CROSS-NODE FEDERATED GRAPH NEURAL NETWORK FOR SPATIO-TEMPORAL DATA MODELING

Anonymous authors
Paper under double-blind review

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 decentralized 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.

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 (Li et al., 2018; Yu et al., 2018), forecasting (Seo et al., 2019; Azencot et al., 2020), and user activity detection (Yan et al., 2018; Liu et al., 2020). While existing works on spatio-temporal dynamics modeling (Battaglia et al., 2016; Kipf et al., 2018; Battaglia et al., 2018) 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) (Kairouz et al., 2019) provides a solution for training a model with decentralized data on multiple devices, these works either do not consider the inherent spatio-temporal dependencies (McMahan et al., 2017; Li et al., 2020b; Karimireddy et al., 2020) or only model it implicitly by imposing the graph structure in the regularization on model weights (Smith et al., 2017), the latter of which suffers from the limitation of regularization based methods due to the assumption that graphs only encode similarity of nodes (Kipf & Welling, 2017), 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 (Kairouz et al., 2019), 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 (Singh et al., 2019) 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 (Liang et al., 2020; He et al., 2020) which incurs only half the communication overhead as compared to a comparable Split Learning architecture. Here, we also use Federated Averaging (FedAvg) (McMahan et al., 2017) 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 Work

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 (Hamilton et al., 2017), node classification (Kipf & Welling, 2017), spatio-temporal data modeling (Yan et al., 2018; Li et al., 2018; Yu et al., 2018) and multi-agent trajectory prediction (Battaglia et al., 2016; Kipf et al., 2018; Li et al., 2020a). While GNNs enjoy the benefit from strong inductive bias (Battaglia et al., 2018; Xu et al., 2019), 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 (Kairouz et al., 2019). 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 (McMahan et al., 2017; Li et al., 2020b; Karimireddy et al., 2020). Smith et al. (2017) 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.

Privacy-Preserving Graph Learning. Suzumura et al. (2019) and Mei et al. (2019) 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). Zhou et al. (2020) 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. Sajadmanesh & Gatica-Perez (2020) 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 (Kairouz et al., 2019).

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 $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}| \times \cdots}$, 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.

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) (Battaglia et al., 2018) 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 (Cho et al., 2014) 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 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.

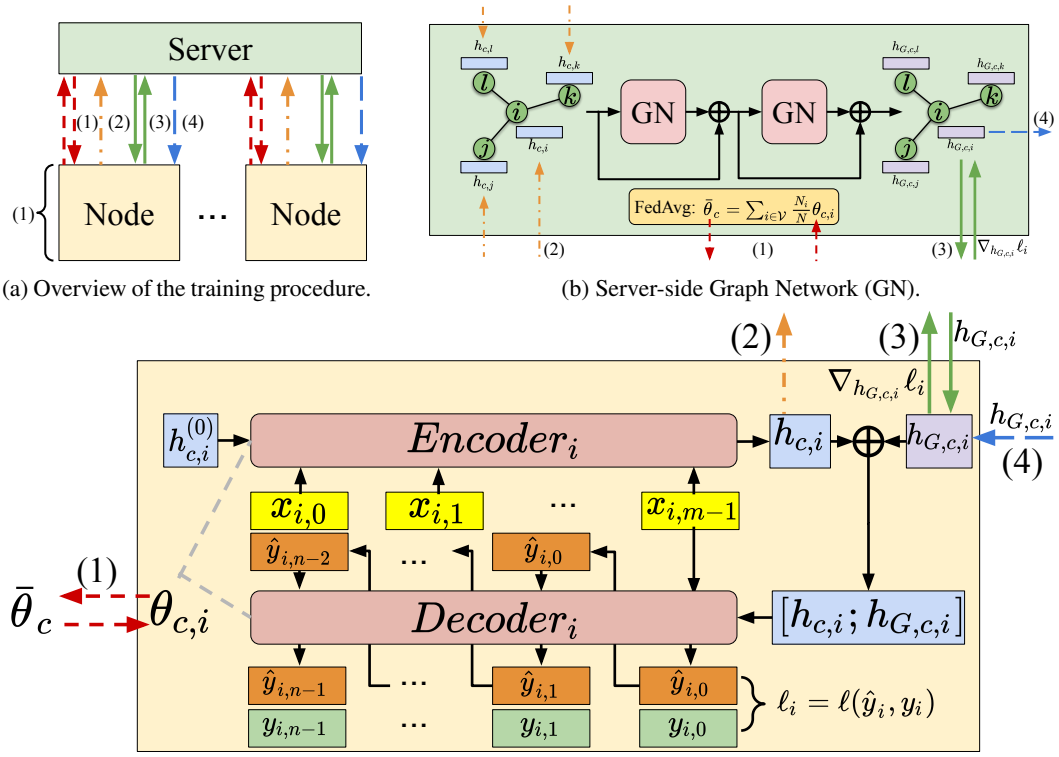
$$\boldsymbol{h}_{c,i} = Encoder_i(\boldsymbol{x}_i, \boldsymbol{h}_{c,i}^{(0)}), \tag{1}$$

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.



(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.

3.2.2 Modeling of Spatial Dynamics

To capture the complex spatial dynamics, we adopt Graph Networks (GNs) proposed in (Battaglia et al., 2018) 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:

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

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. In our case, we set $\mathbf{v}_i = \boldsymbol{h}_{c,i}$ and $\mathbf{e}_k, \mathbf{u}$ as 0-dim vectors. As shown in Figure 1b, we choose a 2-layer GN with residual connections for all experiments. 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 (Singh et al., 2019) 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

Algorithm 1 Training algorithm of CNFGNN on the server side.

Server executes:

Initialize server-side GN weights $\boldsymbol{\theta}_{GN}^{(0)}$, client model weights $\bar{\boldsymbol{\theta}}_c^{(0)} = \{\bar{\boldsymbol{\theta}}_c^{(0),enc}, \bar{\boldsymbol{\theta}}_c^{(0),dec}\}$.
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).
  end for
  $\bar{\boldsymbol{\theta}}_c \leftarrow \sum_{i \in \mathcal{V}} \frac{N_i}{N} \boldsymbol{\theta}_{c,i}$.
  for each client $i \in \mathcal{V}$ in parallel **do**
    Initialize client model: $\boldsymbol{\theta}_{c,i}^{(0)} = \bar{\boldsymbol{\theta}}_c$.
  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,$\boldsymbol{h}_{G,c,i}$).
      $\nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell \leftarrow \nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell + \boldsymbol{h}_{G,c,i}.\text{backward}(\nabla_{h_{G,c,i}} \ell_i)$.
    end for
    $\boldsymbol{\theta}_{GN}^{(r_g,r_s)} \leftarrow \boldsymbol{\theta}_{GN}^{(r_g,r_s-1)} - \eta_s \nabla_{\boldsymbol{\theta}_{GN}^{(r_g,r_s-1)}} \ell$.
  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)**:**

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.

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.

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

To more effectively extract temporal features from each node, we also train the encoder-decoder models on nodes with the FedAvg algorithm proposed in (McMahan et al., 2017). 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. We reuse the following two real-world large-scale datasets in (Li et al., 2018) 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.

4.1 Spatio-Temporal Data Modeling: Traffic Flow Forecasting

Baselines We compare CNFGNN with the following baselines. (1) **GRU (centralized)**: a Gated Recurrent Unit (GRU) model trained with centralized sensor data; (2) **GRU (local)**: for each node we train a GRU model with only the local data on it; (3) **GRU + FedAvg**: a GRU model trained with the Federated Averaging algorithm (McMahan et al., 2017); (4) **GRU + FMTL**: for each node we train a GRU model using the federated multi-task learning (FMTL) with cluster regularization (Smith et al., 2017) given by the adjacency matrix. 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 nodes has 64K parameters and the GN model has 1M parameters.

Table 2: 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 (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**

Discussion Table 2 shows the comparison of forecasting performance and Table 3 shows the comparison of computation cost on device and communication cost of CNFGNN and baselines. We

Table 3: 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

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.

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

Table 4: Inductive learning performance measured with rooted mean squared error (RMSE).

Method	PEMS-BAY			METR-LA		
	25%	50%	75%	25%	50%	75%
GRU (63K) + FedAvg	4.863	4.847	4.859	**11.993**	12.104	12.014
CNFGNN (64K + 1M)	**4.541**	**4.598**	**4.197**	12.013	**11.815**	**11.676**

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 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 4 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.

4.3 Ablation Study: Effect of Alternating Training of 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**: CNFGNN trained with split learning (Singh et al., 2019), where models on nodes and the model on the server are jointly trained by exchanging hidden vectors and gradients. (3) Alternating training without Federated Averaging of models on nodes (**AT, w/o FedAvg**). (4) Alternating training with Federated Averaging on nodes described in Section 3.2.3 (**AT + FedAvg**).

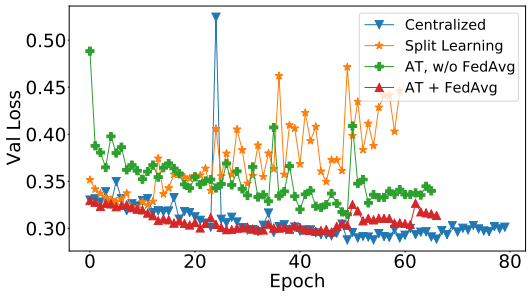


Figure 2: Validation loss during the training stage of different training strategies on METR-LA.

Discussion Figure 2 shows the effect of training strategies. We notice that (1) Split Learning quickly overfits the training data while AT w/o FedAvg and AT + FedAvg do not, which demonstrates that the alternating training helps the model avoid overfitting. (2) AT w/o FedAvg suffers from slow convergence and converges at suboptimal validation loss, while AT + FedAvg achieves the lowest validation error and performs closely to the model trained on centralized data. This result indicates that the federated learning of models on nodes is critical for CNFGNN to achieve the best performance.

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].

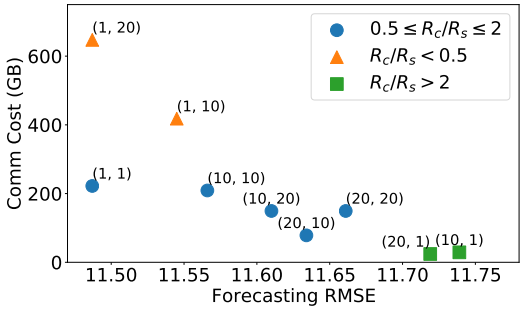


Figure 3: Effect of client rounds and server rounds (R_c, R_s) on forecasting performance and communication cost.

Discussion Figure 3 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 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.

REFERENCES

Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael W Mahoney. Forecasting sequential data using consistent koopman autoencoders. In *ICML*, 2020.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

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. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 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)*, pp. 1724–1734, 2014.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Chaoyang He, Salman Avestimehr, and Murali Annavaram. Group knowledge transfer: Collaborative training of large cnns on the edge. *arXiv preprint arXiv:2007.14513*, 2020.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Thomas N Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

Max Guangyu Li, Bo Jiang, Hao Zhu, Zhengping Che, and Yan Liu. Generative attention networks for multi-agent behavioral modeling. In *AAAI*, 2020a.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of the 3rd MLSys Conference*, 2020b.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR '18)*, 2018.

Paul Pu Liang, Terrance Liu, Liu Ziyin, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. Disentangling and unifying graph convolutions for skeleton-based action recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 143–152, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

Guangxu Mei, Ziyu Guo, Shijun Liu, and Li Pan. Sgnn: A graph neural network based federated learning approach by hiding structure. In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 2560–2568. IEEE, 2019.

Sina Sajadmanesh and Daniel Gatica-Perez. When differential privacy meets graph neural networks. *arXiv preprint arXiv:2006.05535*, 2020.

Sungyong Seo, Chuizheng Meng, and Yan Liu. Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*, 2019.

Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv preprint arXiv:1909.09145*, 2019.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.

Toyotaro Suzumura, Yi Zhou, Natahalie Barcardo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Daniel Klyashtorny, Heiko Ludwig, et al. Towards federated graph learning for collaborative financial crimes detection. *arXiv preprint arXiv:1909.12946*, 2019.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations (ICLR)*, 2019.

Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Jun Zhou, Chaochao Chen, Longfei Zheng, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. Privacy-preserving graph neural network for node classification. *arXiv preprint arXiv:2005.11903*, 2020.

A APPENDIX

A.1 DETAILED EXPERIMENT SETTINGS

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 (local) We train one GRU model for each node with the local data only.

GRU + FedAvg We train a single GRU model with Federated Averaging (McMahan et al., 2017). We select 1 as the number of local epochs.

GRU + FMTL We train one GRU model for each node using the federated multi-task learning (FMTL) with cluster regularization (Smith et al., 2017) given by the adjacency matrix. More specifically, the cluster regularization (without the L2-norm regularization term) takes the following form:

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

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

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

We implement the cluster regularization via sharing model weights between each pair of nodes connected by an edge and select $\lambda_1 = 0.1$.

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.

A.2 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:

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

CNFGNN 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:

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

Table 5 and Table 6 demonstrate all parameters used in calculating the communication cost of GRU + FMTL and CNFGNN.

Table 5: 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 6: Parameters used for calculating the communication cost of CNFGNN.

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