# Network of Tensor Time Series

Baoyu Jing, Hanghang Tong
{baoyuj2, htong}@illinois.edu
University of Illinois at Urbana-Champaign
IL, USA

Yada Zhu
yzhu@us.ibm.com
IBM Research
NY, USA

## ABSTRACT

Co-evolving time series appears in a multitude of applications such as environmental monitoring, financial analysis, and smart transportation. This paper aims to address the following challenges, including (C1) how to incorporate *explicit relationship networks* of the time series; (C2) how to model the *implicit relationship* of the temporal dynamics. We propose a novel model called Network of Tensor Time Series (NeT$^3$), which is comprised of two modules, including Tensor Graph Convolutional Network (TGCN) and Tensor Recurrent Neural Network (TRNN). TGCN tackles the first challenge by generalizing Graph Convolutional Network (GCN) for flat graphs to tensor graphs, which captures the synergy between multiple graphs associated with the tensors. TRNN leverages tensor decomposition to model the implicit relationships among co-evolving time series. The experimental results on five real-world datasets demonstrate the efficacy of the proposed method.

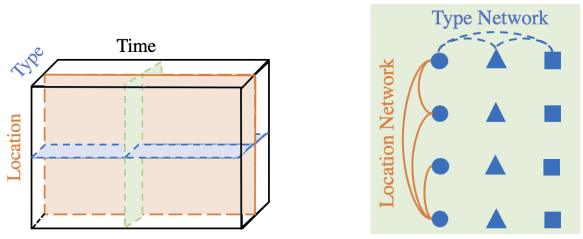
## KEYWORDS

Co-evolving Time Series; Network of Tensor Time Series; Tensor Graph Convolutional Network; Tensor Recurrent Neural Network

**ACM Reference Format:**
Baoyu Jing, Hanghang Tong and Yada Zhu. 2021. Network of Tensor Time Series. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449969

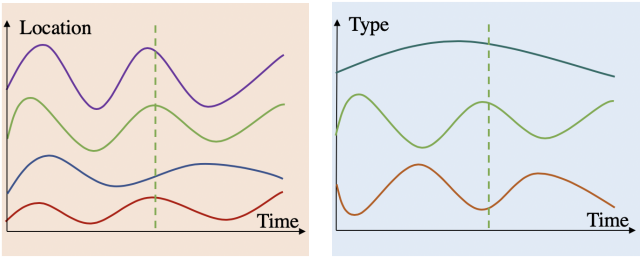
## 1 INTRODUCTION

Co-evolving time series naturally arises in numerous applications, ranging from environmental monitoring [2, 30], financial analysis [32] to smart transportation [21, 23, 37]. As shown in Figure 1a and 1b, each temporal snapshot of the co-evolving time series naturally forms a multi-dimensional array, i.e., a *multi-mode tensor* [27]. For example, the spatial-temporal monitoring data of atmosphere is a time series of an $N_1 \times N_2 \times N_3 \times N_4$ tensor, where $N_1$, $N_2$, $N_3$ and $N_4$ denote latitude, longitude, elevation and air conditions respectively (e.g. temperature, pressure and oxygen concentration). Companies' financial data is a time series of an $N_1 \times N_2 \times N_3$ tensor, where $N_1$, $N_2$ and $N_3$ denote the companies, the types of financial data (e.g. revenue, expenditure) and the statistics of them respectively. Nonetheless, the vast majority of the recent deep learning methods for co-evolving time series [21, 23, 24, 35, 37] have almost exclusively focused on a single mode.




**(a) A tensor time series consists of three modes: location, data type and time.**

**(b) A temporal snapshot. Rows and columns present locations and data types, which are constrained by their networks.**




**(c) A slice along one data type: co-evolving time series of one data type at different locations.**

**(d) A slice along one location: co-evolving time series of different data types at the same location.**

**Figure 1: An exemplary tensor time series and three slices along different dimensions. (Best viewed in color.)**

Data points within a tensor are usually related to each other, and different modes are associated with different relationships (Figure 1b). Within the above example of environmental monitoring, along geospatial modes ($N_1$, $N_2$ and $N_3$), we could know the (latitudinal, longitudinal and elevational) location relationship between two data points. In addition, different data types ($N_4$) are also related with each other. As governed by Gay-Lussac's law [3], given fixed mass and volume, the pressure of a gas is proportional to the Kelvin temperature. These relationships can be *explicitly* modeled by *networks* or *graphs* [1, 8]. Compared with the rich machinery of deep graph convolutional methods for flat graphs [11, 18], multiple graphs associated with a tensor (referred to as *tensor graphs* in this paper) are less studied. To fill this gap, we propose a novel Tensor Graph Convolution Network (TGCN) which extends Graph Convolutional Network (GCN) [18] to tensor graphs based on multi-dimensional convolution.

Another key challenge for modeling the temporal dynamics behind co-evolving time series is how to capture the *implicit relationship* of different time series. As shown in Figure 1c, the temporal patterns of time series with the same data type (e.g. temperature)


*WWW '21, April 19–23, 2021, Ljubljana, Slovenia*

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449969

are similar. The relationship of the co-evolving temperature time series can be partially captured by the location network, e.g., two neighboring locations often have similar temporal dynamics. However, the temperature time series from two locations far apart could also share similar patterns. Most of the existing studies either use the same temporal model for all time series [21, 23, 35, 37], or use separate Recurrent Neural Networks (RNN) [30, 45] for different time series. Nonetheless, none of them offers a principled way to model the implicit relationship. To tackle with this challenge, we propose a novel Tensor Recurrent Neural Network (TRNN) based on Multi-Linear Dynamic System (MLDS) [27] and Tucker decomposition, which helps reduce the number of model parameters.

Our main contributions are summarized as follows:

- We introduce a novel graph convolution for tensor graphs and present a novel TGCN that generalizes GCN [18]. The new architecture can capture the synergy among different graphs by simultaneously performing convolution on them.
- We introduce a novel TRNN based on MLDS [27] for efficiently modeling the implicit relationship between complex temporal dynamics of tensor time series.
- We present comprehensive evaluations for the proposed methods on a variety of real-world datasets to demonstrate the effectiveness of the proposed method.

The rest of the paper is organized as follows. In Section 2, we briefly introduce relevant definitions about graph convolution and tensor algebra, and formally introduce the definition of network of tensor time series. In Section 3, we present and analyze the proposed TGCN and TRNN. The experimental results are presented in Section 4. Related works and conclusion are presented in Section 5 and Section 6 respectively.

## 2 PRELIMINARIES

In this section, we formally define network of tensor time series (Subsection 2.3), after we review the preliminaries, including graph convolution on flat graphs (Subsection 2.1), tensor algebra (Subsection 2.2), and multi-dimensional Fourier transformation (Subsection 2.3) respectively. We introduce the definitions of the problems in Section 2.5.

### 2.1 Graph Convolution on Flat Graphs

Analogous to the one-dimensional Discrete Fourier Transform (Definition 2.2), the graph Fourier transform is given by Definition 2.3. Then the spectral graph convolution (Definition 2.4) is defined based on one-dimensional convolution and the convolution theorem. The free parameter of the convolution filter is further replaced by Chebyshev polynomials and thus we have Chebyshev approximation for graph convolution (Definition 2.5).

*Definition 2.1 (Flat Graph).* A flat graph contains a one-dimentional graph signal $\mathbf{x} \in \mathbb{R}^N$ and an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$.

*Definition 2.2 (Discrete Fourier Transform).* Given an one dimensional signal $\mathbf{x} \in \mathbb{R}^N$, where $N$ is the length of the sequence, its Fourier transform is defined by:

$$\tilde{\mathbf{x}}[n] = \sum_{k=1}^{N} \mathbf{x}[k] e^{-\frac{i2\pi}{N} kn} \tag{1}$$

where $\mathbf{x}[k]$ is the $k$-th element of $\mathbf{x}$ and $\tilde{\mathbf{x}}[n]$ is the $n$-th element of the transformed vector $\tilde{\mathbf{x}}$. The above definition can be rewritten as:

$$\tilde{\mathbf{x}} = \mathbf{F}\mathbf{x} \tag{2}$$

where $\mathbf{F} \in \mathbb{R}^{N \times N}$ is the filter matrix and $\mathbf{F}[n,k] = e^{-\frac{i2\pi}{N} kn}$.

*Definition 2.3 (Graph Fourier Transform [5]).* Given a graph signal $\mathbf{x} \in \mathbb{R}^N$, along with its adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where $N$ is the number of nodes, the graph Fourier transform is defined by:

$$\tilde{\mathbf{x}} = \mathbf{\Phi}^T \mathbf{x} \tag{3}$$

where $\mathbf{\Phi}$ is the eigenvector matrix of the graph Laplacian matrix $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T$, $\mathbf{I} \in \mathbb{R}^{N \times N}$, $\mathbf{D} \in \mathbb{R}^{N \times N}$ denote the identity matrix and the degree matrix, and $\mathbf{\Lambda}$ is a diagonal matrix whose diagonal elements are eigenvalues.

*Definition 2.4 (Spectral Graph Convolution [5]).* Given a signal $\mathbf{x} \in \mathbb{R}^N$ and a filter $\mathbf{g} \in \mathbb{R}^N$, the spectral graph convolution is defined in the Fourier domain according to the convolution theorem:

$$\mathbf{\Phi}^T(\mathbf{g} \star \mathbf{x}) = (\mathbf{\Phi}^T \mathbf{g}) \odot (\mathbf{\Phi}^T \mathbf{x}) \tag{4}$$

$$\mathbf{g} \star \mathbf{x} = \mathbf{\Phi}(\mathbf{\Phi}^T \mathbf{g}) \odot (\mathbf{\Phi}^T \mathbf{x}) = \mathbf{\Phi}\mathrm{diag}(\tilde{\mathbf{g}})\mathbf{\Phi}^T \mathbf{x} \tag{5}$$

where $\star$ and $\odot$ denote convolution operation and Hadamard product; the second equation holds due to the orthonormality.

*Definition 2.5 (Chebyshev Approximation for Spectral Graph Convolution [11]).* Given an input graph signal $\mathbf{x} \in \mathbb{R}^N$ and its adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, the Chebyshev approximation for graph convolution on a flat graph is given by [11, 18]:

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{\Phi}(\sum_{p=0}^{P} \theta_p T_p(\tilde{\mathbf{\Lambda}}))\mathbf{\Phi}^T \mathbf{x} = \sum_{p=0}^{P} \theta_p T_p(\tilde{\mathbf{L}})\mathbf{x} \tag{6}$$

where $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\mathbf{\Lambda} - \mathbf{I}$ is the normalized eigenvalues, $\lambda_{max}$ is maximum eigenvalue of the matrix $\mathbf{\Lambda}$; $\tilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}$; $T_p(x)$ is Chebyshev polynomials defined by $T_p(x) = 2xT_{p-1}(x) - T_{p-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$, and $p$ denotes the order of polynomials; $\mathbf{g}_\theta$ and $\theta_p$ denote the filter vector and the parameter respectively.

### 2.2 Tensor Algebra

*Definition 2.6 (Mode-m Product).* The mode-m product generalizes matrix-matrix product to tensor-matrix product. Given a matrix $\mathbf{U} \in \mathbb{R}^{N_m \times N'}$, and a tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots N_{m-1} \times N_m \times N_{m+1} \cdots \times N_M}$, then $\mathcal{X} \times_m \mathbf{U} \in \mathbb{R}^{N_1 \times \cdots N_{m-1} \times N' \times N_{m+1} \cdots \times N_M}$ is its mode-m product. Its element $[n_1, \cdots, n_{m-1}, n', n_{m+1}, \cdots, n_M]$ is defined as:

$$\begin{aligned}&(\mathcal{X} \times_m \mathbf{U})[n_1, \cdots, n_{m-1}, n', n_{m+1}, \cdots, n_M] \\ &= \sum_{n_m=1}^{N_m} \mathcal{X}[n_1, \cdots, n_{m-1}, n_m, n_{m+1}, \cdots, n_M]\mathbf{U}[n_m, n']\end{aligned} \tag{7}$$

*Definition 2.7 (Tucker Decomposition).* The Tucker decomposition can be viewed as a form of high-order principal component analysis [19]. A tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ can be decomposed into a smaller core tensor $\mathcal{Z} \in \mathbb{R}^{N'_1 \times \cdots \times N'_M}$ by $M$ orthonormal matrices $\mathbf{U}_m \in \mathbb{R}^{N'_m \times N_m}$ ($N'_m < N_m$):

$$\mathcal{X} = \mathcal{Z} \prod_{m=1}^{M} \times_m \mathbf{U}_m \tag{8}$$

The matrix $\mathbf{U}_m$ is comprised of principal components for the $m$-th mode and the core tensor $\mathcal{Z}$ indicates the interactions among the components. Due to the orthonormality of $\mathbf{U}_m$, we have:

$$\mathcal{Z} = \mathcal{X} \prod_{m=1}^{M} \times_m \mathbf{U}_m^T \tag{9}$$

## 2.3 Multi-dimensional Fourier Transform

*Definition 2.8 (Multi-dimensional Discrete Fourier Transform).* Given a multi-dimensional/mode signal $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$, the multi-dimensional Fourier transform is defined by:

$$\tilde{\mathcal{X}}[n_1, \cdots, n_M] = \prod_{m=1}^{M} \sum_{k_m=1}^{N_m} e^{-\frac{i2\pi}{N_m} k_m n_m} \mathcal{X}[k_1, \cdots, k_M] \tag{10}$$

Similar to the one-dimensional Fourier transform (Definition 2.2), the above equation can be re-written by a multi-linear form:

$$\tilde{\mathcal{X}} = \mathcal{X} \times_1 \mathbf{F_1} \cdots \times_M \mathbf{F_M} = \mathcal{X} \prod_{m=1}^{M} \times_m \mathbf{F}_m \tag{11}$$

where $\times_m$ denotes the mode-m product, $\mathbf{F}_m \in \mathbb{R}^{N_m \times N_m}$ is the filter matrix, and $\mathbf{F}_m[n,k] = e^{-\frac{i2\pi}{N_m} kn}$.

*Definition 2.9 (Separable Multi-dimensional Convolution).* The separable multi-dimensional convolution is defined based on Definition 2.8. Given a signal $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ and a separable filter $\mathcal{Y} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ such that $\mathcal{Y}[n_1, \cdots, n_m] = \mathbf{y}_1[n_1] \cdots \mathbf{y}_M[n_m]$, where $\mathbf{y}_m \in \mathbb{R}^{N_m}$ is the filter vector for the $m$-th mode, then the multi-dimensional convolution is the same as iteratively applying one dimensional convolution onto $\mathcal{X}$:

$$\mathcal{Y} \star \mathcal{X} = \mathbf{y}_1 \star_1 \cdots \star_{M-1} \mathbf{y}_M \star_M \mathcal{X} \tag{12}$$

where $\star_m$ denotes convolution on the $m$-th mode.

Suppose $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathcal{Y} = \mathbf{y}_1 \cdot \mathbf{y}_2^T$, where $\mathbf{y}_1 \in \mathbb{R}^{N_1}$ and $\mathbf{y}_2 \in \mathbb{R}^{N_2}$. Then $\mathcal{Y} \star \mathcal{X}$ means applying $\mathbf{y}_1$ and $\mathbf{y}_2$ to the rows and columns of $\mathcal{X}$ respectively. Formally we have:

$$\mathcal{Y} \star \mathcal{X} = \mathbf{y}_1 \star_1 \mathbf{y}_2 \star_2 \mathcal{X} = \mathbf{Y}_1^T \mathcal{X} \mathbf{Y}_2 = \mathcal{X} \prod_{m=1}^{2} \times_m \mathbf{Y}_m \tag{13}$$

where $\mathbf{Y}_1 \in \mathbb{R}^{N_1 \times N_1}$ and $\mathbf{Y}_2 \in \mathbb{R}^{N_2 \times N_2}$ are the transformation matrix corresponding to $\mathbf{y}_1$ and $\mathbf{y}_2$ respectively.

## 2.4 Network of Tensor Time Series

*Definition 2.10 (Tensor Time Series).* A tensor time series is a $(M+1)$-mode tensor $\mathcal{S} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times T}$ or $\{\mathcal{S}_t \in \mathbb{R}^{N_1 \times \cdots \times N_M}\}_{t=1}^{T}$, where the $(M+1)$-th mode is the time and its dimension is $T$.

*Definition 2.11 (Tensor Graph).* The tensor graph is comprised of a $M$-mode tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ and the adjacency matrices for each mode $\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}$. Note that if $m$-th mode is not associated with an adjacency matrix, then $\mathbf{A}_m = \mathbf{I}_m$, where $\mathbf{I}_m \in \mathbb{R}^{N_m \times N_m}$ denotes the identity matrix.

*Definition 2.12 (Network of Tensor Time Series).* A network of tensor time series is comprised of (1) a tensor time series $\mathcal{S} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times T}$ and (2) a set of adjacency matrices $\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}$ ($m \in [1, \cdots, M]$) for all but the last mode (i.e., the time mode).

## 2.5 Problem Definition

In this paper, we focus on the representation learning for the network of tensor time series by predicting its future values. The model trained by predicting the future values can also be applied to recover the missing values of the time series.

*Definition 2.13 (Future Value Prediction).* Given a network of tensor time series with $\mathcal{S} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times T}$ and $\{\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}\}_{m=1}^{M}$, and a time step $T'$, the task of the future value prediction is to predict the future values of $\mathcal{S}$ from $T+1$ to $T+T'$.

*Definition 2.14 (Missing Value Recovery).* We formulate the task of missing value recovery from the perspective of future value prediction. Suppose the data point $\mathcal{S}[n_1, \cdots, n_M, T']$ ($T' \leq T$) of $\mathcal{S} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times T}$ is missing, then we takes $\omega \leq T'$ historical values of $\mathcal{S}$ prior to the time step $T'$: $\{\mathcal{S}_t\}_{\omega=T'-\omega}^{T'-1}$ as input, and predict the value of the $\hat{\mathcal{S}}[n_1, \cdots, n_M, T']$.

# 3 METHODOLOGY

An overview of the proposed NeT$^3$ is presented in Figure 2, which works as follows. At each time step $t$, the proposed Tensor Graph Convolutional Network (TGCN) (Section 3.1) takes as input the $t$-th snapshot $\mathcal{S}_t \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ along with its adjacency matrices $\{\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}\}_{m=1}^{M}$ and extracts its node embedding tensor $\mathcal{H}_t$, which will be fed into the proposed Tensor Recurrent Neural Network (TRNN) (Section 3.2) to encode temporal dynamics and produce $\mathcal{R}_t$. Finally, the output module (Section 3.3) takes both $\mathcal{H}_t$ and $\mathcal{R}_t$ to predict the snapshot of the next time step $\hat{\mathcal{S}}_{t+1}$. Note that $\mathcal{Y}_t$ in Figure 2 denotes the hidden state of TRNN at the time step $t$.

## 3.1 Tensor Graph Convolution Network

In this subsection, we first introduce spectral graph convolution on tensor graphs and its Chebychev approximation in Subsection 3.1.1. Then we provide a detailed derivation for the layer-wise updating function of the proposed TGCN in Subsection 3.1.2.

*3.1.1 Spectral Convolution for Tensor Graph.* Analogues to the multi-dimensional Fourier transform (Definition 2.8) and the graph Fourier transform on flat graphs (Definition 2.3), we first define the Fourier transform on tensor graphs in Definition 3.1. Then based on the separable multi-dimensional convolution (Definition 2.9), and tensor graph Fourier transform (Definition 3.1), we propose spectral convolution on tensor graphs in Definition 3.2. Finally, in Definition 3.3, we propose to use Chebychev approximation in order to parameterize the free parameters in the filters of spectral convolution.

*Definition 3.1 (Tensor Graph Fourier Transform).* Given a graph signal $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$, along with its adjacency matrices for each mode $\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}$ ($m \in [1, \cdots, M]$), the tensor graph Fourier transform is defined by:

$$\tilde{\mathcal{X}} = \mathcal{X} \prod_{m=1}^{M} \times_m \mathbf{\Phi}_m \tag{14}$$

where $\mathbf{\Phi}_m$ is the eigenvector matrix of graph Laplacian matrix $\mathbf{L}_m = \mathbf{\Phi}_m \mathbf{\Lambda}_m \mathbf{\Phi}_m^T$ for $\mathbf{A}_m$; $\times_m$ denotes the mode-m product.

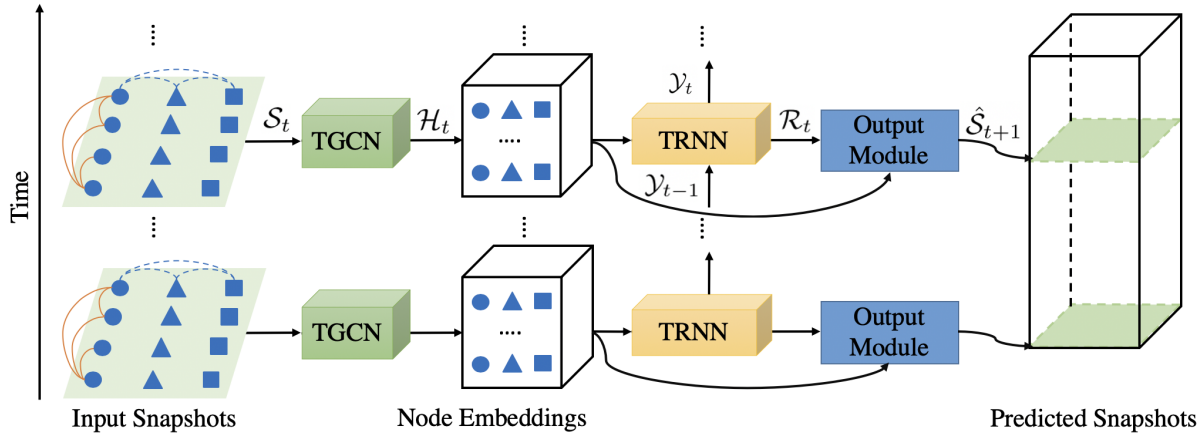



**Figure 2: The framework of the proposed model NeT$^3$. At each time step $t$, the model takes a snapshot $\mathcal{S}_t$ from the tensor time series $\mathcal{S}$ and extracts its node embedding tensor $\mathcal{H}_t$ via Tensor Graph Convolution Network (TGCN) module. $\mathcal{H}_t$ will be fed into the Tensor RNN (TRNN) module to encode the temporal dynamics. Finally, the output module takes both of $\mathcal{H}_t$ and $\mathcal{R}_t$ to predict the snapshot of the next time step $\hat{\mathcal{S}}_{t+1}$. Note that $\mathcal{Y}_t$ and $\mathcal{Y}_{t+1}$ are the hidden states of TRNN at time step $t$ and $t+1$ respectively.**

*Definition 3.2 (Spectral Convolution for Tensor Graph).* Given an input graph signal $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$, and a multi-dimensional filter $\mathcal{G} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ defined by $\mathcal{G}[n_1, \cdots, n_M] = \mathbf{g}_1[n_1] \cdots \mathbf{g}_M[n_M]$, where $\mathbf{g}_m \in \mathbb{R}^{N_m}$ is the filter vector for the $m$-th mode. By analogizing to spectral graph convolution (Definition 2.4) and separable multi-dimensional convolution (Definition 2.9) , we define spectral convolution for tensor graph as:

$$\mathcal{G} \star \mathcal{X} = \mathcal{X} \prod_{m=1}^{M} \times_m \mathbf{\Phi}_m^T \text{diag}(\tilde{\mathbf{g}}_m) \mathbf{\Phi}_m \tag{15}$$

where $\tilde{\mathbf{g}}_m = \mathbf{\Phi}_m^T \mathbf{g}_m$ is the Fourier transformed filter for the $m$-th mode; $\star$ and $\times_m$ denote the convolution operation and the mode-m product respectively; $\text{diag}(\mathbf{g}_m)$ denotes the diagonal matrix, of which the diagonal elements are the elements in $\mathbf{g}_m$.

*Definition 3.3 (Chebyshev Approximation for Spectral Convolution on Tensor Graph).* Given a tensor graph $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$, where each mode is associated with an adjacency matrix $\mathbf{A}_m \in \mathbb{R}^{N_m \times N_m}$, the Chebychev approximation for spectral convolution on tensor graphs is given by approximating $\tilde{\mathbf{g}}_m$ by Chebyshev polynomials:

$$\begin{aligned} \mathcal{G}_\theta \star \mathcal{X} &= \mathcal{X} \prod_{m=1}^{M} \times_m \mathbf{\Phi}_m^T \Big( \sum_{p_m=0}^{P} \theta_{m,p_m} T_{p_m}(\tilde{\mathbf{\Lambda}}_m) \Big) \mathbf{\Phi}_m \\ &= \mathcal{X} \prod_{m=1}^{M} \times_m \sum_{p_m=0}^{P} \theta_{m,p_m} T_{p_m}(\tilde{\mathbf{L}}_m) \end{aligned} \tag{16}$$

where $\mathcal{G}_\theta$ denotes the convolution filter parameterized by $\theta$; $\mathbf{\Lambda}_m \in \mathbb{R}^{N_m \times N_m}$ is the matrix of eigenvalues for the graph Laplacian matrix $\mathbf{L}_m = \mathbf{I}_m - \mathbf{D}_m^{-\frac{1}{2}} \mathbf{A}_m \mathbf{D}_m^{-\frac{1}{2}} = \mathbf{\Phi}_m \mathbf{\Lambda}_m \mathbf{\Phi}_m^T$; $\tilde{\mathbf{\Lambda}}_m = \frac{2}{\lambda_{m,max}} \mathbf{\Lambda}_m - \mathbf{I}_m$ is the normalized eigenvalues, $\lambda_{m,max}$ is maximum eigenvalue in the matrix $\mathbf{\Lambda}_m$; $\tilde{\mathbf{L}}_m = \frac{2}{\lambda_{m,max}} \mathbf{L}_m - \mathbf{I}_m$; $T_{p_m}(x)$ is Chebyshev polynomials defined by $T_{p_m}(x) = 2xT_{p_m-1}(x) - T_{p_m-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$, and $p_m$ denotes the order of polynomials; $\theta_{m,p_m}$ denote the co-efficient of $T_{p_m}(x)$. For clarity, we use the same polynomial degree $P$ for all modes.

*3.1.2 Tensor Graph Convolutional Layer.* Due to the linearity of mode-m product, Equation (16) can be re-formulated as:

$$\begin{aligned} \mathcal{G}_\theta \star \mathcal{X} &= \sum_{p_1, \cdots, p_M = 0}^{P} \mathcal{X} \prod_{m=1}^{M} \times_m \theta_{m,p_m} T_{p_m}(\tilde{\mathbf{L}}_m) \\ &= \sum_{p_1, \cdots, p_M = 0}^{P} \prod_{m=1}^{M} \theta_{m,p_m} \mathcal{X} \prod_{m=1}^{M} \times_m T_{p_m}(\tilde{\mathbf{L}}_m) \end{aligned} \tag{17}$$

We follow [18] to simplify Equation (17). Firstly, let $\lambda_{m,max} = 2$ and we have:

$$\begin{aligned} \tilde{\mathbf{L}}_m &= \frac{2}{\lambda_{m,max}} \mathbf{L}_m - \mathbf{I}_m \\ &= \mathbf{I}_m - \mathbf{D}_m^{-\frac{1}{2}} \mathbf{A}_m \mathbf{D}_m^{-\frac{1}{2}} - \mathbf{I}_m \\ &= -\mathbf{D}_m^{-\frac{1}{2}} \mathbf{A}_m \mathbf{D}_m^{-\frac{1}{2}} \end{aligned} \tag{18}$$

For clarity, we use $\tilde{\mathbf{A}}_m$ to represent $\mathbf{D}_m^{-\frac{1}{2}} \mathbf{A}_m \mathbf{D}_m^{-\frac{1}{2}}$. Then we fix $P = 1$ and drop the negative sign in Equation (18) by absorbing it to parameter $\theta_{m,p_m}$. Therefore, we have

$$\sum_{p=0}^{P} \theta_{m,p_m} T_p(\tilde{\mathbf{L}}_m) = \theta_{m,0} + \theta_{m,1} \tilde{\mathbf{A}}_m \tag{19}$$

Furthermore, by plugging Equation (19) back into Equation (17) and replacing the product of parameters $\prod_{m=1}^{M} \theta_{m,p_m}$ by a single parameter $\theta_{p_1, \cdots, p_M}$, we will obtain:

$$\mathcal{G}_\theta \star \mathcal{X} = \sum_{\exists p_m = 1} \theta_{p_1, \cdots, p_M} \mathcal{X} \prod_{p_m=1} \times_m \tilde{\mathbf{A}}_m + \theta_{0, \cdots, 0} \mathcal{X} \tag{20}$$

We can observe from the above equation that $p_m$ works as an indicator for whether applying the convolution filter $\tilde{\mathbf{A}}_m$ to $\mathcal{X}$ or

not. If $p_m = 1$, then $\tilde{\mathbf{A}}_m$ will be applied to $\mathcal{X}$, otherwise, $\mathbf{I}_m$ will be applied. When $p_m = 0$ for $\forall m \in [1, \cdots, M]$, we will have $\theta_{0,\cdots,0}\mathcal{X}$. To better understand how the above approximation works on tensor graphs, let us assume $M = 2$. Then we have:

$$\mathcal{G}_\theta \star \mathcal{X} = \theta_{1,1}\mathcal{X}\times_1\tilde{\mathbf{A}}_1\times_2\tilde{\mathbf{A}}_2 + \theta_{1,0}\mathcal{X}\times_1\tilde{\mathbf{A}}_1 + \theta_{0,1}\mathcal{X}\times_2\tilde{\mathbf{A}}_2 + \theta_{0,0}\mathcal{X} \quad (21)$$

Given the approximation in Equation (20), we propose the tensor graph convolution layer in Definition 3.4.

*Definition 3.4 (Tensor Graph Convolution Layer).* Given an input tensor $\mathcal{X} \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$, where $d$ is the number of channels, along with its adjacency matrices $\{\mathbf{A}_m\}_{m=1}^M$, the Tensor Graph Convolution Layer (TGCL) with $d'$ output channels is defined by:

$$\begin{aligned}&\text{TGCL}(\mathcal{X}, \{\mathbf{A}_m\}_{m=1}^M)\\ =&\sigma\Big(\sum_{\exists p_m=1}\mathcal{X}\prod_{p_m=1}\times_m\tilde{\mathbf{A}}_m\times_{M+1}\mathbf{\Theta}_{p_1,\cdots,p_M} + \mathcal{X}\times_{M+1}\mathbf{\Theta}_0\Big)\end{aligned} \quad (22)$$

where $\mathbf{\Theta} \in \mathbb{R}^{d\times d'}$ is parameter matrix; $\sigma(\cdot)$ is activation function.

In the NeT$^3$model (Figure 2), given a snapshot $\mathcal{S}_t \in \mathbb{R}^{N_1\times\cdots\times N_M}$ along with its adjacency matrices $\{\mathbf{A}_m\}_{m=1}^M$, we use a one layer TGCL to obtain the node embeddings $\mathcal{H}_t \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$, where $d$ is the dimension of the node embeddings:

$$\mathcal{H}_t = \text{TGCN}(\mathcal{S}_t) \quad (23)$$

*3.1.3 Synergy Analysis.* The proposed TGCL effectively models tensor graphs and captures the synergy among different adjacency matrices. The vector $\mathbf{p} = [p_1, \cdots, p_M] \in [0, 1]^M$ represents a combination of $M$ networks, where $p_m = 1$ and $p_m = 0$ respectively indicate the presence and absence of the $\tilde{\mathbf{A}}_m$. Therefore, each node in $\mathcal{X}$ could collect other nodes' information along the adjacency matrix $\tilde{\mathbf{A}}_m$ if $p_m = 1$. For example, suppose $M = 2$ and $p_1 = p_2 = 1$ (as shown in Figure 3 and Equation (21)), then node $\mathcal{X}[1, 1]$ (node $v$) could reach node $\mathcal{X}[2, 2]$ (node $w'$) by passing node $\mathcal{X}[2, 1]$ along the adjacency matrix $\tilde{\mathbf{A}}_1$ ($\mathcal{X}\times_1\tilde{\mathbf{A}}_1$) and then arriving at node $\mathcal{X}[2, 2]$ via $\tilde{\mathbf{A}}_2$ ($\mathcal{X}\times_1\tilde{\mathbf{A}}_1\times_2\tilde{\mathbf{A}}_2$). In contrast, with a traditional GCN layer, node $v$ can only gather information of its direct neighbors from a given model (node $v'$ via $\tilde{\mathbf{A}}_1$ or $w$ via $\tilde{\mathbf{A}}_2$).

An additional advantage of TGCL lies in that it is robust to missing values in $\mathcal{X}$ since TGCL is able to recover the value of a node from various combination of adjacency matrices. For example, suppose the value of node $v = 0$, then TGCL could recover its value by referencing the value of $v'$ (via $\mathcal{X}\times_1\tilde{\mathbf{A}}_1$), or the value of $w$ (via $\mathcal{X}\times_2\tilde{\mathbf{A}}_2$), or the value of $w'$ (via $\mathcal{X}\times_1\tilde{\mathbf{A}}_1\times_2\tilde{\mathbf{A}}_2$). However, a GCN layer could only refer to the node $v'$ via $\tilde{\mathbf{A}}_1$ or $w$ via $\tilde{\mathbf{A}}_2$.

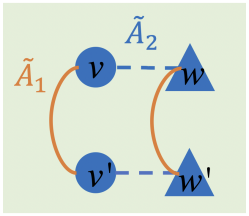



**Figure 3: An illustration of synergy analysis of TGCL.**

*3.1.4 Complexity Analysis.* For a $M$-mode tensor with $K(1 \le K \le M)$ networks, the complexity of the tensor graph convolution (Equation (20)) is $O(2^{K-1}\prod_{m=1}^M N_m(2 + \sum_{k=1}^K N_k))$.

## 3.2 Tensor Recurrent Neural Network

Given the output from TGCN: $\mathcal{H}_t \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$ (Equation (23)), the next step is to incorporate temporal dynamics for $\mathcal{H}_t$.

As shown in Figure 4, we propose a novel Tensor Recurrent Neural Network (TRNN), which captures the implicit relation among co-evolving time series by decomposing $\mathcal{H}_t$ into a low dimensional core tensor $\mathcal{Z}_t \in \mathbb{R}^{N'_1\times\cdots\times N'_M\times d}$ ($N'_m < N_m$) via a Tensor Dimension Reduction module (Section 3.2.1). The Tensor RNN Cell (Section 3.2.2) further introduces non-linear temporal dynamics into $\mathcal{Z}_t$ and produces the hidden state $\mathcal{Y}_t \in \mathbb{R}^{N'_1\times\cdots\times N'_M\times d}$. Finally, the Tensor Dimension Reconstruction module (Section 3.2.3) reconstructs $\mathcal{Y}_t$ and generates the reconstructed tensor $\mathcal{R}_t \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$.

*3.2.1 Tensor Dimension Reduction.* As shown in the left part of Figure 4, the proposed tensor dimension reduction module will reduce the dimensionality of each mode of $\mathcal{H}_t \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$, except for the last mode (hidden features), by leveraging Tucker decomposition (Definition 2.7):

$$\mathcal{Z}_t = \mathcal{H}_t\prod_{m=1}^M\times_m\mathbf{U}_m^T \quad (24)$$

where $\mathbf{U}_m \in \mathbb{R}^{N'_m\times N_m}$ denotes the orthonormal parameter matrix, which is learnable via backpropagation; $\mathcal{Z}_t \in \mathbb{R}^{N'_1\times\cdots\times N'_m\times d}$ is the core tensor of $\mathcal{H}_t$.

*3.2.2 Tensor RNN Cell.* Classic RNN cells, e.g. Long-Short-Term-Memory (LSTM) [15] are designed for a single input sequence, and therefore do not directly capture the correlation among co-evolving sequences. To address this problem, we propose a novel Tensor RNN (TRNN) cell based on tensor algebra.

We first propose a Tensor Linear Layer (TLL):

$$\text{TLL}(\mathcal{X}) = \mathcal{X}\prod_{m=1}^{M+1}\times_m\mathbf{W}_m + \mathbf{b} \quad (25)$$

where $\mathcal{X} \in \mathbb{R}^{N_1\times\cdots\times N_M\times d}$ is the input tensor, and $\mathbf{W}_m \in \mathbb{R}^{N_m\times N'_m}$ ($\forall m \in [1, \cdots, M]$) and $\mathbf{W}_{M+1} \in \mathbb{R}^{d\times d'}$ are the linear transition parameter matrices; $\mathbf{b} \in \mathbb{R}^{d'}$ denotes the bias vector.

TRNN can be obtained by replacing the linear functions in any RNN cell with the proposed TLL. We take LSTM as an example to reformulate its updating equations. By replacing the linear functions in the LSTM with the proposed TLL, we have updating functions for Tensor LSTM (TLSTM)[1]:

$$\mathcal{F}_t = \sigma(\text{TLL}_{fz}(\mathcal{Z}_t) + \text{TLL}_{fy}(\mathcal{Y}_{t-1})) \quad (26)$$
$$\mathcal{I}_t = \sigma(\text{TLL}_{iz}(\mathcal{Z}_t) + \text{TLL}_{iy}(\mathcal{Y}_{t-1})) \quad (27)$$
$$\mathcal{O}_t = \sigma(\text{TLL}_{oz}(\mathcal{Z}_t) + \text{TLL}_{oy}(\mathcal{Y}_{t-1})) \quad (28)$$
$$\tilde{\mathcal{C}}_t = \tanh(\text{TLL}_{cz}(\mathcal{Z}_t) + \text{TLL}_{cy}(\mathcal{Y}_{t-1})) \quad (29)$$
$$\mathcal{C}_t = \mathcal{F}_t \odot \mathcal{C}_{t-1} + \mathcal{I}_t \odot \tilde{\mathcal{C}}_t \quad (30)$$
$$\mathcal{Y}_t = \mathcal{O}_t \odot \sigma(\mathcal{C}_t) \quad (31)$$

[1]Bias vectors are omitted for clarity.

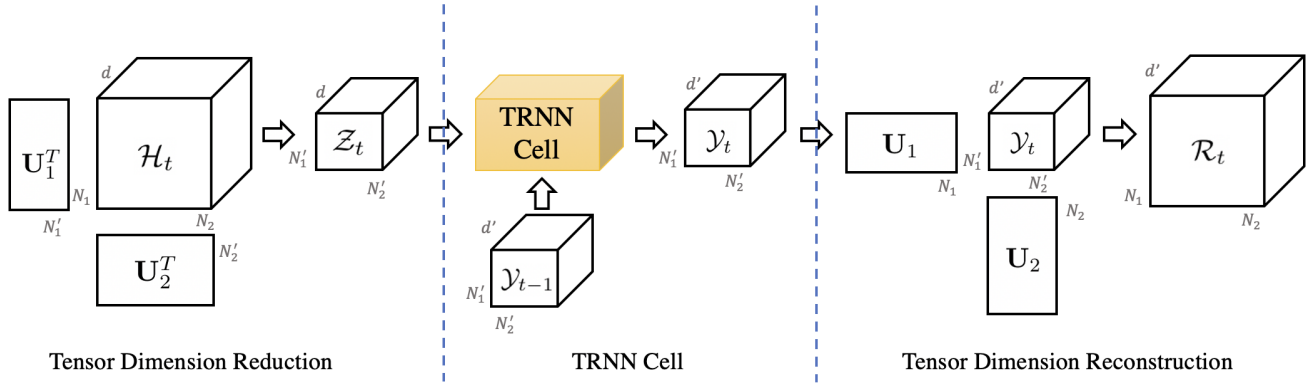



**Figure 4: Tensor Recurrent Neural Network (TRNN).**

where $\mathcal{Z}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d}$ and $\mathcal{Y}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d'}$ denote the input core tensor and the hidden state tensor at the time step $t$; $\mathcal{F}_t$, $\mathcal{I}_t, \mathcal{O}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d'}$ denote the forget gate, the input gate and the output gate, respectively; $\tilde{\mathcal{C}}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d'}$ is the tensor for updating the cell memory $\mathcal{C}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d'}$; $\text{TLL}_*(\cdot)$ denotes the tensor linear layer (Equation (25)), and its subscripts in the above equations are used to distinguish different initialization of TLL[2]; $\sigma(\cdot)$ and $\tanh(\cdot)$ denote the sigmoid activation and tangent activation functions respectively; $\odot$ denotes the Hadamard product.

*3.2.3 Tensor Dimension Reconstruction.* To predict the values of each time series, we need to reconstruct the dimensionality of each mode. Thanks to the orthonormality of $\mathbf{U}_m$ ($\forall m \in [1, \cdots, M]$), we can naturally reconstruct the dimensionality of $\mathcal{Y}_t \in \mathbb{R}^{N_1' \times \cdots \times N_M' \times d'}$ as follows:

$$\mathcal{R}_t = \mathcal{Y}_t \prod_{m=1}^{M} \times_m \mathbf{U}_m \tag{32}$$

where $\mathcal{R}_t \in \mathbb{R}^{N_1 \times \cdots \times N_M \times d'}$ is the reconstructed tensor.

*3.2.4 Implicit Relationship.* The Tucker decomposition (Definition 2.7 and Equation (24) can be regarded as high-order principal component analysis [19]. The matrix $\mathbf{U}_m$ extracts eigenvectors of the $m$-th mode, and each element in $\mathcal{Z}$ indicates the relation between different eigenvectors. We define $\rho \geq 0$ as the indicator of *interaction degree*, such that $N_m' = \rho N_m$ ($\forall m \in [1, \cdots, M]$), to represent to what degree does the TLSTM capture the correlation. The ideal range for $\rho$ is $(0, 1)$. When $\rho = 0$, the TLSTM does not capture any relations and it is reduced to a single LSTM. When $\rho = 1$, the TLSTM captures the relation for each pair of the eigenvectors. When $\rho > 1$, the $\mathbf{U}_m$ is over-complete and contains redundant information.

Despite the dimentionality reduced by Equation (24), it is not guaranteed that the number of parameters in TLSTM will always be less than the number of parameters in multiple separate LSTMs, because of the newly introduced parameters $\mathbf{U}_m$ ($\forall m \in [1, \cdots, M]$).

[2]For all TLL related to $\mathcal{Z}_t$: $\text{TLL}_{*z}(\cdot)$, $\mathbf{W}_m \in \mathbb{R}^{N_m \times N_m'}$ ($\forall m \in [1, \cdots, M]$) and $\mathbf{W}_{M+1} \in \mathbb{R}^{d \times d'}$. For all TLL related to $\mathcal{Y}_{t-1}$: $\text{TLL}_{*y}(\cdot)$, $\mathbf{W}_m \in \mathbb{R}^{N_m' \times N_m'}$ ($\forall m \in [1, \cdots, M]$) and $\mathbf{W}_{M+1} \in \mathbb{R}^{d' \times d'}$.

The following lemma provides an upper-bound for $\rho$ given the dimensions of the input tensor and the hidden dimensions.

Lemma 3.5 (Upper-bound for $\rho$). *Let $N_m$ and $N_m'$ be the dimensions of $\mathbf{U}_m$ in Equation (24), and let $d \in \mathbb{R}$ and $d' \in \mathbb{R}$ be the hidden dimensions of the inputs and outputs of TLSTM. TLSTM uses less parameters than multiple separate LSTMs, as long as the following condition holds:*

$$\rho \leq \sqrt{\frac{(\prod_{m=1}^{M} N_m - 1)d'(d + d' + 1)}{2\sum_{m=1}^{M} N_m^2} + \frac{1}{256}} - \sqrt{\frac{1}{256}} \tag{33}$$

Proof. There are totally $\prod_{m=1}^{M} N_m$ time series in the tensor time series $\mathcal{S} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times T}$, and thus the total number of parameters for $\prod_{m=1}^{M} N_m$ separate LSTM is:

$$\begin{aligned} N^{(LSTM)} &= \prod_{m=1}^{M} N_m [4(dd' + d'd' + d')] \\ &= 4d'(d + d' + 1) \prod_{m=1}^{M} N_m \end{aligned} \tag{34}$$

The total number of parameters for the TLSTM is:

$$N^{(TLSTM)} = 4d'(d + d' + 1) + 8\sum_{m=1}^{M} N_m'^2 + \sum_{m=1}^{M} N_m' N_m \tag{35}$$

where the first two terms on the right side are the numbers of parameters of the TLSTM cell, and the third term is the number of parameters required by $\{\mathbf{U}_m\}_{m=1}^{M}$ in the Tucker decomposition.

Let $\Delta = N^{(TLSTM)} - N^{(LSTM)}$, and let's replace $N_m'$ by $\rho N_m$, then we have:

$$\Delta = (8\rho^2 + \rho)\sum_{m=1}^{M} N_m^2 - 4(\prod_{m=1}^{M} N_m - 1)d'(d + d' + 1) \tag{36}$$

Obviously, $\Delta$ is a convex function of $\rho$. Hence, as long as $\rho$ satisfies the condition specified in the following equation, it can be ensured that the number of parameters is reduced.

$$\rho \leq \sqrt{\frac{(\prod_{m=1}^{M} N_m - 1)d'(d + d' + 1)}{2\sum_{m=1}^{M} N_m^2} + \frac{1}{256}} - \sqrt{\frac{1}{256}} \tag{37}$$

□

### 3.3 Output Module

Given the reconstructed hidden representation tensor obtained from the TRNN: $\mathcal{R}_{t+1} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times d'}$, which captures the temporal dynamics, and the node embedding of the current snapshot $\mathcal{S}_t$: $\mathcal{H}_t \in \mathbb{R}^{N_1 \times \cdots \times N_M \times d}$, the output module is a function mapping $\mathcal{R}_t$ and $\mathcal{H}_t$ to $\mathcal{S}_{t+1} \in \mathbb{R}^{N_1 \times N_2 \cdots \times N_M}$.

We use a Multi-Layer Perceptron (MLP) with a linear output activation as the mapping function:

$$\hat{\mathcal{S}}_{t+1} = \text{MLP}([\mathcal{H}_t, \mathcal{R}_t]) \tag{38}$$

where $\hat{\mathcal{S}}_{t+1} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ represents the predicted snapshot; $\mathcal{H}_t$ and $\mathcal{R}_t$ are the outputs of TGCN and TRNN respectively; and $[\cdot, \cdot]$ denotes the concatenation operation.

### 3.4 Training

Directly training RNNs over the entire sequence is impractical in general [31]. A common practice is to partition the long time series data by a certain window size with $\omega$ historical steps and $\tau$ future steps [21, 23, 37].

Given a time step $t$, let $\{\mathcal{S}_{t'}\}_{t'=t-\omega+1}^{t}$ and $\{\mathcal{S}_{t'}\}_{t'=t+1}^{t+\tau}$ be the historical and the future slices, the objective function of one window slice is defined as:

$$\begin{aligned} \arg\min_{\Theta, \mathcal{W}, \mathcal{B}} & ||\text{NeT}^3(\{\mathcal{S}_{t'}\}_{t'=t-\omega+1}^{t}) - \{\mathcal{S}_{t'}\}_{t'=t+1}^{t+\tau})||_F^2 \\ & + \mu_1 \sum_{t'=t-\omega+1}^{t} ||\mathcal{H}_{t'} - \mathcal{Z}_{t'} \prod_{m=1}^{M} \times_m \mathbf{U}_m||_F^2 \\ & + \mu_2 \sum_{m=1}^{M} ||\mathbf{U}_m \mathbf{U}_m^T - \mathbf{I}_m||_F^2 \end{aligned} \tag{39}$$

where NeT$^3$denotes the proposed model; $\Theta$ and $\mathcal{W}$ represent the parameters of TGCN and TRNN respectively; $\mathcal{B}$ denotes the bias vectors; the second term denotes the reconstruction error of the Tucker decomposition; the third term denotes the orthonormality regularization for $\mathbf{U}_m$, and $\mathbf{I}_m$ denotes identity matrix ($\forall m \in [1, \cdots, M]$); $||\cdot||_F$ is the Frobenius norm; $\mu_1$ and $\mu_2$ are coefficients.

## 4 EXPERIMENTS

In this section, we present the experimental results for the following questions:

- Q1. How accurate is the proposed NeT$^3$on recovering missing value and predicting future value?
- Q2. To what extent does the synergy captured by the proposed TGCN help improve the overall performance of NeT$^3$?
- Q3. How does the interaction degree $\rho$ impact the performance of NeT$^3$?
- Q4. How efficient and scalable is the proposed NeT$^3$?

We first describe the datasets, comparison methods and implementation details in Subsection 4.1, then we provide the results of the effectiveness and efficiency experiments in Subsection 4.2 and Subsection 4.3, respectively.

### 4.1 Experimental Setup

*4.1.1 Datasets.* We evaluate the proposed NeT$^3$ model on five real-world datasets, whose statistics is summarized in Table 1.

**Table 1: Statistics of the datasets.**

| Dataset | Shape | # Nodes | Modes with **A** |
|---|---|---|---|
| *Motes* | $54 \times 4 \times 2880$ | 216 | 1, 2 |
| *Soil* | $42 \times 5 \times 2 \times 365$ | 420 | 1, 2, 3 |
| *Revenue* | $410 \times 3 \times 62$ | 1,230 | 1, 2 |
| *Traffic* | $1000 \times 2 \times 1440$ | 2,000 | 1 |
| *20CR* | $30 \times 30 \times 20 \times 6 \times 180$ | 108,000 | 1, 2, 3, 4 |

*Motes Dataset.* The *Motes* dataset[3] [4] is a collection of reading log from 54 sensors deployed in the Intel Berkeley Research Lab. Each sensor collects 4 types of data, i.e., temperature, humidity, light, and voltage. Following [7], we evaluate all the methods on the log of one day, which has 2880 time steps in total, yielding a $54 \times 4 \times 2880$ tensor time series. We use the average connectivity of each pair of sensors to construct the network for the first mode (54 sensors). As for the network of four data types, we use the Pearson correlation coefficient between each pair of them:

$$\mathbf{A}[i, j] = \frac{1}{2}(r_{ij} + 1) \tag{40}$$

where $r_{ij} \in [-1, 1]$ denotes the Pearson correlation coefficient between the sequence $i$ and the sequence $j$.

*Soil Dataset.* The *Soil* dataset contains one-year log of water temperature and volumetric water content collected from 42 locations and 5 depth levels in the Cook Agronomy Farm (CAF)[4] near Pullman, Washington, USA, [12] which forms a $42 \times 5 \times 2 \times 365$ tensor time series. Since the dataset neither provides the specific location information of sensors nor the relation between the water temperature and volumetric water content, we use Pearson correlation, as shown in Equation (40), to build the adjacency matrices for all the modes.

*Revenue Dataset.* The *Revenue* dataset is comprised of an actual and two estimated quarterly revenues for 410 major companies (e.g. Microsoft Corp.[5], Facebook Inc.[6]) from the first quarter of 2004 to the second quarter of 2019, which yields a $410 \times 3 \times 62$ tensor time series. We construct a co-search network [20] based on log files of the U.S Securities and Exchange Commission (SEC)[7] to represent the correlation among different companies, which is used as the adjacency matrix for the first mode. We also use the Pearson correlation coefficient to construct the adjacency matrix for the three revenues as in Equation (40).

*Traffic Dataset.* The *Traffic* dataset is collected from Caltrans Performance Measurement System (PeMS).[8] Specifically, hourly average speed and occupancy of 1,000 randomly chosen sensor stations in District 7 of California from June 1, 2018, to July 30, 2018, are collected, which yields a $1000 \times 2 \times 1440$ tensor time series. The adjacency matrix $\mathbf{A}_1$ for the first mode is constructed by indicating whether two stations are adjacent: $\mathbf{A}_1[i, j] = 1$ represents the stations $i$ and $j$ are next to each other. As for the second mode,

[3] http://db.csail.mit.edu/labdata/labdata.html
[4] http://www.cafltar.org/
[5] https://www.microsoft.com/
[6] https://www.facebook.com/
[7] https://www.sec.gov/dera/data/edgar-log-file-data-set.html
[8] https://dot.ca.gov/programs/traffic-operations/mobility-performance-reports

since the Pearson correlation between speed and occupancy is not significant, we use identity matrix $\mathbf{I}$ as the adjacency matrix.

*20CR Dataset.* We use the version 3 of the 20th Century Reanalysis data[9][10] [9, 29] collected by the National Oceanic and Atmospheric Administration (NOAA) Physical Sciences Laboratory (PSL). We use a subset of the full dataset, which covers a $30 \times 30$ area of north America, ranging from 30° N to 60° N, 80° W to 110° W, and it contains 20 atmospheric pressure levels. For each of the location point, 6 attributes are used, including air temperature, specific humidity, omega, u wind, v wind and geo-potential height.[11] We use the monthly average data ranging from 2001 to 2015. Therefore, the shape of the data is $30 \times 30 \times 20 \times 6 \times 180$. The adjacency matrix $\mathbf{A}_1$ for the first mode, latitude, is constructed by indicating whether two latitude degrees are next to each other: $\mathbf{A}_1[i, j] = 1$ if $i$ and $j$ are adjacent. The adjacency matrices $\mathbf{A}_2$ and $\mathbf{A}_3$ for the second and the third modes are built in the same way as $\mathbf{A}_1$. We build $\mathbf{A}_4$ for the 6 attributes based on Equation (40).

*4.1.2 Comparison Methods.* We compare our methods with both classic methods (DynaMMo [22], MLDS [27]) and recent deep learning methods (DCRNN [23], STGCN [37]). We also compare the proposed full model NeT$^3$with its ablated versions. To evaluate TGCN, we compare it with MLP, GCN [18] and iTGCN. Here, iTGCN is an ablated version of TGCN, which ignores the synergy between adjacency matrices. The updating function of iTGCN is given by the following equation:

$$\sigma(\sum_{m=1}^{M} \mathcal{X} \times_m \tilde{\mathbf{A}}_m \times_{M+1} \mathbf{\Theta}_m + \mathcal{X} \times_{M+1} \mathbf{\Theta}_0) \tag{41}$$

where $\sigma(\cdot)$ denotes the activation function, $\mathbf{\Theta}$ denotes parameter matrix and $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M \times d}$. For a fair comparison with GCN and the baseline methods, we construct a flat graph by combining the adjacency matrices:

$$\mathbf{A} = \mathbf{A}_M \otimes_k \cdots \otimes_k \mathbf{A}_1 \tag{42}$$

where $\otimes_k$ is Kronecker product, the dimension of $\mathbf{A}$ is $\prod_{m=1}^{M} N_m$, and $N_m$ is the dimension of $\mathbf{A}_m$. To evaluate TLSTM, we compare it with multiple separate LSTMs (mLSTM) and a single LSTM.

*4.1.3 Implementation Details.* For all the datasets and tasks, we use one layer TGCN, one layer TLSTM, and one layer MLP with the linear activation. The hidden dimension is fixed as 8. We fix $\rho = 0.8, 0.8, 0.2, 0.1$ and $0.9$ for TLSTM on *Motes*, *Soil*, *Revenue*, *Traffic*, and *20CR* datasets respectively. The window size is set as $\omega = 5$ and $\tau = 1$, and Adam optimizer [17] with a learning rate of 0.01 is adopted. Coefficients $\mu_1$ and $\mu_2$ are fixed as $10^{-3}$.

[9]20th Century Reanalysis V3 data provided by the NOAA/OAR/ESRL PSL, Boulder, Colorado, USA, from their Web site https://psl.noaa.gov/data/gridded/data.20thC_ReanV3.html

[10]Support for the Twentieth Century Reanalysis Project version 3 dataset is provided by the U.S. Department of Energy, Office of Science Biological and Environmental Research (BER), by the National Oceanic and Atmospheric Administration Climate Program Office, and by the NOAA Physical Sciences Laboratory.

[11]For details of the attributes, please refer to the 20th Century Reanalysis project https://psl.noaa.gov/data/20thC_Rean//

## 4.2 Effectiveness Results

In this section, we present the effectiveness experimental results for missing value recovery, future value prediction, synergy analysis and sensitivity experiments.

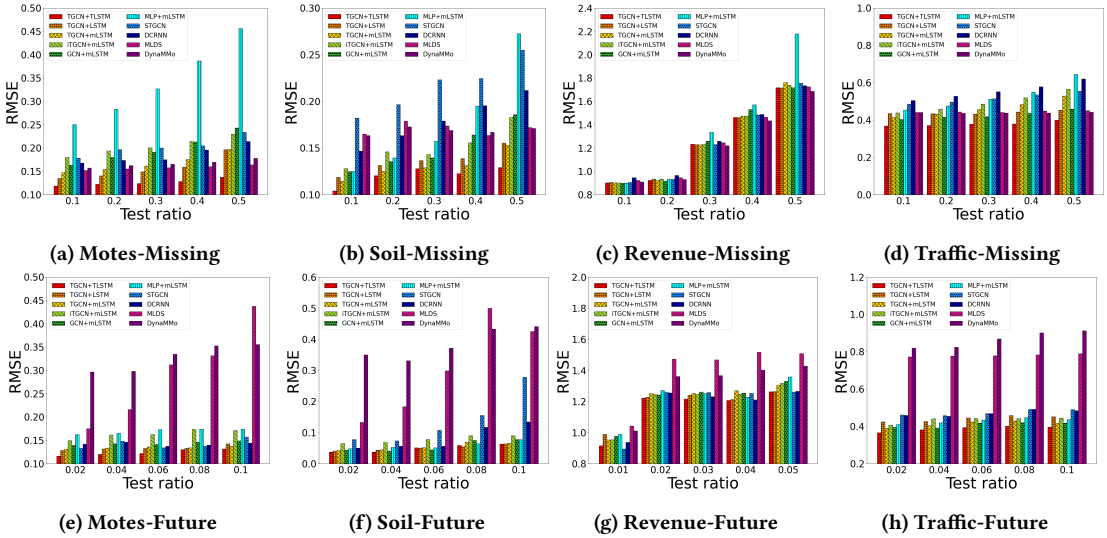
*4.2.1 Missing Value Recovery.* For all the datasets, we randomly select 10% to 50% of the data points as test sets, and we use the mean and standard deviation of each time series in the training sets to normalize each time series. The evaluation results on *Motes*, *Soil*, *Revenue* and *Traffic* are shown in Figure 5a-5d, and the results for *20CR* are presented in 7a. The proposed full model NeT$^3$(TGCN+TLSTM) outperforms all of the baseline methods on almost all of the settings. Among the baselines methods, those equipped with GCNs generally have a better performance than LSTM. When comparing TGCN with iTGCN, we observe that TGCN performs better than iTGCN on most of the settings. This is due to TGCN's capability in capturing various synergy among graphs. We can also observe that TLSTM (TGCN+TLSTM) achieves lower RMSE than both mLSTM (TGCN+mLSTM) and LSTM (TGCN+LSTM), demonstrating the effectiveness of capturing the implicit relations.

*4.2.2 Future Value Prediction.* We use the last 2% to 10% time steps as test sets for the *Motes*, *Traffic*, *Soil* and *20CR* datasets, and we use the last 1% to 5% time steps as test sets for the *Revenue* dataset. Similar to the missing value recovery task, The datasets are normalized by mean and standard deviation of the training sets. The evaluation results are shown in Figure 5e-5h and Figure 7b. The proposed NeT$^3$ outperforms the baseline methods on all of the five datasets. Different from the missing value recovery task, the classic methods perform much worse than deep learning methods on the future value prediction, which might result from the fact that these methods are unable to capture the non-linearity in the temporal dynamics. Similar to the missing value recovery task, generally, TGCN also achives lower RMSE than iTGCN and GCN, and TLSTM performs better than both mLSTM and LSTM.

We present the visualization of the future value prediction task on the *Traffic* dataset in Figure 8.

*4.2.3 Experiments on Synergy.* In this section, we compare the proposed TGCN with iTGCN, $\text{GCN}_1$, $\text{GCN}_2$, $\text{GCN}_3$ and $\text{GCN}_4$ (if applicable) on the missing value recovery and future value prediction tasks. Here, $\text{GCN}_1$, $\text{GCN}_2$, $\text{GCN}_3$ and $\text{GCN}_4$ denote the GCN with the adjacency matrix of the 1st, 2nd, 3rd and 4th mode respectively. iTGCN is an independent version of TGCN (Equation (41)), which is a simple linear combination of different GCNs ($\text{GCN}_1$, $\text{GCN}_2$, $\text{GCN}_3$ and $\text{GCN}_4$). As shown in Figure 6 and Figure 7c-7d, generally, TGCN outperforms GCNs designed for single modes and the simple combination of them (iTGCN).

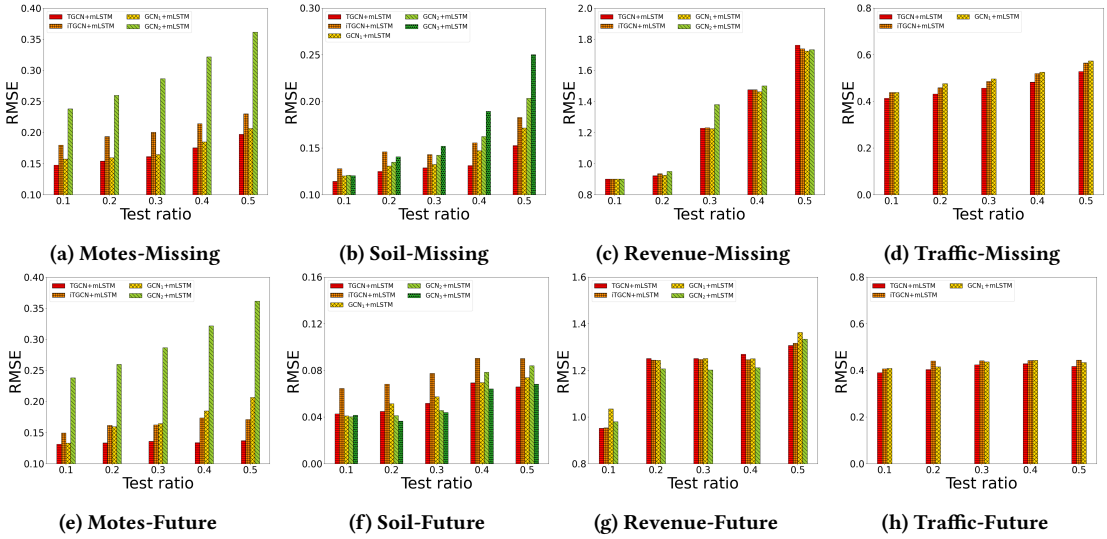
*4.2.4 Sensitivity Experiments.* We use different values of $\rho$ for TLSTM on the *Motes* dataset for the missing value recovery and future value prediction tasks and report their RMSE values in Figure 9a and Figure 9b. It can be noted that, in general, the greater $\rho$ is, the better results (i.e., smaller RMSE) will be obtained. We believe the main reason is that a greater $\rho$ indicates that TLSTM captures more interaction between different time series. Figure 9c shows that the number of parameters of TLSTM is linear with respect to $\rho$.




(a) Motes-Missing (b) Soil-Missing (c) Revenue-Missing (d) Traffic-Missing

(e) Motes-Future (f) Soil-Future (g) Revenue-Future (h) Traffic-Future

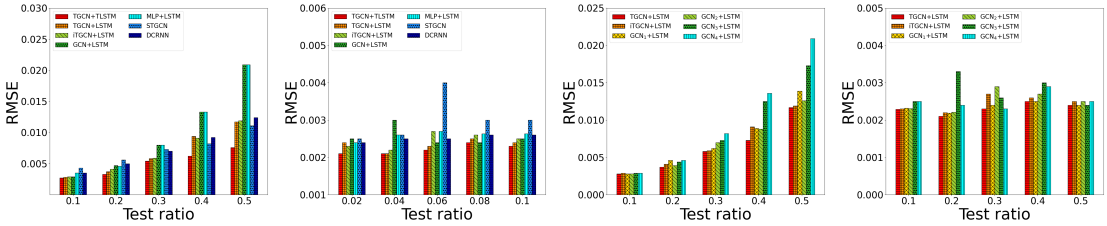
**Figure 5: RMSE of missing value recovery (upper) and future value prediction (lower).**




(a) Motes-Missing (b) Soil-Missing (c) Revenue-Missing (d) Traffic-Missing

(e) Motes-Future (f) Soil-Future (g) Revenue-Future (h) Traffic-Future

**Figure 6: Synergy Analysis: RMSE of missing value recovery (upper) and future value prediction (lower).**




(a) Missing Value Recovery (b) Future Value Prediction (c) Synergy-Missing (d) Synergy-Future

**Figure 7: Experiments on the *20CR* dataset**

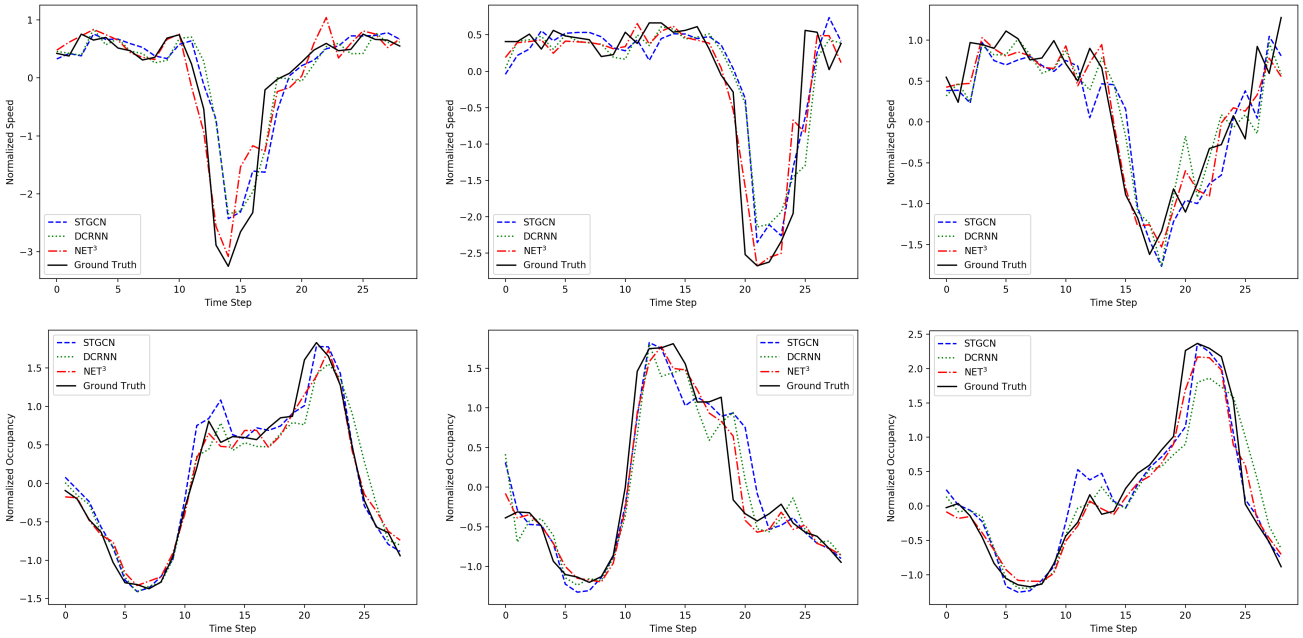
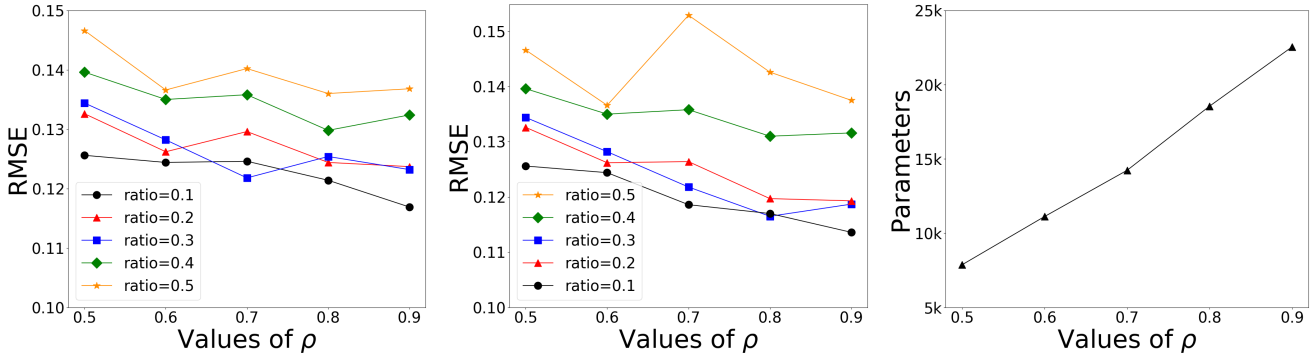



**Figure 8: Visualization of future value prediction on the *Traffic* dataset. Upper part presents the results for normalized speed. Lower part presents the results for normalized occupancy.**




**(a) Missing value recovery** **(b) Future value prediction** **(c) Number of parameters**

**Figure 9: Sensitivity experiments of $\rho$ on the Motes dataset.**

**Table 2: In the upper part, $\rho_{max}$ and $\rho_{exp}$ are the upper bounds and the values of $\rho$ used in experiments. The middle and lower parts present the number of parameters in TLSTM and LSTM, and the parameter reduction ratio.**

| | *Motes* | *Soil* | *Revenue* | *Traffic* | *20CR* |
|---|---|---|---|---|---|
| $\rho_{max}$ | 2.17 | 2.43 | 0.64 | 0.31 | 57.25 |
| $\rho_{exp}$ | 0.80 | 0.80 | 0.20 | 0.10 | 0.90 |
| TLSTM | 18,552 | 10,996 | 87,967 | 180,554 | 16,696 |
| mLSTM | 117,504 | 57,120 | 669,120 | 1,088,000 | 58,752,000 |
| Reduce | 84.21% | 80.75% | 86.85% | 83.40% | 99.97% |

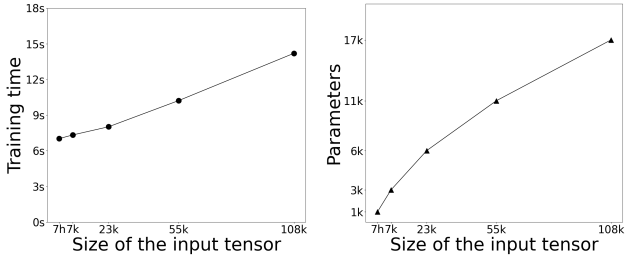



**Figure 10: Scalability experiments.**

## 4.3 Efficiency Results

In this section, we present experimental results for memory efficiency and scalability.

*4.3.1 Memory Efficiency.* As shown in Table 2, the upper bounds ($\rho_{max}$) of $\rho$ for the five datasets are 2.17, 2.43, 0.64, 0.31 and 57.25. In the experiments, we fix $\rho_{exp}$ = 0.80, 0.80, 0.20, 0.10 and 0.90 for the *Motes*, *Soil*, *Revenue*, *Traffic* and *20CR* datasets, respectively. Given the above values of $\rho_{exp}$, the TLSTM row in Table 2 shows the number of parameters in TLSTM. The mLSTM row shows the required number of parameters for multiple separate LSTMs for each single time series. Compared with mLSTM, TLSTM significantly reduces the number of parameters by more than 80% and yet performs better than mLSTM (Figure 5 and Figure 7a-7b).

*4.3.2 Scalability.* We evaluate the scalability of NeT$^3$ on the *20CR* dataset in terms of the training time and the number of parameters. We fix the $\rho = 0.9$, and change the size of the input tensor by shrinking the dimension of all the modes by the specified ratios: [0.2, 0.4, 0.6, 0.8, 1.0]. Given the ratios, the input sizes (the number of nodes) are therefore 684, 6,912, 23,328, 55,296 and 108,000 respectively. The averaged training time of one epoch for TLSTM against the size of the input tensor is presented in left part of Figure 10, and the number of parameters of TLSTM against the size of the input tensor is presented in right part of Figure 10. Note that $h$ and $k$ on the x-axis represent *hundreds* and *thousands* respectively. $s$ and $k$ on the y-axis represent *seconds* and *thousands* respectively. The figures show that the training time and the number of parameters grow almost linearly with the size of input tensor.

# 5 RELATED WORKS

In this section, we review the related work in terms of (1) co-evolving time series, (2) graph convolutional networks (GCN), and (3) networked time series.

## 5.1 Co-evolving Time Series

Co-evolving time series is ubiquitous and appears in a variety of applications, such as enviornmental monitoring, financial analysis and smart transportation. Li et al. [22] proposed a linear dynamic system based on Kalman filter and Bayesian networks to model co-evolving time series. Rogers et al. [27] extended [22] and further proposed a Multi-Linear Dynamic System (MLDS), which provides the base of the proposed TRNN. Yu et al [38] proposed a Temporal Regularized Matrix Factorization (TRMF) for modeling co-evolving time series. Zhou et al. [42] proposed a bi-level model to detect the rare patterns of time series. Recently, Yu et al. [39] used LSTM [15] for modeling traffic flows. Liang et al. [24] proposed a multi-level attention network for geo-sensory time series prediction. Srivastava et al. [30] and Zhou et al. [45] used separate RNNs for weather and air quality monitoring time series. Yu et al. [40] proposed a HOT-RNN based on tensor-trains for long-term forecasting. Zhou et al. [43] proposed a multi-domality neural attention network for financial time series. One limitation of this line of research is that it often ignores the relation network between different time series.

## 5.2 Graph Convolutional Networks

Plenty of real-world data could naturally be represented by a network or graph, such as social networks and sensor networks. Bruna et al. [5] defined spectral graph convolution operation in the Fourier domain by analogizing it to one-dimensional convolution. Henaff et al. [14] used a linear interpolation, and Defferrard et al. [11] adopted Chebyshev polynomials to approximate the spectral graph convolution. Kipf et al. [18] simplified the Chebyshev approximation and proposed a GCN. These methods were typically designed for flat graphs. There are also graph convolutional network methods considering multiple types of relationships. Monti et al. [26] proposed a multi-graph CNN for matrix completion, which does not apply to tensor graphs. Wang et al. [33] proposed HAN which adopted attention mechanism to extract node embedding from different layers of a multiplex network [10, 16, 36], which is a flat graph with multiple types of relations, but not the *tensor graph* in our paper. Liu et al. [25] proposed a TensorGCN for text classification. It is worth pointing out that the term *tensor* in [25] was used in a different context, i.e., it actually refers to a multiplex graph. For a comprehensive review of the graph neural networks, please refer to [34, 41, 44].

## 5.3 Networked Time Series

Relation networks have been encoded into traditional machine learning methods such as dynamic linear [22] and multi-linear [27] systems for co-evolving time series [6, 7, 13]. Recently, Li et al. [23] incorporated spatial dependency of co-evolving traffic flows by the diffusion convolution. Yu et al. [37] used GCN to incorporate spatial relations and CNN for capturing temporal dynamics. Yan et al. [35], introduced a spatial-temporal GCN for skeleton recognition. Li et al. [21] leveraged RGCN [28] to model spatial dependency and LSTM [15] for temporal dynamics. These methods only focus on the relation graphs of a single mode, and ignore relations on other modes e.g. the correlation between the speed and occupancy of the traffic. In addition, these methods rely on the same function for capturing temporal dynamics of all time series.

It is worth pointing out that the proposed NeT$^3$ unifies and supersedes both co-evolving time series and networked time series as a more general data model. For example, if the adjacency matrix $\mathbf{A}_m (m = 1, ..., M)$ for each mode is set as an identity matrix, the proposed NeT$^3$ degenerates to co-evolving (tensor) time series (e.g., [22]); networked time series in [6] can be viewed as a special case of NeT$^3$ whose tensor $\mathcal{X}$ only has a single mode.

# 6 CONCLUSION

In this paper, we introduce a novel NeT$^3$ for jointly modeling of tensor time series with its relation networks. In order to effectively model the tensor with its relation networks at each time step, we generalize the graph convolution from flat graphs to tensor graphs and propose a novel TGCN which not only captures the synergy among graphs but also has a succinct form. To balance the commonality and specificity of the co-evolving time series, we propose a novel TRNN, which helps reduce noise in the data and the number of parameters in the model. Experiments on a variety of real-world datasets demonstrate the efficacy and the applicability of NeT$^3$.

## ACKNOWLEDGMENTS

This work is supported by National Science Foundation under grant No. 1947135, by Agriculture and Food Research Initiative (AFRI) grant no. 2020-67021-32799/project accession no.1024178 from the USDA National Institute of Food and Agriculture, and IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

## REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29, 3 (2015), 626–688.

[2] Viva Banzon, Thomas M Smith, Toshio Mike Chin, Chunying Liu, and William Hankins. 2016. A long-term record of blended satellite and in situ sea-surface temperature for climate monitoring, modeling and environmental studies. *Earth System Science Data* 8, 1 (2016), 165–176.

[3] Martin K Barnett. 1941. A brief history of thermometry. *Journal of Chemical Education* 18, 8 (1941), 358.

[4] Peter Bodik, Wei Hong, Carlos Guestrin, Sam Madden, Mark Paskin, and Romain Thibau. 2004. *Motes Dataset.* http://db.csail.mit.edu/labdata/labdata.html

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[6] Yongjie Cai, Hanghang Tong, Wei Fan, and Ping Ji. 2015. Fast mining of a network of coevolving time series. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 298–306.

[7] Yongjie Cai, Hanghang Tong, Wei Fan, Ping Ji, and Qing He. 2015. Facets: Fast comprehensive mining of coevolving high-order time series. In *KDD*.

[8] Deepayan Chakrabarti and Christos Faloutsos. 2006. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)* 38, 1 (2006), 2–es.

[9] Gilbert P Compo, Jeffrey S Whitaker, Prashant D Sardeshmukh, Nobuki Matsui, Robert J Allan, Xungang Yin, Byron E Gleason, Russell S Vose, Glenn Rutledge, Pierre Bessemoulin, et al. 2011. The twentieth century reanalysis project. *Quarterly Journal of the Royal Meteorological Society* 137, 654 (2011), 1–28.

[10] Manlio De Domenico, Albert Solé-Ribalta, Emanuele Cozzo, Mikko Kivelä, Yamir Moreno, Mason A Porter, Sergio Gómez, and Alex Arenas. 2013. Mathematical formulation of multilayer networks. *Physical Review X* 3, 4 (2013), 041022.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375* (2016).

[12] Caley K Gasch, David J Brown, Erin S Brooks, Matt Yourek, Matteo Poggio, Douglas R Cobos, and Colin S Campbell. 2017. A pragmatic, automated approach for retroactive calibration of soil moisture sensors using a two-step, soil-specific correction. *Computers and Electronics in Agriculture* 137 (2017), 29–40.

[13] N Hairi, Hanghang Tong, and Lei Ying. 2020. NetDyna: Mining Networked Coevolving Time Series with Missing Values. In *IEEE International Conference on Big Data*.

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

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Baoyu Jing, Chanyoung Park, and Hanghang Tong. 2021. HDMI: High-order Deep Multiplex Infomax. *arXiv preprint arXiv:2102.07810* (2021).

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

[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[19] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[20] Charles MC Lee, Paul Ma, and Charles CY Wang. 2015. Search-based peer firms: Aggregating investor perceptions through internet co-searches. *Journal of Financial Economics* 116, 2 (2015), 410–431.

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

[22] Lei Li, James McCann, Nancy S Pollard, and Christos Faloutsos. 2009. Dynammo: Mining and summarization of coevolving sequences with missing values. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 507–516.

[23] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[24] Yuxuan Liang, Songyu Ke, Junbo Zhang, Xiuwen Yi, and Yu Zheng. 2018. Geoman: Multi-level attention networks for geo-sensory time series prediction.. In *IJCAI*. 3428–3434.

[25] Xien Liu, Xinxin You, Xiao Zhang, Ji Wu, and Ping Lv. 2020. Tensor graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8409–8416.

[26] Federico Monti, Michael M Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. *arXiv preprint arXiv:1704.06803* (2017).

[27] Mark Rogers, Lei Li, and Stuart J Russell. 2013. Multilinear dynamical systems for tensor time series. *Advances in Neural Information Processing Systems* 26 (2013), 2634–2642.

[28] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[29] Laura C Slivinski, Gilbert P Compo, Jeffrey S Whitaker, Prashant D Sardeshmukh, Benjamin S Giese, Chesley McColl, Rob Allan, Xungang Yin, Russell Vose, Holly Titchner, et al. 2019. Towards a more reliable historical reanalysis: Improvements for version 3 of the Twentieth Century Reanalysis system. *Quarterly Journal of the Royal Meteorological Society* 145, 724 (2019), 2876–2908.

[30] Shikhar Srivastava and Stefan Lessmann. 2018. A comparative study of LSTM neural networks in forecasting day-ahead global horizontal irradiance with satellite data. *Solar Energy* 162 (2018), 232–247.

[31] Ilya Sutskever. 2013. *Training recurrent neural networks.* University of Toronto Toronto, Canada.

[32] Ruey S Tsay. 2014. Financial time series. *Wiley StatsRef: Statistics Reference Online* (2014), 1–23.

[33] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.

[34] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[35] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[36] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, and Hanghang Tong. 2021. Dynamic Knowledge Alignment. In *AAAI*.

[37] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).

[38] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. 2016. Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction.. In *NIPS*. 847–855.

[39] Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. 2017. Deep learning: A generic approach for extreme condition traffic forecasting. In *Proceedings of the 2017 SIAM international Conference on Data Mining*. SIAM, 777–785.

[40] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. 2017. Long-term forecasting using tensor-train rnns. *Arxiv* (2017).

[41] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* (2019).

[42] D. Zhou, J. He, Y. Cao, and J. Seo. 2016. Bi-Level Rare Temporal Pattern Detection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 719–728. https://doi.org/10.1109/ICDM.2016.0083

[43] Dawei Zhou, Lecheng Zheng, Yada Zhu, Jianbo Li, and Jingrui He. 2020. Domain adaptive multi-modality neural attention network for financial forecasting. In *Proceedings of The Web Conference 2020*. 2230–2240.

[44] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[45] Jingguang Zhou and Zili Huang. 2017. Recover missing sensor data with iterative imputing network. *arXiv preprint arXiv:1711.07878* (2017).