LightNestle: Quick and Accurate Neural Sequential Tensor Completion via Meta Learning

Yuhui Li †, Wei Liang †‡, Kun Xie †, Dafang Zhang †, Songyou Xie †, KuanChing Li ‡
† College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, China
‡ School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, 411201, China
Hunan Key Laboratory for Service Computing and Novel Software Technology, Xiangtan, 411201, China
Email: {liyuhui, dfzhang, syxie}@hnu.edu.cn, cskxie@gmail.com, {wliang, aliric}@hnust.edu.cn

***Abstract*—Network operation and maintenance rely heavily on network traffic monitoring. Due to the measurement overhead reduction, lack of measurement infrastructure, and unexpected transmission error, network traffic monitoring systems suffer from incomplete observed data and high data sparsity problems. Recent studies model missing data recovery as a tensor completion task and show good performance. Although promising, the current tensor completion models adopted in network traffic data recovery lack an effective and efficient retraining scheme to adapt to newly arrived data while retaining historical information. To solve the problem, we propose LightNestle, a novel sequential tensor completion scheme based on meta-learning, which designs (1) an expressive neural network to transfer spatial knowledge from previous embeddings to current embeddings; (2) an attention-based module to transfer temporal patterns into current embeddings in linear complexity; and (3) meta-learning-based algorithms to iteratively recover missing data and update transfer modules to catch up with learned knowledge. We conduct extensive experiments on two real-world network traffic datasets to assess our performance. Results show that our proposed methods achieve both fast retraining and high recovery accuracy.**

***Index Terms*—network measurement, sequential tensor completion, meta learning**

I. Introduction

A. Background and Motivation

Network-wide traffic measurement is a prerequisite for several network operation and maintenance tasks such as anomaly detection [1]–[3], network troubleshooting [4], and network congestion control [5]. To record the network traffic, a traffic matrix is applied to store data volumes between every Origin-Destination (OD) pair. To mine the temporal evolution of network traffic, recent studies compose a series of traffic matrices in chronological order to form a tensor termed Network Monitoring Tensor (NMT) [6]–[8].

The measurement ways to obtain NMT are twofold. The former, network tomography, indirectly measures routing information and link loads to estimate NMT, and unfortunately, the strong assumption and under-determined linear model used in network tomography may not have an exact solution. The latter, direct measurement through OpenTM and NetFlow at the flow-level, results in more precise measurement than indirect measurement.

In this paper, we consider the direct measurement to retrieve NMT. Unfortunately, complete network-wide NMT is almost impossible due to the following reasons: (1) measuring all OD pairs is prohibitively expensive and usually introduces additional overhead to the network that causes observer effects. To reduce the cost and observer effect, only partial OD pairs are measured, (2) network infrastructure may not support direct measurement. Inferring NMT requires the extraction of flow-level statistical information that some devices may not be compatible with, and (3) unpredictable transmission errors (e.g., unreliable protocol, network congestion, and unreachable destination) may also lead to incomplete measurement data [9].

As a result, the observed entries in NMT are always a subset of all entries, indicating that NMT is a sparse tensor. As application such as network state forecasting, anomaly detection, and capacity planning requires intact NMT and less tolerance to incomplete data, this paper focuses on a critical problem termed ***network monitoring data recovery***, aimed to estimate the intact NMT data through a few measurement results organized as a sparse NMT.

B. Prior Arts and Limitations

Sparse representation techniques, such as compress sensing [10], [11] and matrix completion [12], are powerful tools to recover missing data from a few observed entries. Subject to the limited expressiveness of two-dimensional matrix-based methods, recent studies [7], [8], [13] generalize matrix completion to a higher dimensional method called tensor completion for better data recovery.

Despite its effectiveness, the tensor completion scheme is inefficient and ineffective to handle continuously arrived new data. To handle this, one should periodically retrain the model to update the tensor completion results. Current methods to make the model adapt to newly arrived data are retraining on all data, retraining on new data, and retraining by fine-tuning. However, they can not reach a balance between model fidelity, accuracy, and resource consumption, as discussed below:

- **Retraining on all data**. Retraining the tensor completion model to recover missing entries based on observed data, including historical and newly arrived data. Considering the ever-increasing data, this method takes linear-scaling resources to achieve data recovery while preserving the highest model fidelity.
- **Retraining on new data**. Only training the model on newly arrived data is the simplest solution with low resource consumption. However, the sampling rate in network monitoring may be very low (e.g., 2%). It is hard to recover the missing entries without extracting features

Corresponding author: Wei Liang (email: wliang@hnust.edu.cn) from the observed historical entries using merely the new data accurately.

- **Retraining by fine-tuning**. Fine-tune reuses the previous parameters as initialization of the current model and updates them using only the newly collected data. Network traffic data usually change slowly, and the traffic patterns may be very similar. Since the updates are based only on the most recent data without considering historical data, the historical knowledge is not preserved, resulting in forgetting issues [14]. In recovering missing network traffic data, the forgetting issue indicates long-term temporal patterns are no longer preserved. Without explicitly incorporating long-term patterns, the data recovery accuracy may be compromised.

C. Our Contributions

To overcome the limitations of current retraining strategies for tensor completion, we propose a novel approach named **LightNestle** (**Light**weight **Ne**ural **S**equential **T**ensor Comp**le**tion). To complete the missing entries from streaming network traffic data, our scheme split the large-size continuously arrived data into subtensors sequences and trains the model to recover missing data sequentially. We introduce meta learning to optimize the training process and apply it to dedicate designed neural networks to effectively transfer knowledge from previously trained subtensors. Through this design, we only need to train on new data while retaining historical features and achieving fast adaption to new data. Our contributions are as follows:

- We introduce meta-learning in the tensor completion model. We partition streaming network traffic data into a subtensors sequence to capture common knowledge from previously trained subtensors and allow tensor completion to quickly adapt to the next subtensor. Thanks to the excellent characteristics of meta-learning, we achieve 1) quick training convergence on newly arrived data that avoids resource-intensive retraining on all data; 2) preserving rich historic information compared to historical information-lossy retraining by fine-tuning and retraining on new data.
- We propose an expressive multi-layered perceptron (MLP)-based transfer module to capture preference evolution for the origin and destination nodes. Compared with naive MLP which fails to express semantic drift and long short-term patterns with limited parameters, we propose to enhance it by explicitly stacking such patterns as input, and feeding into the two-layer feed-forward network to extract intra- and inter-interactions of input features.
- We propose an attention-based transfer module for time slot embeddings transferring. We highlight the misalignment problem when directly copying time slots embeddings for newly arrived data and propose reordering and reusing strategies to solve this problem. We further propose to apply an attention mechanism to achieve adaptive reordering and reusing time slot embeddings for fast knowledge transfer.
- We conduct experiments on two real-world network traffic datasets to assess our model performance comprehensively. Extensive results demonstrate that LightNestle not only achieves fast adaption on new data but also significantly improves recovery accuracy.

The remainder of this paper is organized as follows. We review related literature in Section II. Section III introduces the problem formulation and the system model design overview. Section IV presents our detailed design and complete solution. Section V evaluates our model performance under different settings. Section VI draws the conclusion.

II. Related Work

Recovering the whole NMT of a network with sparsely observed data is indispensable for downstream tasks such as anomaly detection, traffic engineering, and fault management. Originally, one can purely use spatial or temporal features [15]–[18] for data recovery. With sparse reconstruction theory advanced, some matrix completion-based algorithms [12], [19]–[21] utilize both spatio-temporal features to recover the TM. However, modelling spatial and temporal features in a two-dimensional matrix loses information, and the estimation accuracy is still insufficient.

Recent studies propose to apply the tensor completion to capture more spatio-temporal features in the traffic measurement data for more accurate recovery, which can be can be divided into two categories, mathematical models and neural network models. Recent mathematical models focus on improving accuracy by more careful data utilization. Reshape-Align [22] performs completion on an aligned tensor by reshaping variable sampling rate measurement data. LTC [6] identifies local strong low-rank subtensors via locality-sensitive hashing (LSH) to improve accuracy. OrTC [23] and ETC [24] consider the anomalies and design different methods to reduce the outlier effect while recovering sparse tensor. These studies demonstrate the great potential of highly accurate tensor completion for network traffic data.

With deep learning advanced, recent neural network models [7], [8], [25]–[27] improve tensor completion by introducing non-linearity design into interaction or embedding module. Among them, only [7] and [8] are designed for network traffic tensor completion. For the interaction function, CoSTCo [27] places two convolution layers to capture features from stacked embeddings. NTC [7] and NTM [26] fully exploit interactions inside embeddings by constructing outer-product. To embedding enhancement, NTF [25] and LTP [8] propose temporal embedding refinement techniques based on Recurrent Neural Network (RNN). All these methods achieve better recovery accuracy compared to the linear model.

However, both mathematical and neural network algorithms mentioned above are only focused on improving recovery accuracy rather than efficiently retraining to adapt to newly arrived data.

Meta-learning aims to help models quickly adapt to new data with the help of previously learned tasks. One weakly related field is the recommender system. Recent studies [28]

[29] [30] show fast adaption and cold-start alleviation potential via meta-learning. SML [28] is proposed to avoid costly matrix completion retraining and apply meta-learning to update model parameters for next stage recommendation quickly. IGC [29] and IGCN [30] further apply meta-learning on graph-based recommender systems. These methods either ignore transferring knowledge in the time domain [28], [30], or target incremental graph convolution updates [29]. They are incompatible with tensor completion. Therefore, a novel meta-learning design for recovering network traffic tensor should be derived.

To the best of our knowledge, this is the first work to combine meta-learning with tensor completion algorithms to achieve fast retraining. We see the opportunity to recover a tensor sequentially while using meta-learning to accelerate model training and improve accuracy. Moreover, none of these neural tensor completion methods considers the efficient update when new data arrives. Therefore, we propose our LightNestle to address these issues.

III. Problem Formulation

A. System Model and Problem Definition

Network traffic data is organized in a three-dimensional array called tensor, with its rows being origin nodes, columns being destination nodes, and depth being time slices. Due to the packet loss and overhead reduction, the data tensor $\mathcal{X}$ is sparse. We use a set $\Omega = \{(i,j,k) | \mathcal{X}_{ijk} \neq 0\}$ to indicate the position of observed entries.

As recovering missing entries from sparsely observed data is critical for downstream network operation and management tasks, we focus on solving the data completion tasks. Based on CP decomposition like [6], let $\mathbf{A}, \mathbf{B}, \mathbf{C}$ be factor matrices, the first step of data recovery is to solve:

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C}}{\arg\min} \, \| \mathcal{X} \bullet \Omega - [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] \bullet \Omega \|_F^2, \tag{1}$$

where $\bullet$ denotes element-wise product, and the recovered entry indexed by (i,j,k) can be computed through:

$$\hat{x}_{ijk} = \mathbf{a}_i \bullet \mathbf{b}_j \bullet \mathbf{c}_k = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}, \tag{2}$$

where $\hat{\mathcal{X}}$ denotes the estimated traffic data tensor. $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ denote the i, j, k-th row of factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, respectively.

In real-world network traffic monitoring systems, the streaming monitoring data is coming continuously. For fast data recovery, instead of modelling all data into a single tensor, we split the data into t stages in a non-overlapping manner as $\mathcal{X} = \{\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, ..., \mathbf{X}^{(t)}\}$. Each subtensor has the same number of time slots. Let $\mathbf{X}^{(t)}$ be the newly arrived data. The retraining on all data at stage t can be represented as:

$$\{\mathcal{X}^{(k)} | 0 \leq k \leq t\} \xrightarrow{\text{train}} [\mathbf{A},\mathbf{B},\mathbf{C}] \xrightarrow{\text{estimate}} \hat{\mathcal{X}}. \tag{3}$$

However, directly training on such a large tensor requires many computation resources and a relatively longer training time. We propose to train it in a sequential manner recursively:

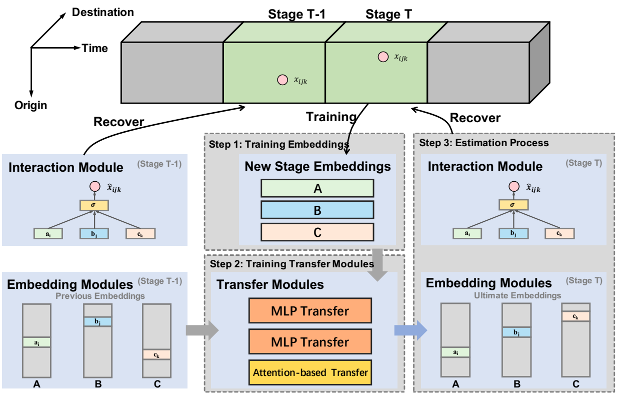

Fig. 1. The framework of the proposed LightNestle method.

$$\begin{tikzcd}
\mathcal{X}^{(t-1)} \arrow[r, "\text{train}"] & {[\mathbf{A},\mathbf{B},\mathbf{C}]} \arrow[r, "\text{estimate}"] \arrow[d, "\text{update}"] & \hat{\mathcal{X}}^{(t-1)} \\
\mathcal{X}^{(t)} \arrow[r, "\text{train}"] & {[\mathbf{A},\mathbf{B},\mathbf{C}]} \arrow[r, "\text{estimate}"] & \hat{\mathcal{X}}^{(t)}.
\end{tikzcd} \tag{4}$$

Both retraining on new data and retraining by fine-tuning can be instances of this training scheme. Retraining on new data abandons the update process while retraining by fine-tuning implements the update process by directly copying the parameters. None of them can solve the dilemma of fast model updates and capturing rich historical information.

The major challenges in sequential training are two folds. First, we fine-tune the parameters in the last stage, the model may meet the forgetting issue and be incapable to incorporate historical patterns. Second, if we do not use the parameters from the last stage and reinitialize the parameters, we lost rich historical information encoded in old parameters while suffering from low convergence speed. To tackle with above challenges, we propose to introduce meta-learning into parameter updates. Meta-learning is naturally suitable for the parameter update process: 1) meta-learning enables incorporating historical information since meta-learning modules capture task-invariant patterns (i.e., different subtensors can be viewed as different tasks), and 2) meta-learning learn better parameter initialization to facilitate fast model adaption.

B. Solution Overview

We propose LightNestle, a meta-learning enhanced tensor completion model, with its architecture shown in Figure 1. To incorporating historical patterns to achieve fast retraining, the core idea behind is to transfer learned historical patterns (encoded in previous embeddings) via a neural network and combine it with current patterns (encoded in new stage embeddings) to generate ultimate embeddings. The LightNestle takes the following three steps to recover the missing data in a new stage:

- **Training Embeddings.** The new stage embeddings encode the information of stage t tensor. In this step, we aim to optimize them quickly with the help of transfer modules. First, we initialize the new stage embeddings the same as the previous stage like retraining by fine-tuning do. Then, instead of directly training on ultimate embedding like retraining on new data and retraining by fine-tuning, we feed new stage embeddings and previous embeddings into transfer modules to predict the ultimate embeddings.

- **Training Transfer Modules.** The transfer modules contain the historical patterns learned from previous subtensors and transfer them from previous subtensors to current tensor. In this step, after training the new stage embeddings, we aim to update the knowledge preserved in transfer modules. We continue to train ultimate embeddings but treat new stage embeddings as constants and update weights in transfer modules. We will detail the design of transfer modules in subsection IV-B and optimize new stage embeddings.
- **Estimation Process.** After new stage embeddings and transfer modules get updated, we use embeddings in the previous stage and the new stage embeddings to compute ultimate embeddings via transfer modules. Given the index of unobserved entries, the prediction values can be estimated.

In LightNestle, thanks to the transfer module design, we have the following merits. (1) Compared to 'retraining on all data", we avoid costly retraining when incorporating historical information to recover newly arrived data while achieving high recovery accuracy. (2) Compared to "retraining on new data", we successfully exploit historical information that "retraining on new data" ignores. The training cost of our scheme remains almost the same, since we only incorporate new data rather than full data, which means it is lightweight and fast. (3) Compared to "retraining by fine-tuning", we explicitly preserves historical information and achieve better knowledge transferring from history to current. In the next section, we dive deep into our model design and present our novel sequential training algorithm.

IV. Design Details of LightNestle

A. Embedding Module

To represent each dimension of a three-way network traffic tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ in the representation learning, following CP decomposition, let R denote the embedding dimension, we set up three randomly initialized factor matrices (a.k.a embedding matrices) $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$ for origins, destinations, and time slots, respectively. Given the indices (i, j, k) as input, we can obtain their corresponding factor vectors $\mathbf{a}_i$, $\mathbf{b}_j$, and $\mathbf{c}_k$ from embedding matrices where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, $\mathbf{b}_j$ is the j-th row of $\mathbf{B}$, and $\mathbf{c}_k$ is the k-th row of $\mathbf{C}$.

B. Transfer Module

Transfer modules play an important role in distilling learned historical patterns from previous embeddings for better ultimate embedding generation. To transfer the knowledge encode in three embedding matrices for a three-way network traffic tensor, we design two transfer modules. There are two different embeddings transfer categories. The origins and destination embeddings are the first categories. The same origin/destination ID indicates the same origin/origination node no matter how time changes. The time slot embedding transferring belongs to the second category. Due to the temporal shifting, the same time slot ID usually does not refer to the same time interval. To deal with these two kinds of embeddings knowledge transfer, we propose an MLP-based transfer module for origins and destinations embeddings transfer and an attention-based transfer module for temporal embedding transfer. Next, we detail the design of these two transfer modules.

MLP-based Transfer Module. Different network traffic patterns can be found in different stages. Current retraining methods handle these patterns differently. Retraining on all data equally treats historical data while retraining on new data fully ignores the historical data. Retraining by fine-tuning implicitly weighs more on recent data. However, they are not sufficient to model dynamically evolved traffic patterns because they do not emphasize the changes between previous weights and current weights wisely.

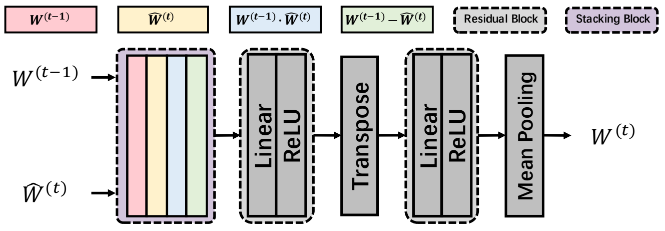

Fig. 2. The architecture of MLP-based transfer modules.

To solve this, we can directly apply an MLP to predict the ultimate embeddings as $\mathbf{w}^{(t)} = \text{MLP}(\mathbf{w}^{(t-1)}, \hat{\mathbf{w}}^{(t)})$[1]. Although MLPs are general function approximators [31], they have limited ability to capture important features for knowledge transfer when the network is not wide and deep enough. For example, feature importance evolution (expressed by $\mathbf{w}_{dot} = \hat{\mathbf{w}^{(t)}} \odot \mathbf{w}^{(t-1)}$) and preference drifting ($\hat{\mathbf{w}^{(t)}} - \mathbf{w}^{(t-1)}$), are fundamental but difficult to learn simultaneously and automatically with naive MLP [32]. To address this problem, we compute and stack these features explicitly and introduce our efficient MLP-based transfer module design for transferring origins and destination embeddings, as shown in Figure 2. Concretely, we first stack $\mathbf{w}^{(t-1)}$, $\hat{\mathbf{w}^{(t)}}$, and their subtraction and dot-product as:

$$H^0 = \begin{bmatrix} \mathbf{w}^{(t-1)} \\ \hat{\mathbf{w}}^{(t)} \\ \mathbf{w}_{dot} \\ \mathbf{w}_{sub} \end{bmatrix}. \tag{5}$$

Next, we learn about interactions inside hidden factors:

$$\mathbf{H}^{(1)} = \mathbf{H}^{(0)} + \text{ReLU}(\mathbf{W}_h \mathbf{H}^{(0)} + b_h), \tag{6}$$

where ReLU(x) $= \max(0, x)$, $\mathbf{W}_h$ and b_h are trainable parameters. After capturing internal interaction, we then extract

[1]Without loss of generality, we treat $\mathbf{w}$ as row vectors, which can be rows of $\mathbf{A}$ and $\mathbf{B}$

the interactions among features in the same dimension. Let $\mathbf{H}^{(2)} = \text{Transpose}(\mathbf{H}^{(1)})$, we formulate this process as:

$$\mathbf{H}^{(3)} = \mathbf{H}^{(2)} + \text{ReLU}(\mathbf{W}_v \mathbf{H}^{(2)} + b_v), \tag{7}$$

where $\mathbf{W}_v$ and b_v are also trainable parameters. Finally, the predicted parameters are computed as:

$$\mathbf{W}^{(t)} = \text{MeanPooling}(\mathbf{H}^{(3)}). \tag{8}$$

Through this design, we ease the training difficulty by explicitly modeling the preference drifting and feature importance evolution. Notably, our MLP-based transfer module only transfers knowledge for the same origins/destinations instead of transferring across origins/destinations. Thus, the time and space complexity of this module is linear.

Attention-based Transfer Module. Learning to transfer knowledge from previous time slots embeddings is different from transferring origins and destinations embeddings. The major difficulty lies in the temporal misalignment problem, which is harmful as it may require more gradient steps to adapt, resulting in a longer adaption time. To understand the temporal misalignment problem and present our solution, we present a graphical illustration in Figure 3. The temporal

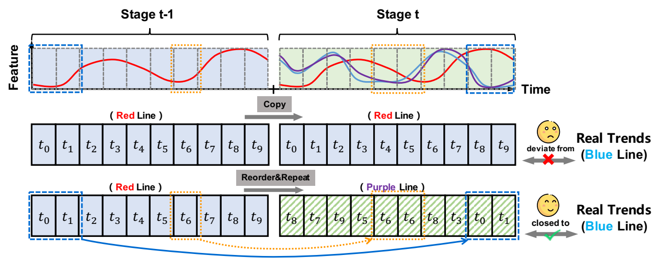
Stage t-1
Stage t
Feature
Time
Copy
(Red Line)
(Red Line)
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
deviate from
Real Trends
(Blue Line)
Reorder&Repeat
(Red Line)
(Purple Line)
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
t_8 t_7 t_9 t_5 t_6 t_6 t_8 t_3 t_0 t_1
closed to
Real Trends
(Blue Line)

Fig. 3. Embedding reordering and repeating for temporal embedding transfer.

misalignment problem reflected in the peak and valley values in the two stages may deviate. As illustrated by red lines in stage $t-1$ and stage t, directly copying previous embeddings leads to identical temporal trends. It may largely deviate from the real trends (marked by blue lines). Therefore, a smarter embedding transferring strategy should be derived to achieve fast adaptation.

Fortunately, the network traffic has periodicity and stability, which inspires us to propose a strategy named "reorder and repeat". The basic rule is to copy the embedding from stage $t-1$ and place it to stage t if the features are the best match. As shown in Figure 3, the blue boxes illustrate the reorder strategy. We can copy embeddings $\mathbf{t}_0$ and $\mathbf{t}_1$ and move them from the head of the stage $t-1$ to the tail of the stage t as the temporal patterns are matched. The yellow boxes show the "repeat" strategy. The embedding t_6 is repeated twice and placed in two slots since the low peak period is twice as long as the previous stage. Noted that, the repeat strategy does not necessarily require the destination time slots to be neighboring. Obviously, with the proposed strategy, we can approximate the real trends in the current stage through embeddings in the previous stage.

The "reorder and repeat" can be expressed by a column transformation matrix with only zeros and ones. However, the binarized matrix is hard to optimize by back-propagation. In addition, the matrix should be generated smartly by incorporating previous embeddings and current new stage embeddings. To overcome these issues, we use a similarity score matrix to approximate it. Such approximation enables the attention mechanism to be exploited. Following the attention mechanism, we measure the similarity scores between queries $\mathbf{Q}$ and keys $\mathbf{K}$ by matrix multiply and then use such scores to compute the weighted sum of values $\mathbf{V}$:

$$\mathbf{C}^{(t)} = \text{Softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V}, \tag{9}$$

where $\mathbf{Q} = \hat{\mathbf{C}}^{(t)}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{C}^{(t-1)}\mathbf{W}_K$, and $\mathbf{V} = \mathbf{C}^{(t-1)}\mathbf{W}_V$ are transformed by trainable weights $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$. We interpret it as using current embeddings to match previous embeddings and predict the ultimate embeddings as a weighted sum of previous embeddings. However, the computation of $\mathbf{Q}\mathbf{K}^T$ is expensive since the time complexity is quadratic w.r.t to the number of time slots. To avoid computing and storing the costly similarity score matrix, we follow [33] to achieve reordering and reusing in linear complexity:

$$\mathbf{C}^{(t)} = \sigma(\mathbf{Q}) \odot \frac{\sum_{t=1}^{T}(\exp(\mathbf{K}_t) \odot \mathbf{V}_t)}{\sum_{t=1}^{T}\exp(\mathbf{K}_t)}, \tag{10}$$

where σ is the sigmoid activation function. T denotes the number of time slot embeddings to be transferred and $\odot$ is an element-wise product. The core idea behind this is to avoid the costly computation of matching queries and keys first but to compute the interaction of keys and values. With interactions of keys and values $\sum_{t=1}^{T}(\exp(\mathbf{K}_t) \odot \mathbf{V}_t) / \sum_{t=1}^{T}\exp(\mathbf{K}_t)$ computed, we have an $T \times d$ matrix which share the same shape with $\sigma(\mathbf{Q})$. Finally, using element-wise product, we maintain the global interaction between queries and values as the attention mechanism does.

C. Interaction Module

In our sequential tensor completion scheme, we use the dot-product-based interaction function, formulated as:

$$\hat{x}_{ijk} = \sigma\left(\sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}\right), \tag{11}$$

where $\sigma(x) = 1/(1+exp(-x))$ is a non-linear activation function, enabling the model to capture non-linearity interaction patterns. We adopt the dot-product-based interaction function for the following reasons. First, it is non-trivial to design an extra parameter transfer module for neural network-based interaction functions as it may contain more than one weight matrix/tensor. Second, using the parameter-free interaction function forces the transfer module to focus on distilling knowledge on embeddings without interfering with interaction functions. Our experiment results demonstrate that the dot-product-based interaction function is still effective in high-accuracy data recovery.

D. Model Training

With all modules introduced, we now consider how to train our model to complete the entire tensor. Algorithm 1 shows the sequential completion process. Specifically, the tensor completion process in a stage has four main steps:

Algorithm 1 Neural Sequential Tensor Completion

```
Input: Sparse Tensor $\mathcal{X}$, Stage Number $T$
Output: Estimated Dense Tensor $\hat{\mathcal{X}}$
1: Split tensor $\mathcal{X}$ into $T$ stages.
2: Initialize Model Parameters
3: $\hat{\mathcal{X}} \leftarrow \emptyset$
4: for $t$=1 to $T$ do
5:     Get current stage tensor $\mathcal{X}^{(t)}$
6:     while not stop condition do
7:         Learn New Stage Embeddings via  Equation 12.
8:     while not stop condition do
9:         Update Transfer Module via  Equation 13.
10:    Estimate stage tensor $\hat{\mathcal{X}}^{(t)}$.
11:    Append $\hat{\mathcal{X}}^{(t)}$ to $\hat{\mathcal{X}}$.
12:    Prepare parameters for next stage.
13: return $\hat{\mathcal{X}}$
```

Step1: Learn New Stage Embeddings. The new stage embeddings $\hat{\mathbf{W}}^{(t)}$ can not randomly initialized. If we random initialize $\hat{\mathbf{W}}^{(t)}$, the $\mathbf{W}^{(t-1)}$,$\hat{\mathbf{W}}^{(t)}$, and $\mathbf{W}^{(t)}$ may not in the same space (i.e. mean values and variances may be different). It may lead to difficulty in optimization and even unsuccessfully learning. In addition, random initialization fails to incorporate historical knowledge. To address these problems, our solution has two measures. First, we let $\hat{\mathbf{W}}^{(t)} = \mathbf{W}^{(t-1)}$ as initialization to ensure all inputs share the same space. Second, we optimize $\hat{\mathbf{W}}^{(t)}$ with sparse tensor $\mathcal{X}^{(t)}$ through transfer modules, which means the back-propagation gradients are calculated with the help of the parameters in transfer module. Let f_Θ denote the entire transfer modules without loss of generality. We minimize tensor completion loss L_c as:

$$L_c(\hat{\mathbf{W}}^{(t)}|\mathcal{X}^{(t)}) = \mathrm{L}(f_\Theta(\mathbf{W}^{(t-1)}, \hat{\mathbf{W}}^{(t)})|\mathcal{X}^{(t)}) + \lambda_1 \|\hat{\mathbf{W}}^{(t)}\|_2^2, \tag{12}$$

where L is the objective function (e.g., mean square loss, mean absolute loss), λ_1 is a hyper-parameters to control model complexity. To prevent the knowledge in transfer modules distorted during the tensor completion stage, we freeze Θ as constant and only update $\hat{\mathbf{W}}^{(t)}$.

Step2: Update Transfer Module. The transfer module parameters Θ are shared in different stages, they contain stage-invariant knowledge of recovering missing data. To prevent modification of well-trained tensor parameter $\hat{\mathbf{W}}^{(t+1)}$ and to update transfer modules only, similar to the previous step, we fix $\hat{\mathbf{W}}^{(t+1)}$ and minimize transfer loss L_t by tensor completion task:

$$L_t(\mathbf{\Theta}|\mathcal{X}^{(t)}) = \mathrm{L}(f_\Theta(\mathbf{W}^{(t-1)}, \hat{\mathbf{W}}^{(t)})|\mathcal{X}^{(t)}) + \lambda_2 \|\mathbf{\Theta}\|_2^2, \tag{13}$$

where λ_2 is regularization hyper-parameters to alleviate overfitting. This process is based on meta-learning since our final parameter $\mathbf{W}^{(t)}$ for tensor recovery is predicted by neural networks. Noted that, it is possible to compute the high-order gradients to further optimize the parameter prediction. However, due to the high computation costs, we conduct the first-order optimization as [34] do.

Step3: Estimation Process. After two steps of model training, we compute the three ultimate embedding matrices for recovering missing data:

$$[\mathbf{A}, \mathbf{B}, \mathbf{C}] = \mathbf{W}^{(t)} = f_\Theta(\mathbf{W}^{(t-1)}, \hat{\mathbf{W}}^{(t)}). \tag{14}$$

Then, we apply interaction modules with three matrices to recover the tensor via Equation 11.

Step4: Prepare Parameters for Next Stage. Before we move to the next stage (Stage=t), we set $\hat{\mathbf{W}}^{(t)} = \mathbf{W}^{(t-1)}$ and store the $\mathbf{W}^{(t-1)}$ as constant which will be used as "previous weight" in the next stage.

E. Complete Solution

The execution process of LightNestle can be described as follow. In the training process, we store the last stage embeddings as constant and reuse them as new stage embeddings. Next, we train new stage embeddings to capture the features from the current stage under parameter-fixed transfer modules. The training process is aided by the knowledge contained in transfer modules, and thus, achieves fast convergence. Then, to update the transfer modules, we freeze new stage embeddings and update the transfer module by optimizing tensor completion. In the estimation process, we use the output of transfer modules as ultimate embeddings and recover the missing data. We repeatedly run the training process and estimation process when the retraining condition is triggered.

V. Experiments

A. Experimental Settings

Datasets. To evaluate the performance of our proposed LightNestle model, we use two widely used public network monitoring datasets:

- **Abilene [35]**: it collects network traffic monitoring data from the U.S. Internet Network that consists of 12 nodes every 5 minutes for 168 days. We select the first 48000 time slices in our experiments.
- **GÉANT [36]**: it records network traffic monitoring data for 23 pan-European research backbone network nodes. The measurement collects data every 15 minutes for consecutive 112 days. We select the first 10000 time slots in our experiments.

Baselines. We compare our model with four types of methods, including four mathematical tensor completion models, four state-of-the-art neural network-based tensor completion methods, and two training schemes:

- **Mathematical Model**. We include HaLRTC [37], FaLRTC [37], TNCP [38], and Tucker-ALS in our performance evaluation. All of them are usual methods in the low-rank tensor completion area. Due to the out-of-memory error, we do not report Tucker-ALS results.
- **Neural Network-based Model**. NTF [25], CoSTCo [27], NTC [7], and NTM [26] are recent state-of-the-art neural network-based tensor completion algorithms. NTF and CoSTCo adopt multi-layered perceptron and convolution neural networks as interaction functions, respectively. NTC and NTM design interaction functions based on outer-product.
- **Training Schemes**. We compare the LightNestle updating scheme with retraining on new data and "retraining by

fine-tuning". For "retraining on new data", we randomly initialize the parameters. For "retraining by fine-tuning", the initial parameters are copied from the previous stage.

- **Variants of LightNestle**. We introduce a novel attention-based transfer module to effectively transfer the temporal embedding. We create a variant that replaces attention-based transfer with an MLP-based transfer module to validate its effectiveness.

Metrics. Let x_{ijk} and $\hat{x}_{ijk}$ denote the ground-truth value and estimated value, $\bar{\Omega}$ denotes the indices of unobserved entries. We apply two commonly used metrics to assess our recovery performance on the unobserved entries:

- **Normalized Root Mean Square Error.** NRMSE takes the form:

$$\text{NRMSE} = \sqrt{\frac{\sum_{(i,j,k)\in\bar{\Omega}}(x_{ijk} - \hat{x}_{ijk})^2}{\sum_{(i,j,k)\in\bar{\Omega}} x_{ijk}^2}} \tag{15}$$

- **Normalized Mean Absolute Error.** NMAE is calculated by:

$$\text{NMAE} = \frac{\sum_{(i,j,k)\in\bar{\Omega}} |x_{ijk} - \hat{x}_{ijk}|}{\sum_{(i,j,k)\in\bar{\Omega}} |x_{ijk}|} \tag{16}$$

For both metrics, NMAE and NRMSE, smaller values indicate better recovery performance.

B. Implementation Details

LightNestle[2] is implemented in PyTorch and performance evaluation on a laptop composed of 16GB memory and 2.2GHz Intel Core CPU running MacOS 12.4. The maximum epochs are 30 (20 epochs for minimizing Equation 12, 10 epochs for minimizing Equation 13). The early stopping is incorporated, i.e., the training loss stops decreasing. The batch size is set to 256. We apply a grid search strategy to determine the best hyper-parameters, including learning rate, weight decay, and optimizer. According to grid search results, we set the learning rate to 0.01 when "learning new stage embeddings" while 0.001 when "update transfer module". The best weight decay is 1e-5. We optimize our model via Adam optimizer. The feature dimension is set to 30 and 20 for Abilene and GÉANT, respectively. The stage size indicates how many time slots are included in tensor completion in a time. We set it to 400 for Abilene and 200 for GÉANT. To ease model training, we clip outliers larger than the 99% percentile to the 99% percentile, then normalize the data by dividing the max value.

C. Performance Comparison with Baselines

Compare LightNestle with Mathematical Model. Figure 4 shows the experimental results of our LightNestle with four mathematical low-rank tensor completion algorithms. We conduct comprehensive experiments to evaluate our model with sampling ratios varying from 2% to 10%. Our proposed model, LightNestle, outperforms the conventional low-rank tensor completion model in all cases with a non-trivial margin, significantly reducing NMAE and NRMSE, demonstrating the superiority of neural network-based tensor completion algorithms. This indicates that the low-rank models, trained with alternative least square or nuclear norm minimization, are insufficient to capture the complex spatial and temporal relationship and thus limit the performance.

The sampling ratios are 2% to 10% in experiment settings. They are low sampling ratios. LightNestle is still effective when the sampling ratio is only 2%, achieving NMAE=0.317 and NRMSE=0.341 on the Abilene dataset. Under the same sample ratio, the best conventional models only achieve NMAE=0.959 and NRMSE=0.834, which are 2.0x and 1.4x better. Similar results can be found in the GÉANT dataset. This proves that LightNestle is very effective in learning complex and non-linear spatial-temporal patterns under a low sampling ratio.

Comparing LightNestle with Neural Model. We present comprehensive experimental results of four state-of-the-art tensor completion models (NTF, CoSTCo, NTC, and NTM) and our proposed model. All baselines are trained on the entire tensor directly while ours is trained sequentially. Compared with all baselines, the proposed LightNestle achieve the best recovery performance on all datasets. Specifically, LightNestle improves NMAE over the best of baselines by 46.06% on Abilene and 113.62% on GÉANT, improves NRMSE by 48.39% and 110.55% on GÉANT, with a sampling ratio low at 2%. NTF, which refines the time slot embeddings with LSTM and applies MLP as an interaction function, has relatively better recovery performance among other baselines. CoSTCo uses 2D-convolution neural networks to capture interactions among embedding vectors without explicitly modeling the temporal dynamics. CoSTCo has relatively poor performance, indicating the importance of temporal pattern learning compared to NTF. NTC and NTM are two representative out-product-based tensor completion algorithms. NTC applies 3D convolutions to capture interactions while NTM linearly transforms the outer product tensor for each mode before feeding into MLP to compute prediction. However, NTC outperforms NTM vastly. This may be because NTC is designed for network monitoring data recovery, while NTM is designed for item recommendation. Perhaps, the linear transformation of outer-product mode by mode is not applicable to network traffic data. Both NTM and NTC are not well-performed when the sampling ratio is low. We achieve surprisingly good performance due to the following reasons. First, our model implicitly incorporates non-linearity feature extraction through transfer modules. Second, the transfer module learns well to initialize the embeddings and include knowledge from previous data. It is similar to the fact that pre-training can improve the model performance.

Table III shows training time compared with four neural tensor completion algorithms. Noted that our LightNestle is evaluated on the CPU, and baselines are assessed on the GPU server for acceleration. We observe that our proposed LightNestle has the fastest training time compared with other models, even though we only use a CPU. The training time reduction is mainly brought by the meta-learning-based transfer

[2]https://github.com/MerrillLi/LightNestle

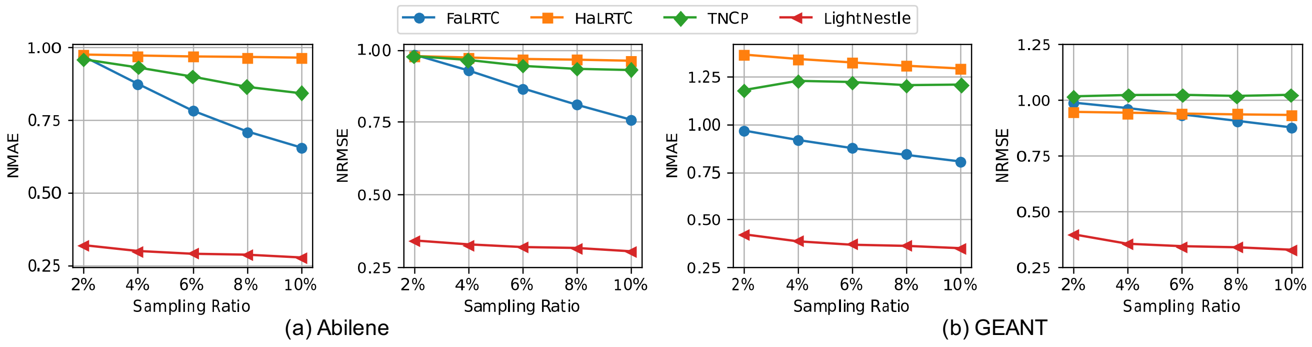

Fig. 4. Comparison between LightNestle and mathematical low-rank tensor completion algorithms.

TABLE I
RECOVERY PERFORMANCE OF NEURAL NETWORK BASED TENSOR COMPLETION APPROACHES

Models	NMAE on Abilene					NRMSE on Abilene				
	2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
NTF	0.463	0.407	0.378	0.362	0.348	0.506	0.443	0.414	0.398	0.381
CoSTCo	0.590	0.560	0.542	0.513	0.480	0.640	0.612	0.595	0.567	0.523
NTC	0.564	0.545	0.510	0.468	0.449	0.609	0.592	0.555	0.509	0.485
NTM	1.123	0.987	0.673	0.649	0.475	1.014	0.922	0.728	0.702	0.535
LightNestle	**0.317**	**0.298**	**0.289**	**0.286**	**0.277**	**0.341**	**0.328**	**0.319**	**0.316**	**0.305**
%Improve.	46.06%	36.58%	30.80%	26.57%	25.63%	48.39%	35.06%	29.78%	25.95%	24.92%
Models	NMAE on GÉANT					NRMSE on GÉANT				
	2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
NTF	0.910	0.719	0.516	0.464	0.442	0.866	0.704	0.506	0.448	0.420
CoSTCo	0.922	0.885	0.856	0.837	0.772	0.875	0.863	0.854	0.825	0.783
NTC	0.910	0.915	0.873	0.785	0.754	0.838	0.834	0.813	0.780	0.749
NTM	1.526	1.488	1.351	1.266	1.286	1.244	1.224	1.149	1.137	1.159
LightNestle	**0.426**	**0.389**	**0.371**	**0.365**	**0.353**	**0.398**	**0.355**	**0.344**	**0.339**	**0.328**
%Improve.	113.62%	84.83%	39.08%	27.12%	25.21%	110.55%	98.31%	47.09%	32.15%	28.05%

TABLE II
COMPARE RESULTS OF TRAINING TIME (SECOND [S]).

Dataset/Model	NTF	CoSTCo	NTC	NTM	**LightNestle**
Abilene	2921s	793s	1025s	1172s	**564s**
GEANT	2152s	559s	730s	842s	**248s**

modules. With better parameter initialization, our model takes minor steps to converge and adapt to the newly collected data, thus saving times.

D. Performance Comparison w.r.t. Training Scheme

We report how we deal with newly collected data in Figure 5. In this experiment, for a fair comparison, ”retraining by fine-tuning” (fine-tune) and ”retraining on new data” (new only) are implemented based on neural-based CP-Decomposition, in which we add a non-linearity transformation layer to enhance embeddings. Fine-tune is one of the intuitive ways to reuse the previous embeddings while incorporating historical information. However, fine-tuning has the worst performance in all cases. Fine-tune can not memorize the long-term origins and destinations preference and fails to capture the evolution of network traffic data. Surprisingly, new only performs even better than fine-tuning. It indicates that directly transferring embeddings of the previous stage as initialization for the next stage is not a proper choice and may encounter the over-fitting issue. Our proposed LightNestle consistently merits other update schemes, demonstrating that our transfer module successfully transfers historical knowledge in recovering the current tensor. We do not include sequentially full retraining in these experiments as their training times were much longer than expected. We can refer to Table I for full retraining performance since they train on the entire tensor directly.

TABLE III
RETRAINING TIME (S) AT EACH WINDOW ON TWO DATASET

Abilene Dataset					
Methods/Density	2%	4%	6%	8%	10%
New Data Only	0.85	1.55	2.58	3.91	4.98
Fine Tune	0.37	0.59	0.77	0.98	1.21
LightNestle	1.27	1.57	1.93	2.43	3.10
GÉANT Dataset					
Methods/Density	2%	4%	6%	8%	10%
New Data Only	1.15	2.37	4.03	5.22	6.55
Fine Tune	0.57	0.93	1.24	1.61	2.17
LightNestle	1.76	2.19	3.42	3.83	3.98

We also show the training time per stage on Table III. We learn that despite retraining by fine-tuning (e.g., 10% sampling rate, 1.21s on Abilene, 2.17s on GÉANT) has the fastest training time while the recovery accuracy is not promising (see Figure 5). Retraining new data has the slowest training time (1.6x slower than LightNestle on both datasets when the sampling ratio is 10%). Compared with LightNestle, it demonstrates that our LightNestle successfully learns better parameter initialization, and thus, is faster in adaption to a new

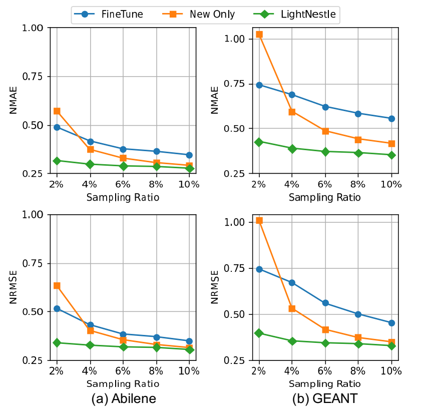
Fig. 5. Comparison of different training schemes.

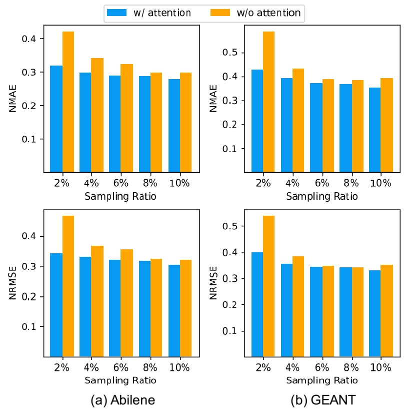
Fig. 6. Ablation study on attention modules.

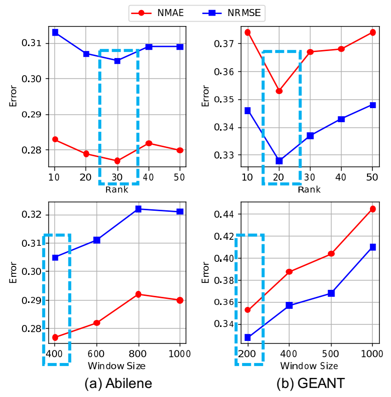
Fig. 7. Analysis of hyper-parameters.

stage. Note that, under the low sampling ratio (e.g., 2%-4%), LightNestle may be slower than training on new data due to the extra overhead from transfer modules. When the sampling rate is high (e.g., 6%-10%), the extra overhead brought by the transfer modules is less than the benefit brought by them.

E. Ablation Study

To show the effectiveness of our attention-based transfer module in reusing temporal embeddings, we compare LightNestle with its variant LightNestle-m, which uses an MLP-based transfer module instead of an attention-based transfer module. Figure 6 reports the recovery performance of our LightNestle (w/ attention) with its variant LightNestle-m (w/o attention). We learn that the performance of LightNestle consistently outperforms its variant without the attention transfer module under different sampling ratios on two datasets. We can draw the following conclusions: 1) The attention module is highly effective when there is a low sampling ratio. That may be because the difference between time slots is more evident when data sparsity. When the sampling ratio is relatively high, the difference may be narrowed due to the local stability nature of monitoring data. 2) The attention transfer module can be considered a time slot embedding refinement trick. Other baselines except NTF do not explicitly consider temporal evolution via neural networks; thus, we achieve better performance.

F. Hyper-Parameter Study

Impact of Dimension. Feature dimension indicates how many hidden factors are captured in the representation learning process. It has a significant impact on both computation resources requirement and model accuracy. We fix other hyper-parameters, vary dimensions from 10 to 50, and draw the recovery performance curve with a sampling ratio set to 10% on all datasets. In Figure 7, the performance improves with the increase of dimension. After the dimension reaches a certain point (30 for Abilene and 20 for GÉANT), the dimension increase degrades the recovery accuracy. The reason may be over-fitting, which increases the model complexity and leads to worse generalization ability. Therefore, we set the dimension to 30 for Abilene and 20 for GÉANT.

Impact of Window Size. The window size indicates how many time slots are included in a stage. It is one of the essential hyper-parameters in our model. It determines the memory consumption, affects the training time, and the model accuracy. In Figure 7, with the increase in window size, the recovery performance of LightNestle keeps degrading. That may be due to fewer transfer module updates, resulting in less task-invariant knowledge captured by meta-learning. In addition, we notice that the transfer modules are parameter-efficient. This may cause learning difficulty when distilling complex patterns from a long tensor. However, the window size should be neither too large nor too small. One should keep a balance between model accuracy and update frequency. Accordingly, we set the window size to 400 for Abilene and 200 for GÉANT.

VI. Conclusions

In this paper, we propose a novel Lightweight Neural Sequential Tensor Completion (LightNestle) method that achieves accurate fast adaption to newly arrived tensors. To achieve this, we first split the long tensor as a subtensors sequence along the time axis. Then, we propose two novel embedding transfer modules to reduce the training cost when retaining rich patterns in historical data. Through extensive experiments on two publicly available network traffic datasets, we demonstrate that LightNestle outperforms both mathematical low-rank models and state-of-the-art neural tensor completion algorithms by a large margin and achieves fast adaption ability on new data.

VII. Acknowledgment

This work was partially supported by the National Key Research and Development Program of China under Grant 2021YFA1000600, the National Natural Science Foundation of China under Grants 62072170, 61976087, 62025201 and 61972144, the Science and Technology Project of the Department of Communications of Hunan Provincial under Grant 202101, the Key Research and Development Program of Hunan Province under Grant 2022GK2015, and the Hunan Provincial Natural Science Foundation of China under Grant 2021JJ30141.

REFERENCES

[1] X. Li, K. Xie, X. Wang, G. Xie, D. Xie, Z. Li, J. Wen, Z. Diao, and T. Wang, "Quick and Accurate False Data Detection in Mobile Crowd Sensing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1339–1352, Jun. 2020.

[2] J. Long, W. Liang, K.-C. Li, Y. Wei, and M. D. Marino, "A regularized cross-layer ladder network for intrusion detection in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1747–1755, 2023.

[3] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, and D. Zhang, "Graph based Tensor Recovery for Accurate Internet Anomaly Detection," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Honolulu, HI: IEEE, Apr. 2018, pp. 1502–1510.

[4] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in *Proceedings of the 2007 ACM CoNEXT conference*, 2007, pp. 1–12.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and Van Jacobson, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, Jan. 2017.

[6] K. Xie, X. Wang, X. Wang, Y. Chen, G. Xie, Y. Ouyang, J. Wen, J. Cao, and D. Zhang, "Accurate Recovery of Missing Network Measurement Data With Localized Tensor Completion," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2222–2235, Dec. 2019.

[7] K. Xie, H. Lu, X. Wang, G. Xie, Y. Ding, D. Xie, J. Wen, and D. Zhang, "Neural tensor completion for accurate network monitoring," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1688–1697.

[8] Y. Ouyang, K. Xie, X. Wang, J. Wen, and G. Zhang, "Lightweight Trilinear Pooling based Tensor Completion for Network Traffic Monitoring," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2128–2137.

[9] W. Liang, Y. Yang, C. Yang, Y. Hu, S. Xie, K.-C. Li, and J. Cao, "Pdpchain: A consortium blockchain-based privacy protection scheme for personal data," *IEEE Transactions on Reliability*, pp. 1–13, 2022.

[10] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 267–278.

[11] D. Jiang, W. Wang, L. Shi, and H. Song, "A Compressive Sensing-Based Approach to End-to-End Network Traffic Reconstruction," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 507–519, 2020.

[12] G. Gürsun and M. Crovella, "On traffic matrix completion in the internet," in *Proceedings of the 2012 Internet Measurement Conference*, 2012, pp. 399–412.

[13] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, G. Zhang, J. Cao, and D. Zhang, "Accurate Recovery of Internet Traffic Data: A Sequential Tensor Completion Approach," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 793–806, Apr. 2018.

[14] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl, "Real-time top-n recommendation in social streams," in *Proceedings of the sixth ACM conference on Recommender systems*, 2012, pp. 59–66.

[15] Z. Li, Y. Zhu, H. Zhu, and M. Li, "Compressive Sensing Approach to Urban Traffic Sensing," in *2011 31st International Conference on Distributed Computing Systems*, Jun. 2011, pp. 889–898.

[16] W. Liang, Y. Li, K. Xie, D. Zhang, K.-C. Li, A. Souri, and K. Li, "Spatial-temporal aware inductive graph neural network for c-its data recovery," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2022.

[17] A. Lakhina, K. Papagiannaki, M. Cro, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural Analysis of Network Traffic Flows," p. 12.

[18] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "Estimating point-to-point and point-to-multipoint traffic matrices: An information-theoretic approach," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 947–960, 2005.

[19] K. Xie, Y. Chen, X. Wang, G. Xie, J. Cao, and J. Wen, "Accurate and Fast Recovery of Network Monitoring Data: A GPU Accelerated Matrix Completion," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 958–971, Jun. 2020.

[20] K. Xie, J. Tian, G. Xie, G. Zhang, and D. Zhang, "Low cost sparse network monitoring based on block matrix completion," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[21] M. Mardani and G. B. Giannakis, "Robust network traffic estimation via sparsity and low rank," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 4529–4533.

[22] K. Xie, C. Peng, X. Wang, G. Xie, J. Wen, J. Cao, D. Zhang, and Z. Qin, "Accurate Recovery of Internet Traffic Data Under Variable Rate Measurements," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1137–1150, Jun. 2018.

[23] Q. Wang, L. Chen, Q. Wang, H. Zhu, and X. Wang, "Anomaly-Aware Network Traffic Estimation via Outlier-Robust Tensor Completion," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2677–2689, Dec. 2020.

[24] K. Xie, S. Li, X. Wang, G. Xie, and Y. Ouyang, "Expectile Tensor Completion to Recover Skewed Network Monitoring Data," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 2021, pp. 1–10.

[25] X. Wu, B. Shi, Y. Dong, C. Huang, and N. V. Chawla, "Neural Tensor Factorization for Temporal Interaction Learning," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. Melbourne VIC Australia: ACM, Jan. 2019, pp. 537–545.

[26] H. Chen and J. Li, "Neural Tensor Model for Learning Multi-Aspect Factors in Recommender Systems," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 2449–2455.

[27] H. Liu, Y. Li, M. Tsang, and Y. Liu, "CoSTCo: A Neural Tensor Completion Model for Sparse Tensors," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, Jul. 2019, pp. 324–334.

[28] Y. Zhang, F. Feng, C. Wang, X. He, M. Wang, Y. Li, and Y. Zhang, "How to retrain recommender system? a sequential meta-learning method," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1479–1488.

[29] S. Ding, F. Feng, X. He, Y. Liao, J. Shi, and Y. Zhang, "Causal Incremental Graph Convolution for Recommender System Retraining," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022.

[30] J. Xia, D. Li, H. Gu, T. Lu, P. Zhang, and N. Gu, "Incremental graph convolutional network for collaborative filtering," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2170–2179.

[31] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[32] S. Rendle, W. Krichene, L. Zhang, and J. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *Fourteenth ACM conference on recommender systems*, 2020, pp. 240–248.

[33] S. Zhai, W. Talbott, N. Srivastava, C. Huang, H. Goh, R. Zhang, and J. Susskind, "An attention free transformer," *arXiv preprint arXiv:2105.14103*, 2021.

[34] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 1126–1135.

[35] "The abilene observatory data collections." [Online]. Available: https://www.cs.utexas.edu/~yzhang/research/AbileneTM/

[36] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.

[37] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor Completion for Estimating Missing Values in Visual Data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 2013.

[38] Y. Liu, F. Shang, L. Jiao, J. Cheng, and H. Cheng, "Trace norm regularized candecomp/parafac decomposition with missing data," *IEEE Transactions on Cybernetics*, vol. 45, no. 11, pp. 2437–2448, 2015.