model to be applied in more general structured sequence forecasting tasks, such as dynamic social networks relationship prediction, and evolving preference prediction in recommendation systems. Extensive experiments and analyses show that our proposed CIGNN consistently outperforms previous state-of-the-art methods, which validates the effectiveness of the model.

Chapter 4

Evolving Super Graph Neural Network for Large-scale Forecasting

4.1 Introduction

In this chapter, we propose another graph time-series model that also involves a series of graphs. Different from CIGNN in Chapter 3, this model generates new graphs as time series dynamically evolve forward. Our proposed model, Evolving Super Graph Neural Network (ESGNN), aims to address time inefficiency, unscalability, and space limitations of existing graph time-series models for forecasting on large-scale data.

Our model motivation is listed below (1) As discussed in Sec. 2.5.1.3, many graph time-series models rely on external information to construct a spatial graph. However, the availability of geo-information is not guaranteed, neither does it necessarily reflect the actual node connections. In a cloud computing system, for example, two closely located compute nodes can belong to different cloud clusters, resulting in a weak or negligible correlation between them. On the other hand, connections between highly correlated nodes may be missing in the datasets but should be taken into consideration. In fact, CIGNN in Chapter 3 has better performance with a DCCA matrix than a distance matrix. In this model, we assume that compute nodes in the same computing cluster share correlated patterns. We consequently derive the cluster information with a similarity-based clustering algorithm. (2) Most existing graph time-series models rely on a single graph throughout the entire time span, which neglects the dynamically evolving node connections. In reality, nodes can not only neighbor with different nodes at different moments but can also have changing link weights. ESGNN creates a new evolving graph for every user-selected time interval, which reflects the most up-to-date connection status and provides a more refined dependency modeling. (3) To reduce time and space complexity, ESGNN builds super nodes based on the original (or target) node time series, and builds super graphs upon super nodes. Many existing models require $\mathcal{O}\left(|V|^2\right)$ time complexity to build graphs [34, 121, 167]. When the number of nodes grows from a few hundred to tens of thousands, the time cost of graph construction alone get worsened by ten thousand times, and even more when considering training runtime with the generated large graph. In contrast, ESGNN trains upon much fewer super nodes, hence its training procedure requires much less computation and memory. As a consequence, ESGNN predicts in a much shorter time, while achieving comparable forecasting accuracy.

Our main contributions in this chapter are summarized below,

- **Efficient Super Graph Construction**: ESGNN employs a novel super graph construction method that leverages K-Means and Locality-Sensitive Hashing (LSH) to quickly build evolving super graphs from the most recent graph connections. To the best of our knowledge, this is the first graph time-series model that approaches large-scale time-series forecasting with a super graph.

- **Improved Time and Space Complexity**: Our proposed ESGNN has a significantly lower time and space complexity compared to previous state-of-the-art baselines. Experiments on two real-world datasets show that ESGNN has a speedup of 3.2×-40.5× over baselines. In addition, ESGNN only needs $0.02\% - 20.96\%$ space usage of compared baselines.

- **Effectiveness**: Compared to previous state-of-the-art baselines, ESGNN achieves comparable forecasting performance, with the best or runner-up results across multiple steps ahead predictions on two cloud computing datasets.

4.2 Evolving Super Graph Neural Network

In this section, we describe the workflow of our proposed ESGNN. We associate each node with a time series of interest, which we refer to as the target time series, as shown in Fig. 4.1 (a). In cloud resource usage forecasting, each time series represents a resource type such as CPU or memory. We assume nodes with high pattern correlations have mutual influence with each other. Hence, nodes with similar time-series patterns are grouped into clusters, as shown in Fig. 4.1 (b). We build a super node for each cluster and create an average time series by averaging over all time series in the cluster. Therefore, each super node is associated with an average time series, or equivalently, a super time series. ESGNN then constructs a super graph by connecting super nodes with super edges. Each super edge connects a pair of super nodes. We assign super edge weights based on the similarity of its connected super time series. Over time, nodes may belong to different super nodes during various time periods. As a consequence, super graphs need to be dynamically updated with the latest period of time-series observations, as shown in Fig. 4.1 (d). We train a model with an evolving super graph encoder-decoder architecture that takes dynamic super graphs as input, as shown in Fig. 4.1 (c). The goal of this step is to learn the super node embeddings. Finally, a predictor structure is trained on the learned super node embeddings together with the target node time series, as shown in Fig. 4.1 (e).

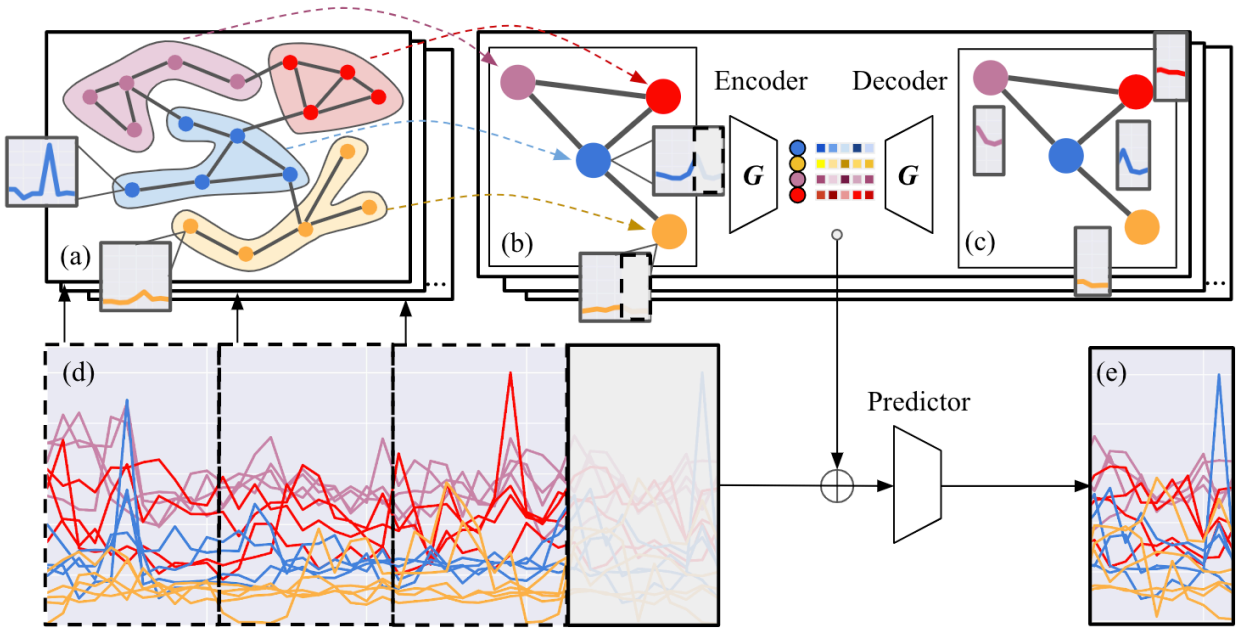


Figure 4.1: The architecture of our proposed ESGNN. (a) Large-scale graphs before clustering: each graph consists of a period of node time series. (b) Super graphs: each super node is created by clustering node time series. (c) An encoder-decoder structure is trained with super graphs to learn super node embeddings. (d) Time series are sliced into periods to construct evolving super graphs per time interval. The testing data are masked off during training. (e) A predictor leverages both target time series and encoded super node embeddings to make prediction.

4.2.1 Problem Formulation

ESGNN periodically creates evolving super graphs upon the time-series data $\mathbf{X} \in \mathbb{R}^{N \times T}$ of N time-series and T steps. Let p denote a user-selected graph update interval. Hence, time values in the first p time steps $\mathbf{X}^{1:p}$ are used to construct the first super graph G_1, and time-series values in the second p time steps $\mathbf{X}^{p+1:2p}$ are used for the second super graph G_2, and so on. The last super graph may contain fewer than p time step values if T does not divide p evenly. Without loss of generality, we assume T is divisible by p. Therefore, there are in total $S = \frac{T}{p}$ super graphs, denoted by $\mathcal{G} = \{G_1, G_2, \ldots, G_S\}$. Each super graph G_i is associated with a super node set and a super edge set, denoted as $G_i\,(V_i, E_i)$. The number of nodes in super graph G_i is denoted as $N_i = |V_i|$.

Given observed values and super graphs, we aim to predict values for a horizon of τ:

$$\left[\mathbf{X}^{1:T}, \mathcal{G}\right] \xrightarrow{f(\cdot)} \hat{\mathbf{X}}^{T+1:T+\tau} \tag{4.1}$$

4.2.2 Super Graph Construction

ESGNN constructs super graphs in two stages, namely, super node aggregation and super edge connection. Firstly, super nodes are created by clustering periods of time series, where each cluster contains time series of similar patterns. For each super node, a super time series is derived by averaging all time series in the corresponding cluster. In the second stage, we connect super nodes with super edges. Our model creates a new super graph every p time steps. For brevity, we discuss the super graph construction for the ith period $\mathbf{X}^{(i-1)p+1:ip} \in \mathbb{R}^{N \times p}$, with $i \in \{1, 2, \ldots, S\}$ covering the whole observation time span. We temporarily override the notation $\mathbf{X}$ with $\mathbf{X}^{(i-1)p+1:ip}$ for this discussion.

Super Node Aggregation. ESGNN groups N nodes into N_i super nodes through a clustering method, denoted by ϕ. *K-Means* is one eligible algorithm for ϕ, in addition to alternative options such as X-Means [150]. K-Means is applied on N nodes with their individual node time series of p time values. We briefly describe K-Means below in three steps: initialization, assignment, and update. In the initialization step, $K = N_i$ nodes are randomly selected as centroids, denoted by $[M_1^0, M_2^0, \ldots, M_K^0]$, at the iteration 0. In the assignment step, each node is assigned to the closest centroid. We let A_u^t denote the assigned cluster for node u in the tth iteration, and let $[C_1^t, C_2^t, \ldots, C_K^t]$ denote the resulting K clusters, where $C_k^t = \{u \mid A_u^t = k\}$. Cluster centroids are recalculated for each cluster in the update step: $M_k^t = \frac{1}{|C_k^t|} \sum_{u \in C_k^t} \mathbf{x}_u$. The node assignment step and the centroid update step are repeated until convergence. As a result, K clusters $C_1, C_2, \ldots, C_K$ are obtained. ESGNN creates a super node for each cluster. Let $\mathbf{y}_k$ denote the kth super time series: $\mathbf{y}_k = \frac{1}{|C_k|} \sum_{u \in C_k} \mathbf{x}_u$. We let $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_i}]^\mathsf{T}$ denote all N_i derived super time series. Here, $\mathbf{Y} \in \mathbb{R}^{N_i \times p}$.

Super Edge Connection. We utilize Locality Sensitivity Hashing (LSH) to quickly build edges among highly correlated super nodes. LSH assigns super time series to various buckets with linear time complexity. We connect super nodes within the same bucket and use a time-series similarity metric ψ to determine edge weights. LSH generates a signature for each super time series through a random matrix, denoted by $\mathbf{B} \in [-1, 1]^{p \times b}$, which maps super time series to b-dimensional vectors, with b being a small preselected number. Therefore, the matrix of b-dimensional vectors for all super time series $\mathbf{X}^b \in \mathbb{R}^{N_i \times b}$ is calculated by $\mathbf{X}^b = \mathbf{YB}$. A binary *signature matrix* $\mathbf{X}^{sign} \in \mathbb{R}^{N_i \times b}$ of N_i time series and b bits is derived from applying the sign function on $\mathbf{X}^b$:

$$\mathbf{X}_{i,j}^{sign} = \begin{cases} 1, & \text{if } \mathbf{X}_{i,j}^b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

The resulting signatures are b-bit strings. Hence, there are at most $Q = 2^b$ unique strings ranging from a b-digit string of all zeros ($\underbrace{00\cdots0}_{b}$) to a b-digit string of all ones ($\underbrace{11\cdots1}_{b}$). Super time series with equivalent signatures are classified into the same bucket. We let $B_1, B_2, \ldots, B_Q$ denote the Q different buckets, where two super nodes u and v are in the

same bucket if and only if $\mathbf{x}_u^{sign} = \mathbf{x}_v^{sign}$. We connect super nodes within the same bucket to form the edge set:

$$E = \{e_{u,v} \mid u, v \in B_i \wedge u \neq v, i = 1, 2, \ldots, Q\}, \quad e_{u,v} = \psi(\mathbf{y}_u, \mathbf{y}_v) \tag{4.3}$$

where ψ denotes a time-series similarity metric that takes two time series as input and yields a similarity score. The selection of ψ is flexible and can naturally leverage any application-specific or state-of-the-art techniques. For instance, our approach can adopt Pearson Correlation Coefficient or Dynamic Time Warping [133]. Correspondingly, we derive an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N_i \times N_i}$ for each super graph with $\mathbf{A}_{u,v} = e_{u,v}$. Leveraging LSH for super edge construction has two advantages. Firstly, LSH is efficient with a time complexity that is linear to the number of super nodes. Secondly, LSH helps to further reduce the number of super edges compared to constructing a complete graph with pairwise connections, which reduces training runtime.

4.2.3 Diffusion on Evolving Super Graphs

ESGNN learns from dynamically evolving super graphs through graph diffusion. Given super graphs $\mathcal{G}$, we leverage the DCRNN model to incorporate time-series values from evolving graphs into a hidden state, as $\mathbf{H}^t = \textsc{dcrnn}\left(\mathbf{Y}^t, G_i\right)$. Details of DCRNN is given in Eq. 2.17-Eq. 2.20. G_i is the $i = \left\lceil \frac{t}{p} \right\rceil$th graph of timestamp t and period length p. Thus, different graphs are used in different periods during training.

We leverage an encoder-decoder architecture where the encoder consists of evolving super graph GRUs and derives encoded states, with the final state used to initialize the decoder. The decoder, also composed of evolving super graph GRUs, generates prediction for the super time series. Both the encoder and the decoder use the DCRNN model.

$$\hat{\mathbf{Y}}^{t+1} = \text{Decoder}\left(\text{Encoder}\left(\mathbf{Y}^t\right)\right) \tag{4.4}$$

We use MAE, defined in Eq. 32, as the loss function to train the neural network by minimizing the difference between the ground truth and the prediction.

$$\underset{\text{super}}{\text{Loss}} = \text{MAE}\left(\mathbf{Y}^t, \hat{\mathbf{Y}}^t\right) \tag{4.5}$$

where t iterates over the time index in the training set.

4.2.4 Predictor

After training on super graphs, ESGNN uses a predictor to decode the learned super time-series embeddings. The predictor allows each target time series to be modeled individually

with their corresponding super time-series embeddings. To ensure a lightweight predictor, we utilize a single-layer network that takes the concatenation of target time series and its super time series as input. Let $\mathbf{Y}_{emb} = \text{Encoder}(\mathbf{Y})$ denote the encoded embeddings for all N_i super nodes. At any moment, a target time series is associated with a super node, as described in Section 4.2.2. The super time-series embeddings from the latest super graph are used in the predictor since they are closest to the prediction period. Let A_u^{last} denote the last assigned cluster of node u, a concatenation is formed by combining the two time series $\mathbf{x}_{cat,u} = \left[\mathbf{x}_u \,\|\, \mathbf{y}_{emb,A_u^{last}}\right]$. We further construct a matrix with all N concatenations $\mathbf{X}_{cat} = [\mathbf{x}_{cat,1}, \mathbf{x}_{cat,2}, \ldots, \mathbf{x}_{cat,N}]^\intercal$. Our predictor uses $\mathbf{X}_{cat}$ to generate forecasts:

$$\hat{\mathbf{X}} = \underset{out}{\text{FC}}\left(\mathbf{X}_{cat}\right) \tag{4.6}$$

Similar to Eq. 4.5, the predictor model is trained with MAE.

4.3 Experiments

We design experiments to answer following research questions: (1) What is the forecasting performance of ESGNN compared to existing methods? (2) How much runtime does ESGNN model take compared to other models? (3) How much space is saved by ESGNN?

We evaluate ESGNN on two cloud trace datasets from Google and Adobe [156]. *Google trace* records the activities of a cluster of $12,224$ machines over a period of 2 days in 2011. The measurements of CPU usage, maximum CPU usage, and memory usage are collected at 5-minute intervals, resulting in a total of $36,672$ time series. *Adobe trace* records the activities of a cluster of 795 machines. The CPU usage for each task is recorded every 30 minutes. For both datasets, we obtain time series of length 576 and split them chronologically in a ratio of $0.5 : 0.25 : 0.25$ for training, validation, and testing, respectively. We set the number of super nodes as $K = 100$ in the K-Mean algorithm for both datasets across all time periods. The length of the binary strings in the LSH algorithm is selected as $b = 12$. We set the graph update time interval as $p = 48$ and generate $S = 6$ super graphs in total. The number of units in each graph diffusion layer is set as 16. We use a rolling window of length 24 to create data samples. For each data sample in the window, the first 12 values are conditioned to predict the next 12 values. We compare our model against state-of-the-art large-scale models, including NBEATS [145], DeepAR [162], MQRNN [192] and DF [191]. All models are implemented using Gluonts [1] with the suggested hyperparameter selection from the DF paper [191]. All experiments are run with 16 CPUs of 2.40 GHz, 37.2 GB memory, and a 12 GB NVIDIA GPU.

Table 4.1: A table of prediction performance on Google and Adobe datasets. MAE and RMSE are reported on 3, 6 and 12 steps ahead predictions.

Data	Methods	3 steps ahead		6 steps ahead		12 steps ahead	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
Google	**ESGNN**	*0.0687*	*0.1766*	*0.0781*	*0.1955*	**0.0867**	**0.2019**
	NBEATS	**0.0590**	**0.1294**	**0.0711**	**0.1498**	*0.0889*	*0.2893*
	DeepAR	0.1172	0.3183	0.1413	0.3326	0.1576	0.3763
	MQRNN	0.1684	0.2576	0.1787	0.2602	0.1837	0.3399
	DF	0.3691	0.5059	0.3864	0.5283	0.3947	0.5802
Adobe	**ESGNN**	*0.0140*	*0.0302*	**0.0175**	**0.0364**	**0.0232**	**0.0455**
	NBEATS	**0.0107**	**0.0187**	*0.0221*	*0.0429*	*0.0274*	*0.0530*
	DeepAR	0.0199	0.0628	0.0340	0.0784	0.0424	0.0852
	MQRNN	0.1066	0.1739	0.1099	0.1754	0.1195	0.1814
	DF	0.0972	0.1329	0.1016	0.1474	0.1141	0.1590

4.3.1 Results

We report MAE and RMSE for 3, 6, and 12 steps ahead prediction in Table 4.1. The best performance is highlighted in **bold** and the second best performance is shown in *italic*. For the Google dataset, ESGNN achieves the second best result for 3 and 6 steps ahead predictions and the best result for 12 steps ahead prediction. We observe that DF performs the worst on the Google dataset which may be attributed to the small number of global factors in the DF model, since it implies a strong assumption that all time-series are related to each other. For experiments with the Adobe dataset, our method ESGNN achieves the second best result for the 3 steps ahead prediction and the best forecasting for the 6 and 12 steps ahead horizon. Overall, our proposed ESGNN demonstrates competitive accuracy compared to state-of-the-art models. It is worth noting that although NBEATS outperforms ESGNN at certain horizons, our model requires less space and has a shorter training time, as discussed in the next section.

4.3.2 Runtime and Space Usage Analysis

We report the model space usage in terms of space ratio compared to the baselines, as shown in Table 4.2. We notice that NBEATS takes much more parameters than other methods, due to its multilayer residual block stacking mechanism, where a great number of parameters are needed. In contrast, our proposed ESGNN greatly saves space usage with much fewer parameters. Specifically, ESGNN only needs 0.02% of the space required by NBEATS, and only accounts for 19.96%, 6.99%, and 0.08% of the space usage compared to DeepAR, MQRNN, and DF, respectively.

Table 4.2: The space complexity of each model, measured in the number of parameters.

Methods	**ESGNN**	NBEATS	DeepAR	MQRNN	DF
#parameters	$5,357$	$24,830,628$	$26,844$	$76,602$	$6,483,211$

In addition, we report the training runtime of our ESGNN against baselines for both the Google dataset and the Adobe dataset. The training runtime measures the time from the start to the end of the training for all time series and all training samples. For a fair comparison, we include the time for super graph construction in ESGNN. ESGNN is significantly faster than baselines regarding training runtime, namely, more than ten times faster than NBEATS (15.9×), DeepAR (14.6×), and DF (14.3×) on Google dataset, and more than three times faster than MQRNN (3.2×). A similar speedup is observed with the Adobe dataset, as NBEATS (40.5×), DeepAR (37.2×), MQRNN (8.1×), and DF (36.3×).

4.4 Conclusion

In this chapter, we propose a novel graph time-series model, Evolving Super Graph Neural Network (ESGNN) to address the challenges of time and space complexity in time-series forecasting. With a novel super graph construction method, ESGNN significantly reduces the number of nodes in the super graph, which reduces the runtime and space requirements. ESGNN exhibits substantial advantages in runtime and space usage compared to previous state-of-the-art models, with competitive prediction accuracy.

Chapter 5

Probabilistic Hypergraph Recurrent Neural Network for Forecasting

5.1 Introduction

In this chapter, we go beyond models with standard graph structures and propose a novel hypergraph time-series forecasting model. Despite the huge success of graph time-series models, there still remain challenges to be resolved. This chapter aims to address two of them. The first challenge lies in modeling beyond-pairwise relationships among nodes. Conventional graph time-series models are limited to modeling pairwise relationships [84, 131]. However, beyond-pairwise connections are ubiquitous in many applications. For instance, beyond-pairwise connections exist among compute nodes in cloud computing systems, where multiple compute nodes are routinely allocated together to accomplish tasks [114]. When nodes collaborate on the same task, messages are typically broadcast among them, resulting in simultaneous broadcasting communication instead of one-on-one interactions. Graph models proposed for these applications simplify broadcasting into pairwise connections. In contrast, hypergraph modeling naturally represents the beyond-pairwise relationships among nodes and therefore provides a more accurate description of the actual interactions [28, 197].

The second challenge is related to the deterministic structure in existing models. These models assume that the leveraged graph structure or hypergraph structure is statically fixed, and their connection weights among nodes remain constant. In reality, however, the underlying internode or node-hyperedge relationships may follow a distribution, with dynamic connection weights sampled from the distribution. To address this challenge, probabilistic **graph** models have been proposed, which model connections between nodes as probabilistic distributions, resulting in significant performance improvement across various tasks [140, 167].

Following this thread, this chapter introduces a novel probabilistic **hypergraph** approach when modeling node connections. An example of three time series and their relationships with a hyperedge is presented in Fig. 5.1. For illustration purposes, we define hyperedge time series as the mean time series of its nodes. Notably, the node-hyperedge relationships are dynamic, as nodes exhibit varying correlations with the hyperedge over time (e.g., the blue area versus the red area). Our probabilistic approach aligns with the nature of node interactions and captures these dynamics, compared to existing hypergraph approaches (either unweighted or weighted), as they assume deterministic node-hyperedge weights.

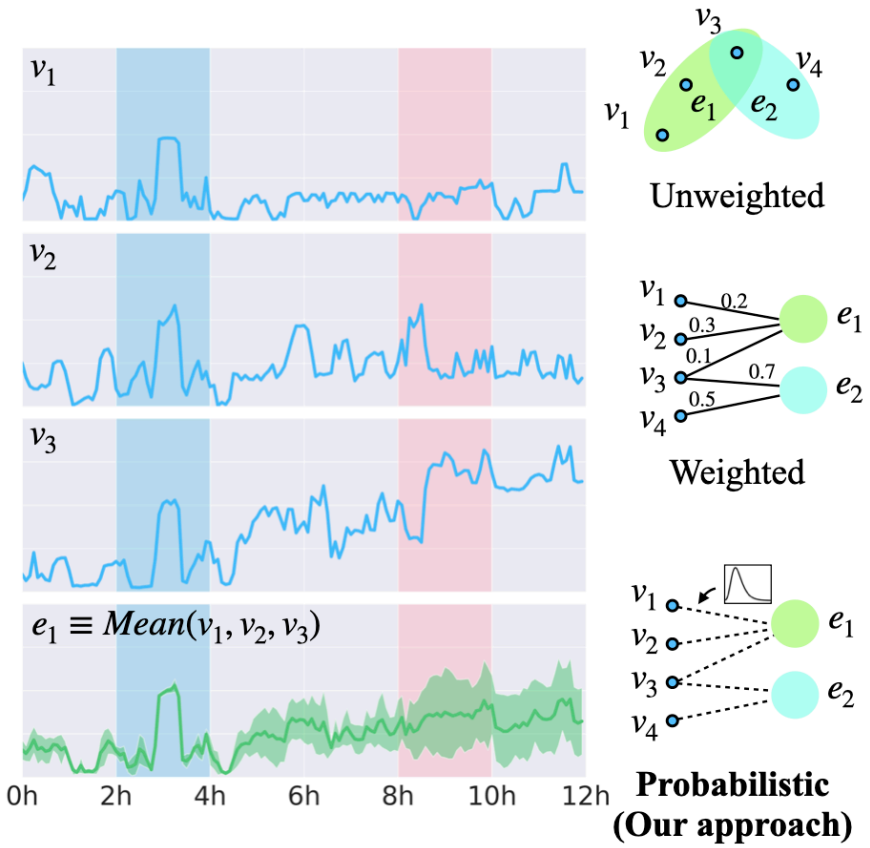


Figure 5.1: Dynamics of node-hyperedge connections. Node time series for (v_1, v_2, v_3) and their hyperedge series e_1 are depicted on the left. The **green** area around the hyperedge time series highlight the standard deviation. Node time series exhibit higher correlations with hyperedge time series in the **blue** area than in the **red** area, indicating fluctuations in the connection strength. Probabilistic relationships modeling can learn the interaction dynamics, in contrast to weighted or unweighted deterministic relationship modeling.

Our proposed model, namely, **Probabilistic Hypergraph Recurrent Neural Network (PHRNN)**, incorporates both probabilistic modeling and hypergraph modeling, and earns the following advantages for time-series forecasting: (1) PHRNN is a hypergraph-based model, enabling it to capture beyond-pairwise interactions among nodes. Specifically, such interactions are modeled in the form of hyperedges that are incident to many nodes. (2) PHRNN does not require an explicit hypergraph structure, but instead, learns a hypergraph structure from the time-series features. This alleviates situations where the underlying structure is unreliable or missing. (3) PHRNN models node-hyperedge interactions as probabilistic distributions, allowing it to exploit the dynamics of node-hyperedge relationships. The probabilistic modeling aligns well with many scenarios across various applications.

Our main contributions in this chapter are summarized below:

- **Probabilistic Hypergraph Learning**: We introduce a novel hypergraph time-series forecasting model, Probabilistic Hypergraph Recurrent Neural Network (PHRNN), which leverages the interrelationships among time series by learning a probabilistic hypergraph. The probabilistic hypergraph approach benefits our model with the advantages of both probabilistic modeling and hypergraph modeling. To the best of our knowledge, PHRNN is the first probabilistic hypergraph model for time-series forecasting. The novel probabilistic hypergraph modeling within our proposed PHRNN serves as an adaptable spatial component and is readily integrated with other models.

- **Effectiveness**: Using multiple datasets, we conduct extensive experiments with closely related baselines, including graph-based methods and hypergraph-based methods. Our experimental results show that PHRNN outperforms previous state-of-the-art baselines in terms of forecasting accuracy. We investigate the effectiveness of PHRNN through a hyperparameter study, including prior knowledge hypergraph constructions and regularization coefficients.

5.2 Related Work

The majority of existing graph models leverage explicit graph structures that are derived from geographical information [121, 217, 229], however, there is a growing trend in graph structure learning, which frees models from relying on connectivity information. Graph structure learning is particularly useful when an explicit graph structure is missing or is challenging to define. Models along this line either learn graph structures from time-series features or from embeddings [167, 202, 204]. This is closely related to the temporal graph construction and the semantic graph construction introduced in Chapter 2. However, most of them learn a deterministic graph in which a static weight is learned for each edge [145, 206]. This assumes that the connection weights between nodes are fixed, which may not accurately reflect real-world scenarios, as connection weights between node time series fluctuate and evolve over time. In order to better reflect these dynamics, some models learn a probabilistic graph structure where each edge represents a distribution, and edge weights are sampled from these distributions [167, 218]. To leverage this advantage, our proposed PHRNN embraces a probabilistic approach by modeling interactions among nodes as distributions.

In addition to graph time-series forecasting models, recent efforts have also been made in hypergraph time-series modeling [213, 214, 215, 231]. A hypergraph consists of nodes and hyperedges, where each hyperedge essentially embodies a set of nodes and describes connections among nodes that extend beyond pairwise relationships. Compared to graph structures, hypergraphs are capable of representing more complex relationships among nodes [64, 197]. Many existing hypergraph models are motivated by web applications as well as traffic networks, as one of their primary goals is to capture inherent connections among groups or communities of nodes to improve forecasting accuracy [134, 187, 214, 232]. For example,

Table 5.1: A notation table of used symbols

Notations	Descriptions	Notations	Descriptions
$H(V, E, \mathbf{C})$	a hypergraph	$E(v)$	edges that v incident to
$E \subseteq 2^V$	(hyper)edge set	$M = \|E\|$	number of edges
$\mathbf{C}$	incident matrix	λ_{reg}	regularization coefficient
s	temperature	g	Gumbel distribution
$\mathbf{z}_v$	node embedding	$\mathbf{y}_e$	hyperedge embedding
$\mathrm{FC}(\cdot)$	a fully connected layer	$\mathrm{Conv}(\cdot)$	a convolutional layer

Wu et al. [198] introduced a hypergraph time-series forecasting model that leverages relational modeling with the aid of predefined hyperedge categories. In contrast, our proposed PHRNN does not assume hyperedge categories and is therefore free from such constraints. Another hypergraph model, introduced by Zhao et al. [232], focuses on capturing the traffic network isomorphism by constructing hyperedges based on actual network connections. Unlike these models that fall short of requiring explicit hypergraph structures, our proposed PHRNN benefits from learning a probabilistic hypergraph structure, which is advantageous when connectivity information is unreliable or missing.

5.3 Problem Formulation

We target a time-series forecasting problem, where our objective is to predict future values for N time-series. Let $\mathbf{X} \in \mathbb{R}^{N \times T}$ denote the time-series matrix of a length of T. With the time-series values, we initially learn a probabilistic hypergraph, denoted by $H(V, E, \mathbf{C})$.

$$\mathbf{X}^{1:T} \rightarrow H(V, E, \mathbf{C}) \tag{5.1}$$

where V and E denote a node set and a hyperedge set, respectively. Each node $v \in V$ is associated with a time series $\mathbf{x}_v \in \mathbb{R}^T$, resulting in $N = |V|$. Furthermore, we let $\mathbf{x}^t \in \mathbb{R}^N$ denote the time-series values for all nodes at time t. Each hyperedge $e \in E$ is incident to a subset of the node set V. We let $M = |E|$ denote the number of hyperedges. The letter $\mathbf{C} \in [0, 1]^{N \times M}$ denotes the probabilistic incidence matrix for the hypergraph H, where each column $\mathbf{C}_{:,e} \in [0, 1]^N$ describes the incident information for hyperedge e, signifying node-hyperedge connections between all nodes and the hyperedge e. We aim to learn a probabilistic distribution for each element $\mathbf{C}_{v,e}$ in the incidence matrix, which subsequently represents the node-hyperedge dynamics.

After learning a probabilistic hypergraph H, our primary objective is to learn a function f that predicts future time-series values $\hat{\mathbf{X}}^{T+1:T+\tau}$ of a horizon τ. The prediction is based on historical time-series observations $\mathbf{X}^{1:T}$ in conjunction with the learned hypergraph H:

$$\left[\mathbf{X}^{1:T}, H\right] \xrightarrow{f(\cdot)} \hat{\mathbf{X}}^{T+1:T+\tau} \tag{5.2}$$

5.4 Probabilistic Hypergraph Recurrent Neural Network

Our proposed Probabilistic Hypergraph Recurrent Neural Network (PHRNN) adopts an encoder-decoder structure, where the encoder recursively digests input sequence values to learn encoded states for nodes, while the decoder utilizes these states to recursively make future predictions. PHRNN contains a probabilistic hypergraph learning component that automatically learns the underlying hypergraph structure from time-series. Once the hypergraph is learned, it is used to update node time-series embeddings through a node-to-hyperedge aggregation and a hyperedge-to-node aggregation. Subsequently, these updated node embeddings are used by a predictor to generate predictions.

5.4.1 Probabilistic Hypergraph Learning

To overcome limitations of existing hypergraph methods that they rely on explicit hypergraphs with fixed incidence matrix values [213, 232], PHRNN learns a probabilistic hypergraph structure that parameterizes the hypergraph representation. We parameterize $\mathbf{C}$ through the parameters $\boldsymbol{\theta} \in [0,1]^{N\times M}$ by using the Gumbel reparameterization trick, which allows efficient differentiability [98, 167]. The distribution of an element $\mathbf{C}_{v,e}$ is described by

$$\mathbf{C}_{v,e} = \sigma\left(\frac{\log\left(\frac{\theta_{v,e}}{1-\theta_{v,e}} + \left(g^1_{v,e} - g^2_{v,e}\right)\right)}{s}\right) \tag{5.3}$$

where $\sigma(\cdot)$ denotes a sigmoid function. $g^1_{v,e}$ and $g^2_{v,e}$ denote two standard Gumbel distributions Gumbel$(0,1)$, and s is a temperature hyperparameter that controls the sparsity of the incidence matrix. During the hypergraph structure optimization, we independently sample $g^1_{v,e}$ and $g^2_{v,e}$ from the standard Gumbel distribution, which consequently instantiates the node-hyperedge weights $\mathbf{C}_{v,e}$ for node v and hyperedge e. For brevity, we let $\mathbf{C}_{v,e}$ denote both the distribution and the instantiation for the incidence matrix, as the context will clarify any ambiguity.

Calculation of Parameter $\boldsymbol{\theta}$. The parameter in Gumbel reparameterization $\boldsymbol{\theta}$ depends on pairs of node-hyperedges, as described in Eq. 5.3. More specifically, the value of $\theta_{v,e}$ depends on node v and hyperedge e. Therefore, we propose calculating θ from node embeddings and hyperedge embeddings. We derive node embeddings from time-series values. Let $\mathbf{z}_v$ denote the node embedding for node $v \in V$. We compute $\mathbf{z}_v$ through a fully connected layer $\mathrm{FC}(\cdot)$ and a convolutional layer $\mathrm{Conv}(\cdot)$ over time series $\mathbf{x}_v$:

$$\text{Node embeddings:} \quad \mathbf{z}_v = \mathrm{FC}\left(\mathrm{Conv}\left(\mathbf{x}_v\right)\right) \in \mathbb{R}^{d_z} \tag{5.4}$$

The use of the convolutional layer over the entire time span of time series allows node embeddings to capture the temporal relations across a wide range of time steps.

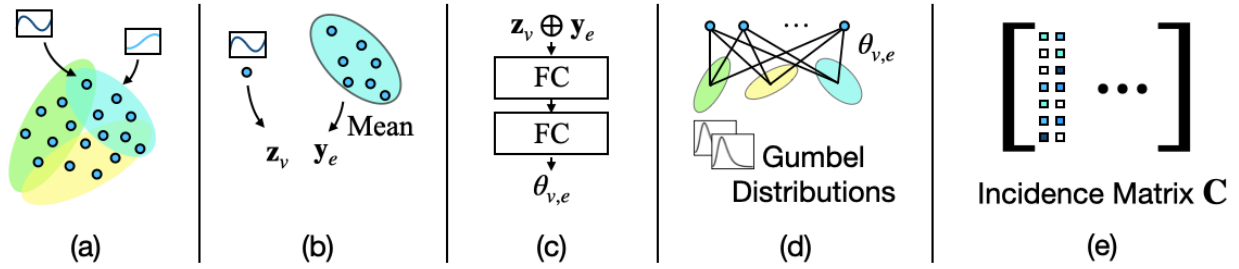


Figure 5.2: Probabilistic hypergraph structure learning. (a) PHRNN utilizes a prior knowledge KNN hypergraph. A hyperedge is derived by taking each node and its closest $K-1$ nodes in terms of time-series similarity. (b) A node embedding $\mathbf{z}_v$ is calculated for each node v using Eq. 5.4. A hyperedge embedding is calculated by averaging all node embeddings the hyperedge is incident to. (c) Using the node embeddings and hyperedge embeddings, the hypergraph parameter $\theta_{v,e}$ is derived through a two-layer neural module. (d) $\theta_{v,e}$ is computed for each node-hyperedge pair. Gumbel distributions are used with $\theta_{v,e}$ to form a probabilistic distribution for each node-hyperedge weight. (e) The learned hypergraph structure is represented in the form of an incidence matrix, where each element represents the sampled node-hyperedge weight.

Hyperedge embeddings, on the other hand, cannot be directly derived from time series due to the missing of hypergraph structure information. To address this, we leverage a K-Nearest-Neighbors (KNN) prior knowledge hypergraph to initialize hyperedge embeddings. The KNN-based hypergraph is constructed by iterating over each node, where at each iteration a hyperedge is formed by including the iterated focal node and its $K-1$ closest neighbors in terms of time-series similarity. Redundant hyperedges are further removed, resulting in $M \leq N$ hyperedges. An example that contains three hyperedges is depicted in Fig. 5.2 (a). Let $H^K\left(V, E^K\right)$ denote the KNN-based prior knowledge hypergraph, we let the hyperedge embedding $\mathbf{y}_e$ be the aggregation of node embeddings for nodes incident to e:

$$\text{Hyperedge embeddings:} \quad \mathbf{y}_e = \frac{1}{|e|} \sum_{v \in e} \mathbf{z}_v, \quad e \in E^K \tag{5.5}$$

As depicted in Fig. 5.2 (b), we employ the mean aggregation, but other aggregation methods such as sum aggregation or attentional aggregation are also possible [79].

Given node embedding $\mathbf{z}_v$ and hyperedge embedding $\mathbf{y}_e$, a node-hyperedge incidence predictor is used to derive $\theta_{v,e}$. We leverage a two-layer module, as depicted in Fig. 5.2 (c):

$$\theta_{v,e} = \mathrm{FC}\left(\mathrm{FC}\left(\mathbf{z}_v \oplus \mathbf{y}_e\right)\right) \tag{5.6}$$

Our incidence predictor has a simple yet effective structure but other readout functions are also viable [202, 204]. Subsequently, the parameter $\theta_{v,e}$ is computed for each pair of node v and hyperedge e, as shown in Fig. 5.2 (d). In conjunction with the Gumbel distributions from Eq. 5.3, the parameter $\boldsymbol{\theta}$ is used to derive the probabilistic hypergraph structure $\mathbf{C}$, as shown in Fig. 5.2 (e).

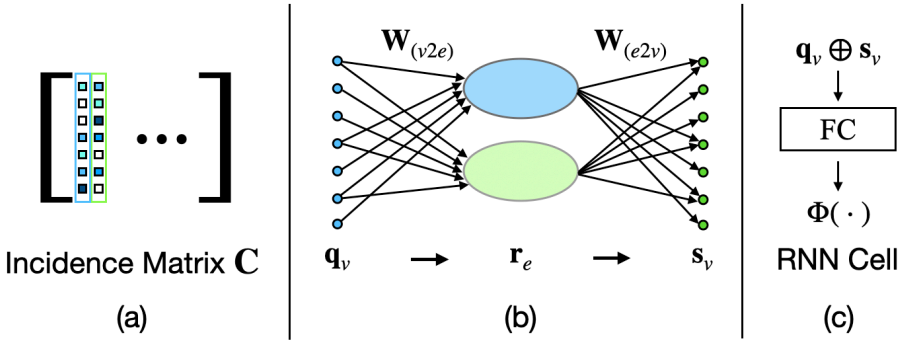


Figure 5.3: Our proposed hypergraph-based RNN cell. (a) A sampled hypergraph is used to determine hyperedges. (b) Two hyperedges are highlighted for the node-hyperedge-node aggregations. Each hyperedge is first aggregated from its incident nodes, as described in Eq. 5.7. Each node is then aggregated from its incident hyperedges, as described in Eq. 5.8. (c) The cell function $\Phi(\cdot)$ outputs a hidden state for each node by applying a fully connected layer to the original node vector $\mathbf{q}$ and the transformed node vector $\mathbf{s}$.

5.4.2 Hypergraph-based RNN Cells

With the learned probabilistic hypergraph structure from Eq. 5.3, we design a hypergraph-based RNN cell that serves as the basic unit for our encoder-decoder framework. Inspired by HGC-RNN [213], our cell takes node time series and node hidden states as input, and outputs updated hidden states, through a node-hyperedge-node update route. An example of two hyperedges are depicted in Fig. 5.3 (a). For the clarity of writing, we let $\mathbf{q}_v \in \mathbb{R}^{d_q}$ denote the input time-series vector for node v to our RNN cell. Each RNN cell consists of two stages, a node-to-hyperedge aggregation stage and a hyperedge-to-node aggregation stage, as depicted in Fig. 5.3 (b). In the node-to-hyperedge aggregation stage, we aggregate each node time-series vector to its incident hyperedges. Let $\mathbf{r}_e \in \mathbb{R}^{d_r}$ denote the vector for hyperedge $e \in E$ through a mean aggregator,

$$\text{v-to-e aggregation:} \quad \mathbf{r}_e = \frac{1}{|e|} \cdot \sigma \left(\sum_{v \in e} \mathbf{W}_{(v2e)}^{\mathsf{T}} \cdot \mathbf{q}_v \right), \quad \forall e \in E \tag{5.7}$$

where $\mathbf{W}_{(v2e)}^{\mathsf{T}} \in \mathbb{R}^{d_q \times d_r}$ denotes a weight layer that transforms node vectors into hyperedge vectors. In the hyperedge-to-node stage, the hyperedge vectors are aggregated to produce a new node vector, denoted as $\mathbf{s}_v \in \mathbb{R}^{d_s}$ for each node v,

$$\text{e-to-v aggregation:} \quad \mathbf{s}_v = \frac{1}{|E(v)|} \cdot \sigma \left(\sum_{v \in E(v)} \mathbf{W}_{(e2v)}^{\mathsf{T}} \cdot \mathbf{r}_{E(v)} \right) \tag{5.8}$$

where $\mathbf{W}_{(e2v)}^{\mathsf{T}} \in \mathbb{R}^{d_r \times d_s}$ denotes a weight layer that transforms hyperedge vectors into node vectors. $E(v)$ denotes hyperedges that are incident to node v. We adopt a residual structure

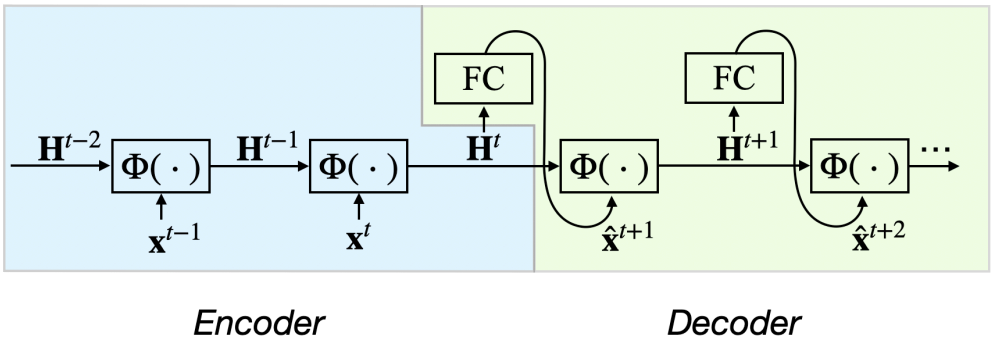


Figure 5.4: Our hypergraph encoder-decoder framework. The encoder consists of hypergraph RNN cells $\Phi(\cdot)$ that recursively consume historical time-series values, described by Eq. 5.9. Given the encoded states, the decoder utilizes hypergraph RNN cells to make forecasts through a fully connected layer.

and apply a fully connected layer to the concatenation of the original node vectors and the aggregated node vectors, as depicted in Fig. 5.3 (c). Let $\Phi(\cdot)$ denote our PHRNN cell:

$$\Phi(\mathbf{q}_v) = \mathrm{FC}\left(\mathbf{q}_v \oplus \mathbf{s}_v\right) \in \mathbb{R}^{d_\Phi} \tag{5.9}$$

5.4.3 A Hypergraph Encoder-decoder Framework

Inspired by previous work [121, 167, 202, 213], our PHRNN utilizes an encoder-decoder structure. The encoder recursively digests historical observations to learn a hidden state, which is subsequently used by the decoder to generate forecasts. Our hypergraph encoder-decoder framework is depicted in Fig. 5.4, where both the encoder and the decoder are built with the hypergraph-based RNN cell described in Sec. 5.4.2. Let $\mathbf{H}^t \in \mathbb{R}^{N \times d}$ denote the hidden state at time step t, where d denotes the dimension of the hidden state features. The hidden state $\mathbf{H}^t$ is computed from the time-series values $\mathbf{x}^t$ and the previous hidden state $\mathbf{H}^{t-1}$. Hence, the input variable $\mathbf{q}_v$ in Eq. 5.9 is substituted accordingly, as $\mathbf{q}_v \equiv \mathbf{x}^t \oplus \mathbf{H}^{t-1}$,

$$\mathbf{H}^t = \Phi\left(\mathbf{x}^t \oplus \mathbf{H}^{t-1}\right) \in \mathbb{R}^{d_\Phi} \tag{5.10}$$

The prediction for time step $t+1$ is generated by decoding the final hidden state $\mathbf{H}^t$ using a projection layer.

$$\hat{\mathbf{x}}^{t+1} = \underset{out}{\mathrm{FC}}\left(\mathbf{H}^t\right) \tag{5.11}$$

The prediction for future time steps is computed by recursively feeding the predicted output as input to the model. PHRNN takes an end-to-end supervised training approach that aims to minimize the loss between the ground truth time-series values and the predictions. We

Algorithm 1 PHRNN Model Training

Initialize model parameters $\text{FC}_*(\cdot)$, $\text{Conv}_*(\cdot)$, $\mathbf{W}_*$, etc.
for each node v and each hyperedge e in the prior knowledge graph H^K **do**
  With the time-series values $\mathbf{x}_v$
    Calculate the node embedding $\mathbf{z}_v$ in Eq. 5.4
    Calculate the hyperedge embedding $\mathbf{y}_e$ in Eq. 5.5
    Calculate the parameter $\theta_{v,e}$ in Eq. 5.6
    Sample a value for the incidence matrix $\mathbf{C}_{v,e}$ in Eq. 5.3
end for
for all time series $\mathbf{x}^t$ at timestamp t in a batch **do**
  With current model parameter estimates
    Encode $\mathbf{x}^t$ with Hypergraph-based RNN cells in Eq. 5.10
    Decode from the encoded hidden state $\mathbf{H}^t$ in Eq. 5.11 to give time-series prediction
  In the current batch, calculate the time-series loss and the regularization loss. Perform stochastic gradient descent to update the trainable parameters accordingly.
end for

use MAE as the training loss on time-series, denoted by Loss_{ts}. The MAE loss is calculated by Eq. 32. Additionally, we integrate a regularization loss on the learned hypergraph. Previous work has shown that incorporating a prior knowledge (hyper)graph potentially improves forecasting performance [167, 218]. The prior knowledge graph can be derived from external connectivity information, such as geographic distance, or from time-series patterns. In PHRNN, we reuse the KNN hypergraph from the stage of hyperedge embeddings, described in Sec. 5.4.1, for the purpose of regularization. Each hyperedge in the KNN hypergraph is constructed based on a node and its $K-1$ closest nodes in terms of time-series similarity, which aligns with the assumption that time series with similar patterns are more relatable to each other than other random time series [29]. We use the elementwise cross entropy between the learned hypergraph and the KNN hypergraph as the regularization loss,

$$\text{Loss}_C = -\frac{1}{N}\frac{1}{M}\sum_{v\in V}\sum_{e\in E}\left(\mathbf{C}_{v,e}\log\left(\mathbf{C}^K_{v,e}\right) + (1-\mathbf{C}_{v,e})\log\left(1-\mathbf{C}^K_{v,e}\right)\right) \tag{5.12}$$

Our final loss function is a combination of the time-series loss and the hypergraph loss using a regularization coefficient λ_{reg} as a weighting factor,

$$\text{Loss} = \text{Loss}_{ts} + \lambda_{reg}\text{Loss}_C \tag{5.13}$$

The workflow of PHRNN is summarized in Algorithm 1.

Table 5.2: A statistics table of used datasets

Data	$\|V\|$	$\|E\|$	Density	Min. Deg.	Mdn. Deg.	Ave. Deg.	Max. Deg.	Time-scale
Adobe	1,135	332	0.88%	1	1	2.92	38	30 min
Google	845	838	1.18%	1	7	9.89	57	5 min
Traffic	862	828	1.02%	1	7	8.45	31	1 hour

Table 5.3: Results for one-step ahead forecasting (MAE and RMSE)

Metrics	DATA	HGC-RNN	DGSL	RGSL	MTGNN	STGCN	StemGNN	**PHRNN**
MAE	Adobe	0.052 ± 0.021	0.071 ± 0.019	0.064 ± 0.023	0.052 ± 0.020	0.061 ± 0.022	0.059 ± 0.020	$\mathbf{0.039 \pm 0.021}$
	Google	0.064 ± 0.024	0.063 ± 0.012	0.062 ± 0.015	0.065 ± 0.016	0.071 ± 0.022	0.077 ± 0.020	$\mathbf{0.060 \pm 0.022}$
	Traffic	0.026 ± 0.015	0.060 ± 0.021	0.030 ± 0.015	0.025 ± 0.021	0.029 ± 0.016	0.038 ± 0.019	$\mathbf{0.013 \pm 0.011}$
RMSE	Adobe	0.101 ± 0.037	0.128 ± 0.032	0.119 ± 0.041	0.100 ± 0.036	0.117 ± 0.039	0.111 ± 0.036	$\mathbf{0.075 \pm 0.034}$
	Google	0.082 ± 0.028	0.082 ± 0.013	0.080 ± 0.018	0.081 ± 0.018	0.092 ± 0.025	0.096 ± 0.022	$\mathbf{0.077 \pm 0.025}$
	Traffic	0.040 ± 0.020	0.076 ± 0.026	0.043 ± 0.023	0.038 ± 0.028	0.044 ± 0.024	0.053 ± 0.025	$\mathbf{0.019 \pm 0.020}$

5.5 Experiments

We conduct extensive experiments on three real-world datasets to evaluate the forecasting accuracy of our PHRNN model. Furthermore, we perform a hyperparameter analysis to investigate the effectiveness of prior knowledge hypergraph.

In our experiments, we select three datasets from cloud computing systems and traffic networks. *Adobe Cloud Trace*. This dataset contains 1,135 time series of CPU utilization rates recorded from October to December in 2018, with a time interval of half an hour. *Google Cloud Trace*. The Google trace dataset records the CPU usage of tens of thousands of compute nodes in May of 2011 [154, 156]. We select a subset of highly active nodes, i.e., those with utilization rates exceeding 25% for more than 70% of their lifetime, resulting in 845 time-series of CPU utilization rates. *Traffic Occupancy*. This dataset contains 862 time series of road occupancy rates recorded in San Francisco from 2015 to 2016, with a time interval of one hour [110]. All values fall within the range of $[0, 1]$.

To evaluate the robustness of our model on various time-series patterns, we divide each dataset into 10 folds. For each fold, we randomly select a period to be split into training, validation, and testing sets in a ratio of $0.5 : 0.25 : 0.25$. Following commonly adopted configurations [167, 204], we construct data points with a time lag window $w = 12$ and a prediction horizon $\tau = 12$. For each period, we generate a prior knowledge hypergraph with $K = 10$ from time-series values, and then train a model instance. Consequently, we train 10 model instances for our PHRNN and for each baseline. A summary of the prior knowledge hypergraph statistics is provided for each dataset in Table 5.2.

We select highly related graph and hypergraph models as baselines. *HGC-RNN* [213] utilizes an explicit hypergraph for time-series forecasting. HGC-RNN also learns from a node-to-

Table 5.4: Results for multistep ahead forecasting (MAE and RMSE)

Metrics	DATA	τ	HGC-RNN	DGSL	RGSL	MTGNN	STGCN	StemGNN	**PHRNN**
MAE	Adobe	3	0.059 ± 0.020	0.071 ± 0.019	0.062 ± 0.022	0.052 ± 0.021	0.061 ± 0.021	0.059 ± 0.021	$\mathbf{0.049 \pm 0.023}$
		6	0.059 ± 0.021	0.069 ± 0.018	0.064 ± 0.023	0.054 ± 0.022	0.061 ± 0.021	0.058 ± 0.021	$\mathbf{0.051 \pm 0.023}$
		12	0.058 ± 0.021	0.068 ± 0.018	0.060 ± 0.023	0.055 ± 0.019	0.060 ± 0.024	0.058 ± 0.021	$\mathbf{0.053 \pm 0.022}$
	Google	3	0.066 ± 0.012	0.062 ± 0.010	0.063 ± 0.012	0.067 ± 0.022	0.068 ± 0.016	0.068 ± 0.014	$\mathbf{0.061 \pm 0.014}$
		6	0.069 ± 0.010	0.065 ± 0.008	0.063 ± 0.011	0.080 ± 0.039	0.064 ± 0.010	0.068 ± 0.009	$\mathbf{0.062 \pm 0.010}$
		12	0.072 ± 0.016	0.068 ± 0.016	0.073 ± 0.012	0.081 ± 0.034	0.068 ± 0.013	0.069 ± 0.012	$\mathbf{0.067 \pm 0.016}$
	Traffic	3	0.036 ± 0.012	0.061 ± 0.017	0.031 ± 0.009	0.038 ± 0.023	0.033 ± 0.015	0.036 ± 0.016	$\mathbf{0.029 \pm 0.009}$
		6	0.037 ± 0.010	0.065 ± 0.017	0.037 ± 0.019	0.041 ± 0.017	0.034 ± 0.013	0.042 ± 0.014	$\mathbf{0.034 \pm 0.008}$
		12	$\mathbf{0.033 \pm 0.012}$	0.068 ± 0.019	0.033 ± 0.010	0.036 ± 0.017	0.035 ± 0.014	0.035 ± 0.015	$\mathbf{0.033 \pm 0.012}$
RMSE	Adobe	3	0.110 ± 0.035	0.128 ± 0.031	0.115 ± 0.039	0.101 ± 0.035	0.118 ± 0.036	0.112 ± 0.036	$\mathbf{0.092 \pm 0.038}$
		6	0.111 ± 0.037	0.126 ± 0.031	0.119 ± 0.039	0.103 ± 0.035	0.117 ± 0.037	0.111 ± 0.036	$\mathbf{0.097 \pm 0.038}$
		12	0.110 ± 0.037	0.125 ± 0.031	0.113 ± 0.039	0.106 ± 0.033	0.125 ± 0.057	0.110 ± 0.036	$\mathbf{0.100 \pm 0.036}$
	Google	3	0.080 ± 0.014	0.081 ± 0.011	0.081 ± 0.015	0.085 ± 0.024	0.089 ± 0.018	0.086 ± 0.016	$\mathbf{0.078 \pm 0.015}$
		6	0.084 ± 0.012	0.080 ± 0.008	0.083 ± 0.013	0.101 ± 0.043	0.083 ± 0.011	0.085 ± 0.010	$\mathbf{0.079 \pm 0.011}$
		12	0.086 ± 0.018	0.087 ± 0.018	0.088 ± 0.015	0.101 ± 0.040	0.089 ± 0.015	0.088 ± 0.013	$\mathbf{0.085 \pm 0.019}$
	Traffic	3	0.051 ± 0.019	0.077 ± 0.022	$\mathbf{0.040 \pm 0.013}$	0.053 ± 0.029	0.050 ± 0.024	0.052 ± 0.024	0.045 ± 0.018
		6	0.053 ± 0.018	0.081 ± 0.023	0.051 ± 0.027	0.057 ± 0.024	0.050 ± 0.022	0.059 ± 0.020	$\mathbf{0.050 \pm 0.015}$
		12	0.048 ± 0.017	0.083 ± 0.021	0.046 ± 0.015	0.049 ± 0.022	$\mathbf{0.045 \pm 0.018}$	0.048 ± 0.019	0.047 ± 0.017

hyperedge aggregation and a hyperedge-to-node aggregation to predict the node-level time series. For a fair comparison, we use the same prior knowledge KNN hypergraph H^K for HGC-RNN as the one we use in PHRNN (described in Sec. 5.4.1). We also include probabilistic graph structure learning models as baselines. These methods model the pairwise interactions between nodes as probabilistic distributions. *DGSL* [167] exploits the pairwise information among time series by optimizing the mean performance over the graph distribution. The probabilistic graph structure is learned based on the historical values of time series. *RGSL* [218] fuses both implicit and explicit graphs, where the implicit graph is learned from node embeddings, and the explicit graph is rendered from domain knowledge. We derive the explicit graph from the KNN hypergraph by connecting all nodes within each hyperedge. Other nonprobabilistic graph-based models are also included. *MTGNN* [204] is a relational learning model for time-series forecasting that learns the uni-directed relations among nodes through node embeddings. *STGCN* [217] consists of a spatial layer that extracts relationships between nodes and a temporal layer that learns temporal dependencies. *StemGNN* [23] predicts time-series values by combining graph Fourier transform and discrete Fourier transform to learn the intraseries correlations and interseries correlations in the spectral domains. For hyperparameters settings, we adopt the recommended hyperparameters from the papers of baselines. Readers may refer to papers of baselines for detailed configurations. As for PHRNN, we conducted a grid search and configure hyperparameters with the following values: $d = d_z = d_s = d_\Phi = 16$, $d_r = 32$. We select $K = 10$ and Euclidean distance when generating prior knowledge hypergraphs. We adopt $s = 0.25$ and $\lambda_{reg} = 0.02$.

5.5.1 Results

We conduct experiments to evaluate the forecasting accuracy regarding one-step ahead prediction and multistep ahead prediction of our model. For PHRNN and each baseline, we report the mean values and standard deviation from their 10 model instances in terms of selected loss metrics, including MAE in Eq. 32 and RMSE in Eq. 33. For one-step ahead prediction, our proposed PHRNN outperforms all other baselines by a significant margin on the selected datasets, as shown in Table 5.3. In addition to PHRNN, the only hypergraph baseline model HGC-RNN also exhibits comparatively better performance than most other baselines, underscoring the positive impact of hypergraph modeling on forecasting accuracy. In contrast, the probabilistic graph model DGSL [167] achieves the least favorable results among the compared models in the Adobe and Traffic dataset, suggesting that only using probabilistic graph modeling may not precisely capture the interactions among time series.

For multistep ahead prediction, we assess prediction performance for horizons of 3, 6 and 12, as presented in Table 5.4. Notably, PHRNN achieves superior performance across most datasets and horizons. Despite PHRNN exhibits slightly subpar RMSE results for prediction horizons $H = 3$ and $H = 12$ on the traffic dataset compared to the baseline, this observation may be attributed to the presence of more outlier points in the traffic dataset, which contribute to larger RMSE losses for PHRNN [193]. Additionally, we observe that some models predict more accurately on larger horizons than smaller ones, which is also noticed in previous research [234]. For instance, our model has lower prediction losses on $H = 12$ than $H = 6$ for the traffic dataset. This behavior is rooted in the randomness of our data selection process, resulting in more challenging values to predict for $H = 6$ in the testing set.

5.5.2 Hyperparameter Analysis

We perform a hyperparameter analysis to investigate how the prior knowledge graph impacts PHRNN with respect to two hyperparameters, hyperedge size K and regularization coefficient λ_{reg}. Hyperedge size K determines the density of the incidence matrix and consequently affects the characteristics of the hypergraph, while regularization coefficient λ_{reg} determines how strictly the learned hypergraph should adhere to the prior knowledge hypergraph.

Effectiveness of Hyperedge Size K**.** To investigate the effectiveness of hyperedge size K in the prior knowledge graph, we vary $K \in [5, 10, 20, 40, 80]$ and run our PHRNN using the Adobe dataset. For each selected K, we report the MAE losses of 1, 3, 6, and 12 steps ahead predictions. As shown in Fig. 5.5 (Left), our PHRNN exhibits improved performance as K increases from 5 to 40. This improvement may be attributed to the inclusion of more nodes within each hyperedge in the prior knowledge hypergraph, leading to the incorporation of extra information. However, this information gain is not unlimited, as the performance deteriorates when K continues to increase to 80.

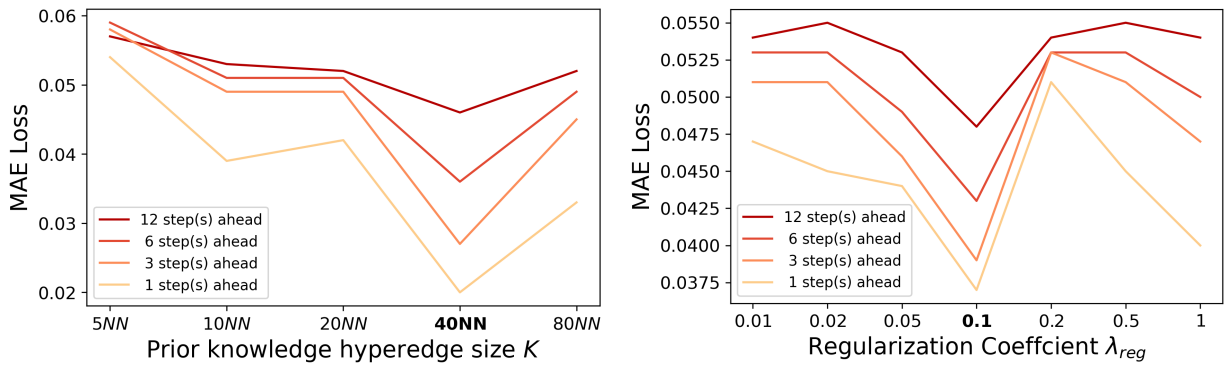


Figure 5.5: (Left) Effectiveness of PHRNN on K. MAE losses of 1, 3, 6, and 12 steps ahead prediction are reported on different hyperedge sizes $K \in [5, 10, 20, 40, 80]$. A large K causes more nodes to be included in each hyperedge of the prior knowledge hypergraph. Our PHRNN reaches the best performance when $K = 40$ (highlighted) for the Adobe dataset. (Right) Effectiveness of PHRNN on λ_{reg}. MAE losses of 1, 3, 6, and 12 steps ahead prediction are reported on different regularization coefficients with the range $\lambda_{reg} \in [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1]$. A larger λ_{reg} indicates the model weights are closer to the prior knowledge hypergraph. Our PHRNN reaches the best performance when $\lambda_{reg} = 0.1$ for the Adobe dataset.

Effectiveness of Regularization Coefficient λ_{reg}**.** To investigate the effectiveness of regularization coefficient λ_{reg}, we vary $\lambda_{reg} \in [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1]$ and run PHRNN on the Adobe dataset. The MAE losses for predictions at 1, 3, 6, and 12 steps ahead are reported for each selected λ_{reg}. As illustrated in Fig. 5.5 (Right), we observe that PHRNN benefits from assigning more weights to the prior knowledge hypergraph as λ_{reg} increases from 0.01 to 0.1, and PHRNN achieves the best performance when $\lambda_{reg} = 0.1$. However, the performance deteriorates when λ_{reg} continues to increase, as a larger regularization coefficient restricts PHRNN from effectively learning the hypergraph structure.

5.6 Conclusion

In this chapter, we introduce a novel hypergraph time-series model for time-series forecasting, namely, Probabilistic Hypergraph Recurrent Neural Network (PHRNN). PHRNN captures the nature of node interactions from two distinct perspectives: (1) PHRNN learns a hypergraph which allows nodes to engage with hyperedges, simulating broadcasting interactions in groups or communities, and (2) PHRNN models node-hyperedge interactions in a probabilistic manner, providing a more precise representation of the underlying dynamics between individual node time series and their associated hyperedges. Our extensive experiments on multiple datasets show that PHRNN consistently outperforms state-of-the-art baselines.

Chapter 6

Graph Deep Factors for Probabilistic Time-series Forecasting

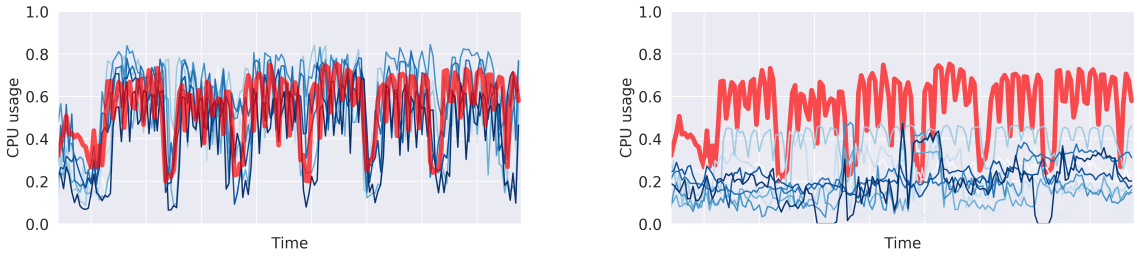
6.1 Introduction

In this chapter, we propose a novel framework that utilizes graph time-series models to make probabilistic forecasting. Time-series forecasting is significantly useful in business world, as most typically leveraged in stock price prediction and sale outlook forecasting. Recently, forecasting has been utilized for the optimization of resource allocation. For example, accurate forecasting of workload patterns on cloud cluster nodes can help service providers such as AWS or Azure, optimize the resource allocation and scheduling and therefore save money. In cloud resource optimization, the goal is to accurately forecast the resources a service or job will require given CPU and memory usages over time. For this problem, learning and inference must be fast and efficient. For instance, every 5 minutes, we receive new CPU and memory usage measurements, and as soon as we receive them, we need to learn a model and use it to forecast the next τ-steps ahead, and then make a decision to scale up/down or not. Additionally, it's important to quantify the prediction uncertainty so that decision makers can apply different strategies based on the probability of forecast values.

Recently, there is a significant increase of data-driven approaches [13, 151] in time-series prediction due to the extensive availability of abundant data from various fields, e.g. shopping behaviors of consumers [161, 162], resource usage optimization for cloud computing [50] and energy consumption [54, 120]. The huge abundance of data makes it necessary to have models that extract limited useful information from big data. At the same time, the intrinsic dependency between time series also needs to be leveraged for accurate predictions.

In local models, the free parameters are learned independently for each time series. While these models are sometimes useful, they require a large amount of data for training [81]. Since these models focus on individual time series, there is often not enough recent data available to make accurate forecasts. As a consequence, they fail to model and extract the mutual connections and dependency across time series that may help forecasting. Another disadvantage of these models is that they are comparatively simple and they require manual feature engineering and design by domain experts, which is labor-intensive and time-consuming.

In the field of multivariate forecasting, the global models have been studied for decades in



(a) Time series of neighbors of a node (b) Time series of randomly selected nodes

Figure 6.1: In (a) the time series of CPU usage for a node (machine) in the Google workload data shown in red and its immediate neighbors in the graph (blue) are highly correlated, whereas in (b) the time series of randomly selected nodes are *significantly different*.

econometrics and statistics. In contrast to local models that consider each time series individually, the free parameters in global models are learned jointly across all time series in the collection [162, 192]. The assumption behind global models is that all time series are driven by a small number of latent factors. Among the global models, the deep learning approaches [112, 153, 162] are able to capture complex nonlinear time series patterns. However, in global models, each time series are equivalently related to any other time series in the data, which is often violated in practice.

There has also recently been local-global models that attempt to combine the benefits of both [70]. Examples include mixed-effects models [42], where the fix (global) effects describe the whole population while the random (local) effects capture the idiosyncratic behavior of individuals. There are local-global models [165] that combine both types of models for time-series forecasting. However, these models do not solve the disadvantages of ignoring the different relations across time series in the global model. Also, the local model is too restricted to model each time series individually. Thus, we argue that a relational global and relational local model can lead to significantly better forecasting performance with faster training/inference while improving the data efficiency.

In terms of relational time series forecasting [159], the local models [21, 73] that treat each time series independently corresponds to a graph where each node time series is not connected to any other nodes and their time series. Conversely, global models[153, 162, 192] that consider all time series jointly correspond to a fully connected graph where each node time series is connected to every other in the same way. These past works all assume time series are either completely mutually independent or completely dependent. However, these assumptions are often violated in practice as shown in Fig. 6.1 where a node time series is shown to be dependent on an arbitrary number of other node time series.

In this work, we propose a deep hybrid *graph-based* probabilistic forecasting model called *Graph Deep Factors (GraphDF)* that allows nodes and their time series to be dependent (connected) in an arbitrary fashion. GraphDF leverages a *relational global model* that uses

the dependencies between time series in the graph to learn the complex nonlinear patterns globally while leveraging a *relational local model* to capture the individual random effects of each time series locally. The relational global model in GraphDF improves the runtime performance and scalability instead of jointly modeling all time series together (fully connected graph), which is computationally intensive, GraphDF learns the global latent factors that capture the complex nonlinear time-series patterns among the time series by leveraging only the graph that encodes the dependencies between the time series. GraphDF serves as a general framework for deep graph-based probabilistic forecasting as many components are completely interchangeable including the relational local and relational global models.

Relational local models use not only the individual time series but also the neighboring time series that are 1 or 2 hops away in the graph. Thus, the proposed relational local models are more data efficient, especially when considering shorter time series. For instance, given an individual time series with a short length (e.g., only 6 previous values), purely local models would have problems accurately estimating the parameters due to the lack of data points. However, relational local models can better estimate such parameters by leveraging not only the individual time series but the neighboring dependent time series that are 1 or 2 hops away in the graph. In comparison, relational global models are typically faster and more scalable since they avoid the pairwise dependence assumed by global models via the graph structure. By leveraging the dependencies between time series encoded in the graph, GraphDF avoids a significant amount of work that would be required if the time series are modeled jointly as done in existing state-of-the-art models.

In addition, considering the time-series streaming nature where newly incoming values arrive at each time step, we further propose an *incremental online GraphDF (IOGraphDF)* model that advances GraphDF model tremendously with respect to training runtime. Instead of training a different GraphDF model instance when new values arrive at each time step, only one IOGraphDF model instance is initialized in the first time step, and then the same model instance is modified and updated to accommodate new values over time.

6.1.1 Main Contributions

We propose a general and extensible deep hybrid graph-based probabilistic forecasting framework called *Graph Deep Factors (GraphDF)* that is capable of learning complex nonlinear time-series patterns globally using the graph time-series data to improve both computational efficiency and forecasting accuracy while learning individual probabilistic models for individual time series based on their own time series and the collection of time series from the immediate neighborhood of the node in the graph. The GraphDF framework is data-driven, fast, scalable for real-time demand forecasting, and highly data efficient.

The state-of-the-art deep probabilistic forecasting methods focus on learning a global model that considers all time series jointly or a local model learned from each individual time series independently. In this work, we propose a deep graph-based probabilistic forecasting model

that lies in between these two extremes. In particular, we propose a relational global model that learns complex nonlinear time-series patterns globally using the structure of the graph to improve both computational efficiency and forecasting performance. Similarly, instead of modeling every time series independently, we learn a relational local model that not only considers its individual time series but the time series of nodes that are connected to an individual node in the graph.

Furthermore, the proposed GraphDF framework applies to a significantly larger class of problems, which includes prior work as a special case. In particular, GraphDF naturally generalizes many existing models including those based purely on local and global models, or a combination of both. This is due to its flexibility to interpolate between purely non-relational models (either local, global, or both) and relational models that leverage the graph structure encoding the dependencies between the different time series. The experiments demonstrate the effectiveness of the proposed deep graph-based probabilistic forecasting model in terms of its forecasting performance, runtime, and scalability.

Finally, we extend GraphDF to meet the incremental online scheme and derive the IOGraphDF model, which converges over timespan to yield approximately accurate predictions as GraphDF, but takes much shorter time to train and update.

6.2 Related Work

Probabilistic Forecasting. Earlier DL-based forecasting models focused on *point forecasting* which aims at predicting optimal expected values. Recently, there is an increasing interest in *probabilistic forecasting* models [86, 113, 137, 155, 200]. Probabilistic models yield prediction as distributions and have the advantage of uncertainty estimates, which are important for downstream decision making. Some recent probabilistic models are proposed in the multivariate manner. For example, Salinas et al. [162] proposed a probabilistic forecasting model that jointly learns a global model from all available time series. Wang et al. [191] proposed DF, a hybrid global-local model that assumes time series are determined by shared factors as well as individual randomness. These methods indiscriminately model mutual dependence between time series. Hence, they imply a strong and unrealistic assumption that all time series are pairwise related to one another in a uniformly equivalent way.

In contrast, we propose a hybrid deep graph-based probabilistic forecasting framework that leverages a relational graph global component that learns the complex nonlinear time-series patterns in the large collection of relational time-series data and a relational local component that handles uncertainty by learning a probabilistic forecasting model for every individual node in the graph that not only considers the time series of the individual node, but also the time series of nodes directly connected in the graph. The relational global component of the proposed GraphDF framework leverages the graph time-series data, leading to a significant improvement in the time-efficiency, scalability, and most importantly, the forecasting accu-

racy of our model compared to the state-of-the-art DF model. Conversely, the relational local model of GraphDF has the advantage of improving both forecasting accuracy and data efficiency.

Resource Usage Prediction. Researchers and engineers put great efforts on resource provisioning and load prediction in cloud scale systems [15, 142, 184]. Early work mainly utilized the traditional state space models such as ARIMA [22, 236]. More recent work covers both traditional methods [22, 236], machine learning approaches [41] such as K-nearest neighbors [62, 168] and linear regression [61, 210] and RNN-based methods [35, 60, 108]. However, none of these methods leverages a graph to model the relationships between nodes.

Prediction on Streaming Data. In many applications, data values are not given beforehand but instead arrive continuously with an equivalent time gap between arrivals. Early work on prediction in this kind of scenario includes modifying ARIMA models to an online manner [3, 129], predicting with kernel-based methods [157], and efforts on elastic resource scaling to reduce cloud system operating cost [16, 168] More recent work leverages deep learning on streaming data. For instance, Vrablecová et al. [185] proposed a stream change detection method to identify the ongoing changes or concept drifts of the power meter data. Guo et al. [76] proposed an adaptive gradient learning method which aims to minimize impacts from outliers as well as leverage the local features, but this work is solely based on RNN and only targets univariate time-series prediction. A more recent RNN-based work [55] targets finding mismatch of temporal distribution between periods of time series. However, these models do not leverage graph structures or the intercorrelations between time series for forecasting. By contrast, our proposed work is graph-based and has the advantage of forecasting accuracy and runtime efficiency.

6.3 Graph Deep Factors

In this section, we describe a general and extensible framework called Graph Deep Factors (GraphDF). It is capable of learning complex nonlinear time-series patterns globally using the graph time-series data to improve both computational efficiency and performance while learning probabilistic models for each individual time series based on their own time series and the collection of related time series from the neighborhood of the node in the graph. The GraphDF framework is data-driven, flexible, accurate, and scalable for large collections of multidimensional time-series data.

6.3.1 Problem Formulation

We first introduce the deep graph-based probabilistic forecasting problem. Notably, this is the first hybrid deep graph-based probabilistic forecasting framework. The framework

is comprised of a graph relational global component (Sec. 6.3.3) that learns the complex non-linear time-series patterns in the large collection of graph-based time-series data and a relational local component (Sec. 6.3.4) that handles uncertainty by learning a probabilistic forecasting model for every individual node in the graph that not only considers the time series of the individual node, but also the time series of nodes directly connected in the graph. This has the advantage of improving both forecasting accuracy and data efficiency.

The proposed framework solves the following graph-based time-series forecasting problem. Let $G = (V, E, \mathcal{X}, \mathcal{Z})$ denote the graph model where V is the set of nodes, E is the set of edges, and $\mathcal{X} = \{\mathbf{X}^{(i)}\}_{i=1}^{N}$ is the set of covariate time series associated with the N nodes in G where $\mathbf{X}^{(i)} \in \mathbb{R}^{D \times T}$ is the covariate time-series data associated with node i. Hence, each node is associated with D different covariate time series. Furthermore, $\mathcal{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{N}$ is the set of time series associated with the N nodes in G. The N nodes can be connected in an arbitrary fashion that reflects the dependence between nodes. Two nodes i and j that contain an edge $(i, j) \in E$ in the graph G encodes an explicit dependency between the time-series data of node i and j. Intuitively, using these explicit dependencies encoded in G can lead to more accurate forecasts as shown in Fig. 6.1. Further, let $\mathbf{z}_{1:T}^{(i)}$ denote a univariate time series for node i in the graph where $\mathbf{z}_{1:T}^{(i)} = \begin{bmatrix} z_1^{(i)} & \cdots & z_T^{(i)} \end{bmatrix} \in \mathbb{R}^T$ and $z_t^{(i)} \in \mathbb{R}$. In addition, each node i in the graph G also has D covariate time series, $\mathbf{X}^{(i)} \in \mathbb{R}^{D \times T}$ where $\mathbf{X}_{:,t}^{(i)} \in \mathbb{R}^D$ (or $\mathbf{x}_t^{(i)} \in \mathbb{R}^D$) represents the D covariate values at time step t for node i. We also denote $\mathbf{A} \in \mathbb{R}^{N \times N}$ as the sparse adjacency matrix of the graph G where $N = |V|$ is the number of nodes. If $(i, j) \in E$, then A_{ij} denotes the weight of the edge (dependency) between node i and j, and $A_{ij} = 0$ when $(i, j) \notin E$ otherwise.

We denote the unknown model parameters as $\mathbf{\Phi}$. Our goal is to learn a generative probabilistic forecasting model described by $\mathbf{\Phi}$ that gives the (joint) distribution on target values in the future horizon τ:

$$\mathbb{P}\left(\left\{\mathbf{z}_{T+1:T+\tau}^{(i)}\right\}_{i=1}^{N} \;\middle|\; \mathbf{A}, \left\{\mathbf{z}_{1:T}^{(i)}, \mathbf{X}_{:,1:T+\tau}^{(i)}\right\}_{i=1}^{N}; \mathbf{\Phi}\right) \tag{6.1}$$

Hence, solving Eq. 6.1 gives the joint probability distribution over future values given all covariates and past observations along with the graph structure that encodes the explicit dependencies between the N nodes and their corresponding time series $\{\mathbf{z}^{(i)}, \mathbf{X}^{(i)}\}_{i=1}^{N}$.

Graph Construction. For each dataset, we derive a graph where each node represents a machine with one or more time-series associated with it, and each edge represents the similarity between the node time-series i and j. The constructed graph encodes the dependency information between nodes. In this work, we estimate the edge weights using the Radial Basis Function (RBF) kernel with the previous time-series observations as $\psi(\mathbf{z}_i, \mathbf{z}_j) = \exp(-\frac{\|\mathbf{z}_i - \mathbf{z}_j\|^2}{2\ell^2})$, where ℓ is the length scale of the kernel.

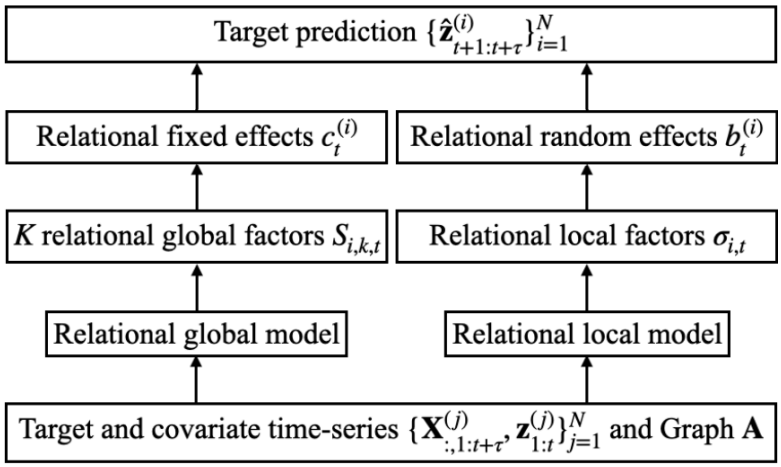


Figure 6.2: An overview of GraphDF framework

6.3.2 Framework Overview

The Graph Deep Factors (GraphDF) framework aims to learn a parametric distribution to predict future values. In GraphDF, each node i and its time series $z_t^{(i)}, \forall t = 1, 2, \ldots$ can be connected to other nodes and their time series in an arbitrary fashion, which is encoded in the graph G. These connections represent explicit dependencies or correlations between the time series of the nodes. Furthermore, we also assume that each node i and their time series $\mathbf{z}_{1:t}^{(i)}$ are governed by two key components, including (1) a relational global model (Sec. 6.3.3), and (2) a relational local random effect model (Sec. 6.3.4). As such, GraphDF is a hybrid forecasting framework. Both the relational global component and the relational local component of our framework leverage the graph via the specific underlying model used for each component.

In the relational global component of GraphDF, we assume there are K latent relational global factors that determine the fixed effect to each node and their time series. Specifically, the relational global model consists of an approach that leverages the adjacency matrix $\mathbf{A}$ of the graph G and $\left\{\mathbf{X}^{(j)}_{:,1:t}, \mathbf{z}^{(j)}_{1:t-1}\right\}$ for learning the K relational global factors that capture the relational nonlinear time-series patterns in the graph-based time-series data,

$$\text{relational global factors:} \quad s_k(\cdot) = \textsc{gcrn}_k(\cdot), \quad k = 1, \ldots, K \tag{6.2}$$

where $s_k(\cdot),\ k = 1, 2, \ldots, K$ are the K relational global factors that govern the underlying graph-based time-series data of all nodes in G. In Eq. 6.2, we learn the relational global factors using a Graph Convolutional Recurrent Network (GCRN) [166], however, GraphDF is flexible for use with any other arbitrary deep time-series model such as DCRNN, among many other possibilities. These are then used to obtain the relational global fixed effects

function $c^{(i)}$ for node i as follows,

$$\text{fixed effect:} \quad c^{(i)}(\cdot) = \sum_{k=1}^{K} w_{i,k} \cdot s_k(\cdot) \tag{6.3}$$

where $w_{i,k}$ represents the K-dimensional embedding for node i. Therefore, the final relational nonrandom fixed effect for node i is simply a linear combination of the K global factors and the embedding $\mathbf{w}_i \in \mathbb{R}^K$ for node i. Now we use a relational local model discussed in Sec. 6.3.4 to obtain the local random effects for each node i. More formally, we define the *relational local random effects* function $b^{(i)}$ for a node i in the graph G as,

$$\text{relational local random effect:} \quad b^{(i)}(\cdot) \sim \mathcal{R}_i, \quad i = 1, \ldots, N \tag{6.4}$$

where $\mathcal{R}_i$ can be any relational probabilistic time-series model. To compute $\mathbb{P}(\mathbf{z}^i_{1:t}|\mathcal{R}_i)$ efficiently, we let $b^{(i)}_t$ follow a normal distribution, and thus can be derived fast. The *relational latent function* of node i denoted as $v^{(i)}$ is then defined as,

$$\text{latent function:} \quad v^{(i)}(\cdot) = c^{(i)}(\cdot) + b^{(i)}(\cdot) \tag{6.5}$$

where $c^{(i)}$ is the relational fixed effect of node i and $b^{(i)}$ is the relational local random effect for node i. Hence, the relational latent function of node i is simply a linear combination of the relational fixed effect $c^{(i)}$ from Eq. 6.3 and its local relational random effect $b^{(i)}$ from Eq. 6.4. Then,

$$\text{emission:} \quad z^{(i)}_t \sim \mathbb{P}\Big(z^{(i)}_t \;\Big|\; v^{(i)}\big(\mathbf{A}, \big\{\mathbf{X}^{(j)}_{:,1:t}, \mathbf{z}^{(j)}_{1:t-1}\big\}_{j=1}^{N}\big)\Big) \tag{6.6}$$

where the observation model $\mathbb{P}$ can be any parametric distribution. For instance, $\mathbb{P}$ can be Gaussian, Poisson, Negative Binomial, among others.

The GraphDF framework is defined in Eq. 6.2-6.6. All the functions $s_k(\cdot), b^{(i)}(\cdot), v^{(i)}(\cdot)$ take past observations and covariates $\big\{\mathbf{z}^{(j)}_{1:t-1}, \mathbf{X}^{(j)}_{:,1:t}\big\}_{j=1}^{N}$, as well as the graph structure in the form of adjacency matrix $\mathbf{A}$ as inputs. We define $\mathbf{w}_i = \big[w_{i,1} \cdots w_{i,k} \cdots w_{i,K}\big] \in \mathbb{R}^K$ as the K-dimension embedding for time-series $\mathbf{z}^{(i)}$ where $w_{i,k} \in \mathbb{R}$ is the weight of the k-th factor for node i. An overview of the GraphDF framework is depicted in Fig. 6.2.

6.3.3 Relational Global Model

The relational global model learns K relational global factors from all time series by a graph-based model. These relational global factors are considered as the driving latent factors. After the relational global factors are derived from the model, they are then used in a linear combination with weights given by embeddings for each time series $\mathbf{w}_i$, as shown in Eq. 6.3.

6.3.3.1 Learning Relational Global Factors via GCRN

We first show how GCRN [166] can be modified for learning relational global factors in GraphDF. Let $\mathbf{x}_t^{(i)} \in \mathbb{R}^D$ denote the D covariates of node i at time step t. Now, we define the input temporal features of the relational global factor component of the graph G as,

$$\mathbf{Y}_t = \begin{bmatrix} z_{t-1}^{(1)} & \mathbf{x}_t^{(1)} \\ \vdots & \vdots \\ z_{t-1}^{(N)} & \mathbf{x}_t^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times P} \tag{6.7}$$

where $P = D+1$ for simplicity. We refer to $\mathbf{Y}_t$ as a time-series graph signal. The aggregation of information from other nodes is performed by a graph convolution operation defined as the multiplication of a temporal graph signal with a filter g_θ. Given input features $\mathbf{Y}_t$, the graph convolution operation is denoted as $f_{\star_\mathcal{G}}\,\mathbf{\Theta}$ with respect to graph G and parameters θ:

$$f_{\star_\mathcal{G}}\,\mathbf{\Theta}(\mathbf{Y}_t) = g_\theta(\mathbf{L})\mathbf{Y}_t \tag{6.8}$$

$$= \mathbf{U} g_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{Y}_t \in \mathbb{R}^{N \times P} \tag{6.9}$$

where $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the normalized Laplacian matrix of the adjacency matrix, $\mathbf{I} \in \mathbb{R}^{N \times N}$ is an identity matrix. $D_{ii} = \sum_j A_{ij}$ is the diagonal weighted degree matrix. $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ is the eigenvalue decomposition. $\mathbf{U}$ is the matrix composed of eigenvectors by order of eigenvalues of $\mathbf{L}$, and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{L}$. $g_\theta(\mathbf{\Lambda}) = \text{diag}(\boldsymbol{\theta})$ denotes a filter parameterized by the coefficients $\boldsymbol{\theta} \in \mathbb{R}^N$ in the Fourier domain [49, 85]. Directly applying Eq 6.9 is computationally expensive due to the matrix multiplication and the eigen-decomposition of $\mathbf{L}$. To accelerate the computation speed, the Chebyshev polynomial approximation up to a selected order $L-1$ is

$$g_\theta(\mathbf{L}) = \sum_{l=0}^{L-1} \theta_l T_l(\tilde{\mathbf{L}}) \tag{6.10}$$

where $\boldsymbol{\theta} = \begin{bmatrix}\theta_0 \cdots \theta_{L-1}\end{bmatrix} \in \mathbb{R}^L$ in Eq. 6.10 is the Chebyshev coefficients vector. Importantly, $T_l(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}}T_{l-1}(\tilde{\mathbf{L}}) - T_{l-2}(\tilde{\mathbf{L}})$ is recursively computed with the scaled Laplacian $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I} \in \mathbb{R}^{N \times N}$, with starting values $T_0 = 1$ and $T_1 = \tilde{\mathbf{L}}$. The Chebyshev polynomial approximation improves the time complexity to linear in the number of edges $O(L|E|)$, i.e., number of dependencies between the multidimensional node time series. The order L controls the local neighborhood time series that are used for learning the relational global factors, i.e., a node's multi-dimensional time series only depends on neighboring node time series that are at maximum L hops away in the graph G.

Let $\mathbf{\Theta} \in \mathbb{R}^{P \times Q \times L}$ be a tensor of parameters that maps the dimension P of input to the dimension Q of output:

$$\mathbf{H}_{:,q} = \tanh\left[\sum_{p=1}^{P} f_{\star_\mathcal{G}}\,\mathbf{\Theta}(\mathbf{Y}_{t,\,:,p})\right], \quad \text{for } q \in 1 \ldots Q \tag{6.11}$$

The relational global component integrates the temporal dependence and relational dependence among nodes with the graph convolution,

$$\mathbf{I}_t = \sigma(\mathbf{\Theta}_I \star_{\mathcal{G}} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_I \odot \mathbf{C}_{t-1} + \mathbf{b}_I) \tag{6.12}$$
$$\mathbf{F}_t = \sigma(\mathbf{\Theta}_F \star_{\mathcal{G}} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_F \odot \mathbf{C}_{t-1} + \mathbf{b}_F) \tag{6.13}$$
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t\text{-}1} + \mathbf{I}_t \odot \tanh(\mathbf{\Theta}_C \star_{\mathcal{G}} [\mathbf{Y}_t, \mathbf{H}_{t\text{-}1}] + \mathbf{b}_C) \tag{6.14}$$
$$\mathbf{O}_t = \sigma(\mathbf{\Theta}_O \star_{\mathcal{G}} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_O \odot \mathbf{C}_t + \mathbf{b}_O) \tag{6.15}$$
$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{6.16}$$

where $\mathbf{I}_t, \mathbf{F}_t, \mathbf{O}_t \in \mathbb{R}^{N \times Q}$ are the input, forget and output gate in the LSTM structure. Q is the number of hidden units, $\mathbf{W}_I, \mathbf{W}_F, \mathbf{W}_O \in \mathbb{R}^{N \times Q}$ and $\mathbf{b}_I, \mathbf{b}_F, \mathbf{b}_C, \mathbf{b}_O \in \mathbb{R}^{Q}$ are weights and bias parameters, $\mathbf{\Theta}_I, \mathbf{\Theta}_F, \mathbf{\Theta}_C, \mathbf{\Theta}_O \in \mathbb{R}^{P \times Q}$ are parameters corresponding to different filters.

The hidden state $\mathbf{H}_t \in \mathbb{R}^{N \times Q}$ encodes the observation information from $\mathbf{H}_{t-1}$ and $\mathbf{Y}_t$, as well as the relations across nodes through the graph convolution described by $\mathbf{\Theta} \star_{\mathcal{G}} (\cdot)$ in Eq. 6.8. From hidden state $\mathbf{H}_t$, we derive the value of K relational global factors at time step t as $\mathbf{S}_t \in \mathbb{R}^{N \times K}$ through a fully connected layer,

$$\mathbf{S}_t = \mathbf{H}_t \mathbf{W} + \mathbf{b} \tag{6.17}$$

where $\mathbf{W} \in \mathbb{R}^{Q \times K}$ and $\mathbf{b} \in \mathbb{R}^{K}$ are the weight matrix and bias vector trained in the model (for the K relational global factors), respectively. The relational global factors $\mathbf{S}_t$ is derived from the Eq. 6.17 that capture the complex nonlinear time-series patterns between the different time-series globally.

Finally, the fixed effect at time t is derived for each node i as a weighted sum with the embedding $\mathbf{w}_i \in \mathbb{R}^K$ and the relational global factors $\mathbf{S}_t$, as

$$c_t^{(i)}(\cdot) = \sum_{k=1}^{K} w_{i,k} \cdot S_{i,k,t} \tag{6.18}$$

The embedding $\mathbf{w}_i$ represents the contribution that each relational factor has on node i.

6.3.3.2 Learning Relational Global Factors via DCRNN

For the relational global component of GraphDF, we can also leverage DCRNN [121]. Different from the GCRN model, the original DCRNN leverages a diffusion convolution operation and a GRU structure for learning the relational global factors of GraphDF.

Given the time-series graph signal $\mathbf{Y}_t \in \mathbb{R}^{N \times P}$ with N nodes, the diffusion convolution with respect to the graph-based time series is defined as,

$$f_{\star_{\mathcal{G}}} \mathbf{\Theta}(\mathbf{Y}_t) = \sum_{l=0}^{L-1} (\theta_l \tilde{\mathbf{A}}^{l}) \mathbf{Y}_t \tag{6.19}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ is the normalized adjacency matrix of the graph G that captures the explicit weighted dependencies between the multidimensional time series of the nodes. The Chebyshev polynomial approximation is used similarly as in Eq. 6.10.

The relational global factors are learned using the graph diffusion convolution combined with GRU enabling them to be carried forward over time using the graph structure,

$$\mathbf{R}_t = \sigma\left(\mathbf{\Theta}_R \star_{\mathcal{G}} \left[\mathbf{Y}_t,\ \mathbf{H}_{t-1}\right] + \mathbf{b}_R\right) \tag{6.20}$$
$$\mathbf{U}_t = \sigma\left(\mathbf{\Theta}_U \star_{\mathcal{G}} \left[\mathbf{Y}_t,\ \mathbf{H}_{t-1}\right] + \mathbf{b}_U\right) \tag{6.21}$$
$$\mathbf{C}_t = \tanh\left(\mathbf{\Theta}_C \star_{\mathcal{G}} \left[\mathbf{Y}_t,\ (\mathbf{R}_t \odot \mathbf{H}_{t-1})\right] + \mathbf{b}_C\right) \tag{6.22}$$
$$\mathbf{H}_t = \mathbf{U}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{U}_t) \odot \mathbf{C}_t \tag{6.23}$$

where $\mathbf{H}_t \in \mathbb{R}^{N \times Q}$ denotes the hidden state of the model at time step t. Q is the number of hidden units, $\mathbf{R}_t, \mathbf{U}_t \in \mathbb{R}^{N \times Q}$ are called reset gate and update gate at time t, respectively. $\mathbf{\Theta}_R, \mathbf{\Theta}_U, \mathbf{\Theta}_C \in \mathbb{R}^L$ denote the parameters corresponding to different filters.

With the hidden state $\mathbf{H}_t$ in Eq. 6.23, the fixed effect is derived from DCRNN similarly with Eq. 6.17 and Eq. 6.18. Compared to the previous GCRN that we adapted for the relational global component, DCRNN is more computationally efficient due to its GRU structure.

6.3.4 Relational Local Model

The (stochastic) relational local component handles uncertainty by learning a probabilistic forecasting model for every individual node in the graph G that not only considers the time series of the individual node, but also the time series of nodes directly connected to it. This has the advantage of improving both forecasting accuracy and data efficiency.

The random effects in the relational local model represent the local fluctuations of the individual node time series. The relational local random effect for each node time series $b^{(i)}$ is sampled from the relational local model $\mathcal{R}_i$, as shown in Eq. 6.4. For $\mathcal{R}_i$, we choose the Gaussian distribution as the likelihood function for sampling, but other parametric distributions such as Student-t or Gamma distributions are also possible. Compared to the relational global component of GraphDF from Sec. 6.3.3 that uses the entire graph G along with all the node multidimensional time series to learn K global factors that capture the most important nonlinear time-series patterns in the graph-based time-series data, the relational local component focuses on modeling an individual node $i \in V$ and therefore leverages only the time series of node i and the set of highly correlated time series from its immediate local neighborhood Γ_i. Hence, $\{\mathbf{z}^{(j)}, \mathbf{X}^{(j)}\}, j \in \Gamma_i$. Intuitively, the relational local component of GraphDF achieves better data efficiency by leveraging the highly correlated neighboring time series along with its own time series. This allows GraphDF to make more accurate forecasts further in the future with less training data. We now introduce probabilistic GCRN and probabilistic DCRNN model that can be used as the stochastic relational local component.

6.3.4.1 Estimating Uncertainties via Probabilistic GCRN

In this section, we propose a relational local probabilistic GCRN model for use with the GraphDF framework. In contrast to the relational global model in Sec. 6.3.3, the relational local model focuses on learning an individual local model for each node based on its own multidimensional time series data as well as the nodes neighboring it. This enables us to model the local fluctuations of the individual multidimensional time series of each node.

Compared to RNN, the benefits of the proposed probabilistic GCRN model in the local component is that it not only models the sequential nature of the data, but also exploits the graph structure by using the surrounding nodes to learn a more accurate model for each individual node in G. This is an ideal property as we assume the fluctuations of each node are related to those of other connected nodes in the ℓ-localized neighborhood, which was shown to be the case in Fig. 6.1.

For simplicity, let $C = \Gamma_i$ denote the set of neighbors of a node i in the graph G. Note that C can be thought of as the set of related neighbors of node i, which may be the immediate 1-hop neighbors, or more generally, the ℓ-hop neighbors of i. Recall that we define $\mathbf{x}_t^{(i)} \in \mathbb{R}^D$ as the D covariates of node i at time t. Then, we define $\mathbf{X}_t^C$ as an $|C| \times D$ matrix consisting of the covariates of all the neighboring nodes $j \in C$ of node i.

$$\mathbf{X}_t^{(C)} = \begin{bmatrix} \mathbf{x}_t^{(C_1)} & \mathbf{x}_t^{(C_2)} & \cdots & \mathbf{x}_t^{(C_{|C|})} \end{bmatrix}^\intercal \tag{6.24}$$

where C_j denotes the j(th) neighbor.

Now, we define the input temporal features of the relational local model for node i,

$$\mathbf{Y}_t^{(i)} = \begin{bmatrix} z_{t-1}^{(i)} & \mathbf{x}_t^{(i)\intercal} \\ \mathbf{z}_{t-1}^{(C)} & \mathbf{X}_t^{(C)} \end{bmatrix} \tag{6.25}$$

Let $\mathbf{L}^{(i)} \in \mathbb{R}^{(|C|+1)\times(|C|+1)}$ denote the submatrix of Laplacian matrix $\mathbf{L}$ that consist of rows and columns corresponding to node i and its neighbors C. For each node i, we derive the relational local random effect using its past observations and covariates of the node i and those of its neighbors through the graph convolution with respect to $\mathbf{L}^{(i)}$.

$$\begin{aligned}
\mathbf{I}_t^{(i)} &= \sigma\left(\boldsymbol{\Theta}_I^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)},\ \mathbf{H}_{t-1}^{(i)}\right] + \mathbf{W}_I^{(i)} \odot \mathbf{C}_{t-1}^{(i)} + \mathbf{b}_I^{(i)}\right) \\
\mathbf{F}_t^{(i)} &= \sigma\left(\boldsymbol{\Theta}_F^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)},\ \mathbf{H}_{t-1}^{(i)}\right] + \mathbf{W}_F^{(i)} \odot \mathbf{C}_{t-1}^{(i)} + \mathbf{b}_F^{(i)}\right) \\
\mathbf{C}_t^{(i)} &= \mathbf{F}_t^{(i)} \odot \mathbf{C}_{t-1}^{(i)} + \mathbf{I}_t^{(i)} \odot \tanh\left(\boldsymbol{\Theta}_C^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)}, \mathbf{H}_{t-1}^{(i)}\right] + \mathbf{b}_C^{(i)}\right) \\
\mathbf{O}_t^{(i)} &= \sigma\left(\boldsymbol{\Theta}_O^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)},\ \mathbf{H}_{t-1}^{(i)}\right] + \mathbf{W}_O^{(i)} \odot \mathbf{C}_t^{(i)} + \mathbf{b}_O^{(i)}\right) \\
\mathbf{H}_t^{(i)} &= \mathbf{O}_t^{(i)} \odot \tanh\left(\mathbf{C}_t^{(i)}\right)
\end{aligned}$$

where $\mathbf{\Theta}_I^{(i)}, \mathbf{\Theta}_F^{(i)}, \mathbf{\Theta}_C^{(i)}, \mathbf{\Theta}_O^{(i)} \in \mathbb{R}^{P \times R}$ denote the parameters corresponding to different filters of the relational local model, R is the number of hidden units in the relational local model, and recall $P = D + 1$. Further, $\mathbf{H}_t^{(i)} \in \mathbb{R}^{(|C|+1) \times R}$ is the hidden state for node i and its neighbors Γ_i. $\mathbf{W}_I^{(i)}, \mathbf{W}_F^{(i)}, \mathbf{W}_O^{(i)} \in \mathbb{R}^{(|C|+1) \times R}$ are weight matrix parameters and $\mathbf{b}_I^{(i)}, \mathbf{b}_F^{(i)}, \mathbf{b}_C^{(i)}, \mathbf{b}_O^{(i)} \in \mathbb{R}^R$ are bias vector parameters. Note that in the above formulation, we assume $\ell = 1$, hence, only the immediate 1-hop neighbors are used.

From the hidden state $\mathbf{H}_t^{(i)}$, we only take the row corresponding to node i to derive the relational local random effect for node i. We denote the value as $\mathbf{h}_t^{(i)} \in \mathbb{R}^R$, and apply a fully connected layer with a softplus activation function to aggregate the hidden units,

$$\sigma_{i,t} = \log\left(\exp(\mathbf{w}^{(i)\intercal}\mathbf{h}_t^{(i)} + \beta^{(i)}) + 1\right) \tag{6.26}$$

where $\mathbf{w}^{(i)} \in \mathbb{R}^R$ and $\beta^{(i)}$ are weight vector and bias, respectively.

Finally, the relational local random effect $b_t^{(i)}(\cdot)$ for node i at time t is sampled from a Gaussian distribution with zero mean and a variance given by σ^2 in Eq. 6.26,

$$b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2) \tag{6.27}$$

The relational local random effect $b_t^{(i)}$ captures both past observations, covariates values of node i and its neighbors Γ_i for uncertainty estimates through $\sigma_{i,t}$, which is given by the probabilistic GCRN. A small $\sigma_{i,t}$ means a low uncertainty of prediction for node i at t. Specifically, the probabilistic model subsumes the point forecasting model when the relational local random effect is zero for all nodes at all timesteps as $\sigma_{i,t} = 0, \forall i \forall t$. The probabilistic property also allows the uncertainty to be propagated forward in time.

6.3.4.2 Estimating Uncertainties via Probabilistic DCRNN

We also describe a probabilistic DCRNN for the relational local component of GraphDF. For a given node i, its relational local random effect is derived with respect to its past observations, covariates and those of its neighbors, denoted by $\mathbf{Y}_t^{(i)} \in \mathbb{R}^{(|C|+1) \times P}$ as defined in Eq. 6.25. The diffusion convolution models the relational local random effect among nodes. The GRU structure is adapted with the diffusion convolution to allow the random effects to be forwarded in time.

$$\mathbf{R}_t^{(i)} = \sigma\left(\mathbf{\Theta}_R^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}\right] + \mathbf{b}_R^{(i)}\right) \tag{6.28}$$

$$\mathbf{U}_t^{(i)} = \sigma\left(\mathbf{\Theta}_U^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}\right] + \mathbf{b}_U^{(i)}\right) \tag{6.29}$$

$$\mathbf{C}_t^{(i)} = \tanh\left(\mathbf{\Theta}_C^{(i)} \star_{\mathcal{G}} \left[\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}\right] + \mathbf{b}_C^{(i)}\right) \tag{6.30}$$

$$\mathbf{H}_t^{(i)} = \mathbf{U}_t^{(i)} \odot \mathbf{H}_{t-1}^{(i)} + \left(1 - \mathbf{U}_t^{(i)}\right) \odot \mathbf{C}_t^{(i)} \tag{6.31}$$

Algorithm 2 Training Graph Deep Factor Models

1: Initialize the model parameters $\mathbf{\Phi}$ ($\mathbf{W}_*, \mathbf{b}_*, \mathbf{\Theta}_*$, etc.)
2: **for** each time series pair $\{(\mathbf{z}^{(i)}, \mathbf{x}^{(i)})\}$ in a batch **do**
3: With current model parameter estimates $\mathbf{\Phi}$
4: Calculate the relational fixed effect $c_t^{(i)} = \sum_{k=1}^{K} w_{i,k} \cdot S_{i,k,t}$ described in Sec. 6.3.3
5: Calculate the relational local random effect, $b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2)$ described in Sec. 6.3.4
6: Compute marginal likelihood, $\mathbb{P}(\mathbf{z}^{(i)}) = \sum_t -\frac{1}{2}\ln(2\pi\sigma_{i,t}) - \sum_t \frac{(z_t^{(i)} - c_{i,t})^2}{2\sigma_{i,t}^2}$ and accumulate loss
7: **end for**
8: Compute the overall loss in the current batch and perform stochastic gradient descent and update the trainable parameters $\mathbf{\Phi}$ accordingly.

where $\mathbf{\Theta}_R^{(i)}, \mathbf{\Theta}_U^{(i)}, \mathbf{\Theta}_C^{(i)} \in \mathbb{R}^{P \times R}$ denote the parameters corresponding to different filters, $\mathbf{H}_t^{(i)} \in \mathbb{R}^{(|C|+1) \times R}$ is the hidden state for node i and its neighbors Γ_i, R is the number of hidden units in the relational local model. $\mathbf{b}_I^{(i)}, \mathbf{b}_F^{(i)}, \mathbf{b}_C^{(i)}, \mathbf{b}_O^{(i)} \in \mathbb{R}^R$ are bias parameters. The graph convolution in equations above is performed with the submatrix $\mathbf{L}^{(i)}$ taken from the Laplacian matrix $\mathbf{L}$ of the graph G that explicitly models the important and meaningful dependencies between the multidimensional time-series data of each node. The matrix $\mathbf{L}^{(i)}$ consists of rows and columns corresponding to node i and its neighbors Γ_i. With the hidden state $\mathbf{H}_t^{(i)}$, the relational local random effect $b_t^{(i)}(\cdot)$ is calculated similarly with Eq. 6.26 and Eq. 6.27.

6.3.5 Learning & Inference

To train a GraphDF model, we estimate the parameters $\mathbf{\Phi}$, which represent all trainable parameters ($\mathbf{W}$, etc.) in the relational global and relational local models, as well as the parameters in the embeddings. We leverage the maximum likelihood estimation,

$$\mathbf{\Phi} = \operatorname{argmax} \sum_i \mathbb{P}\left(\mathbf{z}^{(i)} \middle| \mathbf{\Phi}, \mathbf{A}, \left\{\mathbf{X}_{:,1:t}^{(j)}, \mathbf{z}_{1:t-1}^{(j)}\right\}_{j=1}^{N}\right) \tag{6.32}$$

where

$$\mathbb{P}(\mathbf{z}^{(i)}) = \sum_t -\frac{1}{2}\ln(2\pi\sigma_{i,t}) - \sum_t \frac{(z_t^{(i)} - c_{i,t})^2}{2\sigma_{i,t}^2} \tag{6.33}$$

is the negative log likelihood of Gaussian function. Notice that maximizing $-\frac{1}{2}\ln(2\pi\sigma_{i,t})$ will minimize the relational local random effect, at the same time, σ is small when the predicted fixed effect $c_{i,t}$ is close to the actual value $z_t^{(i)}$, as shown in the second term $\frac{(z_t^{(i)} - c_{i,t})}{2\sigma_{i,t}^2}$ in Eq. 6.33. We describe a general training procedure in Algorithm 2.

6.3.6 Model Variants

In this section, we define a few of the GraphDF model variants investigated in Sec. 6.5.

- **GraphDF-GG:** This is the default model in our GraphDF framework, where we use a graph model to learn the K relational global factors (Sec. 6.3.3) and the probabilistic local graph component from Sec. 6.3.4.1 as the relational local model.
- **GraphDF-GR:** This model variant from the GraphDF framework uses the GCRN from Sec. 6.3.3 to learn the K relational global factors from the graph-based time-series data and leverages a simple RNN for modeling the local random effects of each node.
- **GraphDF-RG:** This model variant from the GraphDF framework uses a simple RNN to learn the K global factors and fixed effects of the nodes and for the relational random effects of the nodes, we leverage the probabilistic graph component from Sec. 6.3.4.1 as the relational local model.

The GraphDF framework is flexible with many interchangeable components. Importantly, the relational global component (Sec. 6.3.3) of GraphDF is completely interchangeable. In particular, this component uses the graph-based time-series data to learn the K global factors and fixed effects of the nodes. Similarly, one can also leverage any arbitrary relational local model (Sec. 6.3.4) for obtaining the relational local random effects of the nodes.

6.4 Incremental Online Learning for GraphDF

In practical setting, time series are changing frequently in a streaming fashion as new time-series observations arrive for all N nodes. For instance, in the Google cloud dataset [154] that we used, the time interval is five minutes, which means that new time-series observations for all N nodes arrive every five minutes. In such scenarios, we want to incrementally update the forecasting model without the need to relearn the entire model from scratch every time a new point arrives in the stream.

However, the original GraphDF model is incapable of incrementally updating the model as new observations arrive in the stream. One solution could be to retrain a new GraphDF model from scratch each time new values arrive, however, this process would require too much time for GraphDF to initialize new parameters and train from scratch, which would cause a waste of limited computing resources, especially when predicting with large-scale time-series data. To handle this, we propose an incremental online approach called Incremental Online GraphDF (IOGraphDF) that efficiently updates the current model, without the need to retrain it entirely from scratch with each newly arrived point. As a result, IOGraphDF operates much more efficiently.

Algorithm 3 Incremental Online Learning for GraphDF

1: Initialize the model parameters $\mathbf{\Phi}$ ($\mathbf{W}_*, \mathbf{b}_*, \mathbf{\Theta}_*$, etc.)
2: **while** new time-series values $\{(z_t^{(i)}, x_t^{(i)})\}_{i=1}^N$ arrive at time t in stream **do**
3: Add new values at time t and remove earliest for all N node time series
4: **for** each time series pair $(\mathbf{z}^{(i)}, \mathbf{x}^{(i)})$ **do**
5: With current model parameter estimates $\mathbf{\Phi}$
6: Calculate the relational fixed effect $c_t^{(i)} = \sum_{k=1}^K w_{i,k} \cdot S_{i,k,t}$ described in Sec. 6.3.3
7: Calculate the relational local random effect $b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2)$ described in Sec. 6.3.4
8: Compute marginal likelihood $\mathbb{P}(\mathbf{z}^{(i)}) = \sum_t -\frac{1}{2}\ln(2\pi\sigma_{i,t}) - \sum_t \frac{(z_t^{(i)} - c_{i,t})^2}{2\sigma_{i,t}^2}$ and accumulate the loss
9: **end for**
10: **end while**
11: Compute the overall loss in the current batch and perform SGD and update the trainable parameters $\mathbf{\Phi}$ accordingly.

Let t denote the moment dynamic streaming time series currently reach, we define the set of covariate time series as $\mathcal{X}_t = \{\mathbf{X}^{(i)}\}_{i=1}^N$ where $\mathbf{X}^{(i)} \in \mathbb{R}^{D \times t}$, with D the number of dimension for covariate features. We define the set of target time series as $\mathcal{Z}_t = \{\mathbf{z}^{(i)}\}_{i=1}^N$ where $\mathbf{z}_{1:t}^{(i)}$ denotes the i(th) univariate target time series. Given $\left(\mathcal{X}_t, \mathcal{Z}_t\right)$ at the moment t, our task is to give probabilistic predictions at the horizon for each target time series $\{\hat{\mathbf{z}}_{t+1}^{(i)}\}_{i=1}^N, \{\hat{\mathbf{z}}_{t+2}^{(i)}\}_{i=1}^N, \ldots, \{\hat{\mathbf{z}}_{t+\ldots}^{(i)}\}_{i=1}^N$. Hence the target function is modified accordingly,

$$\mathbb{P}\left(\left\{\mathbf{z}_{t+1:t+\tau}^{(i)}\right\}_{i=1}^N \,\middle|\, \mathbf{A}, \left\{\mathbf{z}_{1:t}^{(i)}, \mathbf{X}_{:,1:t+\tau}^{(i)}\right\}_{i=1}^N; \mathbf{\Phi}\right) \qquad t = 1, 2, \cdots \tag{6.34}$$

Therefore, every time new observations arrive, the new problem formulation is conditioned upon all available known values to make further predictions. The algorithm for IOGraphDF is described in Algorithm 3.

Now we analyze the time complexity of IOGraphDF, when a single value arrives at time t, the worst-case time complexity to generate a forecast will be:

$$\mathcal{O}\left((|E| \cdot KLkN + K) + LC_{\max}kN\right) \tag{6.35}$$

where $|E|$ is size of the edge set in the graph, K is the number of relational global factors, L is the order of the graph convolution. Noticeably, for incremental online GraphDF, we maintain only the most recent k values in a streaming window for each of N nodes. The time complexity of the relational global component can be decomposed into the computation for K relational global factors, and their linear combination (i.e., the K term in Eq. 6.35) which is timewise negligible, therefore, the time complexity of the relational global component $\mathcal{O}\left(|E| \cdot KLkN + K\right)$ is approximately linear to all the aforementioned values.

Further in the time complexity $\mathcal{O}(LC_{\max}kN)$ for the relational local component, $C_{\max}$ is the maximum node degree of the graph, and thus Eq. 6.35 is the worst case. However, in practice $NC_{\max} \gg \sum_{i=1}^{N} |C_i|$, where $|C_i|$ is the degree of the ith node. Notice that since the number of local iterations in the online model is a small fixed constant (e.g., 1, 10) for every new data point that arrives at time t, it can safely be omitted. In the multistep ahead prediction scenario, Eq. 6.35 is multiplied by a factor of future horizon τ, i.e., the time complexity is linear to τ. However, the offline model time complexity is significantly larger by a factor of epoch numbers M. In the offline case of GraphDF, we are given T time-series values for each N nodes, and need to relearn an entire model from scratch every time. Thus, the time complexity of GraphDF is significantly higher than that of IOGraphDF, $\mathcal{O}(M \cdot ((|E| \cdot KLTN + K) + LC_{\max}TN)) \gg \mathcal{O}((|E| \cdot KLkN + K) + LC_{\max}kN)$. It is important to note that T increases as a function of the stream size, hence, the offline model consumes much more computation time than the IOGraphDF model.

In terms of input data space requirements, the offline GraphDF requires $\mathcal{O}(TN)$ space while the incremental online GraphDF requires only $\mathcal{O}(kN)$ space where w is the most recent w values in the stream. Hence, since $k \ll T$, then $\mathcal{O}(kN) \ll \mathcal{O}(TN)$. Furthermore, as more data arrives over time, we can also see that the input data space of GraphDF can actually increase (assuming that it is trained using all available data). This is in contrast to the incremental online GraphDF that always uses a fixed amount of space, as when a new data point arrives for time t, we simply discard the most distant value and append the new value.

6.5 Experiments

In this section, we examine the performance of GraphDF models with previous state-of-the-art methods, then we evaluate the performance of IOGraphDF models against GraphDF models. Moreover, we investigate both models in the task of opportunistic scheduling. For GraphDF, the experiments are designed to investigate the following:

RQ1. Does GraphDF outperform state-of-the-art deep probabilistic forecasting methods?

RQ2. Are GraphDF models fast and scalable for large-scale time-series forecasting?

RQ3. Can GraphDF generate cloud usage forecasts to effectively perform opportunistic workload scheduling?

6.5.1 Experimental Setup

We used two real-world datasets in our experiments; Google trace data and Adobe trace data. Table 6.1 shows the statistics and properties (e.g. edge density, average degree).

Table 6.1: Statistics of the two real-world large-scale collections of time-series.

Data	$\|V\|$	$\|E\|$	Density	Avg. Deg.	Median Deg.	Mean wDeg.	Time-scale	T	Mean CPU usage	Median CPU usage
Google	12,580	1,196,658	0.0075	95.1	40	30.3	5 min	8,354	22.7%	21.4%
Adobe	3,270	221,984	0.0207	67.9	15	67.7	30 min	1,687	108.5%	9.1%

Google Trace. The Google trace dataset records the activities of a cluster of $12,580$ machines for 29 days since 19:00 EDT in May 1, 2011. The CPU and memory usage for each task are recorded every 5 minutes. The usage of tasks is aggregated to the usage of associated machines, resulting time series of length $8,354$.

Adobe Workload Trace. The Adobe trace dataset records the CPU and memory usage of $3,270$ nodes in the period from Oct. 31 to Dec. 5 in 2018. The timescale is 30 minutes, resulting time series of length $1,687$.

For the opportunistic workload scheduling case study in Sec. 6.6, we need to train a model *fast* within a few minutes and then forecast a single as well as multiple timesteps ahead, which are then used to make a decision on whether the current resources are enough or if we should instead scale up or down. Therefore, models must be able to be trained fast within a few minutes. To ensure rapid model training, we use 6 observations in the time-series data for training across all experiments. Furthermore, as in most time-series forecasting problems, the future CPU usage of machines is highly dependent on the most recent observations than those in the distant past. We set the number of embedding dimension as $K = 10$ in $\mathbf{w}_i \in \mathbb{R}^K$ and use time feature series as covariates. We set the embedding dimension to $K = 10$ in $\mathbf{w}_i \in \mathbb{R}^K$ and used $D = 5$ covariates for each time series. Similar to DF [191], the time features (e.g. minute of hour, hour of day) are used as covariates. We derive a fixed graph using Radial Basis Function (RBF) on the past observations.

The three models described in Section 6.3.6 are evaluated against four state-of-the-art probabilistic forecasting methods, including Deep Factors, DeepAR [162], MQRNN [192], and NBEATS [145]. *Deep Factors* is a generative approach that combines a global model and a local model. To ensure a fair comparison, we modified DF to solve the same problem formulated in Eq. 6.1, such that the DF variant for comparison uses the same inputs as GraphDF. Unless otherwise mentioned, we use the same experimental setup as mentioned in the DF paper. In particular, as suggested by the authors, we use the Gaussian likelihood in terms of the random effects in the deep factors model. We use 10 global factors with a LSTM cell of 1-layer and 50 hidden units in its global component, and 1-layer and 5 hidden units RNN in the local component. We also use suggested hyperparameters for other compared baselines. *DeepAR* is an RNN-based global model, we use a LSTM layer with 50 hidden units in DeepAR. *MQRNN* is a sequence model with quantile regression and NBEATS is an pure interpretable deep learning model. For MQRNN, we use a GRU bidirectional layer with 50 hidden units as an encoder and a modified forking layer in its decoder. For *N-BEATS*, we use an ensemble modification of the model and take the median value from 10 bagging

Table 6.2: Results for one-step ahead forecasting (P10QL, P50QL and P90QL).

Metrics	DATA	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
P10QL	**Google**	18.12 ± 201.26	0.190 ± 0.004	0.046 ± 0.000	0.083 ± 0.001	$\mathbf{0.037 \pm 0.000}$	0.038 ± 0.000	0.044 ± 0.000
	Adobe	0.615 ± 0.091	0.132 ± 0.000	0.164 ± 0.001	1.128 ± 0.004	$\mathbf{0.118 \pm 0.001}$	0.119 ± 0.000	1.027 ± 2.700
P50QL	**Google**	10.064 ± 62.12	0.172 ± 0.001	0.098 ± 0.001	0.239 ± 0.001	$\mathbf{0.072 \pm 0.000}$	0.076 ± 0.000	0.077 ± 0.000
	Adobe	3.070 ± 2.286	0.272 ± 0.001	0.619 ± 0.026	1.649 ± 0.001	$\mathbf{0.188 \pm 0.000}$	0.210 ± 0.001	0.746 ± 0.835
P90QL	**Google**	2.013 ± 2.485	0.106 ± 0.001	0.051 ± 0.000	0.144 ± 0.002	$\mathbf{0.041 \pm 0.000}$	0.044 ± 0.000	0.048 ± 0.000
	Adobe	5.524 ± 7.410	0.217 ± 0.000	0.949 ± 0.086	1.802 ± 0.002	$\mathbf{0.153 \pm 0.001}$	0.169 ± 0.001	0.342 ± 0.037

bases as results. All methods are implemented using MXNet Gluon [1, 30]. The Adam optimization method is used with a default initial learning rate as 0.001 to train all models. The training epochs are selected by grid search in $\{100, 200, \ldots, 1000\}$. An early stopping strategy is leveraged if weight losses do not decrease for 10 continuous epochs. We used a learning rate decay factor of 0.5 and a minimum learning rate of $5 * 10^{-5}$. We use Xavier as the weight initializer, and train for 500 epochs on Adobe data and 100 epochs for the Google dataset. Some hyperparameters are specific to our method: In GraphDF, we set the order $L = 2$ in Eq. 6.10. A small value for the order indicates that the model makes forecasts based more on neighboring nodes than those more distant. For other methods, we use default hyperparameters in the Gluonts implementation, unless otherwise specified.

To evaluate the probabilistic forecasts, we use the quantile loss defined as follows: given a quantile $\rho \in (0, 1)$, a target value $\mathbf{z}_t$ and ρ-quantile prediction $\widehat{\mathbf{z}}_t(\rho)$, the ρ-quantile loss is

$$\mathrm{QL}_\rho[\mathbf{z}_t, \widehat{\mathbf{z}}_t(\rho)] = 2\big[\rho(\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho))\mathbb{I}_{\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho) > 0} + (1-\rho)(\widehat{\mathbf{z}}_t(\rho) - \mathbf{z}_t)\mathbb{I}_{\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho) \leqslant 0}\big] \tag{6.36}$$

To derive quantile losses over a timespan across all time series, we use a normalized version of quantile loss $\sum_{i,t} \mathrm{QL}_\rho[z_{i,t}, \hat{z}_{i,t}(\rho)] / \sum_{i,t} |z_{i,t}|$. When $\rho = 0.5$, the resulted quantile loss is equivalent to Mean Absolute Percentage Error (MAPE). In our experiments, the quantile losses are computed based on 100 sample values. Our algorithms are implemented in MXNet Gluon [30], and all experiments run on a machine with 8 CPU cores.

6.5.2 Forecasting Performance

To answer the research questions, we investigate the proposed GraphDF framework with various forecast horizons, including $\tau = \{1, 3, 4, 5\}$.

The results for single and multistep ahead forecasting are provided in Table 6.2 and Table 6.3-6.5, respectively, where the best results for every dataset and forecast horizon are highlighted in bold. We run 10 trials for each model and report the average for $\rho = \{0.1, 0.5, 0.9\}$, denoted as the P10QL, P50QL and P90QL, respectively. In all cases, we observe that GraphDF models outperform state-of-the-art methods across all datasets and all forecast horizons. Furthermore, we observe that in most cases, the GraphDF-GG variant that uses

Table 6.3: Results for multistep ahead forecasting (P10QL).

DATA	τ	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
Google	3	0.652 ± 0.396	0.152 ± 0.006	0.070 ± 0.000	0.132 ± 0.004	$\mathbf{0.064 \pm 0.001}$	0.077 ± 0.000	0.087 ± 0.000
	4	0.260 ± 0.017	0.272 ± 0.018	0.138 ± 0.000	0.193 ± 0.016	$\mathbf{0.071 \pm 0.001}$	0.083 ± 0.000	0.089 ± 0.001
	5	0.447 ± 0.054	0.147 ± 0.005	0.484 ± 0.017	0.327 ± 0.036	$\mathbf{0.054 \pm 0.000}$	0.113 ± 0.001	0.088 ± 0.001
Adobe	3	0.811 ± 0.295	0.184 ± 0.003	0.207 ± 0.002	0.303 ± 0.006	$\mathbf{0.183 \pm 0.002}$	0.216 ± 0.008	0.267 ± 0.009
	4	0.985 ± 0.537	0.219 ± 0.008	0.273 ± 0.003	0.313 ± 0.018	$\mathbf{0.184 \pm 0.002}$	0.242 ± 0.014	0.423 ± 0.019
	5	0.626 ± 0.023	0.398 ± 0.229	0.402 ± 0.011	0.343 ± 0.047	$\mathbf{0.251 \pm 0.016}$	0.298 ± 0.031	0.544 ± 0.020

Table 6.4: Results for multistep ahead forecasting (P50QL).

DATA	τ	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
Google	3	0.741 ± 0.050	0.257 ± 0.011	0.148 ± 0.001	0.400 ± 0.004	$\mathbf{0.091 \pm 0.001}$	0.134 ± 0.002	0.097 ± 0.000
	4	0.618 ± 0.105	0.410 ± 0.017	0.191 ± 0.000	0.454 ± 0.007	$\mathbf{0.097 \pm 0.002}$	0.185 ± 0.002	0.109 ± 0.000
	5	0.485 ± 0.021	0.684 ± 0.012	0.466 ± 0.006	0.563 ± 0.017	0.128 ± 0.000	0.284 ± 0.012	$\mathbf{0.126 \pm 0.001}$
Adobe	3	1.683 ± 0.100	0.556 ± 0.028	0.592 ± 0.017	1.116 ± 0.006	$\mathbf{0.272 \pm 0.004}$	0.315 ± 0.006	0.319 ± 0.005
	4	1.424 ± 0.210	0.574 ± 0.011	0.629 ± 0.024	1.029 ± 0.001	$\mathbf{0.314 \pm 0.004}$	0.353 ± 0.007	0.405 ± 0.007
	5	1.069 ± 0.027	0.687 ± 0.064	0.633 ± 0.012	1.039 ± 0.004	$\mathbf{0.375 \pm 0.007}$	0.401 ± 0.014	0.484 ± 0.005

Table 6.5: Results for multistep ahead forecasting (P90QL).

DATA	τ	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
Google	3	0.830 ± 0.262	0.091 ± 0.001	0.067 ± 0.000	0.208 ± 0.002	$\mathbf{0.051 \pm 0.000}$	0.051 ± 0.000	0.089 ± 0.000
	4	0.976 ± 0.429	0.090 ± 0.000	0.070 ± 0.000	0.213 ± 0.002	$\mathbf{0.050 \pm 0.001}$	0.076 ± 0.001	0.095 ± 0.001
	5	0.523 ± 0.085	0.124 ± 0.000	0.134 ± 0.000	0.220 ± 0.002	$\mathbf{0.069 \pm 0.001}$	0.167 ± 0.013	0.094 ± 0.001
Adobe	3	2.556 ± 0.328	0.317 ± 0.002	0.751 ± 0.117	1.545 ± 0.008	$\mathbf{0.248 \pm 0.004}$	0.254 ± 0.003	0.301 ± 0.003
	4	1.862 ± 1.212	0.335 ± 0.004	0.696 ± 0.170	1.673 ± 0.008	$\mathbf{0.317 \pm 0.006}$	0.318 ± 0.009	0.482 ± 0.014
	5	1.512 ± 0.082	0.463 ± 0.003	0.546 ± 0.000	1.690 ± 0.015	$\mathbf{0.410 \pm 0.021}$	0.434 ± 0.007	0.512 ± 0.007

GCRN for both the relational global and relational local component outperforms the other variants. The second best model is GraphDF-GR, followed by Graph-RG.

To understand the overall performance and variance of the models, we show boxplots for each model in Fig. 6.3. Strikingly, we observe that the GraphDF models provide more accurate forecasts with significantly lower variance.

6.5.3 Runtime Analysis

Training and inference runtime performance results are shown in Table 6.6 and Table 6.7, respectively. All forecasting models are trained using only six previous values for each time series in the collection (Table 6.6). As expected, the relational global model is significantly faster and more scalable than the global model used in Deep Factors. In particular, we observe that the runtime of our GraphDF model that uses GCRN for the relational global model with RNN as the local model is significantly faster than Deep Factors that uses the same local model, but differs in the global model used. This is due to the fact that in

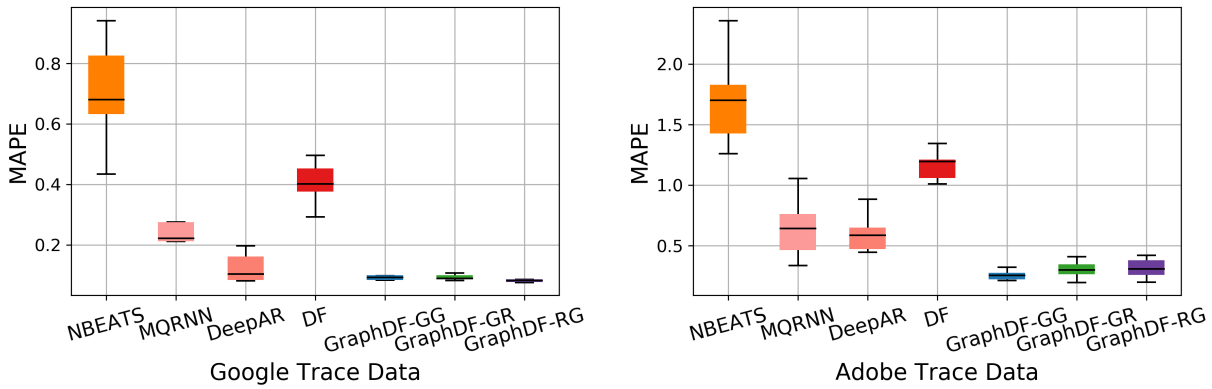


Figure 6.3: Probabilistic forecasting results for 3-step ahead forecast horizon (P50QL) from Adobe and Google trace dataset.

Table 6.6: Training runtime performance (in seconds).

DATA	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
Google	663.31 ± 54.09	284.76 ± 71.08	413.79 ± 49.62	315.06 ± 67.80	279.45 ± 41.19	$\mathbf{222.08 \pm 69.52}$	281.76 ± 49.51
Adobe	462.06 ± 120.07	393.08 ± 4.22	351.99 ± 285.30	378.97 ± 441.64	282.30 ± 36.80	$\mathbf{211.20 \pm 21.56}$	264.00 ± 56.29

Table 6.7: Inference runtime performance (in seconds).

DATA	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
Google	88.08 ± 10.96	9.22 ± 0.06	17.06 ± 0.16	8.28 ± 0.02	1.67 ± 0.03	$\mathbf{0.99 \pm 0.003}$	1.16 ± 0.003
Adobe	162.63 ± 7.59	2.69 ± 0.006	4.30 ± 0.02	2.12 ± 0.001	0.51 ± 0.005	$\mathbf{0.28 \pm 0.001}$	0.33 ± 0.000

the state-of-the-art DF model, all time series are considered equivalently and jointly when learning the K global factors. This can be thought of as a fully connected graph where each time series is connected to every other time series. In comparison, the relational global components of GraphDF leverage the graph that encodes explicit dependencies between the different time series, and therefore, does not need to leverage all pairwise time series, but only a smaller fraction of those that are actually related.

In terms of inference, all models are fast taking only a few seconds as shown in Table 6.7. For inference, we report the time (in seconds) to infer the next six values in each time series in the collection. In all cases, the GraphDF model variants are significantly faster than DF on both Google and Adobe workload datasets.

6.5.4 Scalability

To evaluate the scalability of GraphDF, we vary the training set size (i.e., the number of previous data points per time series to use) from $\{2, 4, 8, 16, 32\}$. Fig. 6.4 shows that

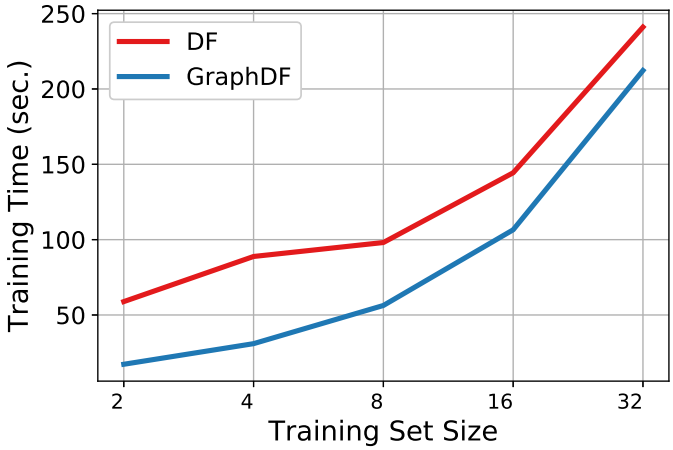


Figure 6.4: Comparing scalability of GraphDF to DF as the training set size increases for the Adobe workload trace dataset. The training set size refers to the number of data points per time series.

GraphDF scales nearly linear as the training set size increases from 2 to 32. For instance, GraphDF takes around 15 seconds to train using only 2 data points per time series and 30 seconds using 4 data points per time series, and so on. We also observe that for the Adobe data, GraphDF is always about 3× faster compared to DF across all training set sizes.

6.5.5 Experimental Result on IOGraphDF

To evaluate the performance of IOGraphDF, we design experiments to answer the following research questions:

RQ4. Does IOGraphDF yield more accurate predictions than GraphDF?

RQ5. Does IOGraphDF outperform GraphDF regarding training and prediction runtime?

RQ6. Does IOGraphDF perform better in the opportunistic workload scheduling task?

Experimental Setting. To compare the performance and runtime between IOGraphDF and GraphDF, we simulate an streaming procedure containing 100 timesteps, where new values are received by both models at each time step. At each time step new values arrive, a new GraphDF instance is initialized and then trained with the newly arrived values. In contrast, IOGraphDF model only needs to be initialized once in the beginning, and the same IOGraphDF instance is incrementally updated from the previous time step with newly arrived data. We assume that incoming values are more dependent on near observations than those in the distant past. Therefore, we maintain a shifting window size of 9 to include

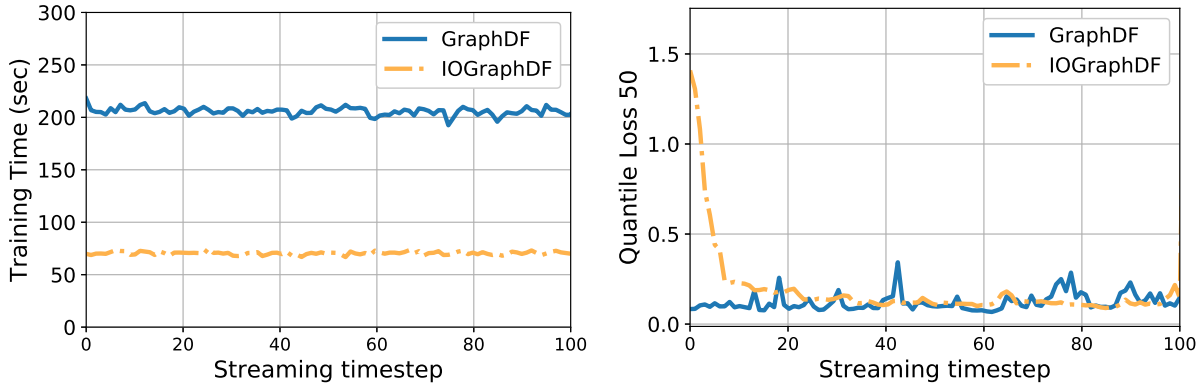


Figure 6.5: Comparing the training runtime and one step ahead P50QL (i.e., MAPE) between GraphDF (blue) and IOGraphDF (yellow dashed line) on the Google dataset.

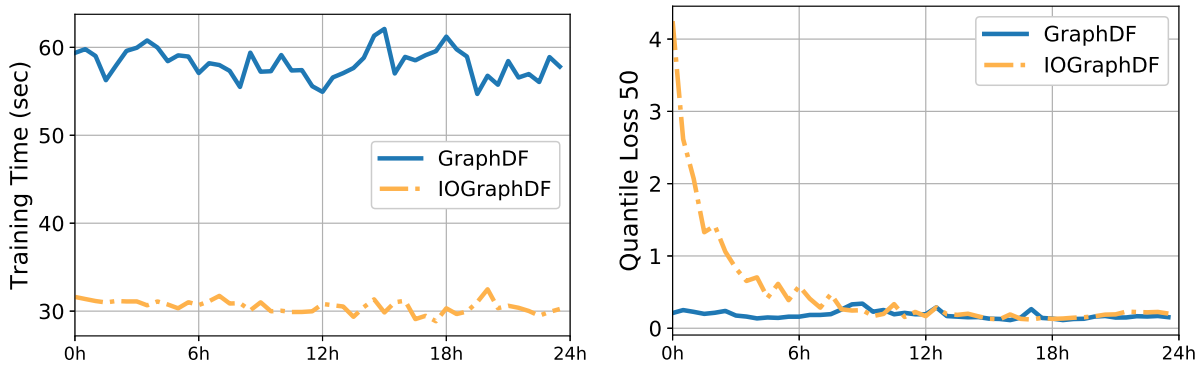


Figure 6.6: Comparing the training runtime and one step ahead P50QL between GraphDF (blue) and IOGraphDF (yellow dashed line) on the Adobe dataset.

the most recent observations relative to t. As t increments, the oldest values are removed from the window, and newly arrived values are appended to the window. The window values are split for data training and fitting procedure. In the case of the GraphDF model using the Google dataset, the number of training epochs is set as 100. Conversely, for the IOGraphDF model, the number of local iterations (i.e., the number of training iterations upon each shifting window) only needs to be set as a smaller number, 50. This is because after the model is initialized, it preserves the learned information from previous time steps. In the inference stage, values in $\tau = 3$ steps ahead are predicted and evaluated.

Experimental Results. We observe from Fig. 6.5 (Left) and Table 6.10 that IOGraphDF is significantly faster. As observed in Fig. 6.5 (Right), the P50QL of GraphDF and IOGraphDF for each 100 streaming timestep is plotted. In the first few timesteps, IOGraphDF has much higher errors than GraphDF, however, IOGraphDF performance converges quickly to that of GraphDF's, as two lines in Fig. 6.5 are mostly overlapping after streaming timestep 20. Hence, IOGraphDF sacrifices a small amount of accuracy for a significant speedup.

Table 6.8: Runtime comparison between GraphDF and IOGraphDF over timesteps

Data	GraphDF	IOGraphDF
Google	208.770 ± 11.862	$\mathbf{76.948 \pm 22.717}$
Adobe	58.170 ± 1.666	$\mathbf{30.534 \pm 0.718}$

Table 6.9: Results for multistep ahead forecasting (P50QL) with IOGraphDF.

DATA	τ	GraphDF-GG	GraphDF-GR	GraphDF-RG	IOGraphDF-GG	IOGraphDF-GR	IOGraphDF-RG
Google	3	$\mathbf{0.091 \pm 0.001}$	0.134 ± 0.002	0.097 ± 0.000	0.104 ± 0.006	0.146 ± 0.007	0.141 ± 0.013
	4	$\mathbf{0.097 \pm 0.002}$	0.185 ± 0.002	0.109 ± 0.000	0.108 ± 0.005	0.146 ± 0.005	0.156 ± 0.005
	5	0.128 ± 0.000	0.284 ± 0.012	0.126 ± 0.001	$\mathbf{0.122 \pm 0.006}$	0.164 ± 0.010	0.171 ± 0.010
Adobe	3	0.272 ± 0.004	0.315 ± 0.006	0.319 ± 0.005	$\mathbf{0.211 \pm 0.023}$	0.328 ± 0.039	0.371 ± 0.042
	4	0.314 ± 0.004	0.353 ± 0.007	0.405 ± 0.007	$\mathbf{0.240 \pm 0.018}$	0.332 ± 0.057	0.393 ± 0.056
	5	0.375 ± 0.007	0.401 ± 0.014	0.484 ± 0.005	$\mathbf{0.286 \pm 0.023}$	0.341 ± 0.083	0.404 ± 0.059

Table 6.10: Results for multistep ahead forecasting (RUNTIME) with IOGraphDF.

DATA	τ	GraphDF-GG	GraphDF-GR	GraphDF-RG	IOGraphDF-GG	IOGraphDF-GR	IOGraphDF-RG
Google	3	279.45 ± 41.19	222.08 ± 69.51	281.76 ± 49.51	70.209 ± 0.844	$\mathbf{51.213 \pm 1.380}$	68.237 ± 1.477
	4	282.67 ± 12.33	222.31 ± 14.31	283.54 ± 26.26	70.325 ± 1.227	$\mathbf{51.481 \pm 1.222}$	68.340 ± 0.729
	5	283.69 ± 15.43	223.53 ± 24.54	289.16 ± 32.79	70.839 ± 1.351	$\mathbf{51.733 \pm 1.465}$	68.807 ± 1.080
Adobe	3	282.30 ± 36.80	211.20 ± 21.36	264.00 ± 56.29	27.69 ± 1.17	$\mathbf{20.20 \pm 0.82}$	25.19 ± 0.55
	4	282.80 ± 26.03	212.60 ± 14.00	264.90 ± 60.90	26.69 ± 0.63	$\mathbf{20.49 \pm 1.17}$	25.22 ± 0.49
	5	287.30 ± 12.03	214.81 ± 21.93	274.78 ± 46.10	26.86 ± 0.82	$\mathbf{20.78 \pm 0.73}$	25.24 ± 0.68

In both offline and online models, the expected runtime on training is positively related to the number of input and output values. In the offline GraphDF model, each time new values arrive, a new model has to be created and trained using these new values. Only recent values are taken as input. Hence, this results in the loss of earlier observations. In our proposed IOGraphDF, the input is only modeled upon a fixed amount of recent values. Since the same model instance is used and kept updated over time, IOGraphDF has the advantage of leveraging earlier observations for forecasting, as learned by the model parameters. As a consequence, we set the local iterations, the number of times data values are used to update IOGraphDF model, to a small number 50, which results in a much faster IOGraphDF while still preserving high accuracy. The runtime comparison between GraphDF and IOGraphDF over 100 timesteps is shown in Fig. 6.5 (Left). Similar result can be observed from the experiment using Adobe dataset using the same warm start period (20 steps i.e. 10 hours), as shown in Fig. 6.6. We also summarize average and deviation runtime in Table 6.8.

Following the setup in Sec. 6.5.2, we designed experiments to investigate the performance of IOGraphDF model variants over multistep ahead prediction, and compare them against previous results from GraphDF variants. Since IOGraphDF models take advantages of incremental training, for a fair comparison, we report the result for IOGraphDF models after a *warm start* period (set as 20 timesteps). The values in the warm start period are

Table 6.11: Results on combination of hyperparameters for 3-step ahead forecasting (p50QL) with the Google dataset. The best performance for each number of warm start period (row) is highlighted in bold.

		Number of relational global factors K			
		5	10	20	50
Number of warm start period	5	1.121 ± 0.124	0.804 ± 0.105	0.746 ± 0.058	$\mathbf{0.453 \pm 0.048}$
	10	0.321 ± 0.015	0.305 ± 0.011	0.295 ± 0.010	$\mathbf{0.281 \pm 0.009}$
	20	0.124 ± 0.012	$\mathbf{0.104 \pm 0.006}$	0.108 ± 0.009	0.126 ± 0.008
	50	0.102 ± 0.005	$\mathbf{0.009 \pm 0.003}$	0.105 ± 0.007	0.107 ± 0.003

used to train the IOGraphDF models, but not used for loss comparison. Subsequently, the values after the warm period are then used to evaluate and compare prediction loss with the GraphDF variants. The result is shown in Table 6.9. We observe that on the Google dataset, IOGraphDF variants reach very close results to the GraphDF counterparts, while IOGraphDF models require significantly less training runtime than GraphDF models, as shown in Table 6.10. For the Adobe dataset, IOGraphDF models not only require less runtime, but also obtain more accurate predictions with smaller losses in most cases (highlighted in the column IOGraphDF-GG).

6.5.6 Ablation Study

We further investigate the effects of hyperparameters on the IOGraphDF model through extensive experiments. The length of the warm start period and the number of relational global factors K are selected from the range of $\{5, 10, 20, 50\}$. We report p50QL for forecasting 3-step ahead on the Google dataset with combinations of these hyperparameters, and the result is shown in Table 6.11. From the result table, we observe (1) When the number of relational global factors K is fixed (i.e., for each column of Table 6.11), IOGraphDF achieves better performance as the warm start period increases. (2) When the number of warm start period is small, the performance drastically improves with increasing K, as shown in the first two rows of Table 6.11, with the best performance observed at $K = 50$. We suggest this can be due to the insufficient training of the model, since the number of warm start period is small. (3) When the number of warm start period is large, the performance improves little or even worsens as the number of relational global factors K increases, as shown in the last two rows of Table 6.11. This might be attributed to the excessive amount of model parameters which leads to overfitting.

Algorithm 4 Opportunistic Scheduling

Initialize hyperparameters and variables lookback window for GraphDF (or streaming window for IOGraphDF) w, horizon τ, threshold ratio ϵ, portion ratio λ. Accumulated utilization improvement Acc
while New observations arrive $t \leftarrow t+1$ **do**
  Initialize weights $\mathbf{\Phi}$ for new model f_t
  $f_t \leftarrow \mathbb{P}\left(\cdot \mid \mathbf{\Phi}, \mathbf{A}, \left\{\mathbf{X}^{(i)}_{:,t-w+1:t+\tau}, \mathbf{z}^{(i)}_{t-w+1:t}\right\}_{i=1}^{N}\right)$
  for each node i **do**
    Obtain forecasts $\hat{z}^{(i)}=\{\hat{z}^{(i)}_{t+1}, \hat{z}^{(i)}_{t+2} \dots \hat{z}^{(i)}_{t+\tau}\} \sim f_t$
    if MEAN of forecasts $\text{MEAN}(\hat{z}^{(i)}) \leq \epsilon$ **then**
      Utilization $Acc \leftarrow Acc + \lambda(1 - \text{MEAN}(\hat{z}^{(i)}))$
    else
      Cancel assigned tasks on node i
    end if
  end for
end while

6.6 Case Study: Opportunistic Scheduling

We leverage our GraphDF forecasting model to enable opportunistic scheduling of batch workloads. Since batch workloads, such as ML training, crawling web pages etc. have loose latency requirements, they can be opportunistically scheduled on underutilized resources, such as CPU cores. This improves resource utilization of the cluster and reduces operating costs by precluding the need to allocate additional machines to run the batch workloads. Our model generates probabilistic CPU usage forecasts for cloud nodes and nodes with low predicted usage are employed for scheduling these workloads.

Our model satisfies the following requirements of such an opportunistic scheduler. First, the model must be able to correctly forecast utilization. If the utilization is underestimated, tasks will be assigned to busy machines and may need to be canceled, which is a waste of resources. Second, the execution time of the forecasting model must be significantly faster than the time period used for data collection. For example, since CPU usage in the Google dataset is observed every 5 minutes, the CPU usage forecast should be generated in less than 5 minutes i.e. before the next observation arrives. We simulate opportunistic scheduling by developing two main components, the *forecaster* and the *scheduler*. The Google dataset provides CPU usages for the cluster in this study. The forecaster reads the 6 most recent observed CPU utilization values of each machine from the data stream and predicts next 3 values. The scheduler identifies underutilized machines as those with mean predicted utilization across the three predictions lower than a predefined threshold ϵ (25%). To safely make use of the idle resources without disturbing already running tasks or cause thrashing, the scheduler only assigns workloads that require at most 75% of compute resources. If a machine is

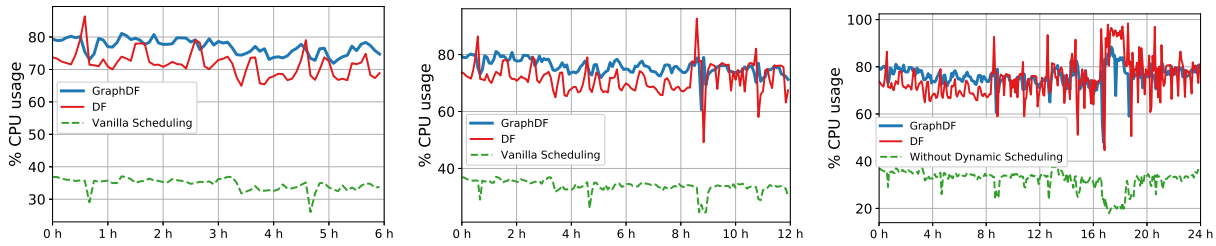


Figure 6.7: CPU utilizations without opportunistic scheduling (green) and with scheduling based on each forecaster (red and blue), on a period of {6, 12, 24} hours on the Google dataset. GraphDF scheduling has higher CPU utilizations than DF and vanilla scheduling.

Table 6.12: Opportunistic scheduling results using different forecasting models.

Data	Model	utilization improvement (%)	correct ratio (%)	cancellation ratio (%)
Google	DF	38.8	68.6	20.9
	GraphDF	**41.9**	**88.6**	**8.2**
Adobe	DF	42.0	65.8	19.1
	GraphDF	**53.2**	**97.0**	**2.2**

assigned batch workloads that exceeds resource availability, they are *terminated/canceled*. This procedure is described in Algorithm 4.

Effects on CPU utilizations. Fig. 6.7 shows the CPU utilizations without opportunistic workload scheduling *(vanilla)* and with scheduling based on each forecaster over a period of {6, 12, 24} hours on the Google dataset. We observe that the GraphDF-based forecaster consistently outperforms both vanilla and DF-based versions by generating forecasts with higher accuracy. Table 6.12 summarizes the performance of the GraphDF-based forecaster with respect to three metrics *CPU utilization improvement*, *correct scheduling ratio*, and *cancellation ratio*. The utilization improvement measures the absolute increase in CPU usage compared to the vanilla version. The correct scheduling ratio corresponds to the ratio when predicted utilization by the scheduler matches the actual utilization. The cancellation ratio measures the fraction of machines on which the batch workload was terminated due to incorrectly generated forecasts. We observe that GraphDF-based workload scheduling leads to higher CPU utilizations, higher correct scheduling ratio, and lower cancellation ratio compared to DF-based scheduling.

Execution Time Comparison. Fig. 6.8 shows that the runtime of DF-based scheduling often exceeds the 5-minute time limit, while GraphDF-based version is much faster and always meets it. The average time of prediction by DF is 340.43 ± 77.95 where the average time of prediction by GraphDF is 231.90 ± 4.25. Hence, GraphDF persuasively provides a solution for enhancing cloud efficiency through effective usage forecasting. Google dataset receive observations every 5 minutes, therefore, DF will fail as it is too slow.

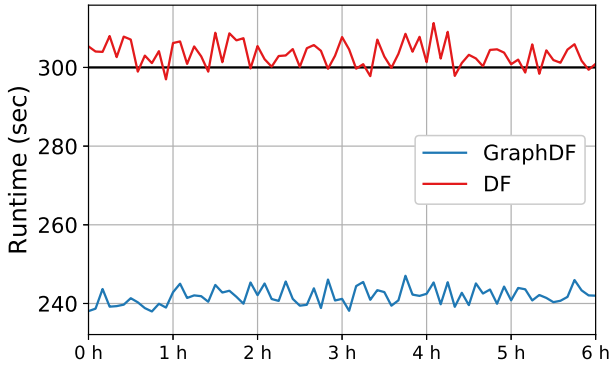


Figure 6.8: The time constraint (black line), the runtime of scheduler with DF (red) and that with GraphDF (blue). Note that in most cases, DF fails to meet the time constraint while GraphDF produces a much faster forecast.

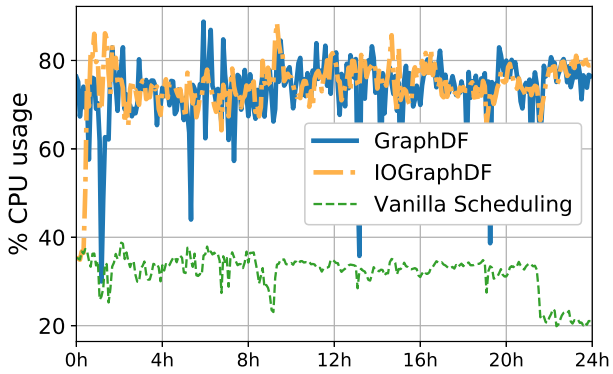


Figure 6.9: CPU utilization without opportunistic workload scheduling (shown in green) and with scheduling based on each forecaster (shown in blue and yellow), over a period of 6 hours on Google dataset.

Opportunistic Scheduling with IOGraphDF. Using the same parameter setup as in previous section, we further conducted opportunistic scheduling with IOGraphDF model, as shown in Fig. 6.9. We observe that scheduling based on IOGraphDF performs worse than GraphDF in the beginning, which may be attributed to the insufficient training of the IOGraphDF model. However, over time IOGraphDF gives performance close to that of the GraphDF model. Additionally, IOGraphDF delivers smoother scheduling curve than the GraphDF model. Since IOGraphDF delivers more accurate prediction over time, the opportunistic scheduling with IOGraphDF also outperforms scheduling with GraphDF model with respect to correct ratio and cancellation ratio, as shown in Table 6.13.

Table 6.13: Results for opportunistic scheduling in the cloud over a 24 hour period using different forecasting models.

Data	Model	utilization improvement (%)	correct ratio (%)	cancellation ratio (%)
Google	GraphDF	41.9	88.0	4.6
	IOGraphDF	**42.7**	**90.1**	**2.9**
Adobe	GraphDF	**56.3**	97.3	0.80
	IOGraphDF	54.3	**98.5**	**0.44**

6.7 Conclusion

In this work, we introduced a deep graph-based probabilistic forecasting framework called Graph Deep Factors. While existing deep probabilistic forecasting approaches do not explicitly leverage a graph, and assume either complete independence among time series (i.e., completely disconnected graph) or complete dependence between all time series (i.e., fully connected graph), this work moved beyond these two extreme cases by allowing nodes and their time series to have arbitrary and explicit weighted dependencies among each other. Such explicit and arbitrary weighted dependencies between nodes and their time series are modeled as a graph in the proposed framework. Notably, GraphDF consists of a relational global component that learns complex nonlinear time-series patterns globally using the structure of the graph to improve both computational efficiency and performance as well as a relational local model that not only considers its individual time series but the time series of nodes that are connected in the graph. The experiments demonstrated the effectiveness of the proposed deep graph-based probabilistic forecasting model in terms of its forecasting performance, runtime, and data efficiency. Further, to address the common streaming nature of many time-series prediction applications where values arrive over timesteps, we propose the incremental online GraphDF model (IOGraphDF). Experiments show that IOGraphDF outperforms GraphDF regarding forecasting accuracy and runtime.

Chapter 7

Conclusion

In this chapter, we summarize the contributions of this dissertation and give a list of resulted publications. We further discuss future directions for graph time-series models.

7.1 Contributions

The main contributions and task statuses of this dissertation are summarized in Table 7.1.

7.1.1 A Comprehensive Review of Graph Time-series Models

- **Proposal of the notion of graph time-series models (A1)** The notion of graph time series is introduced and defined, which serves as a fundamental data form for various tasks such as classification, forecasting, and anomaly detection.
- **Novel categorization of graph time-series models (A2)** Recent graph time-series models are categorized into Graph Recurrent/Convolutional Neural Networks and Graph Attention Neural Networks. Model analyses are given in details in terms of temporal modeling, graph modeling, and representational modeling.
- **Extensive performance comparison on various models and applications (A3)** Experimental performance of more than ten models is compared on two traffic forecasting datasets. Anomaly detection performance is also compared on water treatment datasets. Analyses of performance are also provided.

7.1.2 Context Integrated Graph Neural Network for Time-series Forecasting

- **A novel graph time-series forecasting model that integrates multiple contextual graphs (B1)** For the task of mobile call demand forecasting and bike supply forecasting, a context integrated model, CIGNN, has been proposed to leverage contextual information such as temperature and humidity records. CIGNN jointly learns from the target time-series and contextual time-series to forecast for target time-series.

Table 7.1: Research tasks and status

Tasks	Description	Status
Research Area A	A Comprehensive Review of Graph Time-series Models	Completed
A1	Introduction of the notion Graph Time Series	Completed
A2	Categorization and analysis of graph time-series models	Completed
A3	Performance comparison on forecasting and anomaly detection	Completed
Research Area B	Context Integrated Graph Neural Network for Forecasting	Completed
B1	Proposal of context integrated graph time-series model	Completed
B2	Proposal of leveraging various dependencies	Completed
B3	Validation on two real world datasets	Completed
Research Area C	Evolving Super Graph Neural Network for Forecasting	Completed
C1	Proposal of evolving graph time-series model	Completed
C2	Proposal of using K-Means and LSH for graph construction	Completed
C3	Validation on two large-scale datasets	Completed
Research Area D	Probabilistic Hypergraph Model for Forecasting	Completed
D1	Proposal of probabilistic hypergraph time-series model	Completed
D2	Proposal of leveraging heuristic KNN hypergraphs	Completed
D3	Validation on three real world datasets	Completed
Research Area E	Graph Deep Factor for Probabilistic Forecasting	Completed
E1	Development a framework of probabilistic forecasting model	Completed
E2	Validation on two cloud resource usage datasets	Completed
E3	Simulation with opportunistic scheduling	Completed
E4	Analysis of time complexity and scalability	Completed
E5	Experiment extension to forecast on streaming values	Completed
F	Dissertation Writing and Revision	Completed

- **Introduction and definition of various data dependencies (B2)** CIGNN models the complex dependency between data values by considering the temporal dependency, spatial dependency, relational dependency, and contextual dependency. The Detrended Cross-Correlation Analysis coefficient (DCCA coefficient) is novelly used for relational dependency modeling when geographic information is unavailable.

- **Evaluation and analysis on two real world datasets (B3)** CIGNN is tested on the mobile call dataset and the bike supply dataset to validate its effectiveness in forecasting accuracy. Different combinations of graph constructions are also tested to validate their effectiveness.

7.1.3 Evolving Super Graph Neural Network for Time-series Forecasting

- **A novel graph time-series forecasting model that generates multiple super graphs (C1)** To predict for large-scale time series, a evolving graph model, ESGNN, has been proposed to construct a series of super graph to quickly make forecasts while leveraging mutual influence between node time series.
- **An efficient super graph construction method (C2)** K-Means and Locality-Sensitive Hashing (LSH) are novelly applied to quickly generate super nodes and connect them with super edges. Super graphs are created per a user-selected time interval.
- **Evaluation and validation on large-scale datasets (C3)** Forecasting results are obtained on two large-scale datasets. Runtime and space usage of ESGNN are studied and validated.

7.1.4 Probabilistic Hypergraph Recurrent Neural Network for Time-series Forecasting

- **A novel probabilistic hypergraph model that learns a hypergraph structure from time series (D1)** To consider the situations where time series interact in a simultaneous broadcasting manner, a hypergraph model, PHRNN, has been proposed to model node connections with a probabilistic hypergraph.
- **Dual utilization of KNN hypergraphs (D2)** A KNN-based hypergraph is constructed to initiate hyperedge embeddings for probabilistic hypergraph learning, and is also used for learning regularization.
- **Evaluation on three real-world datasets (D3)** PHRNN is evaluated on cloud resource usage datasets and traffic dataset. Its effectiveness is validated on various sets of hyperparameters.

7.1.5 Graph Deep Factor for Cloud Resource Usage Forecasting

- **Development of a novel probabilistic forecasting framework (E1)** A probabilistic forecasting framework for cloud resource usage forecasting, GraphDF, has been proposed, which consists of a relational global model and a relational local model. The GraphDF framework is data-driven, fast, scalable for real-time demand forecasting, and highly data efficient.
- **Evaluation and validation on two cloud resource usage datasets (E2)** Extensive experiments have been conducted to evaluate GraphDF with various components in the relational global model and the relational local model.

Table 7.2: A summary of model characteristics

Models	Graph Structure	Number of Graphs	Forecasting	Data Scale
CIGNN	graphs	multiple	point, offline	small
ESGNN	super graphs	multiple	point, offline	large-scale
PHRNN	hypergraphs	single	point, offline	small
GRAPHDF	graphs	dual	probabilistic, offline	large-scale
IOGRAPHDF	graphs	dual	probabilistic, online	large-scale

- **Realistic simulation of opportunistic scheduling (E3)** Extensive experiments have been conducted to evaluate GraphDF for the task of opportunistic scheduling, to validate how much more resources will be utilized with the method.

- **Analysis of time complexity and scalability (E4)** The time complexity of GraphDF is studied. Extensive experiments have been conducted to evaluate its scalability.

- **Extension to accommodate streaming values (E5)** The GraphDF model has been extended to introduce a new model IOGraphDF that targets streaming values. Extensive experiments and analyses have been conducted with IOGraphDF.

7.2 Relations between Works

This dissertation is more than a simple union of several models. Instead, different chapters are closely related to each other. Chapter 2 introduces and defines the concept of graph time-series models. It serves as the cornerstone of the following models. Chapter 3-Chapter 6 introduce various graph and hypergraph time-series forecasting models, targeting different research problems and challenges. From the perspective of time-series modeling, all models utilize a RNN model with an encoder-decoder. From the perspective of graph modeling, CIGNN and ESGNN from Chapter 3 and Chapter 4 operate on a series of graphs, PHRNN in Chapter 5 introduces a probabilistic hypergraph model, and GraphDF and IOGraphDF in Chapter 6 leverages two separate graphs for different components in probabilistic forecasting. In terms of graph construction, CIGNN builds upon both spatial graphs and temporal graphs; ESGNN, GraphDF, and IOGraphDF are based on temporal graphs; PHRNN learns a semantic hypergraph from node embedding. From the perspective of data characteristics, CIGNN utilizes contextual data; ESGNN, GraphDF, and IOGraphDF target large-scale data that have tens of thousands of node time series; PHRNN targets small datasets. Overall, Chapter 3-Chapter 6 collectively cover a large range of different graph construction methods, data characteristics, and forecasting manners.

7.3 Publications

7.3.1 Journal Papers

1. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. 'Graph Deep Factors for Probabilistic Time-series Forecasting.' ACM Transactions on Knowledge Discovery from Data 17, no. 2 (2023): 1-30.
2. Chen, Hongjie, and Hoda Eldardiry. 'Graph Time-series Modeling in Deep Learning: A Survey.' ACM Transactions on Knowledge Discovery from Data (2023).

7.3.2 Conference Papers

1. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. 'Graph deep factors for forecasting with applications to cloud resource allocation.' In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD), pp. 106-116. 2021.
2. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, and Hoda Eldardiry. 'Context Integrated Relational Spatio-Temporal Resource Forecasting.' In 2021 IEEE International Conference on Big Data (Big Data), pp. 1359-1368. IEEE, 2021.
3. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, and Hoda Eldardiry. 'Hypergraph Neural Networks for Time-series Forecasting.' In 2023 IEEE International Conference on Big Data (Big Data), 2023.
4. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. 'Evolving Super Graph Neural Networks.' In The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2024.

7.3.3 Submitted Papers

1. Chen, Hongjie, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. 'Probabilistic Hypergraph Recurrent Neural Network for Time-series Forecasting' is submitted to The ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD), 2024

7.4 Future Directions

Time series and graph are evergreen research areas, and there remain many challenges to be solved. Chapter 2 list some future directions for graph time-series models. This section

presents additional future directions in terms of learning approaches, task objectives, and data characteristics.

Unified Learning of Time Series Most time-series models learns from one source of dataset (e.g., traffic networks) and are used on the same source. However, there are situations where the target dataset has limited data points while abundant data points are available from other sources. To leverage datasets outside of target domain, numerous methods have been proposed in the areas of transfer learning and domain adaptation. These models mainly learn from one source domain and are tested on one target domain. Recent successes in large language models may inspire research ideas in learning from large collections of time-series datasets, resulting in a large general time-series model. For example, a general large time-series prediction model may be learned to enable users to achieve satisfied accuracy with a few additional customized time-series data provided by users. Therefore, one future direction is to incorporate various data sources and learn a unified model for time-series tasks.

Various Graph Structures This dissertation has introduced models that cover both graph structures and hypergraph structures. Moreover, other graph structure types also exists, including homogeneous graphs and heterogeneous graphs, homophilous graphs and heterophilous graphs, and bipartite graphs, among others. Different graph structures should be considered according to actual objectives as well as the involved datasets.

Incorporating Irregular Data Future directions should also take irregular data characteristics into account. For example, in medical and neurological datasets, time series are sparse with a few sudden spikes when events occur. These require more efficient graph representations. Another example of irregular data characteristics lie in the domain of time series, such as acoustic series where data are recorded in waveforms. Hence, the graph modeling of acoustic series need to consider relations between series in the waveform.

Bibliography

[1] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner Türkmen, and Yuyang Wang. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116):1–6, 2020. URL http://jmlr.org/papers/v21/19-820.html.

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2020.

[3] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. Online learning for time series prediction. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 172–184, Princeton, NJ, USA, 12–14 Jun 2013. PMLR. URL https://proceedings.mlr.press/v30/Anava13.html.

[4] Huthaifa I. Ashqar, Mohammed Elhenawy, and Hesham A. Rakha. Modeling bike counts in a bike-sharing system considering the effect of weather conditions. *Case Studies on Transport Policy*, 2019. ISSN 2213-624X.

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

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *arXiv preprint arXiv:2007.02842*, 2020.

[8] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018. URL http://arxiv.org/abs/1803.01271.

[9] Sandy D Balkin and J Keith Ord. Automatic neural network modeling for univariate time series. *International Journal of Forecasting*, 16(4):509–515, 2000.

[10] Kasun Bandara, Christoph Bergmeir, and Hansika Hewamalage. Lstm-msnet: Leveraging forecasts on sets of related time series with multiple seasonal patterns. *IEEE*

Transactions on Neural Networks and Learning Systems, 32(4):1586–1599, 2021. doi: 10.1109/TNNLS.2020.2985720.

[11] Gianni Barlacchi, Marco De Nadai, Roberto Larcher, Antonio Casella, Cristiana Chitic, Giovanni Torrisi, Fabrizio Antonelli, Alessandro Vespignani, Alex Pentland, and Bruno Lepri. A multi-source dataset of urban life in the city of milan and the province of trentino. *Scientific Data*, 2, 2015.

[12] Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37, 2021.

[13] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle C. Maddix, Ali Caner Türkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, Laurent Callot, and Tim Januschowski. Neural forecasting: Introduction and literature overview. *ArXiv*, abs/2004.10240, 2020.

[14] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, François-Xavier Aubet, Laurent Callot, and Tim Januschowski. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi: 10.1145/3533382. URL https://doi.org/10.1145/3533382.

[15] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. Toward ml-centric cloud platforms. *Commun. ACM*, 63(2):50–59, January 2020. ISSN 0001-0782.

[16] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut, editors, *Advances in Intelligent Data Analysis VIII*, pages 249–260, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03915-7.

[17] Mikolaj Binkowski, Gautier Marti, and Philippe Donnat. Autoregressive convolutional neural networks for asynchronous time series. In *International Conference on Machine Learning*, pages 580–589. PMLR, 2018.

[18] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)*, 54 (3):1–33, 2021.

[19] Paul Boniol and Themis Palpanas. Series2graph: Graph-based subsequence anomaly detection for time series. *Proc. VLDB Endow.*, 13(12):1821–1834, jul 2020. ISSN 2150-8097. doi: 10.14778/3407790.3407792. URL https://doi.org/10.14778/3407790.3407792.

[20] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *European business intelligence summer school*, pages 62–77. Springer, 2012.

[21] Sofiane Brahim-Belhouari and Amine Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, 2004.

[22] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE transactions on cloud computing*, 3(4):449–458, 2014.

[23] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

[24] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[25] Chris Chatfield. *Time-series forecasting*. Chapman and Hall/CRC, 2000.

[26] Hongjie Chen, Ryan A Rossi, Kanak Mahadik, and Hoda Eldardiry. Context integrated relational spatio-temporal resource forecasting. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1359–1368. IEEE, 2021.

[27] Hongjie Chen, Ryan A Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. Graph deep factors for forecasting with applications to cloud resource allocation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 106–116, 2021.

[28] Hongjie Chen, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. Hypergraph neural networks for time-series forecasting. In *2023 IEEE International Conference on Big Data (BigData)*, pages 1076–1080, 2023. doi: 10.1109/BigData59044.2023.10386109.

[29] Hongjie Chen, Ryan A Rossi, Kanak Mahadik, Sungchul Kim, and Hoda Eldardiry. Graph deep factors for probabilistic time-series forecasting. *ACM Transactions on Knowledge Discovery from Data*, 17(2):1–30, 2023.

[30] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1512.html#ChenLLLWWXXZZ15.

[31] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. *arXiv preprint arXiv:1911.12093*, 2019.

[32] Wenchao Chen, Long Tian, Bo Chen, Liang Dai, Zhibin Duan, and Mingyuan Zhou. Deep variational graph convolutional recurrent network for multivariate time series anomaly detection. In *International Conference on Machine Learning*, pages 3621–3633. PMLR, 2022.

[33] Xu Chen, Junshan Wang, and Kunqing Xie. Trafficstream: A streaming traffic flow forecasting framework based on graph neural networks and continual learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 3620–3626. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/498. URL https://doi.org/10.24963/ijcai.2021/498. Main Track.

[34] Yuzhou Chen, Ignacio Segovia, and Yulia R. Gel. Z-gcnets: Time zigzags at graph convolutional networks for time series forecasting. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1684–1694. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/chen21o.html.

[35] Yuming Cheng, Chao Wang, Huihuang Yu, Yahui Hu, and Xuehai Zhou. Gru-es: Resource usage prediction of cloud workloads using a novel hybrid method. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1249–1256. IEEE, 2019.

[36] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[37] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[38] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[39] Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the g_ap_s: Multivariate time series imputation by graph neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=kOu3-S3wJ7.

[40] Jonathan Crabbé and Mihaela Van Der Schaar. Explaining time series predictions with dynamic masks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2166–2177. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/crabbe21a.html.

[41] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *NSDI*, pages 613–627, 2017.

[42] Michael J Crawley. Mixed-effects models. the r book second edition, 2013.

[43] Csaba Csiszár, Bálint Csonka, Dávid Földes, Ervin Wirth, and Tamás Lovas. Urban public charging station locating method for electric vehicles based on land use approach. *Journal of Transport Geography*, 2019.

[44] Marco Cuturi and Mathieu Blondel. Soft-DTW: a differentiable loss function for time-series. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 894–903. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/cuturi17a.html.

[45] Enyan Dai and Jie Chen. Graph-augmented normalizing flows for anomaly detection of multiple time series. *arXiv preprint arXiv:2202.07857*, 2022.

[46] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[47] Miguel Ramos de Araujo, Pedro Manuel Pinto Ribeiro, and Christos Faloutsos. Tensorcast: Forecasting with context using coupled tensors (best paper award). *ICDM*, pages 71–80, 2017.

[48] Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews*, 74:902–924, 2017.

[49] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, volume 29, pages 3844–3852, 2016.

[50] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 127–144, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450323055.

[51] Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4027–4035, 2021.

[52] Jinliang Deng, Xiusi Chen, Renhe Jiang, Xuan Song, and Ivor W Tsang. St-norm: Spatial and temporal normalization for multi-variate time series forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 269–278, 2021.

[53] Songgaojun Deng, Shusen Wang, Huzefa Rangwala, Lijing Wang, and Yue Ning. Cola-gnn: Cross-location attention based graph neural networks for long-term ili prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 245–254, 2020.

[54] Ilias Dimoulkas, Peyman Mazidi, and Lars Herre. Neural networks for gefcom2017 probabilistic load forecasting. *International Journal of Forecasting*, 35(4):1409–1423, 2019.

[55] Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, Renjun Xu, and Chongjun Wang. Adarnn: Adaptive learning and forecasting of time series. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 402–411, 2021.

[56] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[57] Vijay Ekambaram, Kushagra Manglik, Sumanta Mukherjee, Surya Shravan Kumar Sajja, Satyam Dwivedi, and Vikas Raykar. Attention based multi-modal new product sales time-series forecasting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3110–3118, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403362. URL https://doi.org/10.1145/3394486.3403362.

[58] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):1–34, 2012.

[59] Chenyou Fan, Yuze Zhang, Yi Pan, Xiaoyue Li, Chi Zhang, Rong Yuan, Di Wu, Wensheng Wang, Jian Pei, and Heng Huang. Multi-horizon time series forecasting with temporal attention learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2527–2535, 2019.

[60] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. Rpps: A novel resource prediction and provisioning scheme in cloud data center. In *2012 IEEE Ninth International Conference on Services Computing*, pages 609–616. IEEE, 2012.

[61] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In *2013 39th Euromicro conference on software engineering and advanced applications*, pages 357–364. IEEE, 2013.

[62] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. Utilization prediction aware vm consolidation approach for green cloud computing. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 381–388. IEEE, 2015.

[63] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. Coevolve: A joint point process model for information diffusion and network co-evolution. *Advances in Neural Information Processing Systems*, 28, 2015.

[64] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019.

[65] Robert Fildes, Michèle Hibon, Spyros Makridakis, and Nigel Meade. Generalising about univariate forecasting methods: further empirical evidence. *International journal of Forecasting*, 14(3):339–358, 1998.

[66] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series. In *International Conference on Learning Representations*, 2018.

[67] Satoshi Furutani, Toshiki Shibahara, Mitsuaki Akiyama, Kunio Hato, and Masaki Aida. Graph signal processing for directed graphs based on the hermitian laplacian. In *ECML-PKDD*, 2019.

[68] Everette S. Gardner. Exponential smoothing: The state of the art. *Journal of Forecasting*, 4:1–28, 1985.

[69] Chuancai Ge, Yang Wang, Xike Xie, Hengchang Liu, and Zhengyang Zhou. An integrated model for urban subregion house price forecasting: A multi-source data perspective. In *ICDM*. IEEE, 2019.

[70] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

[71] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 2019.

[72] Amir Ghaderi, Borhan M Sanandaji, and Faezeh Ghaderi. Deep forecast: deep learning-based spatio-temporal forecasting. *arXiv preprint arXiv:1707.08110*, 2017.

[73] Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In *Advances in neural information processing systems*, pages 545–552, 2003.

[74] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[75] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.

[76] Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 816–825. Ieee, 2016.

[77] Shubham Gupta and Srikanta Bedathur. A survey on temporal graph representation learning and generative modeling. *arXiv preprint arXiv:2208.12126*, 2022.

[78] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, NJ, 1994.

[79] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[80] Siho Han and Simon S Woo. Learning sparse latent graph representations for anomaly detection in multivariate time series. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2977–2986, 2022.

[81] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.

[82] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[84] Yixuan He, Quan Gan, David Wipf, Gesine D Reinert, Junchi Yan, and Mihai Cucuringu. Gnnrank: Learning global rankings from pairwise comparisons via directed graph neural networks. In *International Conference on Machine Learning*, pages 8581–8612. PMLR, 2022.

[85] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[86] Mikael Henaff, Junbo Zhao, and Yann LeCun. Prediction under uncertainty with error-encoding networks. *arXiv:1711.04994*, 2017.

[87] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[88] Minoru Higuchi, Kanji Matsutani, Masahito Kumano, and Masahiro Kimura. Discovering spatio-temporal latent influence in geographical attention dynamics. In *ECML-PKDD*, 2019. ISBN 978-3-030-10928-8.

[89] Thi Kieu Khanh Ho, Ali Karami, and Narges Armanfard. Graph-based time-series anomaly detection: A survey. *arXiv preprint arXiv:2302.00058*, 2023.

[90] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[91] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[92] Min Hou, Chang Xu, Zhi Li, Yang Liu, Weiqing Liu, Enhong Chen, and Jiang Bian. Multi-granularity residual learning with confidence estimation for time series prediction. In *Proceedings of the ACM Web Conference 2022*, pages 112–121, 2022.

[93] Siteng Huang, Donglin Wang, Xuehan Wu, and Ao Tang. Dsanet: Dual self-attention network for multivariate time series forecasting. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2129–2132, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763.

[94] Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu. Graph recurrent networks with attributed random walks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 732–740, 2019.

[95] Won-Seok Hwang, Jeong-Han Yun, Jonguk Kim, and Hyoung Chun Kim. Time-series aware precision and recall for anomaly detection: considering variety of detection result and addressing ambiguous labeling. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2241–2244, 2019.

[96] Aya Abdelsalam Ismail, Mohamed Gunady, Hector Corrada Bravo, and Soheil Feizi. Benchmarking deep learning interpretability in time series predictions. *Advances in neural information processing systems*, 33:6441–6452, 2020.

[97] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[98] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

[99] Sheo Yon Jhin, Jaehoon Lee, Minju Jo, Seungji Kook, Jinsung Jeon, Jihyeon Hyeong, Jayoung Kim, and Noseong Park. Exit: Extrapolation and interpolation-based neural controlled differential equations for time-series classification and forecasting. In *Proceedings of the ACM Web Conference 2022*, pages 3102–3112, 2022.

[100] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 207:117921, 2022.

[101] Ming Jin, Huan Yee Koh, Qingsong Wen, Daniele Zambon, Cesare Alippi, Geoffrey I Webb, Irwin King, and Shirui Pan. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *arXiv preprint arXiv:2307.03759*, 2023.

[102] Kieran Kalair and Colm Connaughton. Anomaly detection and classification in traffic flow data from fluctuations in the flow–density relationship. *Transportation Research Part C: Emerging Technologies*, 127:103178, 2021.

[103] Guolin Ke, Zhenhui Xu, Jia Zhang, Jiang Bian, and Tie-Yan Liu. Deepgbm: A deep learning framework distilled by gbdt for online prediction tasks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 384–394, 2019.

[104] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[105] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[106] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

[107] Ladislav Kristoufek. Measuring correlations between non-stationary series with dcca coefficient. *Physica A: Statistical Mechanics and its Applications*, 402:291–298, 2014.

[108] Jitendra Kumar and Ashutosh Kumar Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41–52, 2018.

[109] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[110] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104, 2018.

[111] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1939–1947, 2015.

[112] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, volume 34, pages 1–5, 2017.

[113] Vincent Le Guen and Nicolas Thome. Probabilistic time series forecasting with shape and temporal diversity. *Advances in Neural Information Processing Systems*, 33, 2020.

[114] Chonho Lee, Ping Wang, and Dusit Niyato. A real-time group auction system for efficient allocation of cloud internet applications. *IEEE Transactions on Services Computing*, 8(2):251–268, 2013.

[115] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(6):1–25, 2019.

[116] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. Predicting path failure in time-evolving graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1279–1289, 2019.

[117] Mengzhang Li and Zhanxing Zhu. Spatial-temporal fusion graph neural networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4189–4196, 2021.

[118] Steven Cheng-Xian Li and Benjamin Marlin. Learning from irregularly-sampled time series: A missing data perspective. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5937–5946. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/li20k.html.

[119] Wei Li, Ruihan Bao, Keiko Harimoto, Deli Chen, Jingjing Xu, and Qi Su. Modeling the stock relation with graph network for overnight stock movement prediction. In *IJCAI*, pages 4541–4547, 2020.

[120] Xuerong Li, Wei Shang, and Shouyang Wang. Text-based crude oil price forecasting: A deep learning approach. *International Journal of Forecasting*, 35(4):1548–1560, 2019.

[121] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SJiHXGWAZ.

[122] Yang Li, Xianli Zhang, Buyue Qian, Zeyu Gao, Chong Guan, Yefeng Zheng, Hansen Zheng, Fenglang Wu, and Chen Li. Towards interpretability and personalization: A predictive framework for clinical time-series analysis. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 340–349. IEEE, 2021.

[123] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05493.

[124] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11): 1857–1874, 2005.

[125] Bryan Lim. Forecasting treatment responses over time using recurrent marginal structural networks. In *Advances in Neural Information Processing Systems*, pages 7483–7493, 2018.

[126] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[127] Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*, 1(4): 19–38, 2019.

[128] Lei Lin, Zhengbing He, and Srinivas Peeta. Predicting station-level hourly demand in a large-scale bike-sharing network: A graph convolutional neural network approach. *Transportation Research Part C: Emerging Technologies*, 2018.

[129] Chenghao Liu, Steven CH Hoi, Peilin Zhao, and Jianling Sun. Online arima algorithms for time series prediction. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[130] Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. Eignn: Efficient infinite-depth graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[131] Zhenyu Liu, Baotian Hu, Zhenran Xu, and Min Zhang. Ppat: Progressive graph pairwise attention network for event causality identification. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5150–5158. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/572. URL https://doi.org/10.24963/ijcai.2023/572. Main Track.

[132] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2019.

[133] Bin Lu, Xiaoying Gan, Haiming Jin, Luoyi Fu, and Haisong Zhang. Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1025–1034, 2020.

[134] Xiaoyi Luo, Jiaheng Peng, and Jun Liang. Directed hypergraph attention network for traffic forecasting. *IET Intelligent Transport Systems*, 16(1):85–98, 2022.

[135] Jiaqi Ma, Bo Chang, Xuefei Zhang, and Qiaozhu Mei. Copulagnn: Towards integrating representational and correlational roles of graphs in graph neural networks. In *International Conference on Learning Representations*, 2020.

[136] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[137] Danielle C. Maddix, Yuyang Wang, and Alex Smola. Deep factors with gaussian processes for forecasting. *CoRR*, abs/1812.00098, 2018. URL http://arxiv.org/abs/1812.00098.

[138] Spyros Makridakis and Michele Hibon. Arma models and the box–jenkins methodology. *Journal of Forecasting*, 16(3), 1997.

[139] Spyros Makridakis, Steven C Wheelwright, and Rob J Hyndman. *Forecasting methods and applications*. John wiley & sons, 2008.

[140] Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. In and out: optimizing overall interaction in probabilistic graphs under clustering constraints. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1371–1381, 2020.

[141] David McDowall, Richard McCleary, and Bradley J Bartos. *Interrupted time series analysis*. Oxford University Press, 2019.

[142] Shanka Subhra Mondal, Nikhil Sheoran, and Subrata Mitra. Scheduling of time-varying workloads using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9000–9008, 2021.

[143] Yunfei Mu, Jianzhong Wu, Nick Jenkins, Hongjie Jia, and Chengshan Wang. A spatial–temporal model for grid impact analysis of plug-in electric vehicles. *Applied Energy*, 2014.

[144] Takaaki Nakamura, Makoto Imamura, Ryan Mercer, and Eamonn Keogh. Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1190–1195. IEEE, 2020.

[145] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: neural basis expansion analysis for interpretable time series forecasting. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=r1ecqn4YwB.

[146] Boris N Oreshkin, Arezou Amini, Lucy Coyle, and Mark Coates. Fc-gaga: Fully connected gated graph architecture for spatio-temporal traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9233–9241, 2021.

[147] Benjamin Paassen, Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Barbara Eva Hammer. Graph edit networks. In *International Conference on Learning Representations*, 2020.

[148] Chao Pan, Siheng Chen, and Antonio Ortega. Spatio-temporal graph scattering transform. In *International Conference on Learning Representations*, 2020.

[149] Cheonbok Park, Chunggi Lee, Hyojin Bahng, Yunwon Tae, Seungmin Jin, Kihwan Kim, Sungahn Ko, and Jaegul Choo. St-grat: A novel spatio-temporal graph attention networks for accurately forecasting dynamically changing road speed. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1215–1224, 2020.

[150] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml*, volume 1, pages 727–734, 2000.

[151] Fotios Petropoulos, Daniele Apiletti, Vassilios Assimakopoulos, Mohamed Zied Babai, Devon K. Barrow, Souhaib Ben Taieb, Christoph Bergmeir, Ricardo J. Bessa, Jakub Bijak, John E. Boylan, Jethro Browell, Claudio Carnevale, Jennifer L. Castle, Pasquale

Cirillo, Michael P. Clements, Clara Cordeiro, Fernando Luiz Cyrino Oliveira, Shari De Baets, Alexander Dokumentov, Joanne Ellison, Piotr Fiszeder, Philip Hans Franses, David T. Frazier, Michael Gilliland, M. Sinan Gönül, Paul Goodwin, Luigi Grossi, Yael Grushka-Cockayne, Mariangela Guidolin, Massimo Guidolin, Ulrich Gunter, Xiaojia Guo, Renato Guseo, Nigel Harvey, David F. Hendry, Ross Hollyman, Tim Januschowski, Jooyoung Jeon, Victor Richmond R. Jose, Yanfei Kang, Anne B. Koehler, Stephan Kolassa, Nikolaos Kourentzes, Sonia Leva, Feng Li, Konstantia Litsiou, Spyros Makridakis, Gael M. Martin, Andrew B. Martinez, Sheik Meeran, Theodore Modis, Konstantinos Nikolopoulos, Dilek Önkal, Alessia Paccagnini, Anastasios Panagiotelis, Ioannis Panapakidis, Jose M. Pavía, Manuela Pedio, Diego J. Pedregal, Pierre Pinson, Patrícia Ramos, David E. Rapach, J. James Reade, Bahman Rostami-Tabar, Michał Rubaszek, Georgios Sermpinis, Han Lin Shang, Evangelos Spiliotis, Aris A. Syntetos, Priyanga Dilini Talagala, Thiyanga S. Talagala, Len Tashman, Dimitrios Thomakos, Thordis Thorarinsdottir, Ezio Todini, Juan Ramón Trapero Arenas, Xiaoqian Wang, Robert L. Winkler, Alisa Yusupova, and Florian Ziel. Forecasting: theory and practice. *International Journal of Forecasting*, 38(3):705–871, 2022. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2021.11.001. URL https://www.sciencedirect.com/science/article/pii/S0169207021001758.

[152] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2627–2633. ijcai.org, 2017. doi: 10.24963/ijcai.2017/366. URL https://doi.org/10.24963/ijcai.2017/366.

[153] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *Advances in neural information processing systems*, pages 7785–7794, 2018.

[154] Md. Rasheduzzaman, Md. Amirul Islam, and Rashedur M. Rahman. Workload prediction on google cluster trace. *Int. J. Grid High Perform. Comput.*, 6(3):34–52, July 2014. ISSN 1938-0259.

[155] Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs M. Bergmann, and Roland Vollgraf. Multivariate probabilistic time series forecasting via conditioned normalizing flows. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=WiGQBFuVRv.

[156] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

[157] Cédric Richard, José Carlos M Bermudez, and Paul Honeine. Online prediction of time series data with kernels. *IEEE Transactions on Signal Processing*, 57(3):1058–1067, 2008.

[158] Yuecheng Rong, Zhimian Xu, Ruibo Yan, and Xu Ma. Du-parking: Spatio-temporal big data tells you realtime parking availability. In *Proceedings of the 24th ACM SIGKDD*, KDD '18. ACM, 2018. ISBN 9781450355520.

[159] Ryan A. Rossi. Relational time series forecasting. *Knowledge Engineering Review (KER)*, 33:e1, 2018.

[160] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen K. Ahmed, Danai Koutra, and John Boaz Lee. From community to role-based graph embeddings. In *arXiv:1908.08572*, 2019.

[161] David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico, and Jan Gasthaus. High-dimensional multivariate forecasting with low-rank gaussian copula processes. In *Advances in Neural Information Processing Systems*, pages 6824–6834, 2019.

[162] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36:1181–1191, 2020.

[163] Ramit Sawhney, Shivam Agarwal, Arnav Wadhwa, Tyler Derr, and Rajiv Ratn Shah. Stock selection via spatiotemporal hypergraph attention network: A learning to rank approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 497–504, 2021.

[164] Mona Schirmer, Mazin Eltayeb, Stefan Lessmann, and Maja Rudolph. Modeling irregular time series with continuous recurrent units. In *International Conference on Machine Learning*, pages 19388–19405. PMLR, 2022.

[165] Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In *Advances in Neural Information Processing Systems*, pages 4837–4846, 2019.

[166] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[167] Chao Shang, Jie Chen, and Jinbo Bi. Discrete graph structure learning for forecasting multiple time series. In *International Conference on Learning Representations*, 2020.

[168] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–14, 2011.

[169] Satya Narayan Shukla and Benjamin Marlin. Interpolation-prediction networks for irregularly sampled time series. In *International Conference on Learning Representations*, 2018.

[170] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[171] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[172] Kamile Stankeviciute, Ahmed M Alaa, and Mihaela van der Schaar. Conformal time-series forecasting. *Advances in Neural Information Processing Systems*, 34, 2021.

[173] James Stock and M.W. Watson. Vector autoregressions. *Journal of Economic Perspectives*, 2001.

[174] Junkai Sun, Junbo Zhang, Qiaofei Li, Xiuwen Yi, Yuxuan Liang, and Yu Zheng. Predicting citywide crowd flows in irregular regions using multi-view graph convolutional networks. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[175] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014.

[176] Souhaib Ben Taieb, Antti Sorjamaa, and Gianluca Bontempi. Multiple-output modeling for multi-step-ahead time series forecasting. *Neurocomputing*, 73(10-12):1950–1957, 2010.

[177] Shanshan Tang, Bo Li, and Haijun Yu. Chebnet: Efficient and stable constructions of deep neural networks with rectified power units using chebyshev approximations. *arXiv preprint arXiv:1911.05467*, 2019.

[178] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[179] Waldo Tobler. On the first law of geography: A reply. *Annals of the Association of American Geographers*, 94(2):304–310, 2004.

[180] Alasdair Tran, Alexander Mathews, Cheng Soon Ong, and Lexing Xie. Radflow: A recurrent, aggregated, and decomposable model for networks of time series. In *Proceedings of the Web Conference 2021*, pages 730–742, 2021.

[181] Vladimir Vapnik, Steven Golowich, and Alex Smola. Support vector method for function approximation, regression estimation and signal processing. *Advances in neural information processing systems*, 9, 1996.

[182] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[183] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[184] Shivaram Venkataraman, Aurojit Panda, Ganesh Ananthanarayanan, Michael J Franklin, and Ion Stoica. The power of choice in data-aware cluster scheduling. In *OSDI*, pages 301–316, 2014.

[185] Petra Vrablecová, Viera Rozinajová, and Anna Bou Ezzeddine. Incremental adaptive time series prediction for power demand forecasting. In *International Conference on Data Mining and Big Data*, pages 83–92. Springer, 2017.

[186] Bao Wang, Xiyang Luo, Fangbo Zhang, Baichuan Yuan, Andrea L. Bertozzi, and P. Jeffrey Brantingham. Graph-based deep modeling and real time forecasting of sparse spatio-temporal data. *CoRR*, abs/1804.00684, 2018. URL http://arxiv.org/abs/1804.00684.

[187] Jingcheng Wang, Yong Zhang, Lixun Wang, Yongli Hu, Xinglin Piao, and Baocai Yin. Multitask hypergraph convolutional networks: A heterogeneous traffic prediction framework. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):18557–18567, 2022. doi: 10.1109/TITS.2022.3168879.

[188] Senzhang Wang, Jiannong Cao, and Philip S. Yu. Deep learning for spatio-temporal data mining: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34 (8):3681–3700, 2022. doi: 10.1109/TKDE.2020.3025580.

[189] Shuo Wang, Xing Zhang, Jiaxin Zhang, Jian Feng, Wenbo Wang, and Ke Xin. An approach for spatial-temporal traffic modeling in mobile cellular networks. In *2015 27th International Teletraffic Congress*. IEEE, 2015.

[190] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*, pages 1082–1092, 2020.

[191] Yuyang Wang, Alex Smola, Danielle C. Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine*

Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of *Proceedings of Machine Learning Research*, pages 6607–6617. PMLR, PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19k.html.

[192] Ruofeng Wen, Kari Torkkola, Balakrishnan (Murali) Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. In *NeurIPS 2017*, 2017. URL https://www.amazon.science/publications/a-multi-horizon-quantile-recurrent-forecaster.

[193] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[194] Tyler Wilson, Pang-Ning Tan, and Lifeng Luo. A low rank weighted graph convolutional approach to weather prediction. In *ICDM*, 2018.

[195] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[196] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. *arXiv preprint arXiv:2002.05990*, 2020.

[197] Hanrui Wu and Michael K Ng. Hypergraph convolution on nodes-hyperedges network for semi-supervised node classification. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(4):1–19, 2022.

[198] Jinming Wu, Qi Qi, Jingyu Wang, Haifeng Sun, Zhikang Wu, Zirui Zhuang, and Jianxin Liao. Not only pairwise relationships: Fine-grained relational modeling for multivariate time series forecasting. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 4416–4423. ijcai.org, 2023. doi: 10.24963/ijcai.2023/491. URL https://doi.org/10.24963/ijcai.2023/491.

[199] Renjie Wu and Eamonn Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[200] Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. Adversarial sparse transformer for time series forecasting. *Advances in neural information processing systems*, 33:17105–17115, 2020.

[201] Xian Wu, Yuxiao Dong, Chao Huang, Jian Xu, Dong Wang, and Nitesh V Chawla. Uapd: Predicting urban anomalies from spatial-temporal data. In *ECML-PKDD*, 2017.

[202] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1907–1913. Association for the Advancement of Artificial Intelligence (AAAI), International Joint Conferences on Artificial Intelligence Organization, 2019.

[203] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[204] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763, 2020.

[205] Lianghao Xia, Chao Huang, Yong Xu, Peng Dai, Liefeng Bo, Xiyue Zhang, and Tianyi Chen. Spatial-temporal sequential hypergraph network for crime prediction with dynamic multiplex relation learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1631–1637. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/225. URL https://doi.org/10.24963/ijcai.2021/225. Main Track.

[206] Sheng Xiang, Dawei Cheng, Chencheng Shang, Ying Zhang, and Yuqi Liang. Temporal and heterogeneous graph neural network for financial time series prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3584–3593, 2022.

[207] Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. Spatial-temporal transformer networks for traffic flow forecasting. *arXiv preprint arXiv:2001.02908*, 2020.

[208] Na Xu, Pengjian Shang, and Santi Kamae. Modeling traffic flow correlation using dfa and dcca. *Nonlinear Dynamics*, 2010.

[209] Chao-Han Huck Yang, Yun-Yun Tsai, and Pin-Yu Chen. Voice2series: Reprogramming acoustic models for time series classification. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11808–11819. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/yang21j.html.

[210] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16(1):7–18, 2014.

[211] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Li Zhenhui. Deep multi-view spatial-temporal network for taxi demand prediction. In *The Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[212] Junchen Ye, Zihan Liu, Bowen Du, Leilei Sun, Weimiao Li, Yanjie Fu, and Hui Xiong. Learning the evolutionary and multi-scale graph structure for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2296–2306, 2022.

[213] Jaehyuk Yi and Jinkyoo Park. Hypergraph convolutional recurrent neural network. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[214] Nan Yin, Fuli Feng, Zhigang Luo, Xiang Zhang, Wenjie Wang, Xiao Luo, Chong Chen, and Xian-Sheng Hua. Dynamic hypergraph convolutional network. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1621–1634. IEEE, 2022.

[215] Nan Yin, Li Shen, Huan Xiong, Bin Gu, Chong Chen, Xian-Sheng Hua, Siwei Liu, and Xiao Luo. Messages are never propagated alone: Collaborative hypergraph neural network for time-series forecasting. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 1–15, 2023.

[216] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240, 2019.

[217] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.

[218] Hongyuan Yu, Ting Li, Weichen Yu, Jianguo Li, Yan Huang, Liang Wang, and Alex Liu. Regularized graph structure learning with semantic knowledge for multi-variates time-series forecasting. *arXiv preprint arXiv:2210.06126*, 2022.

[219] G Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European journal of operational research*, 160(2):501–514, 2005.

[220] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 2018.

[221] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[222] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[223] Qi Zhang, Qizhao Jin, Jianlong Chang, Shiming Xiang, and Chunhong Pan. Kernel-weighted graph convolutional network: A deep learning approach for traffic forecasting. In *ICPR*, pages 1018–1023. IEEE, 2018.

[224] Weiqi Zhang, Chen Zhang, and Fugee Tsung. Grelen: Multivariate time series anomaly detection from the perspective of graph relational learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2390–2397, 2022.

[225] Xiang Zhang, Marko Zeman, Theodoros Tsiligkaridis, and Marinka Zitnik. Graph-guided network for irregularly sampled multivariate time series. *arXiv preprint arXiv:2110.05357*, 2021.

[226] Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2022.

[227] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2022. doi: 10.1109/TKDE.2020.2981333.

[228] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850. IEEE, 2020.

[229] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2019.

[230] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2019.

[231] Yusheng Zhao, Xiao Luo, Wei Ju, Chong Chen, Xian-Sheng Hua, and Ming Zhang. Dynamic hypergraph structure learning for traffic flow forecasting. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2303–2316, 2023. doi: 10.1109/ICDE55515.2023.00178.

[232] Zhenzhen Zhao, Guojiang Shen, Junjie Zhou, Junchen Jin, and Xiangjie Kong. Spatial-temporal hypergraph convolutional network for traffic forecasting. *PeerJ Computer Science*, 9:e1450, 2023.

[233] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.

[234] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

[235] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[236] Qazi Zia Ullah, Shahzad Hassan, and Gul Muhammad Khan. Adaptive resource utilization prediction system for infrastructure as a service cloud. *Computational intelligence and neuroscience*, 2017, 2017.

Appendices

A Neural Network Functions

Activation Functions are element-wise non-linear functions. Heavily used activation functions include the sigmoid, ReLU, LeakyReLU [136], tanh, and GeLU [87] functions

$$\sigma(X) = \frac{1}{1+e^{-X}} \qquad \text{ReLU}(X) = \max(0, X) \tag{1}$$

$$\tanh(X) = \frac{e^{X} - e^{-X}}{e^{X} + e^{-X}} \qquad \text{GELU}(X) = \frac{1}{2}X\left[1 + \text{erf}\left(\frac{X}{\sqrt{2}}\right)\right] \tag{2}$$

$$\text{LeakyReLU}(X) = \begin{cases} X, & x >= 0 \\ \mu X, & x < 0 \end{cases} \tag{3}$$

where $\text{LeakyReLU}(\cdot)$ is used with a pre-defined small hyper-parameter μ.

Vectorization is defined as a mathematical operation that converts any higher dimension values into a vector. Given a tensor $X \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_m}$, vectorization is denoted by

$$\text{Vec}(X) \in \mathbb{R}^{d_1 * d_2 * \ldots * d_m \times 1} \tag{4}$$

Softmax Function non-linearly transforms an array of values. Given an array of d values $\mathbf{x} \in \mathbb{R}^d$, the i(th) transformed value is calculated by

$$f_{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{d} \exp(x_j)} \tag{5}$$

A Fully Connected Neural Layer, or a dense layer, with respect to a parameter weight matrix $\mathbf{W} \in \mathbb{R}^{c_{in} \times c_{out}}$ and a parameter bias vector $\mathbf{b} \in \mathbb{R}^{c_{out}}$ is defined as

$$\text{FC}(X) = f_{FC_{\mathbf{W},\mathbf{b}}}(X) = \sigma(X\mathbf{W} + \mathbf{b}) \tag{6}$$

Without loss of generality, $X \in \mathbb{R}^{c_1 \times c_2 \times \ldots \times c_m}$ is defined as a tensor with m dimensions. Thus, the size of last dimension $c_{in} = c_m$ is mapped to size c_{out}, resulting in $f_{FC_{\mathbf{W},\mathbf{b}}}(X) \in \mathbb{R}^{c_1 \times c_2 \times \ldots \times c_{out}}$. We let $f_{FC_{\mathbf{W}}}$ denote the layer when the bias/intercept $\mathbf{b}$ is not included. In practice, when the fully connected layer is not applied on the last dimension, tensor dimensions are reorganized by switching the target dimension and the last dimension before applying the dense layer, and switching back their dimension order after the dense layer is applied. We drop $\mathbf{W}$ and $\mathbf{b}$ from the subscript for conciseness, as $f_{FC}(X) = f_{FC_{\mathbf{W},\mathbf{b}}}(X)$ or $f_{FC}(X) = f_{FC_{\mathbf{W}}}(X)$, when the context is clear to understand. When a dense layer is needed to apply for a selected dimension of X, we denote with a subscript under function $f(\cdot)$. For instance, when the ith dimension is selected, the input channel size $c_{in} = c_i$ is transformed into c_{out} with the aforementioned description, denoted as $f_{FC_{\mathbf{W},\mathbf{b},i}}(X) \in \mathbb{R}^{c_1 \times c_2 \times \ldots c_{i-1} \times c_{out} \times c_{i+1} \times c_{i+2} \ldots c_m}$. The calculation is also applicable when X is a matrix or a

vector. In addition, we let f_{FCs} denote a fully connected network that stacks multiple dense layers.

An Embedding Layer maps discrete one-hot vectors, denoted by $\mathbf{X} \in \mathbb{R}^{N \times c_{in}}$ to a continuous space $\mathbb{R}^{N \times c_{out}}$. We let $f_{EMB}(\cdot) : \mathbb{R}^{N \times c_{in}} \rightarrow \mathbb{R}^{N \times c_{out}}$ denote an embedding layer, which can be implemented in various ways depending on the task characteristics, e.g., f_{EMB} can be a dense layer in the simplest case.

A 1D Convolutional Layer adopts a filter to select only a specific subset of input neurons to derive each output neuron. Given an input vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$, the convolutional layer is a function denoted by f_{CONV} that maps it to an output vector $\mathbf{x}' \in \mathbb{R}^{d_{out}}$

$$f_{CONV}(\mathbf{x}_i) = \sum_{j=0}^{d_{in}} \mathbf{W}_{ij} \star \mathbf{x}_j + \mathbf{b}_i, \quad i \in \{1, 2, \ldots, d_{out}\} \tag{7}$$

where $\mathbf{W}$ and $\mathbf{b}$ are parameters and $\star$ is the cross-correlation operator defined in the subject of signal processing. A dilation method can be applied to the convolutional layer by spacing input neurons to reduce parameter complexity.

B Graph Time-series Models in Details

This section lists detailed equations for the discussed models.

B.1 GCRN

Following the conventional use of symbols $\mathbf{I}, \mathbf{F}$, and $\mathbf{O}$ to respectively denote input gate, forget gate, and output gate, the GCRN model [166] can be described with,

$$\mathbf{I}^t = \sigma\left(f_{GCN,\mathbf{XI}}\left(\mathbf{X}^t\right) + f_{GCN,\mathbf{HI}}\left(\mathbf{H}^{t-1}\right) + \mathbf{W}_{\mathbf{CI}} \odot \mathbf{C}^{t-1} + \mathbf{b}_{\mathbf{I}}\right) \tag{8}$$

$$\mathbf{F}^t = \sigma\left(f_{GCN,\mathbf{XF}}\left(\mathbf{X}^t\right) + f_{GCN,\mathbf{HF}}\left(\mathbf{H}^{t-1}\right) + \mathbf{W}_{\mathbf{CF}} \odot \mathbf{C}^{t-1} + \mathbf{b}_{\mathbf{F}}\right) \tag{9}$$

$$\mathbf{C}^t = \mathbf{F}^t \odot \mathbf{C}^{t-1} + \mathbf{I}^t \odot \tanh\left(f_{GCN,\mathbf{XC}}\left(\mathbf{X}^t\right) + f_{GCN,\mathbf{HC}}\left(\mathbf{H}^{t-1}\right) + \mathbf{b}_{\mathbf{C}}\right) \tag{10}$$

$$\mathbf{O}^t = \sigma\left(f_{GCN,\mathbf{XO}}\left(\mathbf{X}^t\right) + f_{GCN,\mathbf{HO}}\left(\mathbf{H}^{t-1}\right) + \mathbf{W}_{\mathbf{CO}} \odot \mathbf{C}^t + \mathbf{b}_{\mathbf{O}}\right) \tag{11}$$

$$\mathbf{H}^t = \mathbf{O}^t \odot \mathbf{C}^t \tag{12}$$

B.2 FC-GAGA

Given concatenated state $\mathbf{Z}$ as input, the residual module $f_{res} = \hat{\mathbf{X}}$ in FC-GAGA is described with the following equations:

$$\mathbf{Z} = \left[\mathbf{H}_{emb} \,\|\, \frac{\mathbf{X}}{\tilde{\mathbf{x}}} \,\|\, \mathbf{G}\right]^{\mathsf{T}} \qquad \mathbf{Z}_0 = \mathbf{Z} \qquad \hat{\mathbf{Z}}_0 = 0 \qquad \mathbf{Z}_b = \mathrm{ReLU}\left[\mathbf{Z}_{b-1} - \hat{\mathbf{Z}}_{b-1}\right] \tag{13}$$

$$\mathbf{H}_b^1 = f_{FC,b_1}\left(\mathbf{Z}_b\right) \qquad \mathbf{H}_b^L = f_{FC,b_L}\left(\mathbf{H}_b^{L-1}\right) \tag{14}$$

$$\hat{\mathbf{X}}_b = f_{FC,b}\left(\mathbf{H}_b^L\right) \qquad f_{res} = \hat{\mathbf{X}} = \sum_{b=1}^{B} \hat{\mathbf{X}}_b \tag{15}$$

and then $\mathbf{Z}$ is used to initialize the input to B residual blocks in Eq. 13, where each block contains L dense layers in Eq. 14. Finally, the forecasting is the aggregation of hidden states at the last layers of each block through dense layers, as in Eq. 15.

B.3 Radflow

Radflow defines a feed-forward layer, denoted by $g_{FF}(\cdot)$:

$$g_{FF}(X) = f_{FC,FF2}\left(\mathrm{GELU}\left(f_{FC,FF1}(X)\right)\right) \tag{16}$$

The residual module and attention module are described by the following equations:

$$\mathbf{Z}_1^t = f_{FC}\left(X^t\right) \qquad \mathbf{H}_b = f_{LSTM}\left(\mathbf{Z}_b\right) \qquad \mathbf{P}_b^t = g_{FF,p}\left(\mathbf{H}_b^t\right) \qquad \mathbf{Z}_b^t = \mathbf{Z}_{b-1}^t - \mathbf{P}_{b-1}^t \tag{17}$$

$$f_{res}: \begin{cases} \mathbf{Q}_b^t = g_{FF,q}\left(\mathbf{H}_b^t\right), & \hat{\mathbf{Q}}^t = \sum_{b=1}^{B} \mathbf{Q}_b^t \\ \mathbf{U}_b^t = g_{FF,u}\left(\mathbf{H}_b^t\right), & \mathbf{U}^t = \sum_{b=1}^{B} \mathbf{U}_b^t \end{cases} \tag{18}$$

$$f_{attn}: \begin{cases} \mathbf{U}_K^t = f_{FC,K}\left(\mathbf{U}^t\right), & \mathbf{U}_V^t = f_{FC,V}\left(\mathbf{U}^t\right), & \mathbf{U}_Q^t = f_{FC,Q}\left(\mathbf{U}^t\right) \\ \lambda_j^t = \frac{\exp\left(\mathbf{U}_{j,Q}^t \cdot \mathbf{U}_{j,K}^t\right)}{\sum_{k \in \Gamma i} \exp\left(\mathbf{U}_{k,Q}^t \cdot \mathbf{U}_{k,K}^t\right)}, & \tilde{\mathbf{U}}_i^t = \mathrm{GELU}\left(\sum_{j \in \Gamma i} \lambda_j^t \mathbf{U}_{j,V}^t\right) & \\ \hat{\mathbf{U}}^t = f_{FC,E}\left(\mathbf{U}^{t-1}\right) + f_{FC,N}\left(\tilde{\mathbf{U}}^t\right) & & \end{cases} \tag{19}$$

$$\hat{\mathbf{Q}}^t = f_{res,Q} \cdot f_{LSTM}(X) \qquad \hat{\mathbf{U}}^t = f_{attn} \cdot f_{res,U} \cdot f_{LSTM}(X) \tag{20}$$

$$\hat{\mathbf{X}}^t = f_{FC,recurrent}\left(\hat{\mathbf{Q}}^t\right) + f_{FC,graph}\left(\hat{\mathbf{U}}^t\right) \tag{21}$$

The hidden state $\hat{Q}^t$ in the recurrent component is computed with the residual module, described in Eq. 18 and only relies on the time-series data. Whereas the hidden state $\hat{\mathbf{U}}^t$ in the graph component renders attention mechanisms to aggregate node embeddings from the neighborhood for each node, as described by Eq. 19, in addition to the use of LSTM model and neural blocks aggregation.

C Experimental Setup

C.1 Data Preprocessing

Normalization techniques are applied to map large-scaled data into a range of $[0, 1]$ or $[-1, 1]$. Given time-series $\mathbf{x}$, the min-max normalization is defined as

$$\underset{min-max}{\text{Norm}}(\mathbf{x}) = \frac{\mathbf{x} - \mathbf{x}_{\min}}{\mathbf{x}_{\max} - \mathbf{x}_{\min}} \tag{22}$$

where $\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$ are the maximum and the minimum values in the time-series, respectively.

The Z-score normalization is defined as

$$\underset{z-score}{\text{Norm}}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sigma} \tag{23}$$

where μ and σ are the mean and standard deviation of the time-series.

The normalization can be extended for data in the matrix format.

C.2 Distance-based Graph Construction

Radial Basis Function (RBF) is applied to on a given distance adjacency matrix $\mathbf{A}_{dist} \in \mathbb{R}^{N \times N}$ to derives a proximity matrix $\mathbf{A}$ with a selected length scale l:

$$\mathbf{A}_{ij} = \exp\left(-\frac{\mathbf{A}^2_{dist_{i,j}}}{l^2}\right) \tag{24}$$

C.3 Time-series Similarity/Coefficient Metrics

Dynamic Time Warping is a closeness measure between time-series. Given two time-series $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$ and $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$ of length m and n, the DTW distance between the two time-series is denoted by $\text{DTW}(n, m)$ which is defined as

$$\text{DTW}(i, j) = \text{cost}_{i,j} + \min\left(\text{DTW}(i-1, j-1), \text{DTW}(i-1, j), \text{DTW}(i, j-1)\right) \tag{25}$$

where $\text{cost}_{i,j}$ is the distance between $\mathbf{x}_i$ and $\mathbf{y}_j$ and can be decided by the use of distance functions, e.g., absolute distance or square distance. $\text{DTW}(i, 0)$ and $\text{DTW}(0, j)$ are initialized with zeros. A number of variants soft-DTW [44] proposed a smoothed formulation of DTW and use it as a differentiable loss function for classification tasks.

Cosine similarity is also used by taking time-series as vectors. Given two time-series, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^T$ of same length, cosine similarity is defined as

$$\operatorname{Cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|} \tag{26}$$

C.4 Loss Functions and Metrics

For classification tasks, without loss of generality, we let TP, TN, FP, and FN denote numbers of true positive, true negative, false positive, and false negative, respective, thus commonly used metrics are as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad \text{F1} = \frac{2\,\text{Precision}\,\text{Recall}}{\text{Precision} + \text{Recall}} \tag{27}$$

For regression tasks, we let $X, \hat{X} \in \mathbb{R}^{c_1 \times c_2 \times \ldots \times c_m}$ denote the input forecast and prediction tensors to loss metrics.

$$\text{MAE}\left(X, \hat{X}\right) = \frac{1}{\prod_i c_i} \sum_{i1,i2,\ldots,im} \left| X_{i1,i2,\ldots,im} - \hat{X}_{i1,i2,\ldots,im} \right| \tag{28}$$

$$\text{RMSE}\left(X, \hat{X}\right) = \sqrt{\frac{1}{\prod_i c_i} \sum_{i1,i2,\ldots,im} (X_{i1,i2,\ldots,im} - \hat{X}_{i1,i2,\ldots,im})^2} \tag{29}$$

$$\text{MAPE}\left(X, \hat{X}\right) = \frac{1}{\prod_i c_i} \sum_{i1,i2,\ldots,im} \left| \frac{X_{i1,i2,\ldots,im} - \hat{X}_{i1,i2,\ldots,im}}{X_{i1,i2,\ldots,im}} \right| \tag{30}$$

$$\text{SMAPE}\left(X, \hat{X}\right) = \frac{1}{\prod_i c_i} \sum_{i1,i2,\ldots,im} \left| \frac{X_{i1,i2,\ldots,im} - \hat{X}_{i1,i2,\ldots,im}}{\left(|X_{i1,i2,\ldots,im}| + \left| \hat{X}_{i1,i2,\ldots,im} \right| \right) / 2} \right| \tag{31}$$

These metrics are still applicable when X is a vector or a matrix. For example, let Ω denote the time index of the training dataset, its overall MAE and RMSE losses for N time series and a horizon of τ are calculated as:

$$\underset{MAE}{\text{Loss}} = \frac{1}{|\Omega|} \frac{1}{N} \frac{1}{\tau} \sum_{t \in \Omega} \sum_{i=1}^{N} \sum_{j=1}^{\tau} \left| x_i^{t+j} - \hat{x}_i^{t+j} \right| \tag{32}$$

$$\underset{RMSE}{\text{Loss}} = \sqrt{\frac{1}{|\Omega|} \frac{1}{N} \frac{1}{\tau} \sum_{t \in \Omega} \sum_{i=1}^{N} \sum_{j=1}^{\tau} \left(x_i^{t+j} - \hat{x}_i^{t+j} \right)^2} \tag{33}$$

where $\mathbf{X} \in \mathbb{R}^{N \times T}$ is a time-series matrix. Losses for other metrics are calculated similarly.

C.5 Other Functions

A sign function is defined as

$$\operatorname{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{34}$$

The probabilistic Density functions (pdf) of the probability distribution are listed as follows

$$\text{Gumbel}(x; \alpha, \beta) = \frac{1}{\beta} \exp\left(\frac{x-\alpha}{\beta} - \exp\left(\frac{x-\alpha}{\beta}\right)\right) \tag{35}$$

$$\text{Gumbel}(x; 0, 1) = \exp\left(x - \exp\left(x\right)\right) \tag{36}$$

D Data Characteristics

We collect and present available representative datasets in Table 3. Statistical properties of selected datasets are also described. The selected datasets are by no means exhaustive but are summarized for the convenience of researchers.

Table 3: Statistical attributes of selected datasets.

Category	Collection	Source	#Datasets	#Time-series	#Features	#Length	#Classes
Classification	UEA	[5]	30	$27-50K$	$2-1.3K$	$8-18K$	$2-39$
	UCI	[56]	88	$58-63M$	$4-2M$	-	-
	UCR	[46]	128	$40-16K$	-	$60-2.7K$	$2-60$

Category	Application	Dataset	Source	#Time-series	#Length
Regression	Speed	METR-LA	[121]	207	$6.5M$
		PEMS-BAY		325	$17K$
		Xiamen	[233]	95	$52.7K$
	Taxi	ShenZhen	[229]	156	$3K$
		DiDi	[133]	$1.3K$	$17.2K$
	Power	Solar[1]	[110]	137	$52.5K$
		Electricity[2]		321	$26.3K$
		PMU	[167]	42	$8.6K$
	Currency	Exchange	[110]	8	$7.6K$
		Stock	[119]	498	$18K$
	Influenza	Japan-Prefecture	[53]	49	348
		US-States		49	785
		US-Regions		10	360
	Large Scale	Music	[180]	$60.7K$	63
		Wiki		$366.8K$	1827

Category	Datasets	Source	#Time-series	#Features	#Length	%Anomalies
Anomaly Detection	SWaT	[51]	50	1	$95K$	11.97
	WADI		112	1	$122K$	5.99
	SMAP	[228]	25	1	$561K$	13.13
	MSL		55	1	$132K$	10.27