Graph Deep Factors for Probabilistic Time-series Forecasting

HONGJIE CHEN, Virginia Tech, USA
RYAN A. ROSSI, Adobe Research, USA
KANAK MAHADIK, Adobe Research, USA
SUNGCHUL KIM, Adobe Research, USA
HODA ELDARDIRY, Virginia Tech, USA

Effective time-series forecasting methods are of significant importance to solve a broad spectrum of research problems. Deep probabilistic forecasting techniques have recently been proposed for modeling large collections of time-series. However, these techniques explicitly assume either complete independence (local model) or complete dependence (global model) between time-series in the collection. This corresponds to the two extreme cases where every time-series is disconnected from every other time-series in the collection or likewise, that every time-series is related to every other time-series resulting in a completely connected graph. In this work, we propose a deep hybrid probabilistic graph-based forecasting framework called Graph Deep Factors (GraphDF) that goes beyond these two extremes by allowing nodes and their time-series to be connected to others in an arbitrary fashion. GraphDF is a hybrid forecasting framework that consists of a relational global and relational local model. In particular, a relational global model learns complex non-linear time-series patterns globally using the structure of the graph to improve both forecasting accuracy and computational efficiency. Similarly, instead of modeling every time-series independently, a relational local model not only considers its individual time-series but also the time-series of nodes that are connected in the graph. The experiments demonstrate the effectiveness of the proposed deep hybrid graph-based forecasting model compared to the state-of-the-art methods in terms of its forecasting accuracy, runtime, and scalability. Our case study reveals that GraphDF can successfully generate cloud usage forecasts and opportunistically schedule workloads to increase cloud cluster utilization by 47.5% on average. Furthermore, we target addressing the common nature of many time-series forecasting applications where time-series are provided in a streaming version, however, most methods fail to leverage the newly incoming time-series values and result in worse performance over time. In this paper, we propose an online incremental learning framework for probabilistic forecasting. The framework is theoretically proven to have lower time and space complexity. The framework can be universally applied to many other machine learning-based methods.

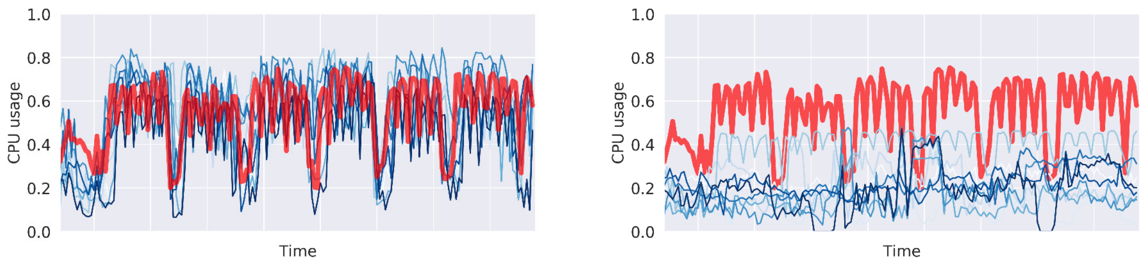
Additional Key Words and Phrases: Incremental Online Learning, Graph Neural Network, Time-series Forecasting

1 INTRODUCTION

Forecasting is fundamentally important with many applications including the optimization of resources allocation. 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

Authors' addresses: Hongjie Chen, Virginia Tech, Blacksburg, Virginia, USA, 24061, jeffchan@vt.edu; Ryan A. Rossi, Adobe Research, San Jose, USA, ryrossi@adobe.com; Kanak Mahadik, Adobe Research, San Jose, USA, mahadik@adobe.com; Sungchul Kim, Adobe Research, San Jose, USA, sukim@adobe.com; Hoda Eldardiry, Virginia Tech, Blacksburg, Virginia, USA, hdardiry@vt.edu.

1556-4681/2022/6-ART $15.00
https://doi.org/10.1145/3543511



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

Fig. 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*.

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 h-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.

Classical time-series forecasting models such as ARIMA [54] and exponential smoothing models [32] only focus on forecasting individual or small groups of time-series, and hence limiting scalability. 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 [38]. 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.

Recently, there is a significant increase of data-driven approaches [7, 58] in time-series prediction due to the extensive availability of abundant data from various fields, *e.g.*shopping behaviors of consumers [31, 65], resource usage optimization for cloud computing [22] and energy consumption [24, 48]. 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 the field of multivariate forecasting , the global models have been studied for decades in 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 [31, 76]. 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 [31, 46, 60] are able to capture complex non-linear 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 [33]. Examples include mixed-effects models [20], 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 [66] 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 [64], the local models [12, 36] 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[31, 60, 76] 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. 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 non-linear patterns globally while leveraging a *relational local model* to capture the individual random effects of each time-series locally. GraphDF's relational global model improves the runtime performance and scalability since 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 non-linear 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.

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 non-linear 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 non-linear 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.

2 RELATED WORK

Classical Time-series forecasting A vast variety of forecasting approaches have been developed for its wide applications and usage in various domain [5, 15, 30, 55, 77]. Classical time-series models including autoregressive integrated moving average (ARIMA) and exponential smoothing [32, 54] have demonstrated a huge success in univariate time-series prediction, however, they fail to extract the non-linear relationships across time-series. Besides that, they are incapable of modeling the exogenous values, which usually helps forecasting. By contrast, multivariate time-series prediction [10, 65, 71] takes the advantage of modeling the inter-dependencies across time-series to improve the prediction accuracy. One example of multivariate time-series models is vector autoregression (VAR) [69] which is commonly considered a generalization of autoregressive model. However, VAR treats the relationships across time-series equivalently without difference, which is unrealistic. Deb et al. [21] summarized nine classical methods for forecasting energy usage including artificial neural network (ANN), support vector machine (SVR) and others.

Deep Learning-based Time-series Forecasting In recent years, advances in deep learning have led to substantial improvements [23, 35, 51] in time-series prediction, among which recurrent neural networks (RNNs) received a great extent of popularity [1, 11, 39, 83] due to their significant accuracy in predictions and flexibility to model the non-linear relationships [7]. As prominent examples of RNN model, the long short-term memory units (LSTM) [6, 41] and the gated recurrent units (GRU) [18] are broadly adopted for their competence to overcome the vanishing gradient problem. Based on the LSTM and GRU architectures, sequence-to-sequence models [4, 50, 70] are developed to allow predictions for a modest number of horizons [26, 76].

While typical RNN models target at univariate time-series prediction, a substantial amount of efforts have been made to share information across time-series to model the highly non-linear inter-dependencies and thus improve the forecast accuracy. For instance, Qin et al. [59] proposed a dual stage attention-based RNN model. Huang et al. [42] introduced a dual-attention mechanism for dynamic-period or non-periodic multivariate time-series forecasting. These methods assume all time-series are equally related to each other, which can be seen as having a fully connected graph.

While earlier work focused on *point forecasting* which aims at predicting optimal expected values, there is an increasing interest in *probabilistic forecasting* models [40, 47, 53, 62, 78]. 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. [31] proposed a probabilistic forecasting model that jointly learns a global model from all available time-series. Wang et. al. proposed DF [75], 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 non-linear 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 accuracy 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.

Graph-based models Modeling the unique relations to each individual time-series from others naturally leads us to graph models. For instance, Graph Neural Network(GNN) [14, 43, 44, 84] has recently showed great successes in extracting the information across nodes. Moreover, the combination of GNNs and RNNs [34, 74] allows the injection of dynamism of the pairwise non-linear relationship across time-series. As an early work, Seo et al. [67] introduced graph convolutional recurrent network (GCRN) to predict structured sequential data. Other recent work is mostly limited in spatio-temporal study, such as traffic prediction [82] and ride-hailing demand forecasting [80, 81]. All these methods are incorporated with a graph structure. Besides, these methods are not probabilistic models and they fail to deliver uncertainty estimates.

Resource usage prediction Researchers and engineers put great efforts on resource provisioning and load prediction in cloud scale systems [8, 56, 72]. Early work mainly utilized the traditional state space models such as ARIMA [13, 85]. More recent work covers both traditional methods [13, 85], machine learning approaches [19] such as K-nearest neighbors [29, 68] and linear regression [28, 79] and RNN-based methods [17, 27, 45]. 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, 52], predicting with kernel-based methods [63], and efforts on elastic resource scaling to reduce cloud system operating cost [9, 68] More recent work leverages deep learning on streaming data. For instance, Vrablecová et al. [73] proposed a stream change detection method to identify the ongoing changes or concept drifts of the power meter data. Guo et al. [37] 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 [25] targets finding mismatch of temporal distribution between periods of time-series. However, these models do not leverage graph structures or the inter-correlations between time-series for forecasting. By contrast, our proposed work is graph-based and has the advantage of forecasting accuracy and runtime efficiency.

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 non-linear 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 multi-dimensional time-series data.

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 (described in Section 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 (Section 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. 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:

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

Hence, solving Eq. (1) gives the joint probability distribution over future values given all covariates and past observations along with the graph structure represented by $\mathbf{A}$ 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 $K(\mathbf{z}_i, \mathbf{z}_j) = \exp(-\frac{\|\mathbf{z}_i - \mathbf{z}_j\|^2}{2\ell^2})$, where ℓ is the length scale of the kernel.

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 (Section 3.3), and (2) a relational local random effect model (Section 3.4). As such, GraphDF is a hybrid forecasting framework. Both the relational global component and relational local component of our framework leverage the graph via the specific underlying model used for each component.

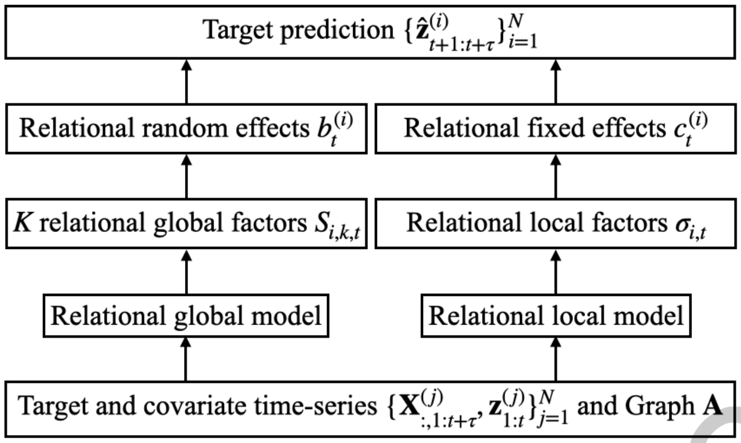

Target prediction $\{\hat{\mathbf{z}}^{(i)}_{t+1:t+\tau}\}_{i=1}^{N}$
Relational random effects $b_t^{(i)}$
Relational fixed effects $c_t^{(i)}$
K relational global factors $S_{i,k,t}$
Relational local factors $\sigma_{i,t}$
Relational global model
Relational local model
Target and covariate time-series $\{\mathbf{X}^{(j)}_{:,1:t+\tau}, \mathbf{z}^{(j)}_{1:t}\}_{j=1}^{N}$ and Graph $\mathbf{A}$

Fig. 2. An overview of GraphDF framework

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 non-linear time-series patterns in the graph-based time-series data,

$$\text{relational global factors:} \quad s_k(\cdot) = \textsc{gcrn}_k(\cdot), \quad k = 1, \ldots, K \tag{2}$$

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. 2, we learn the relational global factors using a Graph Convolutional Recurrent Network (GCRN) [67], 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,

$$\text{fixed effect:} \quad c^{(i)}(\cdot) = \sum_{k=1}^{K} w_{i,k} \cdot s_k(\cdot) \tag{3}$$

where $w_{i,k}$ represents the K-dimensional embedding for node i. Therefore, the final relational non-random 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 Section 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,

$$\text{relational local random effect:} \quad b^{(i)}(\cdot) \sim \mathcal{R}_i, \quad i = 1, \ldots, N \tag{4}$$

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 ensure $b_t^{(i)}$ obeys a normal distribution, and thus can be derived fast. The *relational latent function* of node i denoted as $v^{(i)}$ is then defined as,

$$\text{latent function:} \quad v^{(i)}(\cdot) = c^{(i)}(\cdot) + b^{(i)}(\cdot) \tag{5}$$

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. 3

and its local relational random effect $b^{(i)}$ from Eq. 4. Then,

$$\text{emission:} \quad z_t^{(i)} \sim \mathbb{P}\Big(z_t^{(i)} \mid v^{(i)}\big(\mathbf{A}, \{\mathbf{X}_{:,1:t}^{(j)}, \mathbf{z}_{1:t-1}^{(j)}\}_{j=1}^{N}\big)\Big) \tag{6}$$

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. 2-6. All the functions $s_k(\cdot), b^{(i)}(\cdot), v^{(i)}(\cdot)$ take past observations and covariates $\{\mathbf{z}_{1:t-1}^{(j)}, \mathbf{X}_{:,1:t}^{(j)}\}_{j=1}^{N}$, 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. 2.

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 $\boldsymbol{w}_i$, as shown in Eq. (3).

3.3.1 Learning Relational Global Factors via GCRN. We first show how GCRN [67] 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,

$$\mathbf{Y}_t = \begin{bmatrix} z_{t-1}^{(1)} & \mathbf{x}_t^{(1)} \\ \vdots & \vdots \\ z_{t-1}^{(N)} & \mathbf{x}_t^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times P} \tag{7}$$

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 $f_{\star_{\mathcal{G}}}\,\boldsymbol{\Theta}$ with respect to the graph G and parameters θ:

$$f_{\star_{\mathcal{G}}}\,\boldsymbol{\Theta}(\mathbf{Y}_t) = g_\theta(\mathbf{L})\mathbf{Y}_t \tag{8}$$

$$= \mathbf{U} g_\theta(\boldsymbol{\Lambda}) \mathbf{U}^T \mathbf{Y}_t \in \mathbb{R}^{N \times P} \tag{9}$$

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. $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$ is the eigenvalue decomposition. $\mathbf{U}$ is the matrix composed of eigenvectors by order of eigenvalues of $\mathbf{L}$, and $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{L}$. $g_\theta(\boldsymbol{\Lambda}) = \text{diag}(\boldsymbol{\theta})$ denotes a filter parameterized by the coefficients $\boldsymbol{\theta} \in \mathbb{R}^N$ in the Fourier domain. Directly applying Eq (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

$$g_\theta(\mathbf{L}) = \sum_{l=0}^{L-1} \theta_l T_l(\tilde{\mathbf{L}}), \tag{10}$$

where $\boldsymbol{\theta} = \big[\theta_0 \cdots \theta_{L-1}\big] \in \mathbb{R}^L$ in Eq. (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 $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I} \in \mathbb{R}^{N \times N}$, and 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 multi-dimensional 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 map the dimension P of input to the dimension Q of output:

$$\mathbf{H}_{:,q} = \tanh\left[\sum_{p=1}^{P} f_{\star_\mathcal{G}}\,\mathbf{\Theta}(\mathbf{Y}_{t,\,:,p})\right],\ \text{for}\ q \in 1\ \ldots\ Q \tag{11}$$

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

$$\mathbf{I}_t = \sigma(\mathbf{\Theta}_I \star_\mathcal{G} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_I \odot \mathbf{C}_{t-1} + \mathbf{b}_I) \tag{12}$$
$$\mathbf{F}_t = \sigma(\mathbf{\Theta}_F \star_\mathcal{G} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_F \odot \mathbf{C}_{t-1} + \mathbf{b}_F) \tag{13}$$
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t\text{-}1} + \mathbf{I}_t \odot \tanh(\mathbf{\Theta}_C \star_\mathcal{G} [\mathbf{Y}_t, \mathbf{H}_{t\text{-}1}] + \mathbf{b}_C) \tag{14}$$
$$\mathbf{O}_t = \sigma(\mathbf{\Theta}_O \star_\mathcal{G} [\mathbf{Y}_t, \mathbf{H}_{t-1}] + \mathbf{W}_O \odot \mathbf{C}_t + \mathbf{b}_O) \tag{15}$$
$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{16}$$

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. (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,

$$\mathbf{S}_t = \mathbf{H}_t\mathbf{W} + \mathbf{b} \tag{17}$$

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. (17) that capture the complex non-linear 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

$$c_t^{(i)}(\cdot) = \sum_{k=1}^{K} w_{i,k} \cdot S_{i,k,t} \tag{18}$$

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

3.3.2 Learning Relational Global Factors via DCRNN. For the relational global component of GraphDF, we can also leverage DCRNN [49]. 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,

$$f_{\star_\mathcal{G}}\,\mathbf{\Theta}(\mathbf{Y}_t) = \sum_{l=0}^{L-1}(\theta_l \tilde{\mathbf{A}}^l)\mathbf{Y}_t \tag{19}$$

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 multi-dimensional time-series of the nodes. The Chebyshev polynomial approximation is used similarly as Eq. (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,

$$\mathbf{R}_t = \sigma(\mathbf{\Theta}_R \star_{\mathcal{G}} [\mathbf{Y}_t,\ \mathbf{H}_{t-1}] + \mathbf{b}_R) \tag{20}$$

$$\mathbf{U}_t = \sigma(\mathbf{\Theta}_U \star_{\mathcal{G}} [\mathbf{Y}_t,\ \mathbf{H}_{t-1}] + \mathbf{b}_U) \tag{21}$$

$$\mathbf{C}_t = \tanh(\mathbf{\Theta}_C \star_{\mathcal{G}} [\mathbf{Y}_t,\ (\mathbf{R}_t \odot \mathbf{H}_{t-1})] + \mathbf{b}_C) \tag{22}$$

$$\mathbf{H}_t = \mathbf{U}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{U}_t) \odot \mathbf{C}_t \tag{23}$$

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 as 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. (23), the fixed effect is derived from DCRNN similarly with Eq. (17) and Eq. (18). Compared to the previous GCRN that we adapted for the relational global component, DCRNN is more computationally efficient due to the GRU structure it uses.

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. 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. (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 Section 3.3 that uses the entire graph G along with all the node multi-dimensional time-series to learn K global factors that capture the most important non-linear 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 in GraphDF.

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 Section 3.3, the relational local model focuses on learning an individual local model for each individual node based on its own multi-dimensional time-series data as well as the nodes neighboring it. This enables us to model the local fluctuations of the individual multi-dimensional time-series data 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 for 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. 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.

$$\mathbf{X}_t^{(C)} = \begin{bmatrix} \mathbf{x}_t^{(C_1)} & \mathbf{x}_t^{(C_2)} & \cdots & \mathbf{x}_t^{(C_{|C|})} \end{bmatrix}^\top \tag{24}$$

where C_j denotes the j(th) neighbor. Now, we define the input temporal features of the relational local model for node i as,

$$\mathbf{Y}_t^{(i)} = \begin{bmatrix} z_{t-1}^{(i)} & \mathbf{x}_t^{(i)\top} \\ \mathbf{z}_{t-1}^{(C)} & \mathbf{X}_t^{(C)} \end{bmatrix} \tag{25}$$

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 $\mathbf{L}^{(i)}$.

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

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 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,

$$\sigma_{i,t} = \log\left(\exp(\mathbf{w}^{(i)\top}\mathbf{h}_t^{(i)} + \beta^{(i)}) + 1\right) \tag{26}$$

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.(26),

$$b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2) \tag{27}$$

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.

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. (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.

$$\mathbf{R}_t^{(i)} = \sigma\big(\Theta_R^{(i)} \star_{\mathcal{G}} [\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}] + \mathbf{b}_R^{(i)}\big) \tag{28}$$

$$\mathbf{U}_t^{(i)} = \sigma\big(\Theta_U^{(i)} \star_{\mathcal{G}} [\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}] + \mathbf{b}_U^{(i)}\big) \tag{29}$$

$$\mathbf{C}_t^{(i)} = \tanh\big(\Theta_C^{(i)} \star_{\mathcal{G}} [\mathbf{Y}_t^{(i)}, \mathbf{H}_t^{(i)}] + \mathbf{b}_C^{(i)}\big) \tag{30}$$

$$\mathbf{H}_t^{(i)} = \mathbf{U}_t^{(i)} \odot \mathbf{H}_{t-1}^{(i)} + (1 - \mathbf{U}_t^{(i)}) \odot \mathbf{C}_t^{(i)} \tag{31}$$

where $\Theta_R^{(i)}, \Theta_U^{(i)}, \Theta_C^{(i)} \in \mathbb{R}^{P\times R}$ 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 vector 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 multi-dimensional 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. (26) and Eq. (27).

Algorithm 1 Training Graph Deep Factor Models

```
1: Initialize the model parameters $\mathbf{\Phi}$ ($\mathbf{W}_*, \mathbf{b}_*, \Theta_*$, 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. 3.3
5:         Calculate the relational local random effect, $b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2)$ described in Sec. 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.
```

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 model, as well as the parameters in the embeddings. We leverage the maximum likelihood estimation,

$$\mathbf{\Phi} = \operatorname{argmax} \sum_i \mathbb{P}\big(\mathbf{z}^{(i)} \big| \mathbf{\Phi}, \mathbf{A}, \{\mathbf{X}_{:,1:t}^{(j)}, \mathbf{z}_{1:t-1}^{(j)}\}_{j=1}^{N}\big) \tag{32}$$

where

$$\mathbb{P}(\mathbf{z}^{(i)}) = \sum_t -\frac{1}{2}\ln(2\pi\sigma_{i,t}) - \sum_t \frac{(z_t^{(i)} - c_{i,t})^2}{2\sigma_{i,t}^2} \tag{33}$$

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. (33). We describe a general training procedure in Algorithm 1.

3.6 Model Variants

In this section, we define a few of the GraphDF model variants investigated in Section 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. 3.3) and the probabilistic local graph component from Sec. 3.4.1 as the relational local model.
- **GraphDF-GR:** This model variant from the GraphDF framework uses the GCRN from Sec. 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. 3.4.1 as the relational local model.

The GraphDF framework is flexible with many interchangeable components. Importantly, the relational global component (Sec. 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. 3.4) for obtaining the relational local random effects of the nodes.

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 Google cloud dataset [61] that we used, the time interval is five minutes, which means we have new time-series observations for all N nodes 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 values arrive in the stream. One solution could be to retrain a new GraphDF model from scratch each time new values arrive, however, it will take 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 arrive point. By doing this, the IOGraphDF modeling operates much more efficiently.

Denote t 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 $\mathbf{X}^{(i)} \in \mathbb{R}^{D\times t}$, where D is the number of dimension for covariate features. We define the set of target time-series is $\mathcal{Z}_t = \{\mathbf{z}^{(i)}\}_{i=1}^N$ where $\mathbf{z}_{1:t}^{(i)}$ denotes the i(th) univariate target time-series. Given $(\mathcal{X}_t, \mathcal{Z}_t)$ 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 as:

$$\mathbb{P}\left(\left\{\mathbf{z}_{t+1:t+\tau}^{(i)}\right\}_{i=1}^N \middle| \mathbf{A}, \left\{\mathbf{z}_{1:t}^{(i)}, \mathbf{X}_{:,1:t+\tau}^{(i)}\right\}_{i=1}^N; \mathbf{\Phi}\right) \qquad t = 1, 2, \cdots \tag{34}$$

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 2.

Algorithm 2 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:      Perform incremental update to relational global model
5:      for each time series pair $(\mathbf{z}^{(i)}, \mathbf{x}^{(i)})$ do
6:          With current model parameter estimates $\mathbf{\Phi}$
7:              Calculate the relational fixed effect $c_t^{(i)} = \sum_{k=1}^K w_{i,k} \cdot S_{i,k,t}$ described in Sec. 3.3
8:              Calculate the relational local random effect $b_t^{(i)}(\cdot) \sim \mathcal{N}(0, \sigma_{i,t}^2)$ described in Sec. 3.4
9:          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
10:     end for
11: end while
12: Compute the overall loss in the current batch and perform SGD and update the trainable parameters $\mathbf{\Phi}$
    accordingly.
```

Now we analyze the time complexity of IOGraphDF, when a single value arrives at time t, the worst-case time complexity is

$$O\left((|E| \cdot KLkN + K) + LC_{\max}kN\right) \tag{35}$$

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. 35) which is timewise negligible, therefore, the time complexity of the relational global component $O\left(|E| \cdot KLkN + K\right)$ is approximately linear to all the aforementioned values.

Further in the time complexity $O\left(LC_{\max}kN\right)$ for the relational local component, $C_{\max}$ is the maximum node degree of the graph, and thus Eq. 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 multi-step ahead prediction scenario, Eq. 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, $O\left(M \cdot ((|E| \cdot KLTN + K) + LC_{\max}TN)\right) \gg O\left((|E| \cdot KLkN + K) + LC_{\max}kN\right)$. 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 $O(TN)$ space while the incremental online GraphDF requires only $O(kN)$ space where w is the most recent w values in the stream. Hence, since $k \ll T$, then $O(kN) \ll O(TN)$. 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.

5 EXPERIMENTS

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

RQ1. Does GraphDF outperform the state-of-the-art deep probabilistic forecasting method?
RQ2. Are the GraphDF models fast and scalable for large-scale time-series forecasting?
RQ3. Can GraphDF generate cloud usage forecasts to effectively perform opportunistic workload scheduling?

Table 1. Statistics of the two real-world large-scale collections of time-series.

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

5.1 Experimental Setup

We used two real-world datasets in our experiments; Google trace data and Adobe trace data. Table 1 shows the statistics and properties (*e.g.*edge density, average degree, etc.).

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 Section 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 the models are trained fast within minutes, 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 $\boldsymbol{w}_i \in \mathbb{R}^K$ and use time feature series as covariates. We set the embedding dimension to $K = 10$ in $\boldsymbol{w}_i \in \mathbb{R}^K$ and used $D = 5$ covariates for each time-series. Similar to DF [75], the time features (*e.g.*minute of hour, hour of day) are used as covariates. We derive a fixed graph using Radial Based Function (RBF) on the past observations.

The three models described in Section 3.6 are evaluated against four state-of-the-art probabilistic forecasting methods including Deep Factors, DeepAR [31], MQRNN [76], and NBEATS [57]:

- *Deep Factors* is a generative approach that combines a global model and a local model. To ensure fair comparison, we modified DF to solve the same problem formulated in Eq. 1, and thus the DF version used 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.

Table 2. Results for one-step ahead forecasting (p10ql, p50ql and p90ql).

	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	**0.037±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	**0.118±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	**0.072±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	**0.188±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	**0.041±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	**0.153±0.001**	0.169±0.001	0.342±0.037

Table 3. Results for multi-step ahead forecasting (p10ql).

DATA	H	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
	3	0.652±0.396	0.152±0.006	0.070±0.000	0.132±0.004	**0.064±0.001**	0.077±0.000	0.087±0.000
Google	4	0.260±0.017	0.272±0.018	0.138±0.000	0.193±0.016	**0.071±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	**0.054±0.000**	0.113±0.001	0.088±0.001
	3	0.811±0.295	0.184±0.003	0.207±0.002	0.303±0.006	**0.183±0.002**	0.216±0.008	0.267±0.009
Adobe	4	0.985±0.537	0.219±0.008	0.273±0.003	0.313±0.018	**0.184±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	**0.251±0.016**	0.298±0.031	0.544±0.020

Table 4. Results for multi-step ahead forecasting (p50ql).

DATA	H	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
	3	0.741±0.050	0.257±0.011	0.148±0.001	0.400±0.004	**0.091±0.001**	0.134±0.002	0.097±0.000
Google	4	0.618±0.105	0.410±0.017	0.191±0.000	0.454±0.007	**0.097±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	**0.126±0.001**
	3	1.683±0.100	0.556±0.028	0.592±0.017	1.116±0.006	**0.272±0.004**	0.315±0.006	0.319±0.005
Adobe	4	1.424±0.210	0.574±0.011	0.629±0.024	1.029±0.001	**0.314±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	**0.375±0.007**	0.401±0.014	0.484±0.005

Table 5. Results for multi-step ahead forecasting (p90ql).

DATA	H	NBEATS	MQRNN	DeepAR	DF	GraphDF-GG	GraphDF-GR	GraphDF-RG
	3	0.830±0.262	0.091±0.001	0.067±0.000	0.208±0.002	**0.051±0.000**	0.051±0.000	0.089±0.000
Google	4	0.976±0.429	0.090±0.000	0.070±0.000	0.213±0.002	**0.050±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	**0.069±0.001**	0.167±0.013	0.094±0.001
	3	2.556±0.328	0.317±0.002	0.751±0.117	1.545±0.008	**0.248±0.004**	0.254±0.003	0.301±0.003
Adobe	4	1.862±1.212	0.335±0.004	0.696±0.170	1.673±0.008	**0.317±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	**0.410±0.021**	0.434±0.007	0.512±0.007

- *MQRNN* is a sequence model with quantile regression and NBEATS is an interpretable pure deep learning model. For MQRNN, we use a GRU bidirectional layer with 50 hidden units as encoder and a modified forking layer in decoder.
- For *N-BEATS*, we use an ensemble modification of the model and take the median value from 10 bagging bases as results.

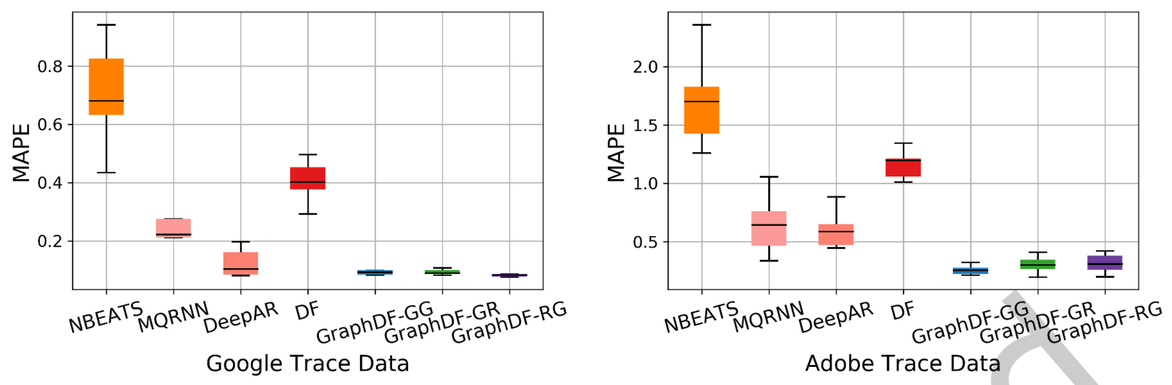


Fig. 3. Probabilistic forecasting results for 3-step ahead forecast horizon (P50QL) from Adobe and Google trace dataset.

All methods are implemented using MXNet Gluon [2, 16]. 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, minimum learning rate of $5 * 10^{-5}$, Xavier as the weight initializer, and trained 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. (10). A small number of the order indicates that the model makes forecasts based more on neighboring nodes than those more distant. For other methods, we use default hyperparameters given by the Gluonts implementation if not otherwise mentioned.

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

$$\mathrm{QL}_\rho[\mathbf{z}_t, \widehat{\mathbf{z}}_t(\rho)] = 2\big[\rho(\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho))\mathbb{I}_{\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho) > 0} + (1-\rho)(\widehat{\mathbf{z}}_t(\rho) - \mathbf{z}_t)\mathbb{I}_{\mathbf{z}_t - \widehat{\mathbf{z}}_t(\rho) \leqslant 0}\big] \tag{36}$$

For deriving 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 experiments, the quantile losses are computed based on 100 sample values. Our algorithms are implemented in MXNet Gluon [16] and all experiments run on a machine with 8 CPU cores.

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 multi-step ahead forecasting are provided in Table 2 and Table 3-5, respectively, where the best result 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 the GraphDF models outperform the state-of-the-art method across all datasets and forecast horizons. Furthermore, we observe that in most cases, the GraphDF-GG variant that uses GCRN for both the relational global and relational local component outperforms the other variants. The second best GraphDF 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. 3. Strikingly, we observe that the GraphDF models provide more accurate forecasts with significantly lower variance.

Table 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	**222.08±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	**211.20±21.56**	264.00±56.29

Table 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	**0.99±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	**0.28±0.001**	0.33±0.000

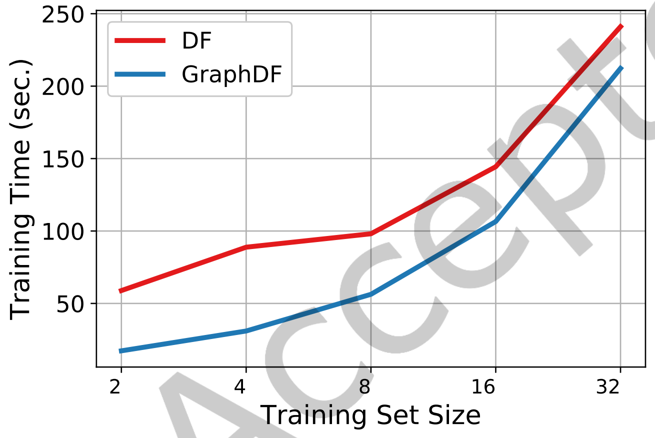


Fig. 4. Comparing scalability of GraphDF to DF as the training set size increases for the Adobe workload trace dataset. Note training set size = data points per time-series.

5.3 Runtime Analysis

Training and inference runtime performance results are shown in Table 6 and Table 7, respectively. All forecasting models are trained using only six previous values for each time-series in the collection (Table 6). As expected, the relational global model is significantly faster and more scalable than the global model used in Deep Factors. In particular, we see 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 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 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 across both Google and Adobe workload datasets.

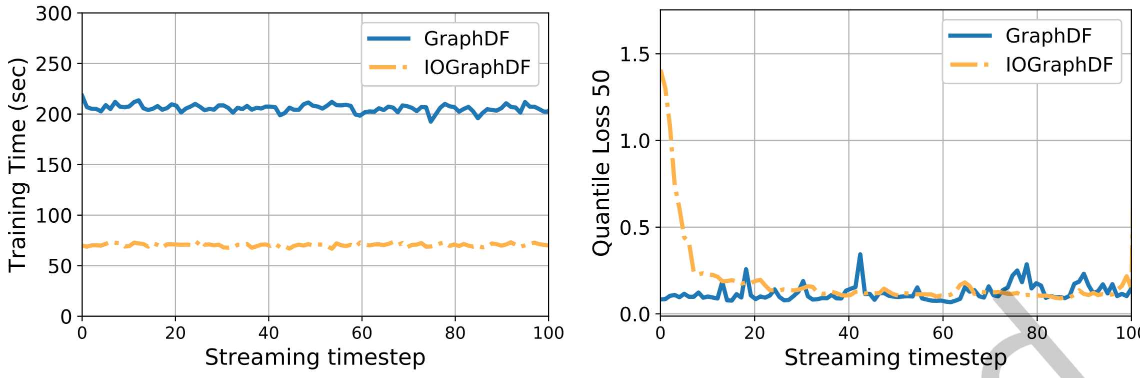


Fig. 5. Comparing training runtime and one step ahead P50QL (*i.e.*MAPE) between GraphDF (blue) and IOGraphDF(yellow dashed line) on Google dataset.

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. 4 shows that 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 3x faster compared to DF across all training set sizes.

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 model, 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 just arrived values. By contrast, IOGraphDF model only needs to be initialized once at the beginning, and the same IOGraphDF instance is incrementally updated from the previous time step with newly arrived data. Incoming values are assumed to be more dependent on near observations than those in the distant past, therefore, we maintain a shifting window size of 9 to include the most recent observations relative to t. When t increments, the oldest values are removed from the window and newly arrived values are appended to the window. The window values are split sets for data training and fitting procedure. Using the Google dataset, for GraphDF model, the number of training epochs is set as 100, while for IOGraphDF model, the number of local iterations (the training times upon each shifting window) only needs to be set as a smaller number 50, because after the model is initialized, the existed model preserve learned information from previous time steps. In the inference stage, values in $\tau = 3$ steps ahead are predicted and evaluated with ground truth.

Experimental results. Overall, we observe from Fig. 5 (left) and Table 8 that IOGraphDF is significantly faster, with very comparable loss (as shown in Fig. 5 (right) and Table 8). Hence, IOGraphDF sacrifices a small amount of accuracy for a significant speedup.

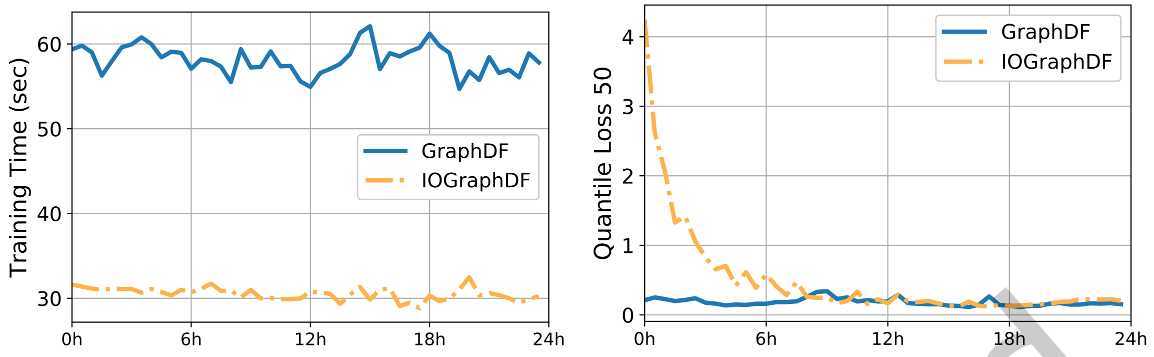


Fig. 6. Comparing training runtime and one step ahead P50QL between GraphDF (blue) and IOGraphDF(yellow dashed line) on Adobe dataset.

We further conducted experiments on 100 timesteps using the best variants GraphDF-GG and IOGraphDF-GG. As observed in Fig. 5 (Right), the p50QL (*i.e.*MAPE) of GraphDF (in blue) and IOGraphDF (yellow dashed line) 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. 5 are mostly overlapping after streaming timestep 20.

Table 8. Results for multi-step ahead forecasting (Runtime) with IOGraphDF.

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

Table 9. Results for multi-step ahead forecasting (p50ql) with IOGraphDF.

DATA	H	GraphDF-GG	GraphDF-GR	GraphDF-RG	IOGraphDF-GG	IOGraphDF-GR	IOGraphDF-RG
	3	**0.091**±**0.001**	0.134±0.002	0.097±0.000	0.104±0.006	0.146±0.007	0.141±0.013
Google	4	**0.097**±**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	**0.122**±**0.006**	0.164±0.010	0.171±0.010
	3	0.272±0.004	0.315±0.006	0.319±0.005	**0.211**±**0.023**	0.328±0.039	0.371±0.042
Adobe	4	0.314±0.004	0.353±0.007	0.405±0.007	**0.240**±**0.018**	0.332±0.057	0.393±0.056
	5	0.375±0.007	0.401±0.014	0.484±0.005	**0.286**±**0.023**	0.341±0.083	0.404±0.059

In both offline and online models, the expected runtime on training are positively related to the number of values on input and output. In the offline GraphDF model, each time new values arrived, a new model has to be created and trained using the new values, and only recent values are taken as input, hence, this causes the loss of

Table 10. Runtime comparison between GraphDF and IOGraphDF over timesteps

Data	GraphDF	IOGraphDF
Google	208.770 ± 11.862	**76.948 ± 22.717**
Adobe	58.170 ± 1.666	**30.534 ± 0.718**

Algorithm 3 Opportunistic Scheduling

```
1: Initialize hyperparameters and variables lookback window for GraphDF (or streaming window for IOGraphDF) $w$, horizon $\tau$, threshold ratio $\epsilon$, portion ratio $\lambda$. Accumulated utilization improvement $Acc$
2: while New observations arrive $t \leftarrow t+1$ do
3:     Initialize weights $\Phi$ for new model $f_t$
4:     $f_t \leftarrow \mathbb{P}\big(\cdot \,\big|\Phi, \mathbf{A}, \{\mathbf{X}^{(i)}_{:,t-w+1:t+\tau}, \mathbf{z}^{(i)}_{t-w+1:t}\}_{i=1}^{N}\big)$
5:     for each node $i$ do
6:         Obtain forecasts $\hat{z}^{(i)}=\{\hat{z}^{(i)}_{t+1}, \hat{z}^{(i)}_{t+2} \ldots \hat{z}^{(i)}_{t+\tau}\}\sim f_t$
7:         if MEAN of forecasts MEAN$(\hat{z}^{(i)}) \leq \epsilon$ then
8:             Utilization $Acc \leftarrow Acc + \lambda(1 - \text{MEAN}(\hat{z}^{(i)}))$
9:         else
10:            Cancel assigned tasks on node $i$
11:        end if
12:    end for
13: end while
```

earlier observations as possibly useful information. 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, a small number 50, which results a much faster IOGraphDF while still preserving a high accuracy. The runtime comparison between GraphDF and IOGraphDF over 100 timesteps is shown in Fig. 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. We also summarize average and deviation runtime in Table 10.

Following the setup in Section 5.2, we design experiments to investigate the performance of IOGraphDF model variants over multi-step ahead prediction, and compare 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 used to train the IOGraphDF models, but not used for loss comparison. The values after the warm period are then used to evaluate and compare the prediction loss with the GraphDF variants. The result is shown in Table. 9. We observe that on the Google dataset, IOGraphDF variants reach very close result to the GraphDF counterparts, while IOGraphDF models only requires significantly less training runtime than GraphDF models, as shown in Table 8. For the Adobe dataset, IOGraphDF models not only require less runtime, but also obtain more accurate prediction with smaller loss in most cases (highlighted in the column IOGraphDF-GG).

5.6 Ablation Study

We further investigate the effects of hyperparameters upon IOGraphDF model with extensive experiments. The length of warm start period and the number of relational global factors K are selected from the range

Table 11. Results on combination of hyperparameters for 3-step ahead forecasting (p50QL) with Google dataset, the best performance for each number of warm start period (row) is highlighted in bold.

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

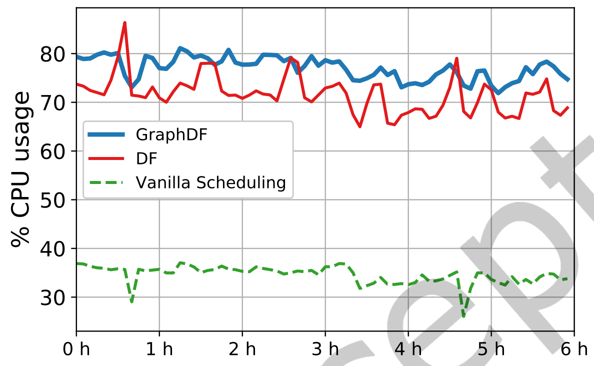


Fig. 7. CPU utilization without opportunistic workload scheduling (shown in green) and with scheduling based on each forecaster (shown in red and blue), over a period of 6 hours on Google dataset. GraphDF-based scheduling leads to higher CPU utilization than DF and vanilla scheduling.

of {5, 10, 20, 50}. We report p50QL for forecasting 3-step ahead on Google dataset with combinations of these hyperparameters and the result is shown in Table 11. From the result table, we observe (1) When the number of relational global factor K is fixed, i.e., for each column of Table 11, the longer the warm start period is, the better performance IOGraphDF achieves. (2) When the number of warm start period is small, the performance is drastically improved as the number of relational global factors K increases, as shown in the first two rows of Table 11, and the best performance is achieved when $K = 50$. 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 11. This can be caused due to the excessive amount of model parameters which leads to overfitting.

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 scheduled on underutilized resources (such as CPU cores) opportunistically. 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.

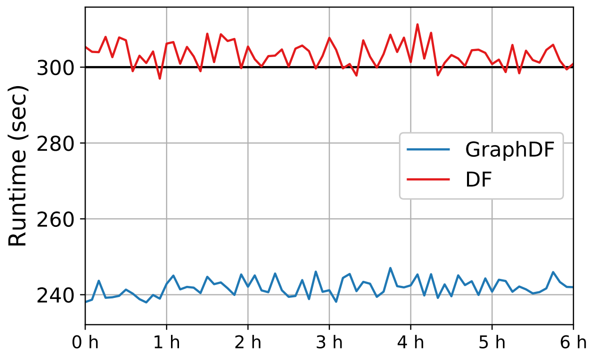


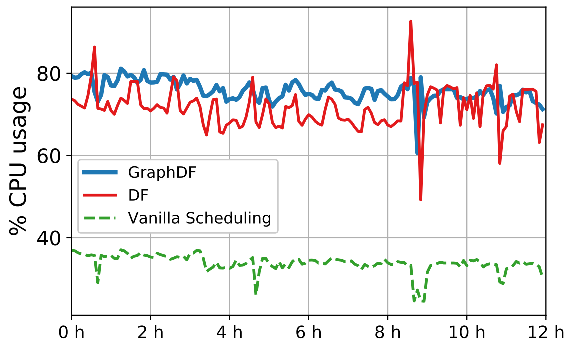
Fig. 8. The time constraint (black line), the runtime of scheduler with DF (red line) and that with GraphDF (blue line). Note that in most cases, DF fails to meet the time constraint while GraphDF produces a forecast much faster.

Table 12. Results for opportunistic scheduling in the cloud over a 6 hour period 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**

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 then 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, *e.g.*, since CPU usage in 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 assigned batch workloads that exceeds resource availability, they are *terminated/canceled*. This procedure is described in Algorithm 3.

Effects on CPU utilization. Fig. 7 shows CPU utilization without opportunistic workload scheduling *(vanilla)* and with scheduling based on each forecaster over a period of 6 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. We also observed similar results (in Fig. 9) over longer periods (12 hours and 24 hours) for Google dataset. Table 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. Correct scheduling ratio corresponds to the ratio when predicted utilization by the scheduler matches the actual utilization. Cancellation ratio



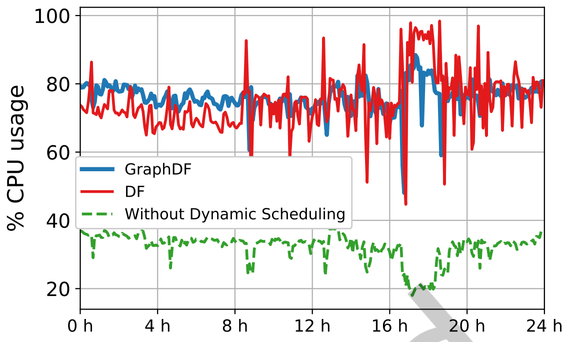


Fig. 9. Using the Google dataset, we simulate a scheduler over 12 hours and 24 hours(144, 288 timestamps). The lag is selected as 6 and horizon is set as 3. The figure shows improvements over baseline (no forecaster) on CPU utilization by using DF and GraphDF as forecasters. For nodes whose average predicted CPU usage in horizon are lower than a threshold 20%, the scheduler will assign tasks that consume 80% CPU resources to the nodes. The overall average utilization improvement of DF is 37.8%, 43.8%. The overall average utilization improvement of GraphDF is 41.9%, 42.6%.

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 utilization, higher correct scheduling ratio, and lower cancellation ratio compared to DF-based scheduling.

Execution Time Comparison. Fig. 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.

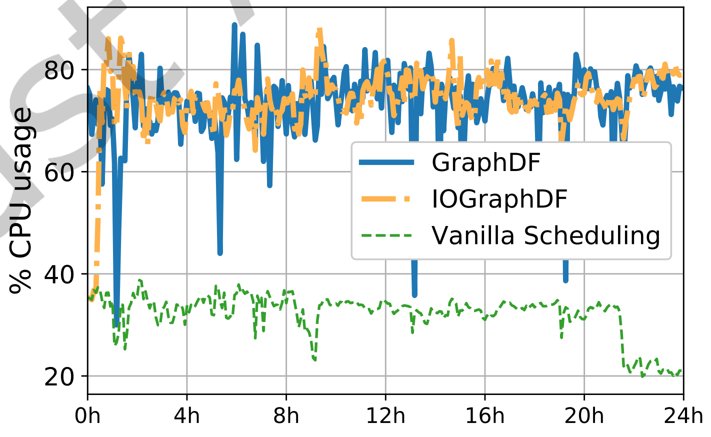


Fig. 10. CPU utilization without opportunistic workload scheduling (shown in green) and with scheduling based on each forecaster (shown in blue and yellow), over a period of 6 hours on Google dataset.

Table 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**

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. 10. We observe that scheduling based on IOGraphDF performs worse than GraphDF at the beginning, which is due to insufficient training of IOGraphDF model, however, over time IOGraphDF give close performance to the GraphDF model. Also, IOGraphDF gives more smooth scheduling curve than 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 13.

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 non-linear 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. 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.

REFERENCES

[1] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. 2010. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews* 29, 5-6 (2010), 594–621.

[2] 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. 2020. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research* 21, 116 (2020), 1–6. http://jmlr.org/papers/v21/19-820.html

[3] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. 2013. Online Learning for Time Series Prediction. In *Proceedings of the 26th Annual Conference on Learning Theory (Proceedings of Machine Learning Research, Vol. 30)*, Shai Shalev-Shwartz and Ingo Steinwart (Eds.). PMLR, Princeton, NJ, USA, 172–184. https://proceedings.mlr.press/v30/Anava13.html

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

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

[6] Kasun Bandara, Christoph Bergmeir, and Hansika Hewamalage. 2021. LSTM-MSNet: Leveraging Forecasts on Sets of Related Time Series With Multiple Seasonal Patterns. *IEEE Transactions on Neural Networks and Learning Systems* 32, 4 (2021), 1586–1599. https://doi.org/10.1109/TNNLS.2020.2985720

[7] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. 2020. Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240* (2020).

[8] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. 2020. Toward ML-Centric Cloud Platforms. *Commun. ACM* 63, 2 (Jan. 2020), 50–59.

[9] Albert Bifet and Ricard Gavaldà. 2009. Adaptive Learning from Evolving Data Streams. In *Advances in Intelligent Data Analysis VIII*, Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 249–260.

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

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

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

[13] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. 2015. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing* 3, 4 (2015), 449–458.

[14] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Conguri Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. 2021. Spectral temporal graph neural network for multivariate time-series forecasting. *arXiv preprint arXiv:2103.07719* (2021).

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

[16] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[17] Y. Cheng, C. Wang, H. Yu, Y. Hu, and X. Zhou. 2019. 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)*. 1249–1256.

[18] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

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

[20] Michael J Crawley. 2013. Mixed-effects models. The R book Second edition.

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

[22] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 127–144.

[23] Jinliang Deng, Xiusi Chen, Renhe Jiang, Xuan Song, and Ivor W Tsang. 2021. 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*. 269–278.

[24] Ilias Dimoulkas, Peyman Mazidi, and Lars Herre. 2019. Neural networks for GEFCom2017 probabilistic load forecasting. *International Journal of Forecasting* 35, 4 (2019), 1409–1423.

[25] Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, and Chongjun Wang. 2021. Adarnn: Adaptive learning and forecasting of time series. *arXiv preprint arXiv:2108.04443* (2021).

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

[27] W. Fang, Z. Lu, J. Wu, and Z. Cao. 2012. RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center. In *2012 IEEE Ninth International Conference on Services Computing*. 609–616.

[28] F. Farahnakian, P. Liljeberg, and J. Plosila. 2013. 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*. 357–364.

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

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

[31] Valentin Flunkert, David Salinas, and Jan Gasthaus. 2017. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *ArXiv* abs/1704.04110 (2017).

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

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

[34] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. 2019. Spatiotemporal Multi-Graph Convolution Network for Ride-Hailing Demand Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019).

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

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

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

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

[39] T. Hastie, R. Tibshirani, and J.H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

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

[41] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* (1997).

[42] Siteng Huang, Donglin Wang, Xuehan Wu, and Ao Tang. 2019. DSANet: Dual Self-Attention Network for Multivariate Time Series Forecasting. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2129–2132.

[43] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[44] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

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

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

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

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

[49] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations (ICLR '18)*.

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

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

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

[53] Danielle C Maddix, Yuyang Wang, and Alex Smola. 2018. Deep factors with gaussian processes for forecasting. *arXiv:1812.00098* (2018).

[54] Spyros Makridakis and Michele Hibon. 1997. ARMA Models and the Box–Jenkins Methodology. *Journal of Forecasting* 16, 3 (1997).

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

[56] Shanka Subhra Mondal, Nikhil Sheoran, and Subrata Mitra. 2021. Scheduling of Time-Varying Workloads Using Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 9000–9008.

[57] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2019. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437* (2019).

[58] Fotios Petropoulos, Daniele Apiletti, Vassilios Assimakopoulos, Mohamed Zied Babai, Devon K Barrow, Christoph Bergmeir, Ricardo J Bessa, John E Boylan, Jethro Browell, Claudio Carnevale, et al. 2020. Forecasting: theory and practice. *arXiv preprint arXiv:2012.03854* (2020).

[59] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. 2017. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971* (2017).

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

[61] Md. Rasheduzzaman, Md. Amirul Islam, and Rashedur M. Rahman. 2014. Workload Prediction on Google Cluster Trace. *Int. J. Grid High Perform. Comput.* 6, 3 (July 2014), 34–52.

[62] Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs Bergmann, and Roland Vollgraf. 2020. Multivariate probabilistic time series forecasting via conditioned normalizing flows. *arXiv preprint arXiv:2002.06103* (2020).

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

[64] Ryan A. Rossi. 2018. Relational Time Series Forecasting. *Knowledge Engineering Review (KER)* 33 (2018), e1.

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

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

[67] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2016. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. *arXiv* (2016).

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

[69] James Stock and M.W. Watson. 2001. Vector Autoregressions. *Journal of Economic Perspectives* (2001).

[70] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc.

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

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

[73] Petra Vrablecová, Viera Rozinajová, and Anna Bou Ezzeddine. 2017. Incremental adaptive time series prediction for power demand forecasting. In *International Conference on Data Mining and Big Data*. Springer, 83–92.

[74] Bao Wang, Xiyang Luo, Fangbo Zhang, Baichuan Yuan, Andrea L Bertozzi, and P Jeffrey Brantingham. 2018. Graph-based deep modeling and real time forecasting of sparse spatio-temporal data. *arXiv preprint arXiv:1804.00684* (2018).

[75] Yuyang Wang, Alex Smola, Danielle C Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. 2019. Deep factors for forecasting. *arXiv preprint arXiv:1905.12417* (2019).

[76] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. 2017. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053* (2017).

[77] Peter R Winters. 1960. Forecasting sales by exponentially weighted moving averages. *Management science* 6, 3 (1960), 324–342.

[78] Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, WEI Ying, and Junzhou Huang. 2020. Adversarial Sparse Transformer for Time Series Forecasting. (2020).

[79] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16, 1 (2014), 7–18.

[80] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction. In *2019 AAAI Conference on Artificial Intelligence (AAAI'19)*.

[81] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Li Zhenhui. 2018. Deep Multi-View Spatial-Temporal Network for Taxi Demand Prediction. In *The Thirty-Second AAAI Conference on Artificial Intelligence*.

[82] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization.

[83] G Peter Zhang and Min Qi. 2005. Neural network forecasting for seasonal and trend time series. *European journal of operational research* 160, 2 (2005), 501–514.

[84] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. 2020. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 841–850.

[85] Qazi Zia Ullah, Shahzad Hassan, and Gul Muhammad Khan. 2017. Adaptive resource utilization prediction system for infrastructure as a service cloud. *Computational intelligence and neuroscience* 2017 (2017).