Cross-Node Federated Graph Neural Network for Spatio-Temporal Data Modeling

Chuizheng Meng
University of Southern California
Los Angeles, California, USA
chuizhem@usc.edu

Sirisha Rambhatla
University of Southern California
Los Angeles, California, USA
sirishar@usc.edu

Yan Liu
University of Southern California
Los Angeles, California, USA
yanliu.cs@usc.edu

ABSTRACT

Vast amount of data generated from networks of sensors, wearables, and the Internet of Things (IoT) devices underscores the need for advanced modeling techniques that leverage the spatio-temporal structure of decentralized data due to the need for edge computation and licensing (data access) issues. While federated learning (FL) has emerged as a framework for model training without requiring direct data sharing and exchange, effectively modeling the complex spatio-temporal dependencies to improve forecasting capabilities still remains an open problem. On the other hand, state-of-the-art spatio-temporal forecasting models assume unfettered access to the data, neglecting constraints on data sharing. To bridge this gap, we propose a federated spatio-temporal model – Cross-Node Federated Graph Neural Network (CNFGNN) – which explicitly encodes the underlying graph structure using graph neural network (GNN)-based architecture under the constraint of cross-node federated learning, which requires that data in a network of nodes is generated locally on each node and remains decentralized. CNFGNN operates by disentangling the temporal dynamics modeling on devices and spatial dynamics on the server, utilizing alternating optimization to reduce the communication cost, facilitating computations on the edge devices. Experiments on the traffic flow forecasting task show that CNFGNN achieves the best forecasting performance in both transductive and inductive learning settings with no extra computation cost on edge devices, while incurring modest communication cost.

CCS CONCEPTS

• **Information systems** → **Sensor networks**; *Data mining*; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Federated Learning; Graph Neural Network; Spatio-Temporal Data Modeling

ACM Reference Format:
Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. 2021. Cross-Node Federated Graph Neural Network for Spatio-Temporal Data Modeling. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467371

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08.
https://doi.org/10.1145/3447548.3467371

1 INTRODUCTION

Modeling the dynamics of spatio-temporal data generated from networks of edge devices or nodes (e.g. sensors, wearable devices and the Internet of Things (IoT) devices) is critical for various applications including traffic flow prediction [18, 32], forecasting [4, 24], and user activity detection [20, 29]. While existing works on spatio-temporal dynamics modeling [5, 6, 14] assume that the model is trained with centralized data gathered from all devices, the volume of data generated at these edge devices precludes the use of such centralized data processing, and calls for decentralized processing where computations on the edge can lead to significant gains in improving the latency. In addition, in case of spatio-temporal forecasting, the edge devices need to leverage the complex inter-dependencies to improve the prediction performance. Moreover, with increasing concerns about data privacy and its access restrictions due to existing licensing agreements, it is critical for spatio-temporal modeling to utilize decentralized data, yet leveraging the underlying relationships for improved performance.

Although recent works in federated learning (FL) [12] provides a solution for training a model with decentralized data on multiple devices, these works either do not consider the inherent spatio-temporal dependencies [13, 17, 21] or only model it implicitly by imposing the graph structure in the regularization on model weights [26], the latter of which suffers from the limitation of regularization based methods due to the assumption that graphs only encode similarity of nodes [15], and cannot operate in settings where only a fraction of devices are observed during training (*inductive learning setting*). As a result, there is a need for an architecture for spatio-temporal data modeling which enables reliable computation on the edge, while maintaining the data decentralized.

To this end, leveraging recent works on federated learning [12], we introduce the *cross-node* federated learning requirement to ensure that data generated locally at a node remains decentralized. Specifically, our architecture – Cross-Node Federated Graph Neural Network (CNFGNN), aims to effectively model the complex spatio-temporal dependencies under the cross-node federated learning constraint. For this, CNFGNN decomposes the modeling of temporal and spatial dependencies using an encoder-decoder model on each device to extract the temporal features with local data, and a Graph Neural Network (GNN) based model on the server to capture spatial dependencies among devices.

As compared to existing federated learning techniques that rely on regularization to incorporate spatial relationships, CNFGNN leverages an explicit graph structure using a graph neural

network-based (GNNs) architecture, which leads to performance gains. However, the federated learning (data sharing) constraint means that the GNN cannot be trained in a centralized manner, since each node can only access the data stored on itself. To address this, CNFGNN employs Split Learning [25] to train the spatial and temporal modules. Further, to alleviate the associated high communication cost incurred by Split Learning, we propose an alternating optimization-based training procedure of these modules, which incurs only half the communication overhead as compared to a comparable Split Learning architecture. Here, we also use Federated Averaging (FedAvg) [21] to train a shared temporal feature extractor for all nodes, which leads to improved empirical performance.

Our main contributions are as follows :

(1) We propose Cross-Node Federated Graph Neural Network (CNFGNN), a GNN-based federated learning architecture that captures complex spatio-temporal relationships among multiple nodes while ensuring that the data generated locally remains decentralized at no extra computation cost at the edge devices.
(2) Our modeling and training procedure enables GNN-based architectures to be used in federated learning settings. We achieve this by disentangling the modeling of local temporal dynamics on edge devices and spatial dynamics on the central server, and leverage an alternating optimization-based procedure for updating the spatial and temporal modules using Split Learning and Federated Averaging to enable effective GNN-based federated learning.
(3) We demonstrate that CNFGNN achieves the best prediction performance (both in transductive and inductive settings) at no extra computation cost on edge devices with modest communication cost, as compared to the related techniques on a traffic flow prediction task.

2 RELATED WORKS

Our method derives elements from graph neural networks, federated learning and privacy-preserving graph learning, we now discuss related works in these areas in relation to our work.

Graph Neural Networks (GNNs). GNNs have shown their superior performance on various learning tasks with graph-structured data, including graph embedding [8], node classification [15], spatio-temporal data modeling [18, 29, 32] and multi-agent trajectory prediction [5, 14, 16]. Recent GNN models [8, 10, 30, 31] also have sampling strategies and are able to scale on large graphs. While GNNs enjoy the benefit from strong inductive bias [6, 28], most works require centralized data during the training and the inference processes.

Federated Learning (FL).. Federated learning is a machine learning setting where multiple clients train a model in collaboration with decentralized training data [12]. It requires that the raw data of each client is stored locally without any exchange or transfer. However, the decentralized training data comes at the cost of less utilization due to the heterogeneous distributions of data on clients and the lack of information exchange among clients. Various optimization algorithms have been developed for federated learning on non-IID and unbalanced data [13, 17, 21]. [26] propose a multi-task learning framework that captures relationships amongst data. While the above works mitigate the caveat of missing neighbors' information to some extent, they are not as effective as GNN models and still suffer from the absence of feature exchange and aggregation.

Alternating Optimization. Alternating optimization is a popular choice in non-convex optimization [1–3, 11]. In the context of Federated Learning, [19] uses alternating optimization for learning a simple global model and reduces the number of communicated parameters, and [9] uses alternating optimization for knowledge distillation from server models to edge models. In our work, we utilize alternating optimization to effectively train on-device modules and the server module jointly, which captures temporal and spatial relationships respectively.

Privacy-Preserving Graph Learning. [27] and [22] use statistics of graph structures instead of node information exchange and aggregation to avoid the leakage of node information. Recent works have also incorporated graph learning models with privacy-preserving techniques such as Differential Privacy (DP), Secure Multi-Party Computation (MPC) and Homomorphic Encryption (HE). [33] utilize MPC and HE when learning a GNN model for node classification with vertically split data to preserve silo-level privacy instead of node-level privacy. [23] preprocesses the input raw data with DP before feeding it into a GNN model. Composing privacy-preserving techniques for graph learning can help build federated learning systems following the privacy-in-depth principle, wherein the privacy properties degrade as gracefully as possible if one technique fails [12].

3 CROSS-NODE FEDERATED GRAPH NEURAL NETWORK

3.1 Problem Formulation

Given a dataset with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a feature tensor $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times \cdots}$ and a label tensor $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}| \times \cdots}$, the task is defined on the dataset with $\mathbf{X}$ as the input and $\mathbf{Y}$ as the prediction target. We consider learning a model under the cross-node federated learning constraint: node feature $\boldsymbol{x}_i = \mathbf{X}_{i,\ldots}$, node label $\boldsymbol{y}_i = \mathbf{Y}_{i,\ldots}$, and model output $\hat{\boldsymbol{y}}_i$ are only visible to the node i.

One typical task that requires the cross-node federated learning constraint is the prediction of spatio-temporal data generated by a network of sensors. In such a scenario, $\mathcal{V}$ is the set of sensors and $\mathcal{E}$ describes relations among sensors (e.g. $e_{ij} \in \mathcal{E}$ if and only if the distance between v_i and v_j is below some threshold). The feature tensor $\boldsymbol{x}_i \in \mathbb{R}^{m \times D}$ represents the i-th sensor's records in the D-dim space during the past m time steps, and the label $\boldsymbol{y}_i \in \mathbb{R}^{n \times D}$ represents the i-th sensor's records in the future n time steps. Since records collected on different sensors owned by different users/organizations may not be allowed to be shared due to the need for edge computation or licensing issues on data access, it is necessary to design an algorithm modeling the spatio-temporal relation without any direct exchange of node-level data.

Algorithm 1 Training algorithm of CNFGNN on the server side.

Input: Initial server-side GN weights $\boldsymbol{\theta}_{GN}^{(0)}$, initial client model weights $\bar{\boldsymbol{\theta}}_c^{(0)} = \{\bar{\boldsymbol{\theta}}_c^{(0),enc}, \bar{\boldsymbol{\theta}}_c^{(0),dec}\}$, the maximum number of global rounds R_g, the maximum number of client rounds R_c, the maximum number of server rounds R_s, server-side learning rate η_s, client learning rate η_c.

Output: Trained server-side GN weights $\boldsymbol{\theta}_{GN}^{(R_g)}$, trained client model weights $\bar{\boldsymbol{\theta}}_c^{(R_g)}$.

Server executes:

Initialize server-side GN weights with $\boldsymbol{\theta}_{GN}^{(0)}$. Initialize client model weights with $\bar{\boldsymbol{\theta}}_c^{(0)}$.
for each node $i \in \mathcal{V}$ in parallel **do**
  Initialize client model $\boldsymbol{\theta}_{c,i}^{(0)} = \bar{\boldsymbol{\theta}}_c^{(0)}$.
  Initialize graph encoding on node $\boldsymbol{h}_{G,c,i} = \boldsymbol{h}_{G,c,i}^{(0)}$.
end for
for global round $r_g = 1, 2, \ldots, R_g$ **do**
  // **(1) Federated learning of on-node models.**
  for each client $i \in \mathcal{V}$ in parallel **do**
    $\boldsymbol{\theta}_{c,i} \leftarrow$ ClientUpdate(i, R_c, η_c).
  end for
  $\bar{\boldsymbol{\theta}}_c^{(r_g)} \leftarrow \sum_{i \in \mathcal{V}} \frac{N_i}{N} \boldsymbol{\theta}_{c,i}$.
  for each client $i \in \mathcal{V}$ in parallel **do**
    Initialize client model: $\boldsymbol{\theta}_{c,i}^{(0)} = \bar{\boldsymbol{\theta}}_c^{(r_g)}$.
  end for
  // **(2) Temporal encoding update.**
  for each client $i \in \mathcal{V}$ in parallel **do**
    $\boldsymbol{h}_{c,i} \leftarrow$ ClientEncode(i).
  end for
  // **(3) Split Learning of GN.**
  Initialize $\boldsymbol{\theta}_{GN}^{(r_g,0)} = \boldsymbol{\theta}_{GN}^{(r_g-1)}$.
  for server round $r_s = 1, 2, \ldots, R_s$ **do**
    $\{\boldsymbol{h}_{G,c,i} | i \in \mathcal{V}\} \leftarrow GN(\{\boldsymbol{h}_{c,i} | i \in \mathcal{V}\}; \boldsymbol{\theta}_{GN}^{(r_g,r_s-1)})$.
    for each client $i \in \mathcal{V}$ in parallel **do**
      $\nabla_{\boldsymbol{h}_{G,c,i}} \ell_i \leftarrow$ ClientBackward(i,$\boldsymbol{h}_{G,c,i}$).
      $\nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell_i \leftarrow \boldsymbol{h}_{G,c,i}$.backward($\nabla_{\boldsymbol{h}_{G,c,i}} \ell_i$).
    end for
    $\nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell \leftarrow \sum_{i \in \mathcal{V}} \nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell_i$.
    $\boldsymbol{\theta}_{GN}^{(r_g,r_s)} \leftarrow \boldsymbol{\theta}_{GN}^{(r_g,r_s-1)} - \eta_s \nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell$.
  end for
  $\boldsymbol{\theta}_{GN}^{(r_g)} \leftarrow \boldsymbol{\theta}_{GN}^{(r_g,R_s)}$.
  // **(4) On-node graph embedding update.**
  $\{\boldsymbol{h}_{G,c,i} | i \in \mathcal{V}\} \leftarrow GN(\{\boldsymbol{h}_{c,i} | i \in \mathcal{V}\}; \boldsymbol{\theta}_{GN}^{(r_g)})$.
  for each client $i \in \mathcal{V}$ in parallel **do**
    Set graph encoding on client as $\boldsymbol{h}_{G,c,i}$.
  end for
end for

Algorithm 2 Training algorithm of CNFGNN on the client side.

ClientUpdate(i, R_c, η_c):

for client round $r_c = 1, 2, \ldots, R_c$ **do**
  $\boldsymbol{h}_{c,i}^{(r_c)} \leftarrow Encoder_i(\boldsymbol{x}_i; \boldsymbol{\theta}_{c,i}^{(r_c-1),enc})$.
  $\hat{\boldsymbol{y}}_i \leftarrow Decoder_i(x_{i,m}, [\boldsymbol{h}_{c,i}^{(r_c)}; \boldsymbol{h}_{G,c,i}]; \boldsymbol{\theta}_{c,i}^{(r_c-1),dec})$.
  $\ell_i \leftarrow \ell(\hat{\boldsymbol{y}}_i, \boldsymbol{y})$.
  $\boldsymbol{\theta}_{c,i}^{(r_c)} \leftarrow \boldsymbol{\theta}_{c,i}^{(r_c-1)} - \eta_c \nabla_{\boldsymbol{\theta}_{c,i}^{(r_c-1)}} \ell_i$.
end for
$\boldsymbol{\theta}_{c,i} = \boldsymbol{\theta}_{c,i}^{(R_c)}$.
return $\boldsymbol{\theta}_{c,i}$ to server.

ClientEncode(i):

return $\boldsymbol{h}_{c,i} = Encoder_i(\boldsymbol{x}_i; \boldsymbol{\theta}_{c,i}^{enc})$ to server.

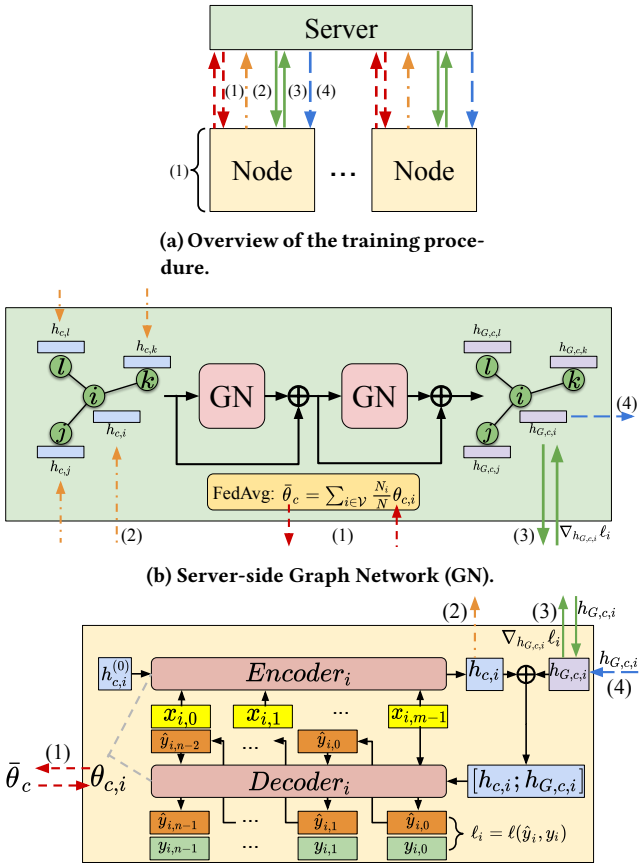
ClientBackward(i, $h_{G,c,i}$):

$\hat{\boldsymbol{y}}_i \leftarrow Decoder_i(x_{i,m}, [h_{c,i}; h_{G,c,i}]; \boldsymbol{\theta}_{c,i}^{dec})$.
$\ell_i \leftarrow \ell(\hat{\boldsymbol{y}}_i, \boldsymbol{y})$.
return $\nabla_{\boldsymbol{h}_{G,c,i}} \ell_i$ to server.

3.2 Proposed Method

We now introduce our proposed Cross-Node Federated Graph Neural Network (CNFGNN) model. Here, we begin by disentangling the modeling of node-level temporal dynamics and server-level spatial dynamics as follows: (i) (Figure 1c) on each node, an encoder-decoder model extracts temporal features from data on the node and makes predictions; (ii) (Figure 1b) on the central server, a Graph Network (GN) [6] propagates extracted node temporal features and outputs node embeddings, which incorporate the relationship information amongst nodes. (i) has access to the not shareable node data and is executed on each node locally. (ii) only involves the upload and download of smashed features and gradients instead of the raw data on nodes. This decomposition enables the exchange and aggregation of node information under the cross-node federated learning constraint.

3.2.1 Modeling of Node-Level Temporal Dynamics. We modify the Gated Recurrent Unit (GRU) based encoder-decoder architecture in [7] for the modeling of node-level temporal dynamics on each node. Given an input sequence $\boldsymbol{x}_i \in \mathbb{R}^{m \times D}$ on the i-th node, an encoder



(a) Overview of the training procedure.

(b) Server-side Graph Network (GN).

(c) Encoder-decoder on the i-th node.

Figure 1: Cross-Node Federated Graph Neural Network. (a) In each round of training, we alternately train models on nodes and the model on the server. More specifically, we sequentially execute: (1) Federated learning of on-node models. (2) Temporal encoding update. (3) Split Learning of GN. (4) On-node graph embedding update. (b) Detailed view of the server-side GN model for modeling spatial dependencies in data. (c) Detailed view of the encoder-decoder model on the i-th node.

sequentially reads the whole sequence and outputs the hidden state $\boldsymbol{h}_{c,i}$ as the summary of the input sequence according to Equation 1.

$$\boldsymbol{h}_{c,i} = Encoder_i(\boldsymbol{x}_i, \boldsymbol{h}_{c,i}^{(0)}), \tag{1}$$

where $\boldsymbol{h}_{c,i}^{(0)}$ is a zero-valued initial hidden state vector.

To incorporate the spatial dynamics into the prediction model of each node, we concatenate $\boldsymbol{h}_{c,i}$ with the node embedding $\boldsymbol{h}_{G,c,i}$ generated from the procedure described in 3.2.2, which contains spatial information, as the initial state vector of the decoder. The decoder generates the prediction $\hat{\boldsymbol{y}}_i$ in an auto-regressive way starting from the last frame of the input sequence $x_{i,m}$ with the concatenated hidden state vector.

$$\hat{\boldsymbol{y}}_i = Decoder_i(x_{i,m}, [\boldsymbol{h}_{c,i}; \boldsymbol{h}_{G,c,i}]). \tag{2}$$

We choose the mean squared error (MSE) between the prediction and the ground truth values as the loss function, which is evaluated on each node locally.

3.2.2 Modeling of Spatial Dynamics. To capture the complex spatial dynamics, we adopt Graph Networks (GNs) proposed in [6] to generate node embeddings containing the relational information of all nodes. The central server collects the hidden state from all nodes $\{\boldsymbol{h}_{c,i} \mid i \in \mathcal{V}\}$ as the input to the GN. Each layer of GN updates the input features as follows:

$$\begin{aligned} \mathbf{e}_k' &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}_i' &= \rho^{e\to v}\left(E_i'\right) \\ \mathbf{v}_i' &= \phi^v\left(\bar{\mathbf{e}}_i', \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e\to u}\left(E'\right) \\ \mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v\to u}\left(V'\right) \end{aligned}, \tag{3}$$

where $\mathbf{e}_k, \mathbf{v}_i, \mathbf{u}$ are edge features, node features and global features respectively. ϕ^e, ϕ^v, ϕ^u are neural networks. $\rho^{e\to v}, \rho^{e\to u}, \rho^{v\to u}$ are aggregation functions such as summation. As shown in Figure 1b, we choose a 2-layer GN with residual connections for all experiments. We set $\mathbf{v}_i = \boldsymbol{h}_{c,i}$, $\mathbf{e}_k = W_{r_k,s_k}$ (W is the adjacency matrix) , and assign the empty vector to $\mathbf{u}$ as the input of the first GN layer. The server-side GN outputs embeddings $\{\boldsymbol{h}_{G,c,i} \mid i \in \mathcal{V}\}$ for all nodes, and sends the embedding of each node correspondingly.

3.2.3 Alternating Training of Node-Level and Spatial Models. One challenge brought about by the cross-node federated learning requirement and the server-side GN model is the high communication cost in the training stage. Since we distribute different parts of the model on different devices, Split Learning proposed by [25] is a potential solution for training, where hidden vectors and gradients are communicated among devices. However, when we simply train the model end-to-end via Split Learning, the central server needs to receive hidden states from all nodes and to send node embeddings to all nodes in the forward propagation, then it must receive gradients of node embeddings from all nodes and send back gradients of hidden states to all nodes in the backward propagation. Assume all hidden states and node embeddings have the same size S, the total amount of data transmitted in each training round of the GN model is $4|\mathcal{V}|S$.

To alleviate the high communication cost in the training stage, we instead alternately train models on nodes and the GN model on the server. More specifically, in each round of training, we (1) fix the node embedding $\boldsymbol{h}_{G,c,i}$ and optimize the encoder-decoder model for R_c rounds, then (2) we optimize the GN model while fixing all models on nodes. Since models on nodes are fixed, $\boldsymbol{h}_{c,i}$ stays constant during the training of the GN model, and the server only needs to fetch $\boldsymbol{h}_{c,i}$ from nodes before the training of GN starts and only to communicate node embeddings and gradients. Therefore, the average amount of data transmitted in each round for Rs rounds of training of the GN model reduces to $\frac{2+2R_s}{R_s}|\mathcal{V}|S$. We provide more details of the training procedure in Algorithm 1 and Algorithm 2.

To more effectively extract temporal features from each node, we also train the encoder-decoder models on nodes with the FedAvg algorithm proposed in [21]. This enables all nodes to share the same feature extractor and thus share a joint hidden space of temporal features, which avoids the potential overfitting of models on nodes and demonstrates faster convergence and better prediction performance empirically.

4 EXPERIMENTS

We evaluate the performance of CNFGNN and all baseline methods on the traffic forecasting task, which is an important application for spatio-temporal data modeling. The primary challenge in FL is to respect constraints on data sharing and manipulation. These constraints can occur in scenarios where data contains sensitive information, such as financial data owned by different institutions. Due to the sensitivity of data, datasets from such scenarios are proprietary and hardly offer public access. Therefore, we demonstrate the applicability of our proposed model on the traffic dataset, which is a good representative example of data with spatio-temporal correlations, and has been extensively studied in spatio-temporal forecasting works without FL constraints [18, 32]. Our proposed model is general and applicable to other spatio-temporal datasets with sensitive information.

We reuse the following two real-world large-scale datasets in [18] and follow the same preprocessing procedures: (1) **PEMS-BAY**: This dataset contains the traffic speed readings from 325 sensors in the Bay Area over 6 months from Jan 1st, 2017 to May 31st, 2017. (2) **METR-LA**: This dataset contains the traffic speed readings from 207 loop detectors installed on the highway of Los Angeles County over 4 months from Mar 1st, 2012 to Jun 30th, 2012.

For both datasets, we construct the adjacency matrix of sensors using the Gaussian kernel with a threshold: $W_{i,j} = d_{i,j}$ if $d_{i,j} >= \kappa$ else 0, where $d_{i,j} = \exp(-\frac{\text{dist}(v_i,v_j)^2}{\sigma^2})$, $\text{dist}(v_i, v_j)$ is the road network distance from sensor v_i to sensor v_j, σ is the standard deviation of distances and κ is the threshold. We set $\kappa = 0.1$ for both datasets.

We aggregate traffic speed readings in both datasets into 5-minute windows and truncate the whole sequence to multiple sequences with length 24. The forecasting task is to predict the traffic speed in the following 12 steps of each sequence given the first 12 steps. We show the statistics of both datasets in Table 1.

Table 1: Statistics of datasets PEMS-BAY and METR-LA.

Dataset	# Nodes	# Directed Edges	# Train Seq	# Val Seq	# Test Seq
PEMS-BAY	325	2369	36465	5209	10419
METR-LA	207	1515	23974	3425	6850

4.1 Spatio-Temporal Data Modeling: Traffic Flow Forecasting

Baselines. Here we introduce the settings of baselines and our proposed model CNFGNN. Unless noted otherwise, all models are optimized using the Adam optimizer with the learning rate 1e-3.

- **GRU (centralized)**: Gated Recurrent Unit (GRU) model trained with centralized sensor data. The GRU model with 63K parameters is a 1-layer GRU with hidden dimension 100, and the GRU model with 727K parameters is a 2-layer GRU with hidden dimension 200.
- **GRU + GN (centralized)**: a model directly combining GRU and GN trained with centralized data, whose architecture is similar to CNFGNN but all GRU modules on nodes always share the same weights. We see its performance as the upper bound of the performance of CNFGNN.
- **GRU (local)**: for each node we train a GRU model with only the local data on it.
- **GRU + FedAvg**: a GRU model trained with the Federated Averaging algorithm [21]. We select 1 as the number of local epochs.
- **GRU + FMTL**: for each node we train a GRU model using the federated multi-task learning (FMTL) with cluster regularization [26] given by the adjacency matrix. More specifically, the cluster regularization (without the L2-norm regularization term) takes the following form:

$$\mathcal{R}(\boldsymbol{W}, \Omega) = \lambda \text{tr}(\boldsymbol{W}\Omega\boldsymbol{W}^T). \tag{4}$$

Given the constructed adjacency matrix $\boldsymbol{A}$, $\Omega = \frac{1}{|\mathcal{V}|}(\boldsymbol{D} - \boldsymbol{A}) = \frac{1}{|\mathcal{V}|}\boldsymbol{L}$, where $\boldsymbol{D}$ is the degree matrix and $\boldsymbol{L}$ is the Laplacian matrix. Equation 4 can be reformulated as:

$$\begin{aligned}\mathcal{R}(\boldsymbol{W}, \Omega) &= \lambda \text{tr}(\boldsymbol{W}\Omega\boldsymbol{W}^T) \\ &= \frac{\lambda}{|\mathcal{V}|}\text{tr}(\boldsymbol{W}\boldsymbol{L}\boldsymbol{W}^T) \\ &= \frac{\lambda}{|\mathcal{V}|}\text{tr}(\sum_{i\in\mathcal{V}} \boldsymbol{w}_i \sum_{j\neq i} a_{ij}\boldsymbol{w}_i^T - \sum_{j\neq i} \boldsymbol{w}_i a_{ij} \boldsymbol{w}_j^T) \\ &= \lambda_1(\sum_{i\in\mathcal{V}}\sum_{j\neq i} \alpha_{i,j}\langle \boldsymbol{w}_i, \boldsymbol{w}_i - \boldsymbol{w}_j\rangle).\end{aligned} \tag{5}$$

We implement the cluster regularization via sharing model weights between each pair of nodes connected by an edge and select $\lambda_1 = 0.1$. For each baseline, we have 2 variants of the GRU model to show the effect of on-device model complexity: one with 63K parameters and the other with 727K parameters. For CNFGNN, the encoder-decoder model on each node has 64K parameters and the GN model has 1M parameters.
- **CNFGNN** We use a GRU-based encoder-decoder model as the model on nodes, which has 1 GRU layer and hidden dimension 64. We use a 2-layer Graph Network (GN) with residual connections as the Graph Neural Network model on the server side. We use the same network architecture for the edge/node/global update function in each GN layer: a multi-layer perceptron (MLP) with 3 hidden layers, whose sizes are [256, 256, 128] respectively. We choose $R_c = 1, R_s = 20$ for experiments on PEMS-BAY, and $R_c = 1, R_s = 1$ for METR-LA.

Calculation of Communication Cost. We denote R as the number of communication rounds for one model to reach the lowest validation error in the training stage.

GRU + FMTL. Using Equation 5, in each communication round, each pair of nodes exchange their model weights, thus the total communicated data amount is calculated as:

$$R \times \#\text{nonself directed edges} \times \text{size of node model weights}. \tag{6}$$

We list corresponding parameters in Table 2.

CNFGNN (AT + FedAvg). In each communication round, the central server fetches and sends back model weights to each node for Federated Averaging, and transmits hidden vectors and gradients for Split Learning. The total communicated data amount is

calculated as:

$$R \times (\#\text{nodes} \times \text{size of node model weights} \times 2 + (1 + 2 * \text{server round} + 1) \times \#\text{nodes} \times \text{hidden state size}). \quad (7)$$

We list corresponding parameters in Table 3.

CNFGNN (SL). In each communication round, each node sends and fetches hidden vectors and graidents twice (one for encoder, the other for decoder) and the total communicated data amount is:

$$R \times 2 \times 2 \times \#\text{nodes} \times \text{hidden state size}. \quad (8)$$

We list corresponding parameters in Table 4.

CNFGNN (SL + FedAvg). Compared to CNFGNN (SL), the method has extra communcation cost for FedAvg in each round, thus the total communicated data amount is:

$$R \times (\#\text{nodes} \times \text{size of node model weights} \times 2 + 2 \times 2 \times \#\text{nodes} \times \text{hidden state size}). \quad (9)$$

We list corresponding parameters in Table 5.

CNFGNN (AT, w/o FedAvg). Compared to CNFGNN (AT + FedAvg), there is no communcation cost for the FedAvg part, thus the total communcated data amount is:

$$R \times (1 + 2 * \text{server round} + 1) \times \#\text{nodes} \times \text{hidden state size}. \quad (10)$$

We list corresponding parameters in Table 6.

Table 2: Parameters used for calculating the communication cost of GRU + FMTL.

Method		GRU (63K) + FMTL	GRU (727K) + FMTL
Node Model Weights Size (GB)		2.347E-4	2.708E-3
PEMS-BAY	#Nonself Directed Edges	2369	
	R	104	56
	Train Comm Cost (GB)	57.823	359.292
METR-LA	#Nonself Directed Edges	1515	
	R	279	176
	Train Comm Cost (GB)	99.201	722.137

Table 3: Parameters used for calculating the communication cost of CNFGNN (AT + FedAvg).

Node Model Weights Size (GB)		2.384E-4
PEMS-BAY	#Nodes	325
	Hidden State Size (GB)	2.173E-3
	Server Round	20
	R	2
	Train Comm Cost (GB)	237.654
METR-LA	#Nodes	207
	Hidden State Size (GB)	1.429E-3
	Server Round	1
	R	46
	Train Comm Cost (GB)	222.246

Table 4: Parameters used for calculating the communication cost of CNFGNN (SL).

PEMS-BAY	#Nodes	325
	Hidden State Size (GB)	2.173E-3
	R	31
	Train Comm Cost (GB)	350.366
METR-LA	#Nodes	207
	Hidden State Size (GB)	1.429E-3
	R	65
	Train Comm Cost (GB)	307.627

Table 5: Parameters used for calculating the communication cost of CNFGNN (SL + FedAvg).

Node Model Weights Size (GB)		2.384E-4
PEMS-BAY	#Nodes	325
	Hidden State Size (GB)	2.173E-3
	R	7
	Train Comm Cost (GB)	80.200
METR-LA	#Nodes	207
	Hidden State Size (GB)	1.429E-3
	R	71
	Train Comm Cost (GB)	343.031

Table 6: Parameters used for calculating the communication cost of CNFGNN (AT, w/o FedAvg).

PEMS-BAY	#Nodes	325
	Hidden State Size (GB)	2.173E-3
	Server Round	20
	R	44
	Train Comm Cost (GB)	5221.576
METR-LA	#Nodes	207
	Hidden State Size (GB)	1.429E-3
	Server Round	1
	R	49
	Train Comm Cost (GB)	2434.985

Discussion. Table 7 shows the comparison of forecasting performance and Table 8 shows the comparison of computation cost on device and communication cost of CNFGNN and baselines. We make the following observations. Firstly, when we compare the best forecasting performance of each baseline over the 2 GRU variants, GRU trained with FedAvg performs the worst in terms of forecasting performance compared to GRU trained with centralized data and GRU trained with local data (4.432 vs 4.010/4.124 on PEMS-BAY and 12.058 vs 11.730/11.801 on METR-LA), showing that the data distributions on different nodes are highly heterogeneous, and training one single model ignoring the heterogeneity is suboptimal.

Table 7: Comparison of performance on the traffic flow forecasting task. We use the Rooted Mean Squared Error (RMSE) to evaluate the forecasting performance.

Method	PEMS-BAY	METR-LA
GRU (centralized, 63K)	4.124	11.730
GRU (centralized, 727K)	4.128	11.787
GRU + GN (centralized, 64K + 1M)	**3.816**	**11.471**
GRU (local, 63K)	4.010	11.801
GRU (local, 727K)	4.152	12.224
GRU (63K) + FedAvg	4.512	12.132
GRU (727K) + FedAvg	4.432	12.058
GRU (63K) + FMTL	3.961	11.548
GRU (727K) + FMTL	3.955	11.570
CNFGNN (64K + 1M)	**3.822**	**11.487**

Table 8: Comparison of the computation cost on edge devices and the communication cost. We use the amount of floating point operations (FLOPS) to measure the computational cost of models on edge devices. We also show the total size of data/parameters transmitted in the training stage (Train Comm Cost) until the model reaches its lowest validation error.

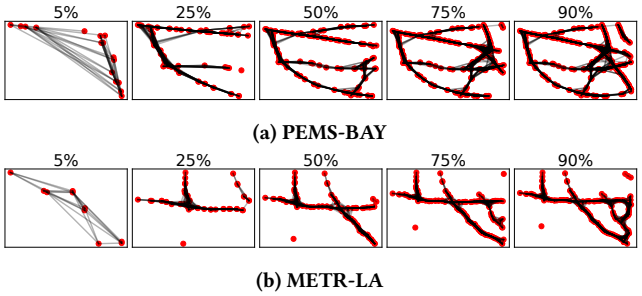
Method	Comp Cost On Device (GFLOPS)	PEMS-BAY		METR-LA	
		RMSE	Train Comm Cost (GB)	RMSE	Train Comm Cost (GB)
GRU (63K) + FMTL	0.159	3.961	57.823	11.548	99.201
GRU (727K) + FMTL	1.821	3.955	359.292	11.570	722.137
CNFGNN (64K + 1M)	0.162	**3.822**	237.654	**11.487**	222.246

Secondly, both the GRU+FMTL baseline and CNFGNN consider the spatial relations among nodes and show better forecasting performance than baselines without relation information. This shows that the modeling of spatial dependencies is critical for the forecasting task.

Lastly, CNFGNN achieves the lowest forecasting error on both datasets. The baselines that increases the complexity of on-device models (GRU (727K) + FMTL) gains slight or even no improvement at the cost of higher computation cost on edge devices and larger communication cost. However, due to its effective modeling of spatial dependencies in data, CNFGNN not only has the largest improvement of forecasting performance, but also keeps the computation cost on devices almost unchanged and maintains modest communication cost compared to baselines increasing the model complexity on devices.

4.2 Inductive Learning on Unseen Nodes

Set-up. Another advantage of CNFGNN is that it can conduct inductive learning and generalize to larger graphs with nodes unobserved during the training stage. We evaluate the performance of CNFGNN under the following inductive learning setting: for each dataset, we first sort all sensors based on longitudes, then use the subgraph on the first $\eta\%$ of sensors to train the model and evaluate



(a) PEMS-BAY

(b) METR-LA

Figure 2: Visualization of subgraphs visible in training under different ratios.

the trained model on the entire graph. For each dataset we select $\eta\% = 25\%, 50\%, 75\%$. Over all baselines following the cross-node federated learning constraint, GRU (local) and GRU + FMTL requires training new models on unseen nodes and only GRU + FedAvg is applicable to the inductive learning setting.

Discussion. Table 9 shows the performance of inductive learning of CNFGNN and GRU + FedAvg baseline on both datasets. We observe that under most settings, CNFGNN outperforms the GRU + FedAvg baseline (except on the METR-LA dataset with 25% nodes observed in training, where both models perform similarly), showing that CNFGNN has the stronger ability of generalization.

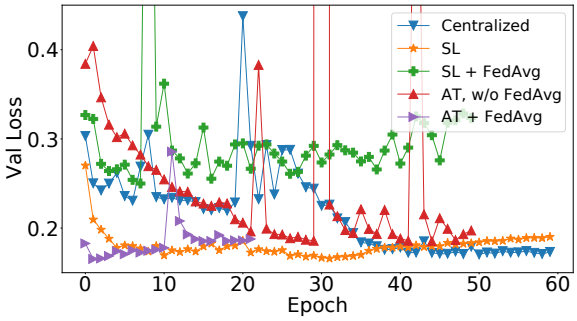
We have further added results using 90% and 5% data on both datasets and we show the table of inductive learning results as Table 9. We observe that: (1) With the portion of visible nodes in the training stage increasing, the prediction error of CNFGNN decreases drastically. However, the increase of the portion of visible nodes has negligible contribution to the performance of GRU + FedAvg after the portion surpasses 25%. Since increasing the ratio of seen nodes in training introduces more complex relationships among nodes to the training data, the difference of performance illustrates that CNFGNN has a stronger capability of capturing complex spatial relationships. (2) When the ratio of visible nodes in training is extremely low (5%), there is not enough spatial relationship information in the training data to train the GN module in CNFGNN, and the performance of CNFGNN may not be ideal. We visualize the subgraphs visible in training under different ratios in Figure 2. However, as long as the training data covers a moderate portion of the spatial information of the whole graph, CNFGNN can still leverage the learned spatial connections among nodes effectively and outperforms GRU+FedAvg. We empirically show that the necessary ratio can vary for different datasets (25% for PEMS-BAY and 50% for METR-LA).

4.3 Ablation Study: Effect of Alternating Training and FedAvg on Node-Level and Spatial Models

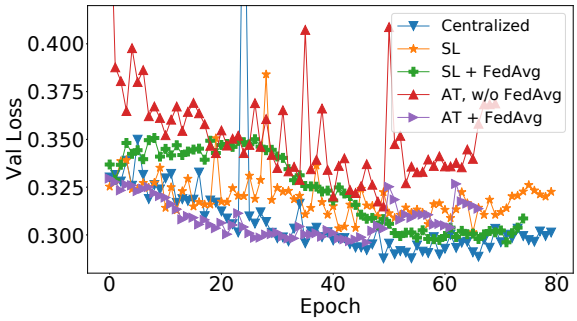
Baselines. We compare the effect of different training strategies of CNFGNN: (1) **Centralized**: CNFGNN trained with centralized data where all nodes share one single encoder-decoder. (2) **Split Learning (SL)**: CNFGNN trained with split learning [25], where models on nodes and the model on the server are jointly trained

Table 9: Inductive learning performance measured with rooted mean squared error (RMSE).

Method	PEMS-BAY					METR-LA				
	5%	25%	50%	75%	90%	5%	25%	50%	75%	90%
GRU (63K) + FedAvg	**5.087**	4.863	4.847	4.859	4.866	**12.128**	**11.993**	12.104	12.014	12.016
CNFGNN (64K + 1M)	5.869	**4.541**	**4.598**	**4.197**	**3.942**	13.931	12.013	**11.815**	**11.676**	**11.629**



(a) PEMS-BAY



(b) METR-LA

Figure 3: Validation loss during the training stage of different training strategies.

Table 10: Comparison of test error (RMSE) and the communication cost during training of different training strategies of CNFGNN.

Method	PEMS-BAY		METR-LA	
	RMSE	Train Comm Cost (GB)	RMSE	Train Comm Cost (GB)
Centralized	**3.816**	-	**11.471**	-
SL	3.914	350.366	12.186	307.627
SL + FedAvg	4.383	80.200	11.631	343.031
AT, w/o FedAvg	4.003	5221.576	11.912	2434.985
AT + FedAvg	**3.822**	237.654	**11.487**	222.246

by exchanging hidden vectors and gradients. (3) **Split Learning + FedAvg (SL + FedAvg)**: A variant of SL that synchronizes the weights of encoder-decoder modules periodically with FedAvg. (4)

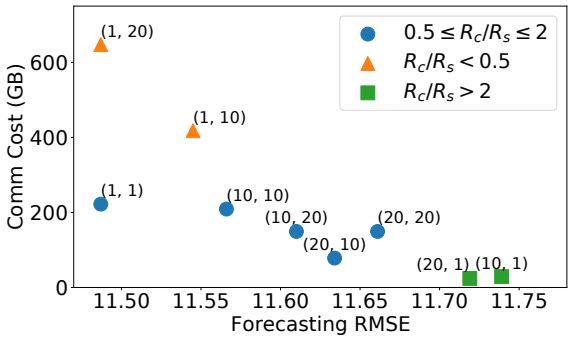


Figure 4: Effect of client rounds and server rounds (R_c, R_s) on forecasting performance and communication cost.

Alternating training without Federated Averaging of models on nodes (**AT, w/o FedAvg**). (5) Alternating training with Federated Averaging on nodes described in Section 3.2.3 (**AT + FedAvg**).

Discussion. Figure 3 shows the validation loss during training of different training strategies on PEMS-BAY and METR-LA datasets, and Table 10 shows their prediction performance and the communication cost in training. We notice that (1) SL suffers from suboptimal prediction performance and high communication costs on both datasets; SL + FedAvg does not have consistent results on both datasets and its performance is always inferior to AT + FedAvg. AT + FedAvg consistently outperforms other baselines on both datasets, including its variant without FedAvg. (2) AT + FedAvg has the lowest communication cost on METR-LA and the 2nd lowest communication cost on PEMS-BAY, on which the baseline with the lowest communication cost (SL + FedAvg) has a much higher prediction error (4.383 vs 3.822). Both illustrate that our proposed training strategy, SL + FedAvg, achieves the best prediction performance as well as low communication cost compared to other baseline strategies.

4.4 Ablation Study: Effect of Client Rounds and Server Rounds

Set-up. We further investigate the effect of different compositions of the number of client rounds (R_s) in Algorithm 2 and the number of server rounds (R_c) in Algorithm 1. To this end, we vary both R_c and R_s over [1,10,20].

Discussion. Figure 4 shows the forecasting performance (measured with RMSE) and the total communication cost in the training of CNFGNN under all compositions of (R_c, R_s) on the METR-LA dataset. We observe that: (1) Models with lower R_c/R_s ratios

($R_c/R_s < 0.5$) tend to have lower forecasting errors while models with higher R_c/R_s ratios ($R_c/R_s > 2$) have lower communication cost in training. This is because the lower ratio of R_c/R_s encourages more frequent exchange of node information at the expense of higher communication cost, while the higher ratio of R_c/R_s acts in the opposite way. (2) Models with similar R_c/R_s ratios have similar communication costs, while those with lower R_c values perform better, corroborating our observation in (1) that frequent node information exchange improves the forecasting performance.

5 CONCLUSION

We propose Cross-Node Federated Graph Neural Network (CNFGNN), which bridges the gap between modeling complex spatio-temporal data and decentralized data processing by enabling the use of graph neural networks (GNNs) in the federated learning setting. We accomplish this by decoupling the learning of local temporal models and the server-side spatial model using alternating optimization of spatial and temporal modules based on split learning and federated averaging. Our experimental results on traffic flow prediction on two real-world datasets show superior performance as compared to competing techniques. Our future work includes applying existing GNN models with sampling strategies and integrating them into CNFGNN for large-scale graphs, extending CNFGNN to a fully decentralized framework, and incorporating existing privacy-preserving methods for graph learning to CNFGNN, to enhance federated learning of spatio-temporal dynamics.

ACKNOWLEDGMENTS

This work is supported in part by NSF Research Grant IIS-1254206, NSF Research Grant CCF-1837131, and WeWork, granted to co-author Yan Liu in her academic role at the University of Southern California. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agency, or the U.S. Government.

REFERENCES

[1] Alekh Agarwal, Animashree Anandkumar, Prateek Jain, Praneeth Netrapalli, and Rashish Tandon. 2014. Learning sparsely used overcomplete dictionaries. In *Conference on Learning Theory*. 123–137.
[2] Sanjeev Arora, Rong Ge, Tengyu Ma, and Ankur Moitra. 2015. Simple, efficient, and neural algorithms for sparse coding. (2015).
[3] Sanjeev Arora, Rong Ge, and Ankur Moitra. 2014. New algorithms for learning incoherent and overcomplete dictionaries. In *Conference on Learning Theory*. 779–806.
[4] Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael W Mahoney. 2020. Forecasting sequential data using consistent Koopman autoencoders. In *ICML*.
[5] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*. 4502–4510.
[6] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).
[7] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.
[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.
[9] Chaoyang He, Salman Avestimehr, and Murali Annavaram. 2020. Group Knowledge Transfer: Collaborative Training of Large CNNs on the Edge. In *Advances in neural information processing systems*.
[10] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*. 4558–4567.
[11] Prateek Jain and Purushottam Kar. 2017. Non-convex Optimization for Machine Learning. *Foundations and Trends® in Machine Learning* 10, 3-4 (2017), 142–363. https://doi.org/10.1561/2200000058
[12] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).
[13] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*.
[14] Thomas N Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S Zemel. 2018. Neural Relational Inference for Interacting Systems. In *ICML*.
[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.
[16] Max Guangyu Li, Bo Jiang, Hao Zhu, Zhengping Che, and Yan Liu. 2020. Generative Attention Networks for Multi-Agent Behavioral Modeling.. In *AAAI*.
[17] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. In *Proceedings of the 3rd MLSys Conference*.
[18] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations (ICLR '18)*.
[19] Paul Pu Liang, Terrance Liu, Liu Ziyin, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523* (2020).
[20] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. 2020. Disentangling and Unifying Graph Convolutions for Skeleton-Based Action Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 143–152.
[21] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.
[22] Guangxu Mei, Ziyu Guo, Shijun Liu, and Li Pan. 2019. SGNN: A Graph Neural Network Based Federated Learning Approach by Hiding Structure. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2560–2568.
[23] Sina Sajadmanesh and Daniel Gatica-Perez. 2020. When Differential Privacy Meets Graph Neural Networks. *arXiv preprint arXiv:2006.05535* (2020).
[24] Sungyong Seo, Chuizheng Meng, and Yan Liu. 2019. Physics-aware Difference Graph Networks for Sparsely-Observed Dynamics. In *International Conference on Learning Representations*.
[25] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. 2019. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145* (2019).
[26] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*. 4424–4434.
[27] Toyotaro Suzumura, Yi Zhou, Natahalie Barcardo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Daniel Klyashtorny, Heiko Ludwig, et al. 2019. Towards Federated Graph Learning for Collaborative Financial Crimes Detection. *arXiv preprint arXiv:1909.12946* (2019).
[28] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2019. What Can Neural Networks Reason About?. In *International Conference on Learning Representations (ICLR)*.
[29] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*.
[30] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.
[31] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning*.
[32] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*.
[33] Jun Zhou, Chaochao Chen, Longfei Zheng, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2020. Privacy-Preserving Graph Neural Network for Node Classification. *arXiv preprint arXiv:2005.11903* (2020).

A APPENDIX

A.1 The Histograms of Data on Different Nodes

We show the histograms of traffic speed on different nodes of PEMS-BAY and METR-LA in Figure A1 and Figure A2. For each dataset, we only show the first 100 nodes ranked by their IDs for simplicity. The histograms show that the data distribution varies with nodes, thus data on different nodes are not independent and identically distributed.

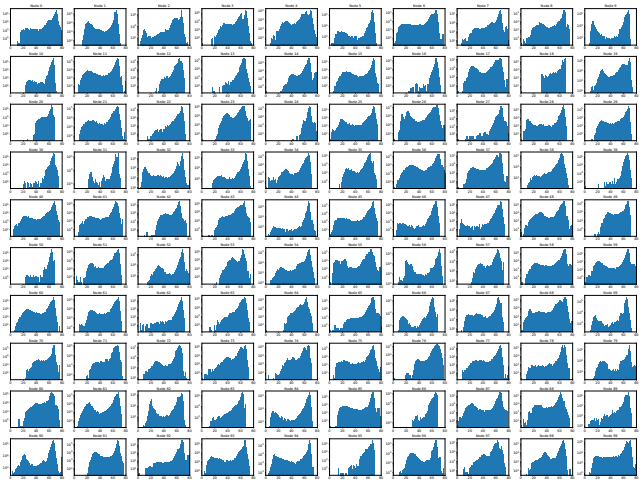


Figure A1: The histograms of PEMS-BAY data on the first 100 nodes ranked by ID.

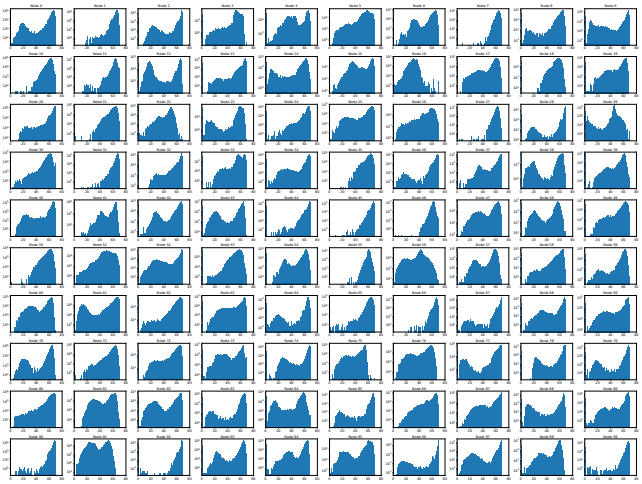


Figure A2: The histograms of METR-LA data on the first 100 nodes ranked by ID.

A.2 Table of Notations

Table A1 summarizes notations used in this paper and their definitions.

Table A1: Table of notations.

Notation	Definition
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Graph $\mathcal{G}$ defined with the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$.
$\mathbf{X}$	Tensor of node features. $\mathbf{X} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\cdots}$.
$\boldsymbol{x}_i$	Features of the i-th node.
$\mathbf{Y}$	Tensor of node labels for the task. $\mathbf{Y} \in \mathbb{R}^{\lvert\mathcal{V}\rvert\times\cdots}$.
$\boldsymbol{y}_i$	Labels of the i-th node.
$\hat{\boldsymbol{y}}_i$	Model prediction output for the i-th node.
$R_g/R_c/R_s$	Maximum number of global/server/client training rounds.
$\boldsymbol{\theta}_{GN}^{(r_g)}$	Weights of the server-side Graph Network in the r_g-th global training round.
$\bar{\boldsymbol{\theta}}_c^{(r_g)}$	Aggregated weights of client models in the r_g-th global training round.
$\boldsymbol{h}_{c,i}$	Local embedding of the input sequence of the i-th node.
$\boldsymbol{h}_{G,c,i}$	Embedding of the i-th node propagated with the server-side GN.
ℓ_i	Loss calculated on the i-th node.
η_s	Learning rate for training $\boldsymbol{\theta}_{GN}^{(r_g)}$.
η_c	Learning rate for training $\bar{\boldsymbol{\theta}}_c^{(r_g)}$.