Dyn-GWN: Time-Series Forecasting using Time-varying Graphs with Applications to Finance and Traffic Prediction

Shibal Ibrahim*
Massachusetts Institute of Technology
Cambridge, MA, USA
shibal@mit.edu

Max Tell*
Massachusetts Institute of Technology
Cambridge, MA, USA
maxtell@mit.edu

Rahul Mazumder
Massachusetts Institute of Technology
Cambridge, MA, USA
rahulmaz@mit.edu

ABSTRACT

Spatio-temporal modeling is an essential lens to understand many real-world phenomena from traffic to finance. There has been exciting work that explores spatio-temporal modeling with temporal graph convolutional networks. Often these methods assume that the spatial structure is static. We propose a new model *Dyn-GWN* for spatio-temporal learning from time-varying graphs. Our model relies on a novel module called the Tensor Graph Convolutional Module (TGCM), which captures dynamic trends in graphs effectively in the time-varying graph representations. This module has two components: (i) it applies temporal dilated convolutions both on the time-varying graph adjacency space and the time-varying features. (ii) it aggregates the higher-level latent representations from both time-varying components through a proposed layer TGCL. Experiments demonstrate the efficacy of these model across time-series data from finance and traffic domains. *Dyn-GWN* can give up to $9\% - 60\%$ better out-of-sample performance than prior methods that learn from time-varying graphs, e.g., EvolveGCN and TM-GCN. Interestingly, *Dyn-GWN* can be $\sim$ 300× faster than EvolveGCN, which is the more competitive baseline from state-of-the-art models that cater to time-varying graphs.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Mathematics of computing** → **Time series analysis**.

KEYWORDS

Spatio-temporal modeling, Time-series forecasting, Graph neural networks, Time-varying graphs.

ACM Reference Format:
Shibal Ibrahim, Max Tell, and Rahul Mazumder. 2023. *Dyn-GWN*: Time-Series Forecasting using Time-varying Graphs with Applications to Finance and Traffic Prediction. In *4th ACM International Conference on AI in Finance (ICAIF '23), November 27–29, 2023, Brooklyn, NY, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3604237.3626864

*Both authors contributed equally to this research.

This work is licensed under a Creative Commons Attribution International 4.0 License.

ICAIF '23, November 27–29, 2023, Brooklyn, NY, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0240-2/23/11.
https://doi.org/10.1145/3604237.3626864

1 INTRODUCTION

Graphs are popular data structures that are effective at compactly representing relationships between entities in structured domains. A rich body of literature have explored graph-based representation learning for a range of applications in the context of time-series forecasting — see [20, 31, 36, 37, 41, 44–46] among others. These works encompass inherently spatial problems such as traffic forecasting [31, 46], power networks [44], and banking links [41]. However, this same framework also extends to more abstract correlation structures in finance, e.g., stock prediction [11], currency exchange rates [44] etc. Thus, effective tools in this domain present promising opportunities to advance the state-of-the-art forecasting performance across these and many more applied problems.

These works employ (temporal) graph neural networks (GNNs) that extend convolutional neural networks to irregular graph domains. Most of these works fundamentally rely on a (given) static graph [31, 45, 46]. In many real-world scenarios, the graph structure can evolve over time e.g., social networks [5], detecting fraud and crime in financial networks [37], and analyzing contact tracing data [34]. It is important to capture such evolutions in graphs into the learnt graph representations for improved generalization. There has been limited work in models that learn from dynamically changing graphs [34, 37, 39]. These models are either prohibitively slow and/or suffer from poor generalization performance — as our experiments show.

We introduce a novel time-series forecasting model, which has the following desirable properties: (i) it can learn from time-varying external graphs (ii) it has superior out-of-sample generalization than state-of-the-art approaches that exploit dynamic graphs, e.g., EvolveGCN [37] and TM-GCN [34] (iii) it is computationally efficient. Our model relies on a novel module called the Tensor Graph Convolutional Module (TGCM), which captures dynamic trends in graphs effectively in the time-varying graph representations. It applies temporal convolutions on the time-varying adjacency space and later effectively aggregates the temporal latent representations from time-varying features and time-varying graphs. We consider finance and traffic forecasting applications and show that that TGCM makes *Dyn-GWN* achieve up to 30% better out-of-sample generalization than prior models that rely on dynamic graphs e.g., EvolveGCN [37] and TM-GCN [34]. Notably, our models can be significantly faster than state-of-the-art EvolveGCN model in the applications we consider.

Interestingly, our module TGCM is flexible that it can adapt many existing state-of-the-art (temporal) GNN architectures that have previously relied on only static graphs, e.g., Spatio-Temporal Graph Convolutional Networks (STGCN) [46], Graph WaveNet (GWN) [45], MTGNN [44]. We demonstrate the usefulness of TGCM in the

context of *Dyn-GWN*— *Dyn-GWN* is our novel adaptation of GWN [45] to handle dynamic graphs.

Contributions. Our contributions can be summarized as follows:

(1) We propose *Dyn-GWN*: a novel adaptation of GWN to learn from time-varying external graphs.
(2) The adaptation relies on a novel Tensor Graph Convolutional Module (TGCM). This module has two components: (i) it applies temporal dilated convolutions on the time-varying graph adjacency space and the time-varying features — existing work only applies temporal convolutions on the feature space. (ii) it aggregates the higher-level latent representations from both time-varying components through a proposed layer `TGCL`.
(3) Interestingly, in comparison with state-of-the-art baselines that rely on external time-varying graphs e.g., EvolveGCN and TM-GCN, our proposed model *Dyn-GWN* can give up to 30% better out-of-sample performance on finance and traffic prediction datasets.
(4) *Dyn-GWN* is $\sim 300\times$ faster than EvolveGCN (more competitive baseline from dynamic graph models).

2 RELATED WORK

Spatio-Temporal Models with Static Graphs. Recent work has focused on developing models that can leverage a static structure. Yu et al. [46] propose STGCN, which applies graph convolutions to the graph adjacency and 1D causal convolutions to the temporal axis. Li et al. [31] leverage diffusion convolution to capture structure dependencies in their DCRNN architecture. Both these approaches have become foundational in this domain, sparking a series of models, improving upon both spatial and temporal aspects of these models. See Table 1 (and description) in [25] for an extensive overview.

Notable among these later models are Graph WaveNet [45] and MTGNN [44]. Wu et al. [45] builds on the work of Li et al. [31] using diffusion convolutions, as well as learning a "self-adaptive" adjacency matrix to capture hidden dependencies which are not represented in the original adjacency. To capture temporal relationships, they apply 1D dilated convolutions to capture long-range relationships. With MTGNN, Wu et al. [44] consider applications where the graph structure is not known a priori. They focus on learning the graph structure from purely time-series data. Their approach uses similar 1D convolutions to the approach by Yu et al. [46], but also employees MixHop-based [1] graph convolution layers which promise greater representation power by learning weights for a mixture of adjacency matrix powers. Graph WaveNet has been independently validated to be a state-of-the-art method in time-series forecasting — See Table 5 in survey by Jiang et al. [25]. None of these methods consider time-varying graphs, as the one we consider in this work.

Spatio-Temporal Models with Time-varying Graphs. Next, we summarize works that consider dynamic graphs in their modeling approaches. With EvolveGCN, Pareja et al. [37] learn from dynamic graphs through a recurrent architecture which "evolves" the GCN weights based on the past weights (at previous time-steps) and/or the current node embeddings. Malik et al. [34] extend the graph convolution paradigm to 3D tensors that explicitly incorporate a time dimension. With TM-GCN, their work replicates the functional form of graph convolutions while redefining the convolution function using Tensor-M products.

Table 1: Summary of prominent graph convolution models for time-series data

Model	Graph Type	Training Time	Generalization
DCRNN [31]	Static (External)	Slow	Strong
STGCN [46]	Static (External)	Fast	Strong
GWN [45]	Static (External+Learnt)	Fast	Strong
MTGNN [44]	Static (Learnt)	Fast	Strong
EvolveGCN [37]	Time-varying (External)	Slow	Strong
TM-GCN [34]	Time-varying (External)	Fast	Poor
Dyn-GWN(ours)	Time-varying (External)	Fast	Strong

Limitations of existing work. All existing methods for dynamic graphs are either slow in training and/or inference or provide relatively poor generalization performance. We provide a qualitative summarize in Table 1. Although EvolveGCN [37] shows strong performance on a number of tasks, its recurrent architecture renders this method prohibitively time-consuming to train. As a result, it is challenging to tune to new datasets and tasks. TM-GCN [34] is orders of magnitudes more computationally efficient on spatio-temporal datasets. However, it shows poor generalization in our experiments. Thus, we propose *Dyn-GWN*, which address both shortcomings in a principled manner, drawing inspiration from TM-GCN [34] and GWN [45].

3 PROBLEM DEFINITION

We refer to a sequence of graphs: $\mathcal{G}^{(t)} = (\mathcal{V}, \boldsymbol{A}^{(t)}, \boldsymbol{X}^{(t)})$, $t \in \{1, \ldots, T\}$, with a fixed set $\mathcal{V}$ of N nodes, time-varying adjacency matrices $\boldsymbol{A}^{(t)} \in \mathbb{R}^{N\times N}$, and time-varying node feature matrices $\boldsymbol{X}^{(t)} \in \mathbb{R}^{N\times p}$. Note $\boldsymbol{X}^{(t)}_{n,:} \in \mathbb{R}^{p}$ is the feature vector consisting of p features associated with node n at time t. The graphs can be weighted, and directed/undirected. A time-varying graph is represented by a tensor $\mathbf{G} \in \mathbb{R}^{T\times N\times N}$ which concatenates $\boldsymbol{A}^{(t)}$ across all timesteps $t \in [T]$. Similarly, a feature tensor, $\mathbf{X} \in \mathbb{R}^{T\times N\times p}$ concatenates the node features $\boldsymbol{X}^{(t)}$ at each timestep. A static graph setting is represented by $\mathcal{G} = (\mathcal{V}, \boldsymbol{A}, \boldsymbol{X}^{(t)})$, with the same set $\mathcal{V}$ of nodes and fixed adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ (capturing long-term dependencies) and time-varying node feature matrices $\boldsymbol{X}^{(t)}$.

We consider the following learning paradigm that learns a function $f(\cdot)$ that maps T' historical graph signals and time-varying graphs to future h graph signals, i.e.,

$$[\mathbf{X}_{(t-T'+1:t)}, \mathbf{G}_{(t-T'+1:t)}, S] \xrightarrow{f(\cdot)} \mathbf{X}_{(t+1:t+h)}, \tag{1}$$

where the learning paradigm may employ additional (given) static graph to capture long-term dependencies. The time-varying graphs are assumed to be given and are passed to the learning problem — this is similar to the learning paradigm studied by Malik et al. [34], Pareja et al. [37].

4 PROPOSED MODEL: *DYN-GWN*

We propose *Dyn-GWN*: an adaptation of GWN [45] to cater to time-varying graphs. The main idea is to capture trends in both time-varying graphs and time-varying signals/features. In contrast,

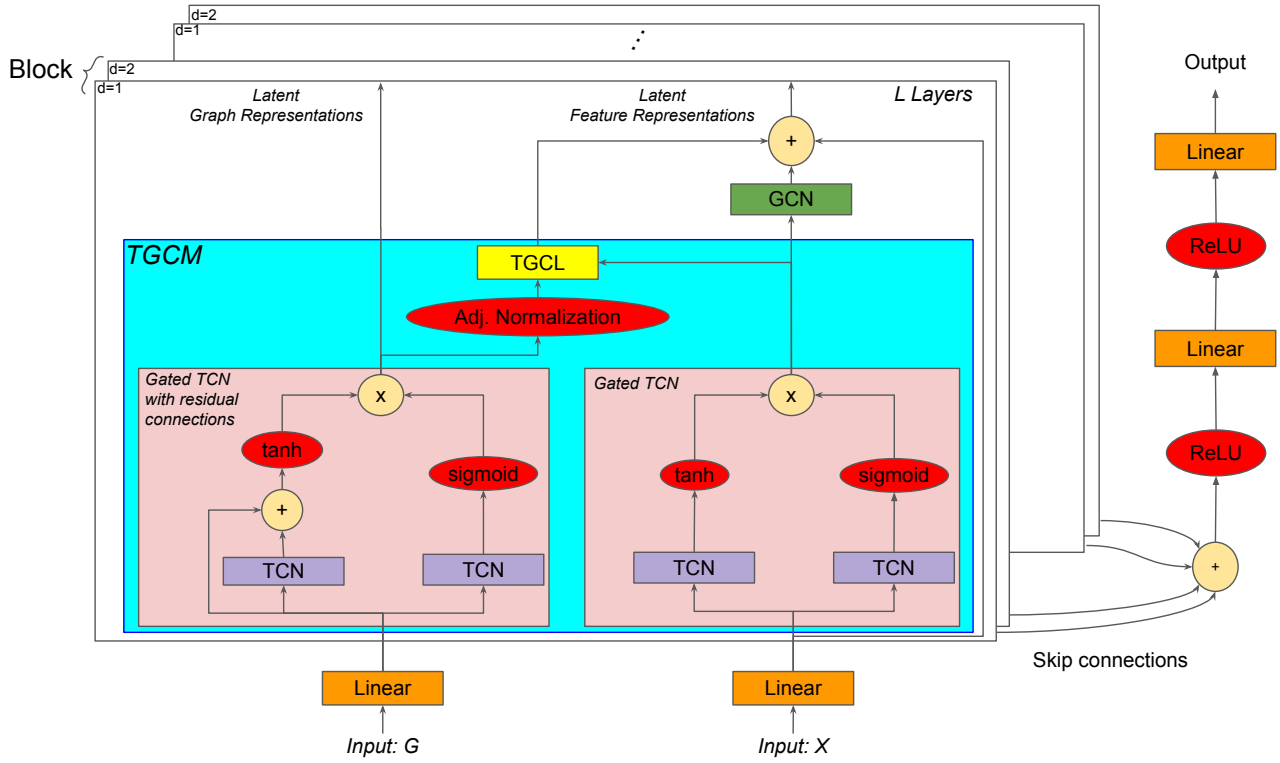

Figure 1: Illustration of *Dyn-GWN*.

GWN only considered long-term structure dependencies through a static graph structure. We first provide a mathematical description of the architecture in Section 4.1. Next, in Section 4.2, we discuss the model components in more detail.

4.1 *Dyn-GWN* Architecture

We first summarize *Dyn-GWN* architecture that apply temporal convolutions on time-varying graphs and time-varying features. We visualize the architecture in Fig. 1.

Dyn-GWN. Let $\mathbf{G}$ be the time-varying graph tensor with shape $(b, T', N^2, 1)$, where b is the batch size, T' is the historical lag, and N is the number of nodes. We can summarize the operations as follows.

Dilated temporal convolutions on time-varying features/signals and time-varying graphs are given by:

$$\text{Convolution on Time-varying Features: } \mathbf{H}_f^l = \Gamma(\mathbf{H}^l), \tag{2}$$

$$\text{Convolution on Time-varying Graphs: } \mathbf{G}_a^l = \Gamma_{res}(\mathbf{G}^l). \tag{3}$$

$\mathbf{G}^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N,N,c_{in}^l}$ and $\mathbf{H}^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N,c_{in}^l}$ are layer l inputs, where c_{in}^l denotes the number of input channels for layer l, K denotes the filter size, and $d_{[l]}$ denotes the dilation rate up to layer l. The operation in (2) is same dilated convolutions used by GWN [44]. The operations in (3) and (2) are discussed in detail in Section 4.2.1. The latent representations from time-varying graphs are normalized:

$$\bar{\mathbf{G}}^l = n(\mathbf{G}_a^l) \tag{4}$$

where the normalization is discussed in Section 4.2.2. The latent representations from time-varying features and time-varying graphs are aggregated:

$$\mathbf{S}_{Dyn}^l = \mathbf{H}_f^l \circledast_{\bar{\mathbf{G}}^l} \mathbf{W}_{Dyn}^l \tag{5}$$

where the aggregation function is discussed in Section 4.2.3. Optionally, similar to the original GWN model, our *Dyn-GWN* adaptation can also learn from an external static graph A and/or learn a static graph A_{adp} as follows:

$$\mathbf{S}_{St}^l = \sum_{v=0}^{V} \overline{\boldsymbol{A}}^v \mathbf{H}_f^l \mathbf{W}_{v,1} + \left(\overline{\boldsymbol{A}^T}\right)^v \mathbf{H}_f^l \mathbf{W}_{v,2} \tag{6}$$

$$\mathbf{S}_{adp}^l = \sum_{v=0}^{V} \boldsymbol{A}_{adp}^v \mathbf{H}_f^l \mathbf{W}_{v,3} \tag{7}$$

where $\overline{\boldsymbol{A}}$ is a row-sum normalized version of $\boldsymbol{A}$ and $\overline{\boldsymbol{A}^T}$ is a row-sum normalized version of $\boldsymbol{A}^T$. The outputs from all three graph convolution operations are added along with a residual connection as:

$$\mathbf{H}^{l+1} = \mathbf{S}_{St}^l + \mathbf{S}_{adp}^l + \mathbf{S}_{Dyn}^l + \mathbf{H}^l \tag{8}$$

Lastly, the temporal latent representations from the time-varying graphs are passed to the next block:

$$\mathbf{G}^{l+1} = \mathbf{G}_a^l. \tag{9}$$

$\mathbf{G}^{l+1} \in \mathbb{R}^{b,T'-d_{[l+1]}(K-1),N^2,c^l_{out}}$ and $\mathbf{H}^{l+1} \in \mathbb{R}^{b,T'-d_{[l+1]}(K-1),N,c^l_{out}}$ are the latent temporal representations of time-varying graphs and time-varying features that are inputs to the next block, where $d_{[l+1]}$ denotes the dilation rate up to layer $l+1$, which controls the reduction in temporal dimension as successive temporal convolutions are applied. $\mathbf{H}^{l+1}$ is also passed through dropout layers (omitted in Fig. 1) before being passed to the next block in order to reduce overfitting.

In the next section, we discuss the operations in (3), (4) and (5) in more detail. These operations make up the Tensor Graph Convolution Module (TGCM) as shown in Fig. 1.

4.2 Tensor Graph Convolutional Module (TGCM)

Tensor Graph Convolutional Module captures dynamic trends in both features and graphs effectively in the latent representations. It separately applies temporal convolutions on both the time-varying adjacency space and the time-varying feature space. Later, it effectively aggregates the temporal latent representations from both of these through a Tensor Graph Convolutional Layer.

To our knowledge, prior work, e.g., STGCN [46] and GWN [45], has only considered temporal convolutions on the time-varying features. This allows them to capture dynamics in the time-varying signals only as they rely on a static graph. Next, we highlight some key challenges in applying temporal convolutions to the time-varying graphs.

Challenges. TGCM addresses three challenges below:

(1) When applying temporal convolutions on the time-varying graphs, how do we extend the operation of temporal convolutions that is typically used for temporal convolutions on time-varying features e.g., in GWN [45]?
(2) How do we normalize the latent dynamic graph embeddings such that the graph convolution operation is not ill-posed?
(3) How do we aggregate information from latent graph representations (from temporal convolutions on adjacency space) with the latent feature representations (from temporal convolutions on the feature space)?

We discuss our proposals in detail in Sections 4.2.1, 4.2.2 and 4.2.3.

4.2.1 Gated Temporal Convolution Network with Residual Connections. We first describe the gated temporal convolutional network with residual connections that operate on the time-varying adjacency graph tensors. This network consists of three components: dilated causal convolutions [47], gating [13], and residual connections [22]. The gated temporal convolution network captures temporal trends in the time-varying graphs. In prior work, such temporal convolutions have been used to capture temporal trends in node features only, e.g., in [45]. However, the temporal trends in dynamic graphs also capture useful information as been shown in prior work [34, 37]. Next, we describe individual components of the gated temporal convolutional layers.

Dilated Causal Convolutions. Dilated causal convolutions have multiple favorable characteristics:

(1) they allow an exponentially large receptive field by increasing the layer depth,
(2) they can support long-range sequences,
(3) they allow efficient parallel computation because they do not require recursive computation,
(4) they allow causal prediction (prediction does not rely on future time steps) by zero padding.

As a special case of standard 1D convolution, the dilated causal convolution operation slides over inputs by skipping values with a certain step. Mathematically, given a 1D sequence input $\boldsymbol{x} \in \mathbb{R}^{T'}$ and a filter $h_\theta \in \mathbb{R}^K$, the dilated causal convolution operation of x with h_θ at step t is represented as: $\boldsymbol{x} \star \boldsymbol{h}_\theta(t) = \sum_{s=0}^{K-1} \boldsymbol{h}_\theta(s)\boldsymbol{x}(t - d \times s)$, where d is the dilation rate controlling the skipping distance.

Gated Temporal Convolution Network. Gating mechanisms have shown promising results in both recurrent and temporal architectures [13]. A simple Gated Temporal Convolution Network (Gated TCN) takes the form:

$$h = g(\boldsymbol{x} \star h_{\theta_1}) \odot \sigma(\boldsymbol{x} \star h_{\theta_2}), \tag{10}$$

where $\boldsymbol{x}$ is the input, θ_1, θ_2 are learnable parameters, $\odot$ is the element-wise Hadamard product, $g(\cdot)$ is an activation function, and $\sigma(\cdot)$ is the sigmoid function that determines the ratio of information passed to the next layer. Note there are additional bias terms, which we omit for convenience. We adopt Gated TCN in our model to learn complex temporal dependencies. Empirically, we used tangent hyperbolic function ($tanh(\cdot)$) as the activation function $g(\cdot)$. In tensor notation, we denote this operation as $\Gamma(\mathbf{X})$.

Residual Connections. Given the success of residual connections in various neural network architectures, e.g. ResNets [22], graph convolutional networks [10] etc., we include additional residual connection in the gated TCN as follows:

$$h = \left(g\left(\boldsymbol{x} \star h_{\theta_1}\right) + \boldsymbol{x}^{res}\right) \odot \sigma(\boldsymbol{x} \star h_{\theta_2}), \tag{11}$$

where $\boldsymbol{x}^{res}$ denotes the residual connection from the prior TCN layer. In tensor notation, we denote this operation as $\Gamma_{res}(\mathbf{X})$.

Application of Gated TCN with residual connections to dynamic graphs. We propose to apply the operation in (11) on the dynamic graphs. This function operates in parallel to the temporal convolutional layers operating on features. Stacking of gated TCN layers allows learning of longer term dependencies in both the dynamic graphs and the dynamic node signals — the existing GWN only learn longer term dependencies in dynamic node signals.

Implementation considerations. Application of the operation in (11) requires some implementation considerations. In order to make use of existing efficient GPU-compatible temporal convolutional implementations in popular deep learning API, we pass the reshaped version of the dynamic graphs to the modules. $\mathbf{G}$ is processed (at the input) into a shape: $(b, T', N^2, 1)$, where b represent the batch-size, T' denotes the historical lag, N^2 denotes all the pair-wise entries in the graphs. The successive Gated TCN modules generate $\mathbf{G}^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N^2,c^l_{out}}$, where c^l_{out} denotes the number of latent graph representations and $d_{[l]}$ denotes dilation factors up to layer l. Note $c^{l-1}_{out} = c^l_{in}$ for $l \in [L]$, $c^1_{in} = 1$, and L denotes the number of successive gated TCN modules in the architecture.

4.2.2 Adjacency Normalization Layer. As pointed out by Derr et al. [15], the aggregation methods in graph convolution operation in GCNs is traditionally designed for unsigned graphs because they assume social theory homophily. Signed graphs require more complex aggregation methods (see Derr et al. [15] for an example) — we do

not pursue signed graphs in this work. Hence in order to feed the latent graph representations (learnt from the temporal convolutions on the time-varying graphs) into the TGCL(discussed in Section 4.2.3), we need to normalize them. We also observed empirically that normalizing adjacency tensors is essential to strong performance with our model. In particular, we transform each latent (or input) adjacency slice into $[0, 1]^{N \times N}$. This is only necessary before passing the adjacency tensor to TGCL, which is described in Section 4.2.3. Thus, we construct the Adjacency Normalization Layer, denoted by the function, $\bar{\mathbf{G}}^l = n(\mathbf{G}_a^l)$. We take the output of the Gated TCN with residual connections on the adjacency space from layer l: $\mathbf{G}_a^l \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N^2,c_{out}^l}$, as an input and transform it into $\bar{\mathbf{G}}^l$ as follows:

$$\bar{\mathbf{G}}^l = n(\mathbf{G}_a^l) = \text{softmax}(\text{R}(\mathbf{G}_a^l)) \tag{12}$$

where $\text{R}(\mathbf{G}_a^l) \in \mathbb{R}^{b,T'-d_{[l]}(K-1),N,N,c_{out}^l}$ is a reshaped version of $\mathbf{G}_a^l$, and softmax$(\cdot)$ is applied on the second to last dimension of $R(\mathbf{G}_a^l)$. Softmax normalization has been used in prior work in static graph learning literature — see Wu et al. [45] among others. The normalization ensures that the latent graphs have non-negative weights when they are used in the graph convolution/aggregation operation.

Relation of Temporal Convolution on adjacency space to the approach in TM-GCN [34]. Conceptually, our proposal to apply temporal convolutions on the dynamic graphs is motivated by the approach taken by Malik et al. [34]. They apply the tensor M-product framework [26, 27] to compute an average on the adjacency tensor over the temporal dimension. By applying gated temporal convolutions, we achieve a similar weighted average in the non-linear space.

4.2.3 Tensor Graph Convolutional Layer. Graph convolution is an essential operation to extract a node's features given its structural information. The two popular approaches in existing literature are summarized below. The first approach smooths a node's signal by aggregating and transforming its neighborhood information through first approximations of Chebyshev filters [14, 28]. This has been used in STGCN [46] among others. The second approach models the diffusion process of graph signals with V steps. This approach was proposed by the authors of DCRNN [31] and also later used by authors of Graph WaveNet [45] in the context of learning from a static graph. We use the second approach in the context of learning from dynamic graphs.

In the context of dynamic graphs, applying graph convolution operation requires a careful consideration of how to combine the latent temporal features with the latent temporal graphs. This is different from the static setting as the same graph is combined across all channels of the latent temporal features. A naive parameterization would consider an outer product of the latent feature representations with the latent graph representations. However, this can result in overparameterization. To prevent this, our parallel architecture with temporal convolutional layers on time-varying graphs (described in Section 4.2.1) matches the number of channels per layer with that of the temporal convolutional layers (operating on the time-varying features). This makes for a compact paramaterization. We combine this parameterization with the diffusion process approach for graph convolution and define the tensor graph convolution operation:

$$\left(\mathbf{H}^l \circledast_{\mathbf{G}^l} \boldsymbol{W}^l\right)_{bijm} = \sum_{v=1}^{V} \left(\sum_{n=1}^{N} \left(\left(\bar{\mathbf{G}}^l\right)^v \right)_{bijno} \mathbf{H}_{bino}^l \right) \cdot \boldsymbol{W}_{om}^{l,v} \tag{13}$$

where $\left(\bar{\mathbf{G}}^l\right)^v$ denotes the power series of the transition matrix $\bar{\mathbf{G}}^l$, $\mathbf{H}^l$ denotes the latent temporal features at layer l, $\boldsymbol{W}^{l,v} \in \mathbb{R}^{c_{in}^l, c_{out}^l}$ denotes the learnable matrix for power v for layer l. In the case of a directed graph, the diffusion process have both forward and backward directions, which our implementation supports. The output from the TGCL, in *Dyn-GWN* architecture, is aggregated with the outputs of the graph convolutional layers (operating on static graphs in the original GWN architecture).

5 EXPERIMENTS

This section demonstrates the usefulness of *Dyn-GWN* on time-series datasets from different domains with time-varying graphs: finance and traffic forecasting. Additional discussion and results to highlight generalizability of our approach to other models and datasets are included in Supplement Section A.

Competing Methods. We consider methods that learn from external time-varying graphs, as these are most related to the setting we pursue in this work. In particular, we consider: (i) EvolveGCN [37] and (ii) TM-GCN [34]. The original implementations provided by the authors were updated to support multi-step horizon forecasting.

Model Implementation. We implemented *Dyn-GWN* in PyTorch. The code is available at https://github.com/mazumder-lab/DynGWN.

Training and Evaluation. All three models were trained with (masked) mean absolute error objective catering to multi-step horizon forecasting on all datasets. We consistently used horizon of 12 steps across all models and datasets. Note that we consider mean absolute error following prior work [31, 45] as mean absolute error appears to be more robust than mean squared error for training in the presence of outliers. For evaluation, we consider mean absolute errors (MAE), root mean squared errors (RMSE), and mean absolute percentage errors (MAPE) for different horizons. Following prior work, different horizon steps $\{3, 6, 9\}$ were considered. We also include average performance across 12 horizon steps.

Tuning. The architectures were fixed for EvolveGCN and TM-GCN to default optimal settings. For *Dyn-GWN*, we used 2 blocks with 2 layers each and tuned over the learning rates in the set $\{0.0001, 0.0005, 0.001, 0.005, 0.01\}$ with weight decay of 0.004. We capped the training time for EvolveGCN to 5 days (108 hours), which was prohibitively slow.

Next, we describe the datasets in Section 5.1, and results in Section 5.2. In addition, we also compare training and inference times of *Dyn-GWN* against those of EvolveGCN (more promising than TM-GCN in our experiments) in Section 5.3.

5.1 Datasets

5.1.1 Finance Datasets: Stock Volatilities. We evaluate time-series forecasting with dynamic graphs in the context of stock volatilities.

Volatilities prediction has been studied by multiple works [4, 16–18, 23]. We consider 83 bluechip companies from S&P100 financial market and define the daily volatility, as given in [4, 17, 23, 38], using the daily high and low stock prices:

$$\tilde{\sigma}_{it}^2 = 0.361 \left(\log p_{it}^{\text{high}} - \log p_{it}^{\text{low}} \right)^2, \tag{14}$$

where p_{it}^{high} and p_{it}^{low} denote the maximum and minimum price of stock i on day t [1]. In terms of the prediction task, we use 60 past steps of the following 10 signals as time-varying features: "Log Adjusted Open Stock Price", "Log Adjusted High Stock Price", "Log Adjusted Low Stock Price", "Log Adjusted Close Stock Price", "percentage change of Adjusted Open Stock Price", "percentage change of Adjusted High Stock Price", "percentage change of Adjusted Low Stock Price", "Stock Returns", "Stock Trading Volume" and "Stock Volatilities". We learn to forecast the next 12 horizon steps of daily Stock Volatilities.

Time-varying Graphs. We consider the following dynamic graphs as a proxy from external time-varying graphs: Partial Correlation Graphs. These time-varying graphs are cheaply constructed from the same time-series data to cater to settings when dynamic graphs are not easily available. We summarize the computational methodology below:

Beginning with the time-series, we compute the Ledoit-Wolf covariance matrix [29] between all companies in the subset of interest using a rolling window of fixed size. We empirically find that a window size of 48 is reasonable with the SP100 data. To construct the adjacency, we use matrix inversion to compute the corresponding precision matrix. We compute partial correlation from the precision matrix as described by Barigozzi and Brownlees [4] and take the absolute values. This provides a non-negative correlation structure between companies at each timestep.

5.1.2 Traffic Datasets: METR-LA & PEMS-BAY. We also explore traffic time-series forecasting. We conduct experiments on two datasets used by Li et al. [31], Wu et al. [45]. These are METR-LA and PEMS-BAY.

(1) METR-LA contains traffic data from loop detectors in the highway of Los Angeles County (Jagadish et al., 2014).
(2) PEMS-BAY dataset was produced by California Transportation Agencies (CalTrans) Performance Measurement System (PeMS).

For both datasets, we randomly select a subset of 80 nodes/sensors. We used the validation and test splits used by Wu et al. [45]. We used the last 5,000 samples from the original training split for METR-LA and PEMS-BAY.

Time-varying Graphs. To further validate the robustness of our model, we apply the same partial correlation-based method to construct the time-varying graphs for these two datasets.

5.2 Results

5.2.1 Stock Volatility Forecasting. We consider daily stock volatility dataset described in Section 5.1.1 with time-varying partial correlation graphs. Table 2 shows the results of *Dyn-GWN* and baselines for stock volatility prediction. Our proposed model *Dyn-GWN* achieves 9% better MAE performance than EvolveGCN and 23% better MAE performance than TM-GCN. Note, we omit MAPE as a metric for this dataset as it is ill-posed when time-series signals can be negative.

Table 2: Test RMSE ($\times 10^{-1}$) and MAE ($\times 10^{-1}$) for multi-step horizon forecasting of financial indicators for dynamic graph methods. Financial indicator: Daily Stock volatilities.

Stock Volatilities					
Metric	Model	H3	H6	H9	H1-12
MAE	TM-GCN	3.371	3.617	4.153	3.666
	EvolveGCN	3.096	3.127	3.147	3.090
	Dyn-GWN	**2.821**	**2.730**	**2.941**	**2.818**
RMSE	TM-GCN	4.347	4.482	7.717	5.106
	EvolveGCN	3.821	3.847	3.851	3.797
	Dyn-GWN	**3.538**	**3.547**	**3.678**	**3.577**

5.2.2 Traffic Forecasting. We compare *Dyn-GWN* against methods that learn from time-varying graphs on a subset of traffic datasets (METR-LA and PEMS-BAY) — See Section 5.1.2 for more details. We use dynamic partial correlation graphs for all methods. Note that missing values are excluded in calculating these metrics as done by others [31, 45] in traffic forecasting.

Results. Table 3 shows the comparison of different approaches for 15 minutes (H3), 30 minutes (H6), 45 minutes (H9) and average across 5 minute to 1 hour (H1-12) horizon forecasting. We can observe a clear advantage of *Dyn-GWN* over the competing methods. *Dyn-GWN* outperforms EvolveGCN and TM-GCN across all metrics and all horizons with a large margin. *Dyn-GWN* achieves up to 35% – 60% improvement in MAE on average across horizon steps in comparison to EvolveGCN.

5.3 Timing comparison with EvolveGCN

We compare the time of *Dyn-GWN* with time-varying graph-based methods. In particular, we compare against EvolveGCN [37], which has more competitive performance (than TM-GCN [34]). We consider both training and inference times in our comparison. We train both *Dyn-GWN* model for 1 epoch on a Tesla V100 GPU. The number of time samples for training and inference are taken to be 5000. Both models are run with batch-size equal to 64. We show the timing in seconds in Table 4. We can observe that *Dyn-GWN* is 300× faster in terms of training time and 2000× faster in terms of inference.

In terms of convergence time, we consider PEMS-BAY dataset as an example. *Dyn-GWN* converged in 2.2 hours. On the other hand, EvolveGCN had not converged even in 5 days and had a much worse training and validation objective.

6 CONCLUSION

To summarize, we propose *Dyn-GWN* for spatio-temporal learning from dynamic graphs. Our model relies on a novel module called

[1] Several advanced estimators of volatility based on high-frequency data have been proposed over the last years [3]. However, a number of contributions have pointed out that simple estimators, like the high-low range, perform satisfactorily (see also Alizadeh et al. [2], Brownlees and Gallo [8]).

Table 3: Test RMSE, MAE and MAPE for multi-step horizon forecasting of traffic for dynamic graph methods. We compare on two traffic datasets: METR-LA and PEMS-BAY.

METR-LA					
Metric	Model	H3	H6	H9	H1-12
MAE	TM-GCN	7.478	7.781	7.999	7.756
	EvolveGCN	4.526	5.071	5.554	5.103
	Dyn-GWN	**2.961**	**3.414**	**3.706**	**3.366**
RMSE	TM-GCN	12.023	12.473	12.799	12.434
	EvolveGCN	6.344	7.327	8.075	7.290
	Dyn-GWN	**5.578**	**6.643**	**7.241**	**6.462**
MAPE (%)	TM-GCN	23.050	24.170	24.950	24.050
	EvolveGCN	11.851	13.762	15.284	13.780
	Dyn-GWN	**7.870**	**9.440**	**10.370**	**9.220**
PEMS-BAY					
Metric	Model	H3	H6	H9	H1-12
MAE	TM-GCN	4.441	4.664	4.881	4.691
	EvolveGCN	4.308	4.646	4.962	4.680
	Dyn-GWN	**1.401**	**1.896**	**2.226**	**1.853**
RMSE	TM-GCN	9.722	10.273	10.832	10.342
	EvolveGCN	5.482	6.141	6.700	6.174
	Dyn-GWN	**3.086**	**4.424**	**5.242**	**4.254**
MAPE (%)	TM-GCN	13.120	13.720	14.310	13.800
	EvolveGCN	8.768	9.701	10.592	9.815
	Dyn-GWN	**3.090**	**4.670**	**5.810**	**4.610**

Table 4: Timing comparison of *Dyn-GWN* against EvolveGCN for 1 epoch with 64 batch size, $T_{train} \approx 5000$, $T_{valid} \approx 5000$, and 80 nodes on Tesla V100 GPU. *Dyn-GWN* is 300× faster than EvolveGCN.

Model	Training Time	Inference Time
EvolveGCN	16400s	15400s
Dyn-GWN	52s	8s

the Tensor Graph Convolutional Module (TGCM), which captures dynamic trends in graphs effectively in the time-varying graph representations. This module has two components: (i) it applies temporal dilated convolutions both on the time-varying graph adjacency space and the time-varying features. (ii) it aggregates the higher-level latent representations from both time-varying components through a proposed layer TGCL. Experiments demonstrate the efficacy of these model in terms of strong generalization and faster runtimes across time-series data from finance and traffic domains. *Dyn-GWN* can give up to 9% − 60% better out-of-sample performance than prior methods that can exploit time-varying graphs e.g., EvolveGCN and TM-GCN. Interestingly, *Dyn-GWN* can be ∼ 300× faster than EvolveGCN, which is the more competitive baseline from state-of-the-art models that cater to time-varying graphs.

7 LIMITATIONS

Dyn-GWN is capable of catering to problem sizes with many time samples e.g., $T \sim 50k$, and a medium sized node set e.g., $N \sim 100$. To cater to larger node sizes e.g., $N \sim 1000$, it may be possible to consider a divide and conquer strategy, where graph-partitioning strategies are used to divide the nodes into smaller sized problems. Such stategies have been proposed by Mallick et al. [35] for extensions of DCRNN [31] to larger-scale problems. Other potential solutions may rely on learning decompositions of the time-varying graphs into low-ranked matrices or consider sparse approximations of time-varying graphs. Such strategies are necessary for scaling to larger problem sizes.

ACKNOWLEDGMENTS

This work is supported by the MIT-IBM Watson AI Lab. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies. The authors acknowledge Yada Zhu (IBM Research) and Wenyu Chen (MIT) for discussions. The authors acknowledge the use of MIT Satori Cluster and MIT SuperCloud [40] resources that have contributed to the research results reported within this paper.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, et al. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML (PMLR, Vol. 97)*. PMLR, 21–29.
[2] Sassan Alizadeh, Michael W. Brandt, and Francis X. Diebold. 2002. Range-Based Estimation of Stochastic Volatility Models. *The Journal of Finance* 57, 3 (June 2002), 1047–1091. https://doi.org/10.1111/1540-6261.00454
[3] Torben G. Andersen, Tim Bollerslev, Francis X. Diebold, and Paul Labys. 2003. Modeling and Forecasting Realized Volatility. *Econometrica* 71, 2 (March 2003), 579–625. https://doi.org/10.1111/1468-0262.00418
[4] Matteo Barigozzi and Christian Brownlees. 2019. NETS: Network estimation for time series. *Journal of Applied Econometrics* 34, 3 (2019), 347–364.
[5] Tanya Y. Berger-Wolf and Jared Saia. 2006. A Framework for Analysis of Dynamic Social Networks. In *KDD* (Philadelphia, PA, USA). ACM, New York, NY, USA, 523–528. https://doi.org/10.1145/1150402.1150462
[6] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *ICML (PMLR, Vol. 28)*. PMLR, Atlanta, Georgia, USA, 115–123.
[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[8] C. T. Brownlees and G. M. Gallo. 2009. Comparison of Volatility Measures: a Risk Management Perspective. *Journal of Financial Econometrics* 8, 1 (July 2009), 29–56. https://doi.org/10.1093/jjfinec/nbp009
[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.
[10] Ming Chen, Zhewei Wei, Zengfeng Huang, et al. 2020. Simple and Deep Graph Convolutional Networks. In *ICML (PMLR, Vol. 119)*. PMLR, 1725–1735.
[11] Yingmei Chen, Zhongyu Wei, and Xuanjing Huang. 2018. Incorporating Corporation Relationship via Graph Convolutional Neural Networks for Stock Price Prediction. In *CIKM* (Torino, Italy). ACM, New York, NY, USA, 1655–1658. https://doi.org/10.1145/3269206.3269269
[12] Fulvio Corsi. 2004. A Simple Long Memory Model of Realized Volatility. *SSRN Electronic Journal* (2004). https://doi.org/10.2139/ssrn.626064
[13] Yann N. Dauphin, Angela Fan, Michael Auli, et al. 2017. Language Modeling with Gated Convolutional Networks. In *ICML* (Sydney, NSW, Australia) *(PMLR)*. 933–941.
[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS* (Barcelona, Spain). Curran Associates Inc., Red Hook, NY, USA, 3844–3852.
[15] Tyler Derr, Yao Ma, and Jiliang Tang. 2018. Signed Graph Convolutional Networks. In *ICDM*. 929–934.
[16] F. X. Diebold and K. Yilmaz. 2014. On the network topology of variance decompositions: Measuring the connectedness of financial firms. *Journal of Econometrics* 182, 1 (Sept. 2014), 119–134.
[17] Francis X. Diebold and Kamil Yilmaz. 2015. *Financial and Macroeconomic Connectedness*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199338290.001.0001
[18] R. Engle, G. Gallo, and M. Velucchi. 2012. Volatility Spillovers in East Asian Financial Markets: A Mem-Based Approach. *Review of Economics and Statistics - REV ECON STATIST* 0 (02 2012), 222–223.

[19] Robert F. Engle and Kenneth F. Kroner. 1995. Multivariate Simultaneous Generalized Arch. *Econometric Theory* 11, 1 (1995), 122–150.
[20] F. Feng, X. He, X. Wang, et al. 2019. Temporal relational ranking for stock predictions. *ACM Transactions on Information Systems (TOIS)* 37, 2 (2019).
[21] Sylvia Frühwirth-Schnatter Gregor Kastner and Hedibert Freitas Lopes. 2017. Efficient Bayesian Inference for Multivariate Factor Stochastic Volatility Models. *Journal of Computational and Graphical Statistics* 26, 4 (2017), 905–917.
[22] Kaiming He, X. Zhang, Shaoqing Ren, et al. 2016. Deep Residual Learning for Image Recognition. *CVPR* (2016), 770–778.
[23] Shibal Ibrahim, Wenyu Chen, Yada Zhu, Pin-Yu Chen, Yang Zhang, and Rahul Mazumder. 2022. Knowledge Graph Guided Simultaneous Forecasting and Network Learning for Multivariate Financial Time Series. In *Proceedings of the Third ACM International Conference on AI in Finance* (New York, NY, USA) *(ICAIF '22)*. ACM, New York, NY, USA, 480–488. https://doi.org/10.1145/3533271.3561702
[24] Eric Jacquier, Nicholas G. Polson, and Peter E. Rossi. 1994. Bayesian Analysis of Stochastic Volatility Models. *Journal of Business & Economic Statistics* 12, 4 (1994), 371–389.
[25] Renhe Jiang, Du Yin, Zhaonan Wang, Yizhuo Wang, Jiewen Deng, Hangchen Liu, Zekun Cai, Jinliang Deng, Xuan Song, and Ryosuke Shibasaki. 2021. DL-Traff: Survey and Benchmark of Deep Learning Models for Urban Traffic Prediction. In *CIKM*. 4515–4525.
[26] Eric Kernfeld, Misha Kilmer, and Shuchin Aeron. 2015. Tensor–tensor products with invertible linear transforms. *Linear Algebra Appl.* 485 (2015), 545–570.
[27] Misha E. Kilmer, Karen Braman, Ning Hao, et al. 2013. Third-Order Tensors as Operators on Matrices: A Theoretical and Computational Framework with Applications in Imaging. *SIAM J. Matrix Anal. Appl.* 34, 1 (2013), 148–172. https://doi.org/10.1137/110837711
[28] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).
[29] Olivier Ledoit and Michael Wolf. 2004. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis* 88, 2 (2004), 365–411.
[30] Charles M.C. Lee, Paul Ma, and Charles C.Y. Wang. 2015. Search-based peer firms: Aggregating investor perceptions through internet co-searches. *Journal of Financial Economics* 116, 2 (2015), 410–431.
[31] Yaguang Li, Rose Yu, Cyrus Shahabi, et al. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.
[32] Shiqing Ling and Michael McAleer. 2003. Asymptotic Theory for a Vector ARMA-GARCH Model. *Econometric Theory* 19, 2 (2003), 280–310.
[33] Chuong Luong and Nikolai Dokuchaev. 2018. Forecasting of Realised Volatility with the Random Forests Algorithm. *Journal of Risk and Financial Management* 11, 4 (2018).
[34] Osman Asif Malik, Shashanka Ubaru, Lior Horesh, et al. 2021. Dynamic Graph Convolutional Networks Using the Tensor M-Product. In *SDM*. Society for Industrial and Applied Mathematics, 729–737. https://doi.org/10.1137/1.9781611976700.82
[35] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. 2020. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. *Transportation Research Record* 2674, 9 (2020), 473–488.
[36] Jean-François Mas. 2021. Spatio-temporal dataset of COVID-19 outbreak in Mexico. *Data in brief* 35 (04 2021), 106843–106843. https://doi.org/10.1016/j.dib.2021.106843
[37] Aldo Pareja, Giacomo Domeniconi, Jie Chen, et al. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*.
[38] M. Parkinson. 1980. The Extreme Value Method for Estimating the Variance of the Rate of Return. *The Journal of Business* 53, 1 (1980), 61–65.
[39] Simone Piaggesi and André Panisson. 2022. Time-varying graph representation learning via higher-order skip-gram with negative sampling. *EPJ Data Science* 11, 1 (2022), 33.
[40] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. 2018. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 1–6.
[41] Valentina Shumovskaia, Kirill Fedyanin, Ivan Sukharev, et al. 2021. Linking bank clients using graph neural networks powered by rich transactional data. *International Journal of Data Science and Analytics* 12, 2 (2021), 135–145. https://doi.org/10.1007/s41060-021-00247-3
[42] Christopher A. Sims. 1980. Macroeconomics and Reality. *Econometrica* 48, 1 (1980), 1–48.
[43] Ines Wilms, Jeroen Rombouts, and Christophe Croux. 2021. Multivariate volatility forecasts for stock market indices. *International Journal of Forecasting* 37, 2 (2021), 484–499.
[44] Zonghan Wu, Shirui Pan, Guodong Long, et al. 2020. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. arXiv:2005.11650 [cs.LG]
[45] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph Wavenet for Deep Spatial-Temporal Graph Modeling. In *IJCAI* (Macao, China). AAAI Press, 1907–1913.
[46] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*.
[47] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *ICLR, San Juan, Puerto Rico, May 2-4*.
[48] Chao Zhang, Xingyue Pu, Mihai Cucuringu, and Xiaowen Dong. 2023. Graph Neural Networks for Forecasting Realized Volatility with Nonlinear Spillover Effects. *SSRN Electronic Journal* (2023). https://doi.org/10.2139/ssrn.4375165

A GENERALIZABILITY OF TGCM TO OTHER MODELS

In this section, we show how Tensor Graph Convolutional Module can be used to adapt other architectures to learn from time-varying graphs to show generalizability of Tensor Graph Convolutional Module. In particular, we propose *Dyn-STGCN*, which is a novel adaptation of STGCN [46]. We describe this next.

A.1 *Dyn-STGCN*

Dyn-STGCN is an adaptation of STGCN [46]. We begin with a normalized (and processed) dynamic graph tensor, $\mathbf{G}$, with shape, $(b, T', N^2, 1)$, where b is the batch size, T' is the historical lag, and N is the number of nodes. Now, we can apply the temporal convolution layer directly in adjacency space. Thus, we can represent the full Spatio-temporal Convolution *Block* in *Dyn-STGCN* as:

$$\text{Convolution on Time-varying Features: } \mathbf{H}_f^l = \Gamma(\mathbf{H}^l) \tag{15}$$

$$\text{Convolution on Time-varying Graphs: } \mathbf{G}_a^l = \Gamma_{res}(\mathbf{G}^l) \tag{16}$$

$$\text{Adjacency Normalization: } \tilde{\mathbf{G}}^l = n(\mathbf{G}_a^l) \tag{17}$$

$$\text{Graph Convolution (Static): } \mathbf{Z}_{St}^l = \hat{\boldsymbol{A}}\mathbf{H}_f^l\boldsymbol{W}_{St}^l \tag{18}$$

$$\texttt{TGCL}\text{: } \mathbf{Z}_{Dyn}^l = \mathbf{H}_f^l \circledast_{\tilde{\mathbf{G}}^l} \boldsymbol{W}_{Dyn}^l \tag{19}$$

$$\text{Combine: } \mathbf{Z}^l = \text{ReLU}\left(\mathbf{Z}_{St}^l + \mathbf{Z}_{Dyn}^l + \mathbf{H}^{l,res}\right) \tag{20}$$

$$\text{Convolution on Time-varying Features: } \mathbf{H}^{l+1} = \Gamma(\mathbf{Z}^l) \tag{21}$$

$$\text{Convolution on Time-varying Graphs: } \mathbf{G}^{l+1} = \Gamma_{res}^l(\mathbf{G}_a^l) \tag{22}$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$, $\boldsymbol{I}_N \in \mathbb{R}^{N,N}$ is an identity matrix, and $\tilde{\boldsymbol{D}}$ is a diagonal matrix with diagonal entries $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $\mathbf{G}^l \in \mathbb{R}^{b,T'-2l(K-1),N,N,c_{in}^l}$ and $\mathbf{H}^l \in \mathbb{R}^{b,T'-2l(K-1),N,c_{in}^l}$ are layer l inputs, producing outputs $\mathbf{G}^{l+1} \in \mathbb{R}^{b,T'-2(l+1)(K-1),N^2,c_{out}^l}$ and $\mathbf{H}^{l+1} \in \mathbb{R}^{b,T'-2(l+1)(K-1),N,c_{out}^l}$. Note that the dilation rate is 1 in all temporal convolutional layers in *Dyn-STGCN* (as in STGCN). $\mathbf{H}^{l+1}$ is also passed through batch normalization and dropout layers in order to stabilize learning before being passed to the next block.

A.2 Experiments for *Dyn-STGCN*

We study the performance of *Dyn-STGCN* in this setting with two different dynamic graphs. We again consider daily stock volatilities as in Section 5.1.1. However, we consider some variations to study the generalizability of our methods:

- We consider a random subset of 80 companies from S&P500.

- We consider two time-varying graphs: (i) partial correlation graphs constructed as described in Section 5.1.1, (ii) EDGAR cosearch (described below)
- We consider single-step horizon forecasting.

Time-varying EDGAR Cosearch Graphs. Following Lee et al. [30], the EDGAR cosearch builds a graph from stocks searched in SEC filings. Each company represents a node, while edge weights are defined by the number of times that company A is searched for before company B by analysts. This weight corresponds to a directed edge from A to B. The numbers are aggregated at the daily level. We leverage this daily weighted connectivity matrix as dynamic graphs. For methods that only accommodate a static graph, we compute an average adjacency over the entire time period.

Competing Methods. We compare *Dyn-STGCN* against (static) STGCN and classical high-dimensional statistical approaches for time-series forecasting e.g., vector autoregression (VAR) [42] and multioutput Random Forests (RF) [7]. We used Ridge regressor in scikit-learn [9] for setting up Multivariate VAR with ridge penalty. We used Multioutput Random Forests (RF) from scikit-learn [9].

Tuning. We perform 1000 trials of hyperparameter tuning with a random search (via hyperopt [6]). All models are tuned with respect to historical lag in the range $[5, 100]$. For VAR, ridge penalty was tuned. For RF, we tuned over number of trees, depths, minimum samples for split and minimum samples per leaf. For STGCN and *Dyn-STGCN*, we used a single STGCN block. We tuned over learning rates, batch sizes, and dropout.

- Ridge penalty α: Uniform on the log scale in the range $[10, 10^6]$ for VAR with ridge.
- Number of trees: Discrete uniform in the range $[1, 100]$ for Multioutput RF.
- Depth: Discrete uniform in the range $[2-20]$ for Multioutput RF.
- Minimum samples for split: Discrete uniform in the set $\{1, 2, 4, 6, 8\}$ for Multioutput RF.
- Minimum samples per leaf: Discrete uniform in the range $[1, 20]$ for Multioutput RF.
- Single Block with fixed 8 channels and 32 channels for layers within the block for STGCN and *Dyn-STGCN*.
- Learning rates: Uniform on the log scale $[10^{-3}, 10^{-1}]$ for STGCN and *Dyn-STGCN*.
- Batch-size: Discrete uniform in the set $\{16, 32\}$ for STGCN and *Dyn-STGCN*.
- Dropout rate: Uniform in the range $[0.03, 0.1]$ for STGCN and *Dyn-STGCN*.
- Epochs: 500 with early stopping (patience=10) based on validation set for STGCN and *Dyn-STGCN*.

Evaluation Metrics. We evaluate the performance of the models with RMSE and MAE. After tuning, we train each model for 50 repetitions (using random initialization) and report the averaged results along with their standard errors. We report results for both metrics for all models across 50 trials.

Results. Table 5 shows the results of *Dyn-STGCN* and baselines for stock volatility prediction. Clearly, there are benefits from incorporating dynamic graph structures into these prediction tasks. Our proposed model *Dyn-STGCN* outperforms all baselines with both types of dynamic graphs in both evaluation metrics. Interestingly, the cheaply computed dynamic partial correlation graphs outperform the externally generated dynamic EDGAR graphs.

Table 5: Test RMSE and MAE of *Dyn-STGCN* on stock volatility with different time-varying graphs. VAR/RF use no graphs. STGCN uses static EDGAR graph. *Dyn-STGCN* uses static EDGAR graph and time-varying graphs (either EDGAR or Partial Correlation).

Model	RMSE ($\times 10^{-1}$)	MAE ($\times 10^{-1}$)
Vector Auto Regression (VAR)	3.418 ± 0.00	2.719 ± 0.000
Multioutput Random Forests (RF)	3.624 ± 0.01	2.908 ± 0.001
STGCN (***S***: EDGAR)	3.265 ± 0.05	2.558 ± 0.030
Dyn-STGCN (***S***: EDGAR, **G**: EDGAR)	**3.222** ± 0.02	**2.522** ± 0.007
Dyn-STGCN (***S***: EDGAR, **G**: Partial Corr.)	**2.676** ± 0.03	**2.096** ± 0.018

B COMPUTING SETUP

We used a cluster running Ubuntu 7.5.0 and equipped with Intel Xeon Platinum 8260 CPUs and Nvidia Volta V100 GPUs.

C RELATED WORK ON VOLATILITY FORECASTING

Forecasting of stock volatilities has gained significant attention. Different definitions of stock volatility are considered in literature. Some works e.g., Andersen et al. [3] consider volatility based on high frequency data. Others consider high-low range [4, 17, 38]. A number of contributions have pointed out that simple estimators, like the high-low range, perform satisfactorily (see also Alizadeh et al. [2], Brownlees and Gallo [8]). Our volatility definition considers the second line of work.

Traditionally, volatility forecasting has been studied with Hierarchical Autoregression [12], Vector Autoregression [43], GARCH [19, 32], Stochastic Volatility models [21, 24], and Random Forests [33]. However, in high-dimensional scenarios, the aforementioned models may deliver poor out-of-sample forecasts due to the curse of dimensionality. Most recently, Ibrahim et al. [23], Zhang et al. [48] consider Graph Neural Network based methods to improve volatility forecasting.