Engineering Applications of Artificial Intelligence 162 (2025) 112801

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Research paper

A federated learning framework for arbitrary spatio-temporal graph neural networks

Heeyong Yoon [a], Kang-Wook Chon [b,*], Min-Soo Kim [c,*]

[a] Department of Electrical Engineering and Computer Science, Daegu Gyeongbuk Institute of Science and Technology (DGIST), 333 Techno jungang-daero, Hyeonpung-eup, Dalseong-gun, Daegu, 42988, Republic of Korea
[b] School of Computer Science and Engineering, Korea University of Technology and Education (KOREATECH), 1600 Chungjeol-ro, Byeongcheon-myeon, Dongnam-gu, Cheonan-si, Chungcheongnam-do, 31253, Republic of Korea
[c] School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, Republic of Korea

ARTICLE INFO

Keywords:
Machine learning framework for arbitrary models
Spatio-temporal graph neural network
Federated learning
Edge computing
Vehicle traffic data
Weather data

ABSTRACT

The proliferation of mobile and Internet of Things (IoT) devices has resulted in a surge of time-series sensor data, posing significant challenges for centralized data collection and processing. This challenge has driven the adoption of edge computing, which offloads data processing to mid-level servers located at the edge of the Internet, thereby reducing computation and bandwidth demands. Federated learning has emerged as a promising method for training models in edge-computing environments. Recently, spatio-temporal graph neural networks (STGNNs) have shown impressive performance in time-series prediction, yet their application in edge computing is limited by the complexity of adapting them to distributed environments. To address this gap, we propose FedSTGNN (Federated Spatio-Temporal Graph Neural Network), a universal framework that converts existing centralized STGNN models into a federated learning version. We formulate the common STGNN training process using matrix operations, employ graph-based imputation methods to handle missing sensor values at edge servers, and facilitate the transition from centralized to federated STGNNs. Our comprehensive evaluations demonstrate that FedSTGNN not only preserves the prediction accuracy of the original STGNN models but is also significantly more network-efficient than the competing model. Furthermore, the framework proves its robustness in challenging real-world scenarios, including sparse graphs, long-term forecasting, and dynamic server participation. Our work presents a practical, robust, and universal solution for deploying STGNNs into various edge computing applications.

1. Introduction

Owing to the widespread use of mobile and Internet of Things (IoT), the increasing number of sensors has made it challenging to collect and process all sensor data on a central server. This classical design requires substantial computational power from the central server and requires high bandwidth for the global network. To address these limitations, a new paradigm, *edge computing* (Chang et al., 2014; Satyanarayanan, 2017), has been proposed. Edge computing reduces computational and network overhead by deploying middle-level servers, called *edge servers*, at the edge of the Internet. These servers offload data processing tasks and summarize the data before sending it to the central server. Edge computing has been applied across various domains, including manufacturing (Chen et al., 2018), autonomous driving (Liu et al., 2019), smart grids (Mehmood et al., 2021), and healthcare (Ray et al., 2019), delivering benefits such as cost reduction, reduced latency, and enhanced data security.

Fig. 1 shows the difference between the conventional central server model and the edge-computing approach. In the central server model, the central server M is located remotely and receives raw data from all five sensors $\{0, 1, 2, 3, 4\}$ through a wide-area network, processing the information independently. In contrast, the edge-computing model introduces edge servers M_0 and M_1 near the sensors through a local area network, with the edge servers connected to the central server M through a wide-area network. Edge server M_0 handles data from sensors $\{0, 1, 2\}$, while edge server M_1 processes data from sensors $\{3, 4\}$. Each edge server processes its raw data locally and transmits only summarized information to the central server M. This architecture effectively offloads data processing tasks from the central server to edge servers, reducing computational loads and wide-area network usage.

The proliferation of IoT has also led to the emergence of a new machine learning method called *federated learning* (FL) (McMahan et al.,

* Corresponding authors.
E-mail addresses: sunrise2575@dgist.ac.kr (H. Yoon), kw.chon@koreatech.ac.kr (K.-W. Chon), minsoo.k@kaist.ac.kr (M.-S. Kim).

https://doi.org/10.1016/j.engappai.2025.112801
Received 15 March 2025; Received in revised form 22 August 2025; Accepted 13 October 2025
Available online 17 October 2025

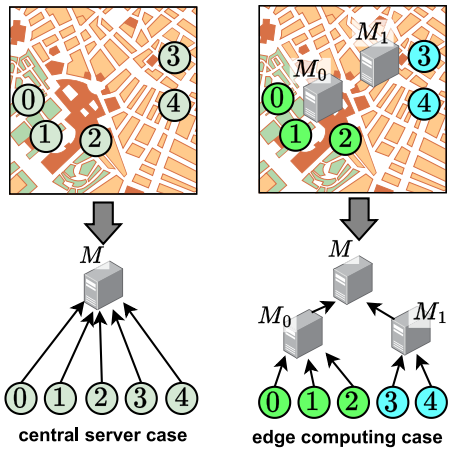

Fig. 1. An example of comparing central server case with edge-computing case.

2017). FL is a data-parallel distributed machine learning method that enables machine learning in a multi-server environment. FL trains a model through the following procedure: the edge server trains its local model using its input data, uploads the local model parameters to the central server to aggregate them into a global model, and downloads the global model for application to each local model. FL and edge computing are similar in that the nodes at the edges of the networks summarize information locally, and only the summarized data are exchanged between the edge and central nodes. The characteristics of FL mentioned above can address the challenging issues of training models using only a central server (i.e., the requirements for high-performance central servers, high-bandwidth networks, and enhanced cybersecurity).

Several sensors have been deployed across various industrial sectors in IoT environments. As these sensors can measure a wide range of information, they drive the demand for predicting future events, making decisions, and responding to crises. Various methods have been proposed for predicting sensor values, such as ARIMA (Douglas, 1994), VAR (Williams and Hoel, 2003), LSTM (Hochreiter and Schmidhuber, 1997), and Seq2seq (Sutskever et al., 2014). Among these, the best-performing model is *Spatio-Temporal Graph Neural Networks* (STGNN) (Li et al., 2018; Yu et al., 2018; Guo et al., 2019; Wu et al., 2019). STGNN utilizes both time-series and spatial information to achieve superior performance. It has been successfully applied in various fields, such as road networks (Li et al., 2018; Chaolong et al., 2018; Yu et al., 2018; Zhao et al., 2019; Zhang et al., 2020; Guo et al., 2019; Lai and Chen, 2024; Xia et al., 2024), object detection in videos (Chaolong et al., 2018; Wang et al., 2021), epidemic modeling (Cao et al., 2020), weather prediction (Davidson and Moodley, 2022), motion sensing (Wang et al., 2024), and water quality control (Liang et al., 2018). Most STGNN models adopt a similar structure to process temporal and spatial information separately. First, a distance graph is formed from sensor locations and processed using a GNN-based module. Second, temporal information is managed using well-established sequence-processing modules such as recurrent neural networks (RNNs) (Rumelhart et al., 1985) or transformers (Vaswani, 2017). Finally, the spatial and temporal information from these modules is combined to generate predictions.

Traditionally, STGNNs have been developed for centralized training on a single server (Wen et al., 2023). However, as the number of sensors and timesteps increases, centralized models encounter significant challenges, including escalating network traffic and computational overload. These challenges are particularly pronounced for STGNNs due to the continuous growth of time-series data over time (Jiang and Luo, 2022; Jin et al., 2023). To address these scalability issues, leveraging edge-computing environments is an effective solution. Recent studies have introduced FL-STGNN approaches to address these challenges (Meng et al., 2021; Zhang et al., 2023; Liu et al., 2024b; Mao et al., 2023; Liu et al., 2025, 2024a; He et al., 2023; Tian et al., 2023; Yuan et al., 2022; Yang et al., 2024). These approaches are often designed from scratch. This approach usually splits a manually designed neural networks into central and edge servers, exchanges model parameters and additional information such as spatial and temporal embeddings. A critical limitation of this approach is that it does not utilize lessons of existing centralized STGNN models. Even though centralized STGNN models have conducted extensive studies to establish their performance, time-space complexity, and hyperparameter settings (Jiang et al., 2021; Luo et al., 2023), current FL-STGNN approaches need to repeat the time-intensive validation processes from scratch because the overall design is not related to centralized STGNNs.

To the best of our knowledge, there is no straightforward method for converting a centralized STGNN model into an FL model that allows STGNN to be quickly applied to edge computing. In this context, we propose a framework that enables centralized STGNN model to be applied in a FL setting, named FedSTGNN (**Fed**erated **S**patio-**T**emporal **G**raph **N**eural **N**etwork). Our approach directly addresses the limitations of existing FL-STGNN models. Unlike these monolithically designed from scratch, our framework acts as a universal converter. This approach can preserve the performance and stability of already validated centralized STGNN models. We formulate the common training process of STGNNs using matrix operations to facilitate this transition and adapt a graph-based imputation method that aggregates values from neighboring nodes to handle missing sensor data inherent to the distributed environment. We examined that FedSTGNN preserves prediction accuracy comparable to centralized models and significantly outperforms a competing FL-STGNN model in training time and network usage. Furthermore, our framework proves its practical superiority over alternative approaches like LLM-based models. We also show its robustness across various challenging real-world scenarios, including sparse graphs, long-term forecasting, and dynamic environments with high server failure rates.

The main contributions of this study are as follows:

- **FL-converting framework for centralized STGNN models**: We propose the first framework to convert a centralized STGNN model into an FL version without architecture modification. The study identifies that various STGNN models share a common training process, which can be expressed using matrix-based formulas. The centralized training process is successfully integrated into FL through simple modifications of these formulas.
- **Preserving performance by the graph-based imputation**: We propose a graph-based imputation method to prevent performance degradation caused by missing values in distributed processing. We categorize imputation methods based on the number of hops of aggregation in a graph, some of which achieve results comparable to those of centralized models. Notably, the graph-based imputation method derives values directly from the input dataset rather than relying on additional neural network models, enhancing the reliability of model training in edge-computing environments.
- **Comprehensive validation of the framework in various aspects**: We tested our framework on several real-world datasets from different fields and compared it with single-node models and an FL-STGNN model. The framework can preserve the performance of a centralized STGNN model and reduce network demands compared to the FL-STGNN model. We also demonstrate the framework's robustness across diverse and challenging scenarios, such as using sparse graphs, long-term forecasting, or server failures. Through extensive evaluation, we demonstrate our framework's superiority over alternative approaches, including LLM-based models, proving its real-world viability and practical advantages.

The remainder of this paper is organized as follows: We explain the basic concepts of FL and STGNN in Section 3. We describe our framework, including the transition to FL and graph-based imputation, in Section 4. We evaluate the prediction performance of FedSTGNN from several perspectives in Section 5. Finally, Section 7 concludes the paper.

2. Related work

FL-specific STGNN model. Cross-Node Federated Graph Neural Network (CNFGNN) (Meng et al., 2021) is a prominent STGNN model designed explicitly for FL. It processes the spatial and temporal components, which are the key parts of STGNN, separately between the central and edge servers. The detailed training process is as follows: (1) First, an encoder–decoder using Gated Recurrent Unit (GRU) (Chung et al., 2014) is deployed on the edge server and designated as the local model. (2) Each edge server's local model processes one sensor's time series through the encoder to generate a temporal encoding vector. (3) The local model parameters and temporal encoding vector are uploaded to the central server. (4) The central server synchronizes the local model parameters through FedAvg operations and distributes them back to edge servers. (5) Meanwhile, the central server performs graph neural network computations on the basis of the temporal encoding vectors to derive graph embeddings. (6) These graph embeddings are then transferred to each edge server and inserted into the local model's decoder input to predict the time series.

Although CNFGNN deserves recognition for being among the first to apply FL to STGNN, it unfortunately has several limitations: The communication frequency and volume between the central and edge servers are substantial. As mentioned in the original paper, training requires tens to thousands of gigabytes of data transfer over wide area networks. It indicates that CNFGNN struggles to move beyond a proof-of-concept to practical implementation in real environments; the architecture is not well suited for edge computing. Because CNFGNN is designed for one edge server to process information from only one sensor, the number of local models increases linearly as the number of sensors increases. It demands significant computational resources. Additionally, the increase in edge servers naturally escalates the aforementioned communication costs.

The following studies have focused on prediction performance optimization or expanding application domains. FedRel (Zhang et al., 2023) uses an Inter–Intra Graph that simultaneously captures intra-partition and inter-partition relationships in partitioned sensor graphs, and adopts a weighted averaging scheme based on the contribution of each edge server. FedOSTC (Liu et al., 2024b) extracts temporal features using a GRU encoder at the edge server, while the central server leveraged a Graph Attention Network to model time-varying spatial adjacency and periodicity. In the healthcare domain, FedGST (Mao et al., 2023) was proposed, where the dynamic connectivity of brain signals is computed locally at each hospital and then integrated through federated learning for disease prediction.

Another group of researchers targeted communication efficiency or privacy. FUELS (Liu et al., 2025) proposed a contrastive learning approach that performs dual semantic alignment to handle heterogeneous spatio-temporal patterns across regions, defines prototype vectors for each edge server to enable efficient spatial alignment, and reduces communication overhead compared with prior methods. REFOL (Liu et al., 2024a) automatically selects participating edge servers according to data conditions, adjusts learning rates for online settings, and applies graph-based aggregation to reduce unnecessary communication and computation while maintaining performance under distribution shifts. Additionally, to address communication failures or unstable edge servers, Fed-mSSA (He et al., 2023) applies a consensus-based optimization method to extract low-rank common patterns across edge servers and achieves noise-robust predictions. M3FGM (Tian et al., 2023) incorporates node masking and multi-granularity message passing, making local inference possible even when some edge servers were offline. On the other hand, there are privacy-preserving methods. FedSTN (Yuan et al., 2022) combines vertically partitioned heterogeneous features with homomorphic encryption to perform traffic flow forecasting without directly exposing raw data, and FedGTP (Yang et al., 2024) designs an adaptive mechanism that complies with privacy regulations while recovering hidden spatial dependencies across edge servers.

3. Background

Our study focuses on FL and STGNNs. This section provides an overview of the FL process in Section 3.1 and the spatio-temporal graph neural network in Section 3.2.

3.1. Federated learning

Federated learning (FL) (McMahan et al., 2017) is a data-parallel approach to distributed machine learning that enhances communication efficiency and data privacy. Various follow-up studies have been conducted to improve model performance in FL (Li et al., 2020; Thapa et al., 2022; Mohri et al., 2019; Diao et al., 2022). Despite their differences, these studies share a common principle: in an edge-computing architecture, only the edge server model's parameters are transmitted to the central server rather than raw data being sent directly. FL is applicable in several scenarios. One notable use case involves reducing network traffic and central storage costs by avoiding the direct transmission of vast amounts of data generated by thousands of mobile edge servers (e.g., smartphones) (Hard et al., 2018). Additionally, FL enables model training in privacy-sensitive environments, such as hospitals (Vaid et al., 2020), the military (Cirincione and Verma, 2019), and banking institutions (Long et al., 2020).

FL is anticipated to reduce the communication overhead of a wide-area network when training STGNNs. We measured the dataset sizes and model parameters during training using our framework, FedSTGNN, as shown in Table 1. The datasets and models referenced are described in Table 2 and Section 5.1, respectively. In most cases, the model parameters accounted for approximately mean 7% of the raw data size. For clarity, we present only megabyte-scale datasets in Table 1. Notably, the difference between raw data and model parameters will become more clear for extended periods.

FL operates through iterative *round*s. While there are variations of FL, the basic process remains consistent. Fig. 2 shows the steps in a typical round: ① First, each selected edge server M_i uses its local data to train a local model θ_i. ② After completing the training, edge servers send their local models θ_i to the central server M. ③ The central server then aggregates all local models θ_i into a global model θ using appropriate functions, such as averaging. ④ Finally, the global model θ is broadcast to the edge servers, replacing their local models θ_i. This process is repeated over multiple rounds, with the global and local models iteratively optimized to converge to a desired performance level.

3.2. Spatio-temporal graph neural networks (STGNN)

STGNN is a group of neural network models designed for various tasks (Cao et al., 2020; Chaolong et al., 2018; Yu et al., 2018; Liang et al., 2018; Li et al., 2018; Guo et al., 2019, 2021; Yu et al., 2017; Wu et al., 2019; Deng et al., 2020; Davidson and Moodley, 2022; Zhao et al., 2019; Jin et al., 2023; Sahili and Awad, 2023; Kreuzberger et al., 2023) in time series, empowered by graph neural networks (GNN) (Scarselli et al., 2008). In this subsection, we explain the typical process of training a centralized (i.e., single-node) version of STGNN, which extends to federated STGNN learning.

Before describing STGNN, we define some mathematical notation necessary for the following explanation: First, each element in the matrix is represented by Definition 1.

Table 1
Sizes of input datasets and their STGNN model parameters. Each model has a different parameter size because model size is dependent on various training settings, such as input graph size, input and output time length, and batch size. For this table, we preserve the model's original settings. Each percentage value means the model parameter's relative size compared to the dataset size.

Dataset		Model parameter size (MB)						Avg. ratio of parameter size to dataset size
Name	Size (MB)	AGCRN	ASTGCN	DGCRN	DSTAGNN	GWNet	STGODE	
METR-LA	28.63	0.36 (1.25%)	0.11 (0.40%)	0.09 (0.33%)	1.24 (4.33%)	0.13 (0.45%)	0.39 (1.35%)	1.35%
PEMS-BAY	67.19	0.36 (0.53%)	0.24 (0.35%)	0.10 (0.15%)	1.80 (2.68%)	0.13 (0.19%)	0.39 (0.58%)	0.75%
PEMS03	37.35	0.36 (0.96%)	0.28 (0.75%)	0.10 (0.27%)	1.99 (5.33%)	0.13 (0.35%)	0.39 (1.05%)	1.45%
PEMS04	60.75	0.36 (0.59%)	0.21 (0.35%)	0.10 (0.16%)	1.71 (2.81%)	0.13 (0.21%)	0.39 (0.64%)	0.79%
PEMS07	99.21	0.36 (0.36%)	1.54 (1.55%)	0.12 (0.12%)	6.63 (6.68%)	0.13 (0.14%)	0.41 (0.41%)	1.54%
PEMS08	35.58	0.36 (1.00%)	0.09 (0.24%)	0.09 (0.26%)	1.10 (3.08%)	0.13 (0.36%)	0.39 (1.09%)	1.00%
PEMSD7	11.75	0.36 (3.04%)	0.13 (1.12%)	0.09 (0.81%)	1.33 (11.30%)	0.13 (1.09%)	0.39 (3.31%)	3.44%
Air-PM25	6.20	0.36 (5.75%)	0.09 (1.40%)	0.09 (1.49%)	1.10 (17.79%)	0.13 (2.06%)	0.39 (6.24%)	5.79%
CSSE-COVID	1.68	0.36 (21.21%)	0.19 (11.47%)	0.10 (5.76%)	1.67 (99.15%)	0.13 (7.66%)	0.44 (26.27%)	28.59%
GDELT-139	7.43	0.36 (4.80%)	0.07 (0.92%)	0.09 (1.23%)	1.05 (14.15%)	0.13 (1.72%)	0.44 (5.90%)	4.78%
Opinet	3.57	0.36 (9.98%)	0.13 (3.73%)	0.09 (2.65%)	1.39 (38.94%)	0.13 (3.59%)	0.44 (12.32%)	11.87%
Uzel2022	0.93	0.36 (38.14%)	0.04 (3.86%)	0.09 (9.44%)	0.85 (91.08%)	0.13 (13.61%)	0.46 (49.76%)	34.32%
NOAA-958	21.22	0.36 (1.70%)	1.81 (8.52%)	0.12 (0.58%)	7.62 (35.88%)	0.14 (0.64%)	0.46 (2.17%)	8.25%

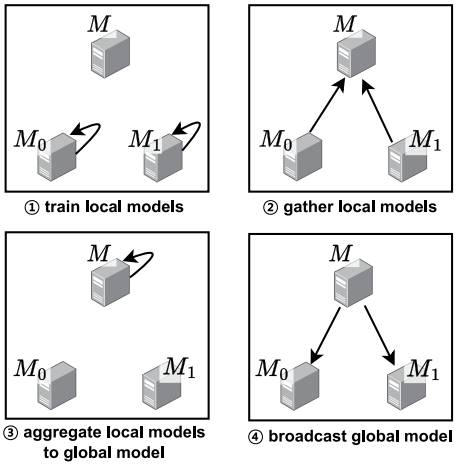

Fig. 2. Model parameter exchanging steps in federated learning.

Definition 1 (*Matrix Element Notation*)**.** Element $A_{i,j} \in \mathbb{R}$ is located in ith row and jth column of matrix $A \in \mathbb{R}^{I \times J}$, where $0 \le i < I$ and $0 \le j < J$.

The matrix element notation in Definition 3 can be extended to describe a submatrix using the colon notation in Definition 2.

Definition 2 (*Colon Notation*)**.** A combination of colons and integers symbolizes a continuous range of integers. Note that we temporarily placed boxes around the representation to separate symbols from normal punctuation. For example, given two integers n and m satisfying $0 \le n < m < l$, $\boxed{n:m}$ is a set of continuous integers k such that $n \le k < m$, i.e., $\boxed{n:m} \equiv \{n, n+1, ..., m-1\}$. If any integers aside from the colon is omitted, the missing integer is considered to be the minimum or maximum possible integer. In other words, $\boxed{n:} \equiv \{n, n+1, ..., l-1\}$, $\boxed{:m} \equiv \{0, ..., m-2, m-1\}$ and $\boxed{:} \equiv \{0, 1, ..., l-1\}$.

Definition 3 (*Submatrix Notation*)**.** The submatrix notation $A_{a:b,c:d} \in \mathbb{R}^{(b-a)\times(d-c)}$ represents the continuous range from the ath to the bth row and the cth to the dth column of the matrix $A \in \mathbb{R}^{I \times J}$, where $0 \le a < b < I$ and $0 \le c < d < J$. In addition, a submatrix with a non-continuous range of rows and columns can be represented using set symbols. Therefore, $A_{\mathcal{M},\mathcal{N}} \in \mathbb{R}^{|\mathcal{M}|\times|\mathcal{N}|}$ is a submatrix extracting the corresponding rows $m \in \mathcal{M}$ and columns $n \in \mathcal{N}$ from A.

Fig. 3 provides a visual example of the raw data processing workflow before training an STGNN model. The raw data, which includes sensor locations on a map and time-series signals from sensors, is processed through two separate pipelines: spatial and temporal information.

First, a distance graph is created based on the sensor locations. This graph does not need to be fully connected because long distances beyond a selected threshold are eliminated using a thresholded Gaussian kernel, described later. Therefore, only distances between reasonably close sensors are calculated. This distance graph is considered as an adjacency matrix, where unknown distances are assigned infinite values. The adjacency matrix is then processed using the thresholded Gaussian kernel, which maps short distances closer to one and long distances closer to zero. For all valid i and j of the adjacency matrix A, the thresholded Gaussian kernel $\tilde{A} = \mathsf{GaussKern}(A, k)$ is as follows:

$$\tilde{A}_{u,v} := \begin{cases} W_{u,v} & \text{if } W_{u,v} > k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $W_{u,v} = e^{-\left(A_{u,v}/\mathrm{stdev}(A)\right)^2}$. Finally, the processed adjacency matrix $\tilde{A}$ generated by Eq. (1) is converted into a Laplacian matrix. While several variants of the Laplacian matrix exist, the symmetrically normalized Laplacian is the most basic version, defined as $L = I - D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}$ in Fig. 3. Here, I is the identity matrix, and each element of $D^{-\frac{1}{2}}$ is determined by Eq. (2). For a certain $u, v \in \mathcal{V}$, where $\mathcal{V}$ represents the set of vertices and $\deg(i) = \sum_{j\in\mathcal{V}} \tilde{A}_{i,j}$,

$$D^{-\frac{1}{2}}_{u,v} := \begin{cases} \frac{1}{\sqrt{\deg(u)}} & u = v \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Meanwhile, the sensor value matrix $X \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}|}$, where $\mathcal{T}$ is the set of timestamps, is constructed by arranging time-series signals in rows (timestamps) and columns (sensors or vertices). The dataset is then split into training and testing datasets based on a train–test ratio. For example, in Fig. 3, the time-series data from t_0 to t_3 form the training dataset $X_{0:4,:}$, with the timestamp set $\mathcal{T}_{\text{train}} = \{t_0, t_1, t_2, t_3\}$. Similarly, the time-series data from t_4 to t_5 form the testing dataset $X_{4:6,:}$, with the timestamp set $\mathcal{T}_{\text{test}} = \{t_4, t_5\}$. Since most STGNN models require fixed-size input lengths, the datasets are divided into *slices* using a sliding window (Hällman, 2017), also known as a moving window. This method generates overlapping windows of length $t_{\text{in}}+t_{\text{out}}$ over all available timestamps t in X, producing slices $X_{t-t_{\text{in}}:t+t_{\text{out}},:} \in \mathbb{R}^{(t_{\text{in}}+t_{\text{out}})\times|\mathcal{V}|}$. For example in Fig. 3, $t_{\text{in}} = t_{\text{out}} = 1$ and $|\mathcal{V}| = 5$, so submatrices of size $\mathbb{R}^{2\times5}$ are extracted, such as $X_{0:2,:}$, $X_{1:3,:}$, and $X_{2:4,:}$ from the training dataset $X_{0:4,:}$, and $X_{4:6,:}$ from the testing dataset $X_{4:6,:}$. Distinct colors represent the input data (green) and the corresponding correct output data (yellow), illustrating that the model predicts future values (yellow) based on past values (green).

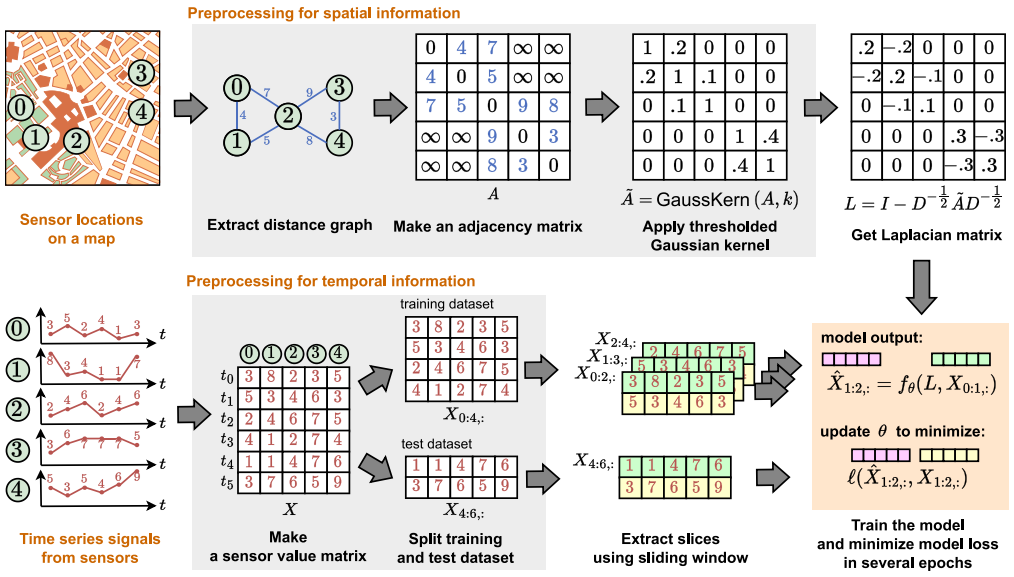

Fig. 3. Typical example of preprocessing pipeline for training STGNN model.

Using the Laplacian matrix and slices, the STGNN model minimizes the sum of losses ℓ of prediction outputs across multiple epochs. The training process consists of three steps: prediction, loss calculation, and optimization. In each epoch, for every slice $X_{t-t_{\text{in}}:t+t_{\text{out}},:}$, the STGNN model f_θ predicts t_{out} future values $\hat{X}_{t:t+t_{\text{out}},:}$ using t_{in} past values $X_{t-t_{\text{in}}:t}$, along with the Laplacian matrix L, as formally described in Eq. (3).

$$\hat{X}_{t:t+t_{\text{out}},:} = f_\theta(L, X_{t-t_{\text{in}}:t,:}). \tag{3}$$

The loss function ℓ calculates the discrepancy between the predicted value $\hat{X}_{t:t+t_{\text{out}},:}$ and the actual value $X_{t:t+t_{\text{out}},:}$. The mean of ℓ across all $\hat{X}_{t:t+t_{\text{out}},:}$ is defined as the epoch loss such that

$$\mathcal{L}_{\text{train}} = \frac{1}{\tau} \sum_{t=t_{\text{in}}}^{|\mathcal{T}_{\text{train}}|-t_{\text{out}}} \ell(X_{t:t+t_{\text{out}},:}, \hat{X}_{t:t+t_{\text{out}},:}). \tag{4}$$

If the slices are extracted from the testing dataset rather than the training dataset, the notation changes accordingly, from $\mathcal{L}_{\text{train}}$ to $\mathcal{L}_{\text{test}}$, and from $\mathcal{T}_{\text{train}}$ to $\mathcal{T}_{\text{test}}$, in Eq. (4). For clarity, the timestamps t for all available slices fall within the range $t_{\text{in}} \le t \le |\mathcal{T}_{\text{train}}| - t_{\text{out}}$, resulting in a total of $\tau = |\mathcal{T}_{\text{train}}| - (t_{\text{in}} + t_{\text{out}}) + 1$ slices $X_{t-t_{\text{in}}:t+t_{\text{out}},:}$, as specified in Eq. (4). The optimizer finally updates θ to minimize $\mathcal{L}$ as much as possible. The training process iterates over multiple epochs until a predefined epoch η is reached.

4. FedSTGNN

We describe our framework in four subsections. An overview of the framework is presented in Section 4.1, followed by an algorithmic perspective in Section 4.2. Detailed explanations of these two aspects are provided in Section 4.3, which covers the transition from the centralized single-node STGNN of Section 3.2 to federated STGNN training. Lastly, Section 4.4 discusses imputation methods including the graph-based approach.

4.1. Framework overview

Fig. 4 shows how FedSTGNN operates, using the same sensors as shown in Fig. 3. In this example, the computing environment comprises two edge servers (M_0 and M_1) and a central server (M). Each ith edge server hosts a local model θ_i, and manages a set of local sensors $\mathcal{V}_i$ within its coverage area.

In Fig. 4, the sensors are distributed among different edge servers: sensors $\{0,1,2\}$ are assigned to M_0, and sensors $\{3,4\}$ are assigned to M_1. Each ith edge server receives the time-series signals of the sensors as a local input value matrix $X_{:,\mathcal{V}_i} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}_i|}$. The sensor values corresponding to the set of invisible sensors $\mathcal{V}\backslash\mathcal{V}_i$ are imputed using an appropriate imputation function $\mathsf{Imput}(X_{:,\mathcal{V}_0})$. This imputation allows each ith edge server to generate the local sensor value $X^{(i)} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}|}$.

The FL process, enclosed in the blue dotted-line box in Fig. 4, is then repeated as follows: ① Each edge server trains its local model f_{θ_i} using slices from the local sensor value $X^{(i)}$ and the Laplacian matrix L as described in Eq. (6), performing η local epochs. ② All local model parameters θ_i are sent to the central server. ③ The central server aggregates the collected parameters into the global model parameter θ using an aggregation function $\mathsf{Agg}(\theta_i, \ldots)$. This function typically computes the mean of all parameters, expressed as $\theta = \frac{1}{n}\sum_i \theta_i$, where n is the number of edge servers. ④ The global model parameter θ is then broadcast to the edge servers, and each edge server overwrites its local model parameter θ_i with θ. Through this FL process, edge servers indirectly share and synchronize information despite having access to limited sensor values within their local coverage. A single round, encompassing steps ① to ④, is repeated until the designated maximum number of rounds ρ is reached. At the end of this process, the global and local models θ and θ_i, respectively are anticipate to converge to the lowest possible prediction loss, closely approximating the performance of a centralized server-only STGNN model.

4.2. Algorithmic explanation

Although FedSTGNN operates as an FL framework where each edge and central server can function simultaneously, it synchronizes at each round, allowing it to be conceptualized as a single-processor operation, as presented in Algorithm 1. The mainline of the framework is represented by the procedure FedSTGNN, while GetLoss in Algorithm 1 serves as a general function for calculating loss during training or testing. The algorithm begins at line 8 in Algorithm 1.

In the preparation phase (lines 9–11 in Algorithm 1), the framework splits the dataset's timestamps into training $\mathcal{T}_{\text{train}}$ and testing $\mathcal{T}_{\text{test}}$ ranges, Each edge server then fills in missing sensor values using the Imput function, generating local sensor values $X^{(i)}$ (lines 10–11 in Algorithm 1), as described in Section 4.4.

Once preparation is completed, FL begins, iterating through lines 13–23 in Algorithm 1 until the maximum number of rounds ρ is

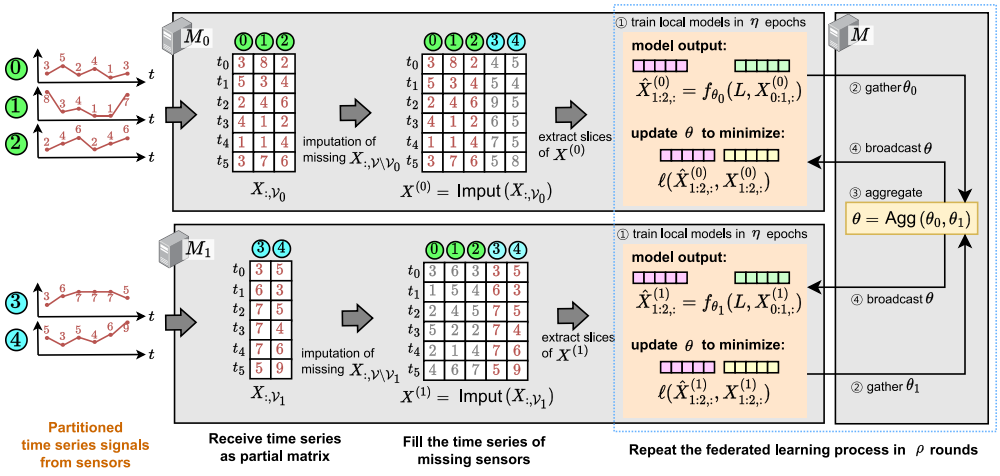

Fig. 4. Example diagram for illustrating overview of FedSTGNN.

reached. In each round, edge servers train their local models over η epochs and validate them (lines 13–18 in Algorithm 1). During training, the procedure GetLoss is conducted by three arguments: local model parameter θ_i, set of training timestamps $\mathcal{T}_{\text{train}}$, and the index of edge server i (line 15 in Algorithm 1). Because, for any edge server, i is not equal to the number of edge servers m, the variable S is considered as the local sensor value $X^{(i)}$ (line 1 in Algorithm 1). To get Eq. (4), the loss result $\mathcal{L}$ is set to zero at line 2 in Algorithm 1. For each timestamp in $\mathcal{T}_{\text{train}}$, if t does not satisfy the conditions required to produce a proper slice $X_{t-t_{\text{in}}:t+\text{out},:}$, the loop continues to the next timestamp (lines 3–4 in Algorithm 1). Otherwise, the local model θ_i produces prediction output at line 5 in Algorithm 1, and the loss ℓ is accumulated to $\mathcal{L}$ (lines 5–6 in Algorithm 1). The mean loss ℓ is calculated and returned (lines 7–8 in Algorithm 1), and the model parameters θ_i are updated to minimize the training loss. While the algorithm assumes a full-batch situation for clarity, minibatch implementations would involve modifying lines 15–16 to iteratively update small disjoint timestamp sets extracted from $\mathcal{T}_{\text{train}}$.

After training, the edge servers validate their local models using the testing timestamps $\mathcal{T}_{\text{test}}$ (line 17 in Algorithm 1), following a process similar to training but without parameter updates. The resulting local testing loss $\mathcal{L}^{(i)}_{\text{test}}$ is recorded in a list of local losses $\mathcal{R}^{(i)}_{\text{edge}}$ (line 18 in Algorithm 1).

The round concludes with aggregating local model parameters θ_i at the central server into the global model parameter θ using an aggregation function such as averaging (line 19 in Algorithm 1). The aggregated global model is validated against $\mathcal{T}_{\text{test}}$, using the original sensor values X rather than the imputed matrix $X^{(i)}$ (line 20 in Algorithm 1). This global model is broadcasted to all edge servers, updating their local parameters θ_i (line 23 in Algorithm 1). Finally, all recorded losses are returned for evaluation (line 24 in Algorithm 1).

4.3. Transition to federated learning

This section details how each ith edge server trains its local model θ_i using its local sensor value $X^{(i)}$, as described in Section 4.1, based on the single-node STGNN framework outlined in Section 3.2.

The idea is that there are two main input matrices L and X for the training, and if each edge server has matrices, L and X, with the same shape as in Eq. (3) (i.e., $L \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and $X \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}|}$), the equations in Eq. (3) and (4) can be directly applied to FL without significant modifications. Because sensors are usually fixed at specific locations, such as traffic or weather stations, and the number of sensors $|\mathcal{V}|$ is not drastically changed in the long term, the Laplacian matrix L is distributed identically across all edge servers before starting federated learning.

Algorithm 1: FedSTGNN

Data: ρ, /* the number of rounds */
m, /* the number of edge servers */
η, /* the number of epochs */
L, /* Laplacian matrix */
X, /* the sensor value matrix */
θ, /* the global model */
θ_i, /* the local model in i-th edge server */
$\mathcal{R}_{\text{central}} = []$; /* loss records of global model θ */
$\mathcal{R}^{(i)}_{\text{edge}} = []$; /* loss records of local model θ_i */

Procedure GetLoss (θ, $\mathcal{T}$, i)
1 **if** $i = m$ **then** $S \leftarrow X$ **else** $S \leftarrow X^{(i)}$;
2 $\mathcal{L} \leftarrow 0$;
3 **for each** $t \in \mathcal{T}$ **do**
4 **if not** $t_{\text{in}} \le t \le |\mathcal{T}| - t_{\text{out}}$ **then continue**;
5 $\hat{S}_{t:t+t_{\text{out}},:} \leftarrow f_\theta(L, S_{t-t_{\text{in}}:t,:})$;
6 $\mathcal{L} \leftarrow \mathcal{L} + \ell(\hat{S}_{t:t+t_{\text{out}},:}, S_{t:t+t_{\text{out}},:})$;
7 $\mathcal{L} \leftarrow \mathcal{L}/|\mathcal{T}|$;
8 **return** $\mathcal{L}$;

Procedure FedSTGNN
9 Split the timestamp to train $\mathcal{T}_{\text{train}}$ and to test $\mathcal{T}_{\text{test}}$;
10 **for** $0 \le i < m$ **do**
11 $X^{(i)} = \text{Imput}(X_{:,\mathcal{V}_i})$;
12 **for** ρ **times do**
13 **for** $0 \le i < m$ **do**
14 **for** η **times do**
15 $\mathcal{L}^{(i)}_{\text{train}} \leftarrow$GetLoss ($\theta_i$, $\mathcal{T}_{\text{train}}$, i);
16 Update θ_i to minimize $\mathcal{L}^{(i)}_{\text{train}}$;
17 $\mathcal{L}^{(i)}_{\text{test}} \leftarrow$GetLoss ($\theta_i$, $\mathcal{T}_{\text{test}}$, i);
18 $\mathcal{R}^{(i)}_{\text{edge}} \leftarrow \mathcal{R}^{(i)}_{\text{edge}} \cup [\mathcal{L}^{(i)}_{\text{test}}]$;
19 $\theta \leftarrow (1/m) \cdot \sum_{i=0}^{m-1} \theta_i$ /* gather and aggregate */;
20 $\mathcal{L}_{\text{test}} \leftarrow$GetLoss ($\theta$, $\mathcal{T}_{\text{test}}$, m);
21 $\mathcal{R}_{\text{central}} \leftarrow \mathcal{R}_{\text{central}} \cup [\mathcal{L}_{\text{test}}]$;
22 **for** $0 \le i < m$ **do**
23 $\theta_i \leftarrow \theta$ /* broadcast */;
24 **return** $[\mathcal{R}_{\text{central}}, \mathcal{R}^{(0)}_{\text{edge}}, \ldots, \mathcal{R}^{(m-1)}_{\text{edge}}]$;

On the other hand, the sensor value matrix X cannot be identical across all edge servers because each ith edge server can only access its assigned sensors $\mathcal{V}_i \subset \mathcal{V}$ and cannot access unseen sensors $\mathcal{V}\backslash\mathcal{V}_i$. To address this, X is substituted for the local sensor value $X^{(i)} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}|}$,

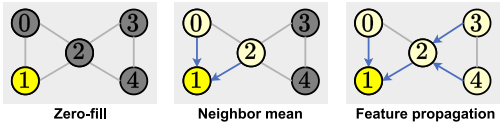

Fig. 5. Three types of missing sensor value imputation.

which may contain missing values ϕ. Therefore,

$$X^{(i)}_{:,v} := \begin{cases} X_{:,v} & v \in \mathcal{V}_i \\ \phi & v \in \mathcal{V} \backslash \mathcal{V}_i. \end{cases} \tag{5}$$

These null values ϕ can disrupt matrix calculations, so the imputation process $\mathsf{Imput}(X_{:,\mathcal{V}_i})$ shown at Line 11 in Algorithm 1 requires that substitutes null values in $X^{(i)}_{:,\mathcal{V}\backslash\mathcal{V}_i}$ with some appropriate numbers on each edge server. The details of the imputation method will be discussed in Section 4.4.

By substituting the global model f_θ of Eq. (3) with the local model f_{θ_i}, the formula for local STGNN prediction is expressed as

$$\hat{X}^{(i)}_{t:t+t_{\text{out}},:} = f_{\theta_i}(L, X^{(i)}_{t-t_{\text{in}}:t,:}). \tag{6}$$

Similarly, the loss function Eq. (4) can be converted into local model loss by replacing X with $X^{(i)}$, like

$$\mathcal{L}^{(i)}_{\text{train}} = \frac{1}{\tau} \sum_{t=t_{\text{in}}}^{|\mathcal{T}_{\text{train}}|-t_{\text{out}}} \ell(X^{(i)}_{t:t+t_{\text{out}},:}, \hat{X}^{(i)}_{t:t+t_{\text{out}},:}), \tag{7}$$

which is the same as Line 15 in Algorithm 1. Each edge server updates its local parameter θ_i to minimize $\mathcal{L}^{(i)}_{\text{train}}$ as much as possible, by iterating the mini-batch training process η times (η is the number of local epochs). After the local training, the local parameter is sent to the central server for parameter aggregation.

4.4. Imputation of missing value

Various types of time-series imputation methods exist (Moritz et al., 2015). A primary approach is interpolation, which fills in missing values based on existing values when missing and present data coexist within a single-sensor time series. Interpolation usually relies on two or more known timestamp values to estimate the missing ones, using techniques such as linear interpolation (averaging two known values) or fitting data to a high-degree polynomial (e.g., Newton interpolation). However, interpolation is not applicable in edge-computing environments because it is targeted to partially filled single sensor data. For a multi-sensor environment, each edge server has either all or none of the data for a specific sensor, making it impossible to mix missing and present values within a single sensor's data.

To address this limitation, we employ an alternative strategy to replace unknown values with existing data. Two approaches were considered: a statistical model and a graph-based model. The statistical model uses linear approximation or a simple neural network, but its imputation results can vary between runs due to dependence on the initial parameter values of the model. This variability may cause fluctuations in training performance, complicating the evaluation of the original STGNN model's effects. Conversely, graph-based methods rely solely on input data and employ a consistent algorithm, ensuring stability across repetitive training sessions. Additionally, graph-based imputation leverages the inherent advantages of STGNNs by utilizing graphs for training. For these reasons, we selected graph-based imputation.

Graph-based imputation operates by gathering information from neighboring vertices connected by graph edges. We classify graph-based imputation into three categories, as depicted in Fig. 5, based on the number of hops used to collect information from adjacent vertices.

Zero-fill is the simplest method for imputation, where missing values are not inferred but replaced with zeros. Formally, this is achieved by concatenating each edge server's partial matrix $X_{:,\mathcal{V}_i}$ with a zero matrix $O_{:,\mathcal{V}\backslash\mathcal{V}_i}$, as defined in Eq. (8).

$$X^{(i)} = X_{:,\mathcal{V}_i} \parallel O_{:,\mathcal{V}\backslash\mathcal{V}_i}. \tag{8}$$

Neighbor mean calculates the imputation value as the average of the neighboring vertices' values. For a vertex u with an unknown value a_u, its neighboring vertices $\mathcal{N}_u$, and edge values e_{uv} where $v \in \mathcal{N}_u$, the neighbor mean $\hat{a}_u$, as defined in Eq. (9), serves as the imputed value for a_u.

$$\hat{a}_u = \frac{\sum_{v\in\mathcal{N}_u} e_{vu} a_v}{\sum_{v\in\mathcal{N}_u} e_{vu}} \tag{9}$$

Extending this calculation in Eq. (9) to all missing vertices requires a transition matrix P. The sum of each column in P must equal 1, because Eq. (9) is a linear combination $\hat{a}_u = \sum_{v\in\mathcal{N}_u} w_{vu} a_v$, where the weight of u's inward edge is $w_{vu} = e_{vu} / \sum_{v\in\mathcal{N}_u} e_{vu}$, and the sum of these weights satisfies $\sum_{v\in\mathcal{N}_u} w_{vu} = 1$. The left-hand stochastic matrix $P = \tilde{A}D^{-1}$ (Wikipedia contributors, 2024) satisfies this requirement, where $\tilde{A}$ is the output of the function $\mathsf{GaussKern}$ from Eq. (1).

Initially, the neighbor mean method is similar to zero-fill, as it also concatenates the zero matrix $O_{:,\mathcal{V}\backslash\mathcal{V}_i}$ into $X_{:,\mathcal{V}_i}$. However, the key distinction lies in the intermediate matrix $Z^{(i)}$, which is defined in Eq. (10).

$$Z^{(i)} = X_{:,\mathcal{V}_i} \parallel O_{:,\mathcal{V}\backslash\mathcal{V}_i} \tag{10}$$

To calculate Eq. (9) for all vertices, the matrix $Z^{(i)}P$ is obtained. The matrix contains the neighbor mean for both the missing sensors $\mathcal{V}\backslash\mathcal{V}_i$ and the existing sensors $\mathcal{V}_i$, due to the nature of the dot product operation. To isolate the missing sensor values, only the $\mathcal{V}\backslash\mathcal{V}_i$ part of $Z^{(i)}P$ is extracted. Using the submatrix notation from Definition 3, this extraction is represented as $(Z^{(i)}P)_{:,\mathcal{V}\backslash\mathcal{V}_i}$. Finally, the local sensor value $X^{(i)}$ is obtained by concatenating this extracted submatrix with $X_{:,\mathcal{V}_i}$, as shown in Eq. (11).

$$X^{(i)} = X_{:,\mathcal{V}_i} \parallel (Z^{(i)}P)_{:,\mathcal{V}\backslash\mathcal{V}_i} \tag{11}$$

Feature propagation (Rossi et al., 2022) extends the neighbor mean method by *repeating the neighbor mean operation multiple times*, allowing the aggregation of values from vertices beyond n-hops. The key difference lies in the transition matrix, which is updated to a normalized adjacency matrix $P = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}$. Because the intermediate matrix $Z^{(i)}$ in Eq. (10) is recalculated iteratively, it is denoted as $Z^{(i,n)}$ in feature propagation, where i represents the edge server index and n the number of repetitions. Therefore, Eq. (10) is expressed as $Z^{(i,0)} = X_{:,\mathcal{V}_i} \parallel O_{:,\mathcal{V}\backslash\mathcal{V}_i}$, and the imputation formula in Eq. (11) is rewritten as Eq. (12) for feature propagation.

$$Z^{(i,n+1)} = X_{:,\mathcal{V}_i} \parallel (Z^{(i,n)}P)_{:,\mathcal{V}\backslash\mathcal{V}_i} \tag{12}$$

By increasing n in Eq. (12), the intermediate matrix $Z^{(i,n)}$ aggregates values from vertices at progressively larger n-hop distances. The range of n is $0 \le n < n_{\max}$, with $n_{\max} = 40$ chosen for evaluation. This enables feature propagation to gather information from up to 40-hop neighbors, leveraging graph edge values to impute missing sensor data. The final intermediate matrix becomes the local sensor value that is $X^{(i)} = Z^{(i,n_{\max})}$.

5. Evaluation

This section demonstrates the proposed framework's performance through three aspects of extensive experiments. First, we describe the evaluation environment in Section 5.1 with detailed specifications of the computing nodes, models, datasets, and measurement metrics. Section 5.2 shows the overall results of prediction performance for FedSTGNN and other methods, with comparison analysis in Section 5.3. Section 5.4 and Section 5.5 compare training time and network communication cost, respectively. In addition, Section 5.6

investigates the effect of each key component of the framework through ablation studies. Section 5.7 evaluates FedSTGNN against alternative designs, and Section 5.8 examines the robustness of the framework under challenging conditions.

5.1. Environment

We obtained datasets from various domains as described below. The specifications of these datasets are provided in Table 2.

- **Traffic**: Several STGNN researches have extracted highway traffic datasets such as **METR-LA**, **PEMS-BAY**, **PEMS03**, **PEMS04**, **PEMS07**, **PEMS08**, and **PEMSD7**, with different locations and time intervals. They originated from the Freeway Performance Measurement (PeMS) System of California Department of Transportation (California Department of Transportation, 2018). The datasets contain 5 min average car speed measured by Vehicle Detection Sensors (VDS) on each road.
- **Urban environment**: Microsoft provides a raw dataset of Particulate Matter (PM) concentrations below 2.5 micrometers (Zheng et al., 2015). The raw dataset contains hourly measurements from various regions across China over several periods. We extracted **Air-PM25** dataset in approximately one year of measurements from 172 air quality monitoring stations around Beijing, Tianjin, and neighboring small cities.
- **Epidemic**: The Center for Systems Science and Engineering (CSSE) at Johns Hopkins University compiled and published region-specific COVID-19 case numbers worldwide during the approximately three-year pandemic (Dong et al., 2020). Because this dataset represented a nonstationary time series with an increasing trend, we extracted the daily changes to create **CSSE-COVID** dataset in case numbers for our analysis. We utilized data from 287 global regions divided by country or state, using the geographical center of each region as its location reference.
- **Global events**: Global Data on Events, Location, and Tone (GDELT) project (Leetaru and Schrodt, 2013), operated by Google Jigsaw, is a platform that collects and freely provides global broadcast, print, and internet news. We extracted data from the GDELT 1.0 dataset, which contains daily international events spanning approximately 45 years. The GDELT dataset consists of tuples (timestamp, source country, target country, Goldstein scale). We created the **GDELT-139** dataset by aggregating the average Goldstein scale of each target country receiving from other countries on specific dates. The Goldstein scale measures the intensity of conflict or cooperation in international events, ranging from −10 to 10. Negative values represent expulsions, detentions, arrests, protests, diplomatic ruptures, or military conflicts, whereas positive values indicate apologies, visits, ceasefires, support statements, agreements, or cooperation. To exclude relatively isolated countries from international relations, we selected only 139 countries with at least one event every three months. For periods without events, we estimated Goldstein scale values using linear interpolation. Geographic locations were based on the central point of each country's territory.
- **Consumer prices**: Korea National Oil Corporation provides daily gasoline prices across South Korea, publishing them on a website called **Opinet** (Korea National Oil Corporation, 2024). We collected raw data from this website over ten years and then calculated daily averages for 228 municipality-level divisions in South Korea. We used the latitude and longitude of each division's town hall as location reference points.
- **Neuroscience**: The **Uzel2022** dataset was acquired from neurobiological research (Uzel et al., 2022). It captures C. elegans neuronal activation intensities for 30 min, with neuronal images recorded at approximately three frames per second. The edge values in the dataset represent the combined count of gap junctions and synapses between neurons, while vertex values correspond to the measured luminescence intensity.
- **Meteorology**: The weather dataset, **NOAA-958**, contains global daily temperatures at various locations around the world. This dataset was collected by the National Centers for Environmental Information (Smith et al., 2011), and we extracted data from 958 randomly chosen weather stations over 12 years.

Next, we evaluate FedSTGNN using several existing STGNN models. Since the STGNN model can be generalized as an arbitrary model f_θ in Eq. (6) and Algorithm 1, we selected f_θ from the following models to assess performance differences depending on the model type. We keep the design and hyperparameters of each model as shown in their original papers or official repositories.

- AGCRN: Adaptive graph convolutional recurrent network (AGCRN) (Bai et al., 2020) utilizes an embedding matrix for spatial information, with the temporal module based on RNNs. AGCRN uniquely calculates the Laplacian matrix L using the dot product of the embedding matrix E, such that $L = E \cdot E^T$.
- ASTGCN: Attention-based spatial–temporal graph convolutional network (ASTGCN) (Guo et al., 2019) uses graph convolution to extract spatial information from sensors at the same timestamp and temporal convolution for extracting information across timesteps. Both convolution mechanisms are enhanced by the attention mechanism (Vaswani, 2017).
- DGCRN: Dynamic graph convolutional recurrent network (DGCRN) (Li et al., 2023), an advanced version of DCRNN (Li et al., 2018), generates a new graph based on sensor distances and node embeddings from the time series. It combines graph convolutions of the generated and original graphs on the model's output side.
- DSTAGNN: Dynamic spatial–temporal aware graph neural network (DSTAGNN) (Lan et al., 2022) uses both location-based distance graphs and synthetic time-series-based graphs created via Wasserstein distance between time series. Multi-head attention (Vaswani, 2017) captures spatial relevance, while temporal convolutions identify dependencies across various time intervals.
- GWNet: Graph WaveNet (GWNet) (Wu et al., 2019) extends dilated convolution from WaveNet (Oord et al., 2016) by integrating it with graph convolution. The input time range for dilated convolution depends on two hyperparameters, layers l and blocks b. We set the minimum l and b values to satisfy $(2^l-1)\times b+1 \geq t_{\text{in}}$.
- STGODE: Spatial–temporal graph ordinary differential equations network (STGODE) (Fang et al., 2021) employs ordinary differential equations (ODEs) for its model design. A new distance matrix is created using dynamic time warping, which calculates mean time series by folding data into specific periods (e.g., daily or hourly). ODEs can help prevent over-smoothing problems while reducing memory and time consumption.

We construct an edge-computing testbed comprising five computing nodes: one central server and four edge servers. Each edge server is equipped with an Intel Xeon E5-2630v4 processor, a single NVIDIA GTX 1080Ti GPU, 64 GB of memory, and a 512 GB SSD, connected via a 1Gbps Ethernet switch. The vertices $\mathcal{V}$ of each dataset are split into four local vertex sets $\mathcal{V}_i$ using spectral clustering, and each $\mathcal{V}_i$ was assigned to an edge server. We set the maximum number of communication rounds to $\rho = 100$, with each edge server performing local model updates over $\eta = 2$ epochs per round. We apply early stopping with a patience of 10 rounds and a minimum improvement threshold 1×10^{-3} to prevent overfitting. The model used in the framework is optimized by Adam optimizer (Kingma and Ba, 2014), with the learning rate of 1×10^{-3} and the weight decay of 5×10^{-4}. We split the dataset into a training and a testing set at 0.7 ratio. The number of input and output timesteps shown in Eq. (6) varies by dataset (12 for traffic datasets and Air-PM25, 15 for Uzel2022, and 14 for the remaining datasets), and we set the input and output timesteps to be the same.

Table 2
Dataset specifications.

Category	Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{T}\rvert$	Start	End	Period
Traffic	**METR-LA** (Li et al., 2018)	207	34 272	2012-03-01	2012-06-27	5 min.
	PEMS-BAY (Li et al., 2018)	325	52 116	2017-01-01	2017-06-30	5 min.
	PEMS03 (Guo et al., 2021)	358	26 208	2018-09-01	2018-11-30	5 min.
	PEMS04 (Guo et al., 2021)	307	16 992	2018-01-01	2018-02-28	5 min.
	PEMS07 (Guo et al., 2021)	883	28 224	2017-05-01	2017-08-06	5 min.
	PEMS08 (Guo et al., 2021)	170	17 856	2016-07-01	2016-08-31	5 min.
	PEMSD7 (Yu et al., 2018)	228	12 672	2012-05-01	2012-06-13	5 min.
Urban environment	**Air-PM25** (Zheng et al., 2015)	172	8664	2014-05-01	2015-04-30	1 h
Epidemic	**CSSE-COVID** (Dong et al., 2020)	287	1143	2020-01-22	2023-03-09	1 day
Global events	**GDELT-139** (Leetaru and Schrodt, 2013)	139	12 833	1979-01-01	2024-02-18	1 day
Consumer prices	**Opinet** (Korea National Oil Corporation, 2024)	228	3653	2015-01-01	2024-12-31	1 day
Neuroscience	**Uzel2022** (Uzel et al., 2022)	58	3312	–	–	1/3 s
Meteorology	**NOAA-958** (Smith et al., 2011)	958	4748	2010-01-01	2022-12-31	1 day

We evaluated model performance using the mean arctangent absolute percentage error (MAAPE) (Kim and Kim, 2016), an improved metric for time-series forecasting. Unlike the mean absolute percentage error, which diverges to infinity when the predicted value exceeds the actual value, MAAPE converges to 100% in such cases, offering a more robust measure of prediction error.

5.2. Overall results of prediction performance

We evaluate the performance of the global model f_θ by applying FedSTGNN to various combinations of datasets and models, as shown in Fig. 6. Specifically, we compare the original single-node STGNN model (i.e., non-federated, described in Section 3.2) with FedSTGNN on three imputation methods (i.e., zero-fill, neighbor mean, and feature propagation; outlined in Section 4.4), to measure how much FedSTGNN preserves the performance of single-node models.

For more objective comparisons, we also choose a baseline method. However, to the best of our knowledge, there are no framework studies such as ours that expand the existing single-node STGNN to the federated environment, so there are no direct competitors comparable to our work. Instead, we select CNFGNN (Meng et al., 2021), an FL-specific STGNN model mentioned in Section 2 that runs STGNN in the same federated environment. The difference between our framework and the FL-specific STGNN model is that our framework is a general method that transforms any single-node STGNN model into the FL version of the model without any modifications to the original model, such as a plugin. In contrast, the FL-specific model is built from scratch as a multi-node model, with neural network layers partitioned to the edge and central server disjointly, requiring exchanges of every intermediate data (e.g., parameters, gradients, and embedding vectors) and constructed as totally monolithic manner, so they cannot be modified or replaceable partially.

In Fig. 6, **Zero**, **Neighbor**, and **FP** represent the types of imputation methods used in FedSTGNN: zero-fill, neighbor mean, and feature propagation, respectively. **Single** refers to the non-federated version of the STGNN model. The asterisk mark $\boxed{*}$ at the end of some bars indicates notable cases among the results, specifically where **Neighbor** or **FP** performs better than **Zero**. The numbers displayed above some bars represent the ratio of improvement, showing how many times better the best case among zero-fill, neighbor mean, and feature propagation is compared to the single-node model. Finally, on the far right, we display the performance of CNFGNN for each dataset. For NOAA-958, we were unable to run CNFGNN due to GPU memory limitations with our available equipment.

We evaluate the effect of the graph-based imputation methods in Fig. 6, as discussed in Section 4.4, on the performance of FedSTGNN. The zero-fill method, which substitutes missing values with zeros, does not leverage the graph structure and is thus considered as a naive method. We identified cases where graph-based imputation methods (neighbor mean and feature propagation) outperformed the zero-fill methods in terms of lower MAAPE values. Out of the 78 combinations of datasets and models, 55 cases (approximately 70.5%) achieved better performance with one of the graph-based imputation methods. Among these, the feature propagation was superior in 46 cases (59.0%), whereas neighbor mean only outperformed the zero-fill in 9 cases (11.5%).

In Fig. 6, the performance dominance of graph-based methods varies depending on the model and dataset. The feature propagation dominantly shows the best results with the combination of AGCRN, DGCRN, and STGODE models. In contrast, ASTGCN, DSTAGNN, and GWNet demonstrate relatively stronger dependences on the input dataset rather than the model itself. Additionally, some graph-based imputation methods significantly reduce prediction errors compared to the zero-fill. For example, the AGCRN and STGODE model exhibited drastic error reductions for PEMS03, PEMS04, PEMS07, and PEMS08. Although FedSTGNN works well in most cases, there appear to be specific combinations of models and datasets where it performs exceptionally well.

We examined the performance ratio between the best-performing FedSTGNN configuration and the single-node model for each combination of model and dataset. FedSTGNN trains local models in edge servers separately on disjoint datasets, and aggregates local parameters to the central server. This decentralized training process inherently introduces information loss, leading to potential performance degradation compared with the original single-node model. The experiment of Fig. 6 showed that even in the worst case, the performance degradation remained within ×2 (only a single case of the PEMS07 dataset on the AGCRN model). On average, we observed a performance decrease of approximately ×1.26 across all combinations, which is an acceptable performance difference.

Our FedSTGNN outperforms CNFGNN on some datasets, such as METR-LA and PEMSD7. However, CNFGNN achieves slightly better performance than others, primarily due to fundamental design differences between the two approaches. FedSTGNN assumes that the number of edge servers is significantly smaller than the number of sensors, minimizing data exchange with the central server and relying on imputation to handle missing values. In contrast, CNFGNN assumes that each sensor corresponds to an edge server, leveraging frequent and diverse communication with the central server to train the model with complete raw data. As a result, CNFGNN inherently benefits from an information-rich training process, leading to improved performance.

Another key factor is that CNFGNN is a specialized *model* explicitly designed for federated learning, whereas FedSTGNN is a general *framework* for converting single-node STGNNs into FL models. In CNFGNN, the edge and central servers collaboratively train a tightly integrated model, exchanging parameters, gradients, and hidden representations. This monolithic design allows for more effective model architecture and hyperparameter optimization. In contrast, FedSTGNN prioritizes generality by preserving the structure of the original single-node model, making end-to-end optimization more challenging and resulting in slightly lower performance.

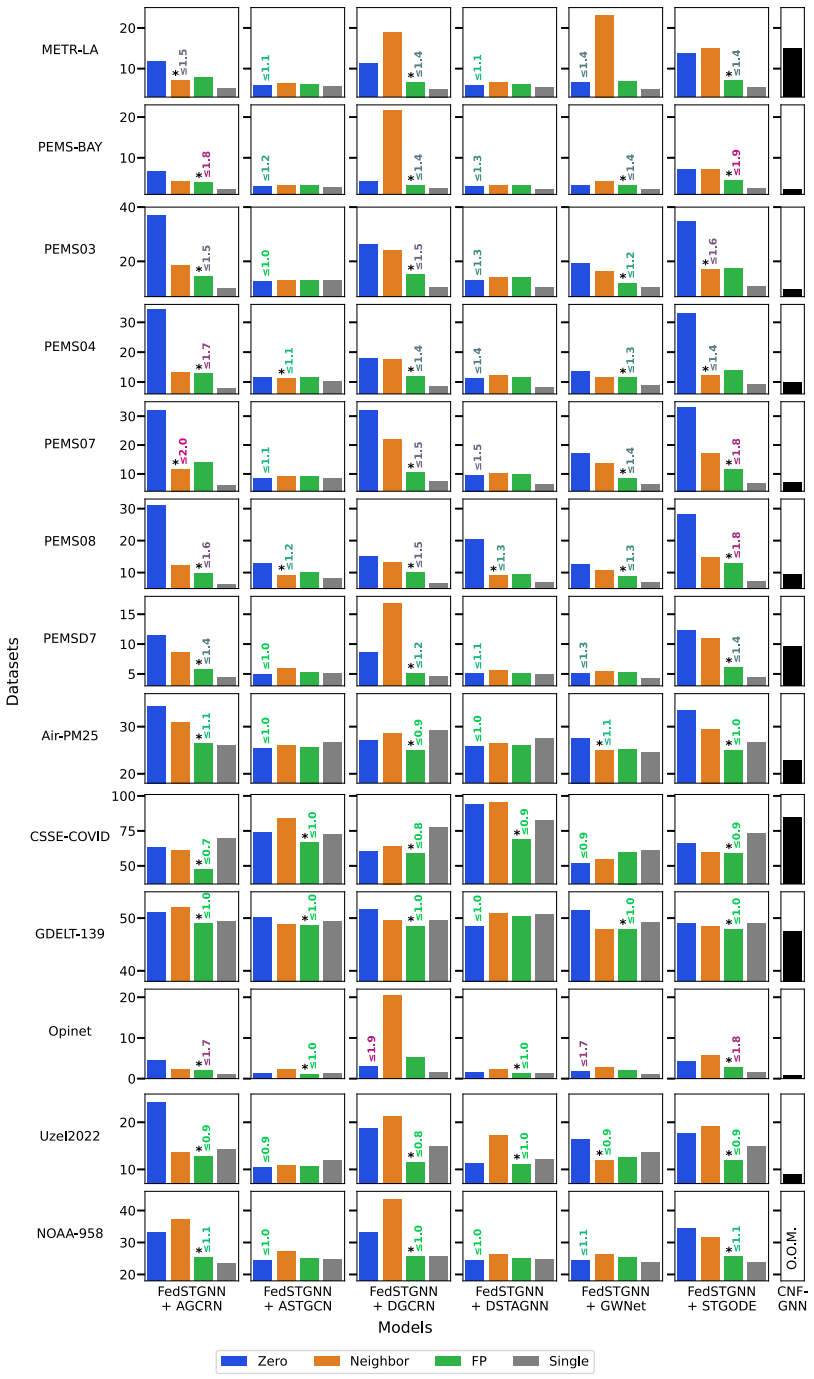

Fig. 6. Prediction performance results in MAAPE for single-node model, FedSTGNN-applied model and CNFGNN (unit: %). O.O.M. stands for Out Of Memory.

5.3. Prediction performance comparison

This section presents a more quantitative comparison by aggregating the results shown in Fig. 6. Fig. 7 summarizes the difference ratio of prediction loss between FedSTGNN and the single-node model. The figure consists of six plots arranged in a grid. The vertical direction represents the results grouped by the imputation methods proposed in Section 4.4: zero-fill, neighbor mean, and feature propagation, with colors matching those in Fig. 6. In the horizontal direction, the top three plots show the average ratio per dataset, with the overall average annotated. The bottom three plots display histograms of prediction loss difference ratios across all the models and datasets, with Kernel Density Estimation (KDE) curves showing peak values.

At the top three charts of Fig. 7, among FedSTGNN's three missing value imputation methods, the zero-fill and neighbor mean show similar average values of ×1.81 and ×1.83, respectively. At the same time, feature propagation minimizes the gap with single-node models with an average of ×1.24. It suggests that filling missing values by increasing the number of hops more closely restores the data to its original pre-partitioned state. At the dataset level, we observe that METR-LA, PEMS-BAY, and Opinet perform better with zero-fill than the neighbor mean, whereas other datasets such as PEMS03, PEMS04, PEMS07, and PEMS08 show the opposite trend. However, feature propagation, which actively utilizes the graph for imputation, significantly reduces differences from the single-node model across all datasets, resulting in lower variance.

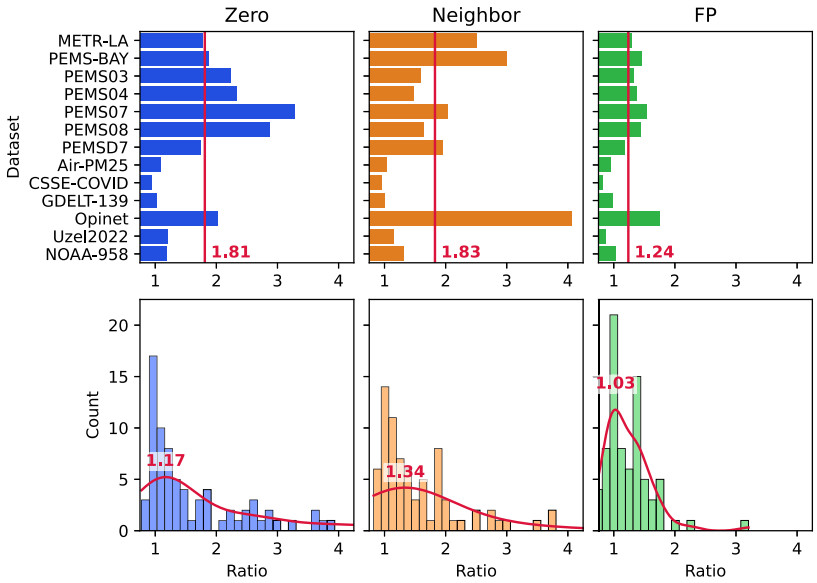

Fig. 7. Bar chart and histogram for the ratio of FedSTGNN-applied model to single-node model prediction performance. Lower is better.

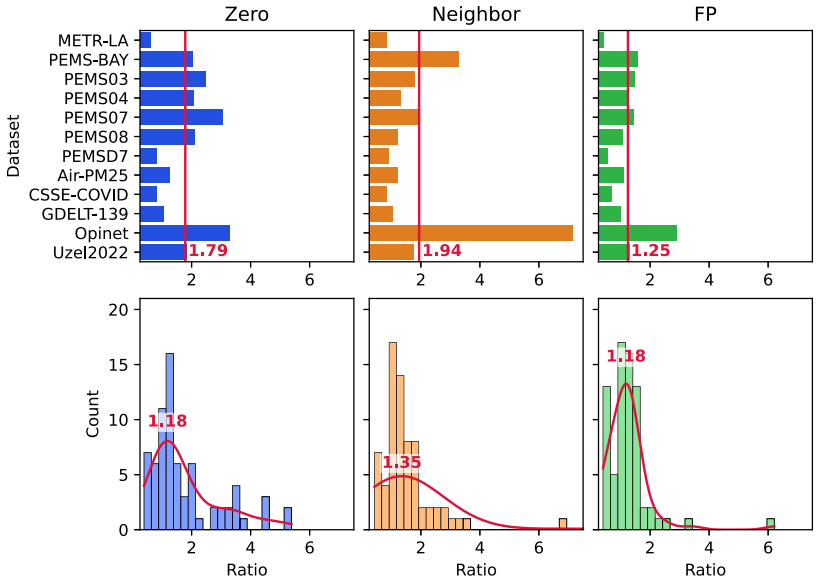

Fig. 8. Bar chart and histogram for the ratio of FedSTGNN-applied model to CNFGNN prediction performance. Lower is better.

This trend is further confirmed in the bottom three charts of Fig. 7. While zero-fill and neighbor mean exhibit high variance and fat-tailed distributions, the KDE plot for feature propagation shows a relatively high peak and is closer to 1 compared to the other two methods. It demonstrates that when using FedSTGNN with feature propagation, performance remains close to that of single-node models in most cases, indicating that FedSTGNN can serve as a universal solution for extending existing single-node STGNN models to federated learning.

We also examine how FedSTGNN compares to the CNFGNN baseline using the results from Fig. 6, as shown in Fig. 8. We maintain the same plot configuration as in Fig. 7, annotating both the mean of the performance difference ratios and the peak values of the KDE plots on Fig. 8. Note that for NOAA-958, we could not make comparisons due to the out-of-memory issues.

The upper charts in Fig. 8 display trends similar to those in Fig. 7. Although the zero-fill and neighbor mean show slightly worse performance with ratios of $\times 1.79$ and $\times 1.94$, respectively, feature propagation consistently maintains a significantly smaller difference (average ratio of $\times 1.25$) than CNFGNN. Notably, except for the neighbor mean for Opinet, both the zero-fill and neighbor mean exhibit reduced variance with single-node models. Additionally, feature propagation continues to demonstrate the lowest variance among all methods.

The lower histograms in Fig. 8 confirm that the zero-fill and neighbor mean maintain fat-tailed distributions compared with feature propagation. We can also observe that although zero-fill and feature propagation share the same KDE peak value, they differ significantly in variance. Overall, FedSTGNN with feature propagation demonstrates the best performance.

5.4. Training time analysis

In this section, we describe the differences in training time between our method, FedSTGNN, and the baseline method, CNFGNN. Since FedSTGNN and CNFGNN are approaches for constructing the federated learning STGNN, they both optimize models through repeated rounds. Therefore, we present the per-round training time in Table 3 and show

Table 3
Per-round training time comparison (unit: seconds). O.O.M. stands for Out Of Memory. Each percentage value means the relative ratio of elapsed time compared to the CNFGNN.

Dataset	CNFGNN	FedSTGNN						Avg. ratio of FedSTGNN
		AGCRN	ASTGCN	DGCRN	DSTAGNN	GWNet	STGODE	
METR-LA	369.1	53.2 (14.4%)	52.1 (14.1%)	357.2 (96.8%)	215.9 (58.5%)	76.5 (20.7%)	142.9 (38.7%)	40.5%
PEMS-BAY	854.4	111.8 (13.1%)	131.1 (15.3%)	649.3 (76.0%)	655.4 (76.7%)	125.4 (14.7%)	351.3 (41.1%)	39.5%
PEMS03	247.0	61.8 (25.0%)	79.2 (32.1%)	427.6 (173.1%)	372.2 (150.7%)	71.1 (28.8%)	190.4 (77.1%)	81.1%
PEMS04	244.6	35.5 (14.5%)	40.6 (16.6%)	195.0 (79.7%)	194.5 (79.5%)	42.2 (17.2%)	106.3 (43.5%)	41.8%
PEMS07	1121.1	192.2 (17.1%)	388.7 (34.7%)	1853.4 (165.3%)	2361.4 (210.6%)	218.0 (19.4%)	554.1 (49.4%)	82.8%
PEMS08	247.3	24.6 (10.0%)	23.3 (9.4%)	125.4 (50.7%)	93.5 (37.8%)	36.1 (14.6%)	60.4 (24.4%)	24.5%
PEMSD7	244.0	20.2 (8.3%)	22.8 (9.3%)	102.8 (42.1%)	96.9 (39.7%)	35.3 (14.5%)	61.1 (25.0%)	23.2%
Air-PM25	96.3	12.5 (13.0%)	12.4 (12.9%)	59.2 (61.4%)	49.3 (51.2%)	32.4 (33.7%)	31.7 (32.9%)	34.2%
CSSE-COVID	38.6	4.4 (11.3%)	4.6 (11.9%)	15.7 (40.7%)	22.7 (58.7%)	5.7 (14.8%)	9.4 (24.4%)	27.0%
GDELT-139	122.9	21.5 (17.5%)	17.0 (13.9%)	92.2 (75.1%)	64.8 (52.8%)	28.1 (22.9%)	44.9 (36.5%)	36.4%
Opinet	65.8	9.2 (14.0%)	9.8 (15.0%)	33.7 (51.3%)	39.6 (60.2%)	20.7 (31.5%)	21.0 (31.9%)	34.0%
Uzel2022	25.3	6.3 (24.9%)	4.0 (15.8%)	22.4 (88.6%)	13.0 (51.6%)	6.0 (23.6%)	6.9 (27.4%)	38.7%
NOAA-958	O.O.M.	48.6	89.3	427.6	561.4	50.2	112.6	–

the ratio of differences between CNFGNN and our method. Note that our method performs the imputation process during the preprocessing stage, and the training process is the same regardless of the chosen imputation method. Consequently, the per-round time is independent of the selection of the imputation method. Therefore, we calculated the average training time from the experimental results in Fig. 6 and recorded it in Table 3.

Unlike CNFGNN, which is a single model, FedSTGNN is a framework applied to existing models, so execution time varies depending on which model is combined with FedSTGNN. As shown in Table 3, FedSTGNN's execution time varies significantly depending on the STGNN model. Nevertheless, among the 72 dataset and model combinations (excluding NOAA-958), only four combinations (DGCRN and DSTAGNN models with PEMS03 and PEMS07 datasets) required more execution time than CNFGNN; the remaining 68 combinations (94.4% of all combinations) were faster than CNFGNN. Therefore, FedSTGNN can be considered generally faster than CNFGNN.

When averaging across all ratios in Table 3, FedSTGNN requires only 42% of CNFGNN's execution time, demonstrating superior temporal efficiency. We guess that this time difference originates from structural differences. That is, FedSTGNN only trains on edge servers and aggregates the results at the central server, whereas CNFGNN, as mentioned in Section 2, processes part of the model on edge servers and the remainder on the central server during training. This approach generates multiple instances of network traffic between central and edge servers and can result in longer execution times.

5.5. Communication cost analysis

We explain the differences in network traffic usage between the baseline method CNFGNN and our method FedSTGNN. In Table 4, we measured how many megabytes of data were transmitted per round for both methods. We display the ratio of our method compared to the baseline method. Overall, all FedSTGNN results showed dramatically lower network traffic usage than those of CNFGNN. Among these results, the highest traffic usages were with DSTAGNN on Uzel2022 (8.62%) and CSSE-COVID (7.70%), whereas the remaining results were significantly lower. In terms of quantity, the experimental results show less than 1% traffic usage compared to CNFGNN, which totaled 61 cases, representing 84.7% of all the experiments. Therefore, FedSTGNN generally uses much less network traffic than CNFGNN.

We average the FedSTGNN results in Table 4, which shows that it uses only 0.74% of the traffic compared to CNFGNN, demonstrating exceptional bandwidth efficiency. We speculate that this results from structural differences, similar to our analysis in Section 5.4. FedSTGNN performs only two communications per round: transmitting parameters from edge servers to the central server and broadcasting aggregated parameters to edge servers. As mentioned in Table 1, since the size of the parameters required for FedSTGNN is not large (under 8MB), this process uses minimal network bandwidth. In contrast, CNFGNN requires multiple communications within a single round as mentioned in Section 2, and exchanges various information between edge and central servers, including not only parameters but also encoded vectors, embedding vectors, and gradients of hidden vectors, resulting in much higher traffic usage than FedSTGNN.

5.6. Characteristics of FedSTGNN

This section presents the characteristics of the proposed FedSTGNN framework through three aspects. Section 5.6.1 presents an ablation study including the effectiveness of each component of FedSTGNN. Section 5.6.2 compares different FL aggregation strategies, including FedAvg, FedProx, and SCAFFOLD. Finally, Section 5.6.3 presents the performance of the proposed framework, varying the imputation methods.

5.6.1. Ablation study

To validate the contributions of the key components of the proposed FedSTGNN framework, we conducted an ablation study on two components: the existence of parameter aggregation in FL and imputation. For removing FL, we set each edge to train independently without sharing any parameters and only aggregate at the final stage to validate the overall process. Also, for removing imputation, we substitute the imputation from the propagation-based imputation to the simple zero-filling method for missing data.

As shown in Fig. 9, the baseline configuration (i.e., both FL and imputation are on) achieves the best performance in most cases. Between the two components, the removal of FL has a more significant impact on performance degradation than the removal of imputation. Especially for AGCRN with FedSTGNN, the effects of both FL and imputation are significant. The results confirm that the absence of both FL and imputation is a critical component for decreasing the performance of FedSTGNN.

5.6.2. Aggregation strategies of federated learning

We conducted experiments with other aggregation methods over FedAvg (McMahan et al., 2017), which is the default aggregation function $\mathrm{Agg}(\theta_i, \dots)$ described in Section 4.1. Specifically, we adapted FedProx (Li et al., 2020) and SCAFFOLD (Karimireddy et al., 2020) into FedSTGNN and compared it to the FedAvg version of FedSTGNN. Both methods aggregate local model parameters differently into the global model to reduce the difference between the local and global models. For this purpose, both FedProx employs a proximal term in edge servers, and SCAFFOLD calculates a correlation term.

As shown in Fig. 10, FedSTGNN using FedAvg presents the lowest prediction error across all datasets and model combinations, while

Table 4
Per-round communication cost comparison (unit: MB). O.O.M. stands for Out Of Memory. Each percentage value means the relative ratio of communication cost compared to the CNFGNN.

Dataset	CNFGNN	FedSTGNN						Avg. ratio of FedSTGNN
		AGCRN	ASTGCN	DGCRN	DSTAGNN	GWNet	STGODE	
METR-LA	2172.03	2.85 (0.13%)	0.91 (0.04%)	0.75 (0.03%)	9.91 (0.46%)	1.02 (0.05%)	3.10 (0.14%)	0.14%
PEMS-BAY	5143.51	2.86 (0.06%)	1.89 (0.04%)	0.79 (0.02%)	14.43 (0.28%)	1.03 (0.02%)	3.13 (0.06%)	0.08%
PEMS03	1633.81	2.86 (0.17%)	2.25 (0.14%)	0.80 (0.05%)	15.92 (0.97%)	1.03 (0.06%)	3.14 (0.19%)	0.27%
PEMS04	1633.81	2.86 (0.17%)	1.72 (0.11%)	0.78 (0.05%)	13.66 (0.84%)	1.03 (0.06%)	3.13 (0.19%)	0.24%
PEMS07	7661.54	2.88 (0.04%)	12.32 (0.16%)	0.96 (0.01%)	53.03 (0.69%)	1.07 (0.01%)	3.26 (0.04%)	0.16%
PEMS08	1633.81	2.85 (0.17%)	0.68 (0.04%)	0.74 (0.05%)	8.76 (0.54%)	1.02 (0.06%)	3.09 (0.19%)	0.17%
PEMSD7	1633.81	2.85 (0.17%)	1.05 (0.06%)	0.76 (0.05%)	10.62 (0.65%)	1.02 (0.06%)	3.11 (0.19%)	0.20%
Air-PM25	487.79	2.85 (0.58%)	0.70 (0.14%)	0.74 (0.15%)	8.82 (1.81%)	1.02 (0.21%)	3.09 (0.63%)	0.59%
CSSE-COVID	173.43	2.86 (1.65%)	1.54 (0.89%)	0.77 (0.45%)	13.35 (7.70%)	1.03 (0.60%)	3.54 (2.04%)	2.22%
GDELT-139	605.68	2.85 (0.47%)	0.54 (0.09%)	0.73 (0.12%)	8.40 (1.39%)	1.02 (0.17%)	3.50 (0.58%)	0.47%
Opinet	321.56	2.85 (0.89%)	1.07 (0.33%)	0.76 (0.24%)	11.14 (3.46%)	1.03 (0.32%)	3.52 (1.10%)	1.06%
Uzel2022	78.88	2.85 (3.61%)	0.29 (0.37%)	0.71 (0.89%)	6.80 (8.62%)	1.02 (1.29%)	3.72 (4.71%)	3.25%
NOAA-958	O.O.M.	2.88	14.46	0.98	60.93	1.08	3.69	–

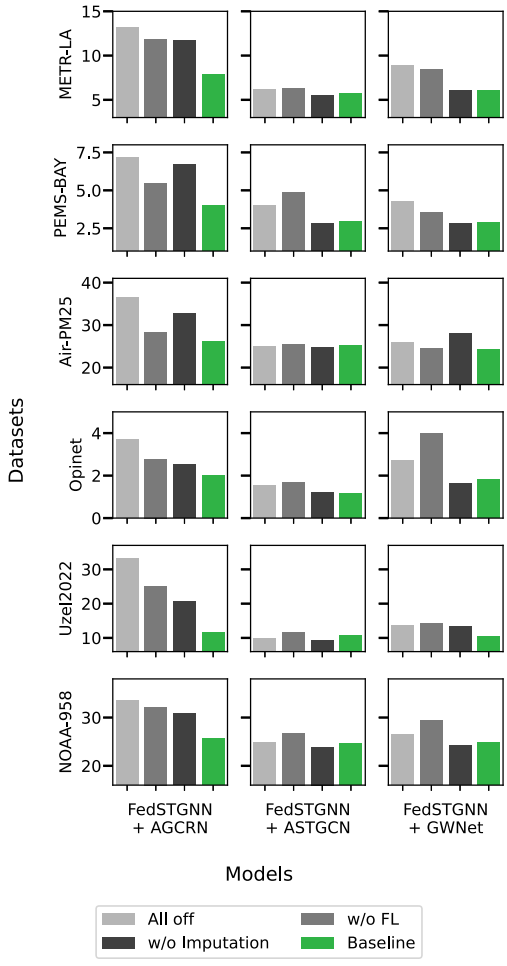

Fig. 9. Prediction error of the ablation study, measured in MAAPE (unit: %). *Baseline* is FedSTGNN with feature propagation, same as Fig. 6. *w/o* stands for *without*. *All off* is both FL and imputation are removed.

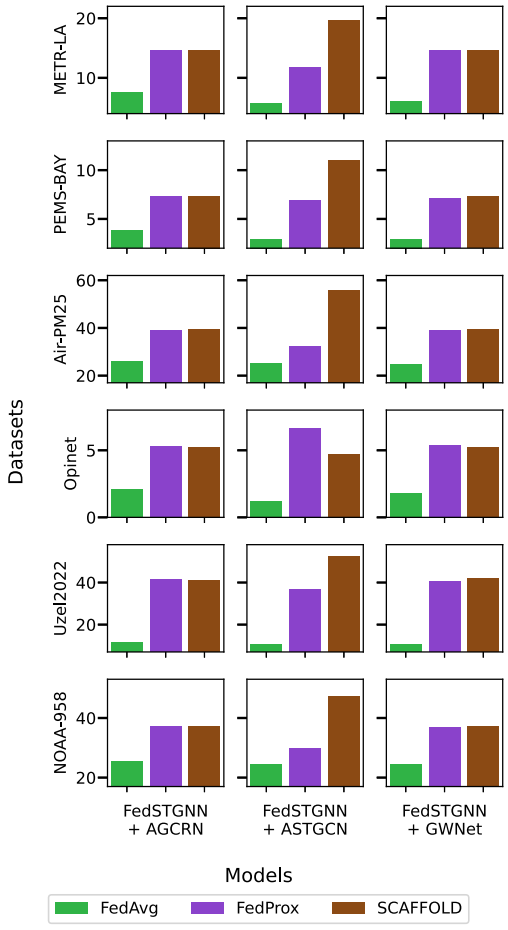

Fig. 10. Prediction error of different FL aggregation methods, measured in MAAPE (unit: %). Each bar color represents the type of aggregation method.

both FedProx and SCAFFOLD version shows significant performance degradation, which means that simple average aggregation operates more stably than weighted and complex aggregation. We presume that both the characteristics of the datasets and the processing approach of FedSTGNN can affect the results. Because sensors located in the same area are likely to output similar data. For example, in traffic prediction, if congestion occurs at one location, it is easy to think that nearby areas will also experience congestion. However, it is relatively unrelated to predicting congestion in a distant city. Similarly, in weather prediction, if the temperature at one location is very low, the temperature at an adjacent location is also likely to be low. Since FedSTGNN allocates each partial region of sensors for each edge server and sensor values are expected to show high correlation in the same region, the local models need to become *specialized* on their regions to achieve better performance. However, FedProx and SCAFFOLD try to reduce the bias of local models, which appears to make them less effective in each local region. Therefore, FedAvg, which does not heavily attempt to reduce bias, is considered the most suitable for FedSTGNN.

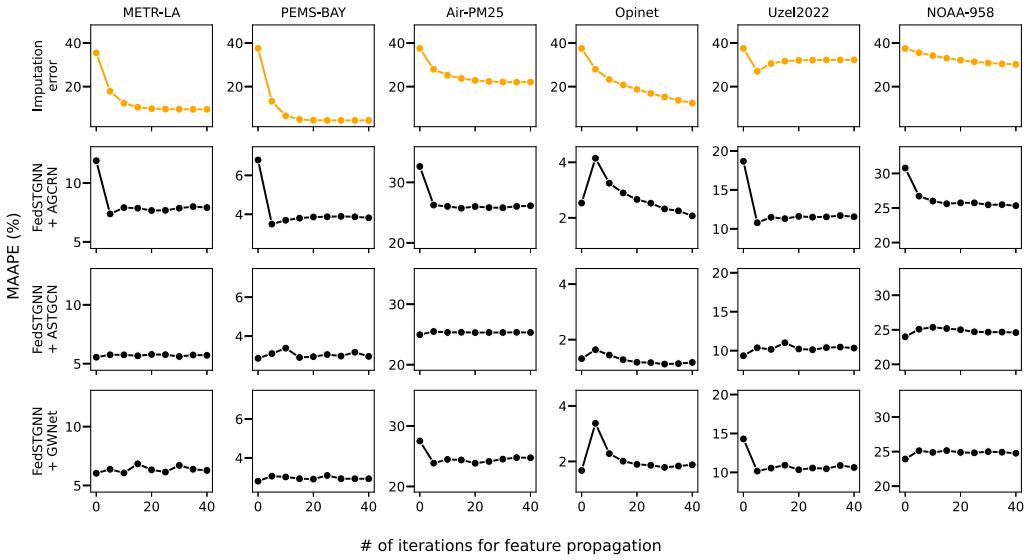

Fig. 11. Prediction error varies with the number of iterations of feature propagation and STGNN models used. Each column of plots is a dataset. The first row (yellow line plots) of plots is the average imputation error of all edge servers. The second to fourth rows (black line plots) mean the performance of FedSTGNN combined with each STGNN model. Each plot's horizontal axis is the number of feature propagation iterations, and each plot's vertical axis is error (MAAPE, unit: %).

5.6.3. Imputation strategies

As outlined in Section 4.4, the three imputation methods (zero-filling, neighbor mean, and feature propagation) are categorized by a different number of iterations of feature propagation. Zero-fill is zero iterations, neighbor mean is a single iteration, and feature propagation is multiple iterations of graph propagation. We measure imputation and prediction errors by incrementing the number of iterations from 0 to 40, showing in Fig. 11. Increasing the number of feature propagation iterations reduces imputation error across all datasets. The reason is that existing information may be multiple hops away from missing sensor data. In other words, information for reconstructing missing values can exist not only in the range of 1-hop, but also at 5-hop or 10-hop distances. For this reason, iterative propagation is required to pull the distant existing values. After all non-existing values are replaced with some value, no more values are changed, and the imputation error finally saturates. This saturation occurs in approximately 5 to 15 iterations of all datasets in Fig. 11. As imputation error decreases, missing values are replaced with values closer to actual data, and the prediction errors are reduced across most models and datasets. For example, in the Uzel2022 dataset with the AGCRN model, the error decreases significantly from approximately 18% at zero iterations to 11%–12% at 40 iterations.

We compare another graph imputation method with our proposed graph-based imputation approach. GraphMAE (Hou et al., 2022) is one of the graph imputation methods based on generative artificial intelligence design that reconstructs missing data using a modified AutoEncoder. Despite its sophisticated structure, GraphMAE performs worse than feature propagation across all datasets except NOAA-958, as shown in Fig. 12. GraphMAE also requires a separate training process involving backpropagation, which demands extra computational time. Due to the stochastic characteristics of machine learning training, machine learning-based imputation, such as GraphMAE results, may vary across each run. This instability can be amplified in the output result of FedSTGNN. In contrast, feature propagation relies on only a few matrix multiplications, which requires relatively fewer resources and produces stable imputation results for a given input. Consequently, feature propagation is more suitable for FedSTGNN in terms of accuracy and efficiency.

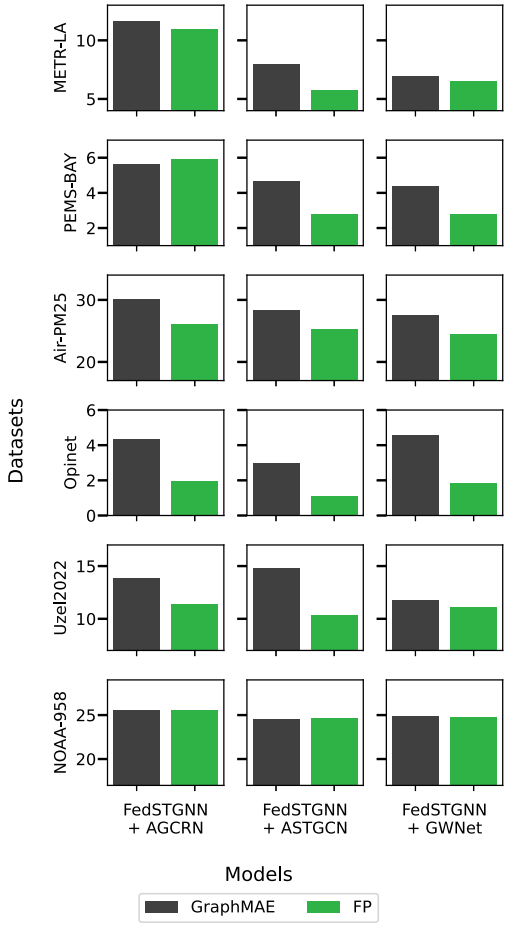

Fig. 12. Prediction error of machine learning-based method (i.e., GraphMAE) and graph propagation-based method, measured in MAAPE (unit: %). *FP* stands for feature propagation.

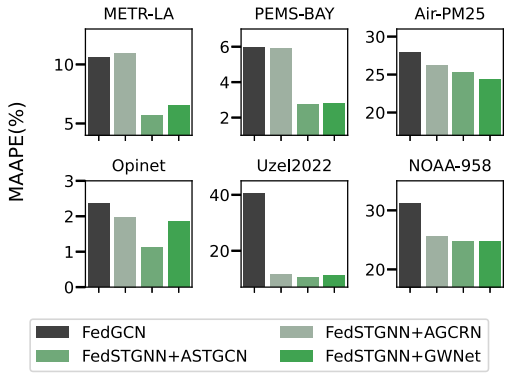

Fig. 13. Prediction error of simple GNN-based FL method (i.e., FedGCN) and STGNN-based FL method (i.e., FedSTGNN with model names), measured in MAAPE (unit: %).

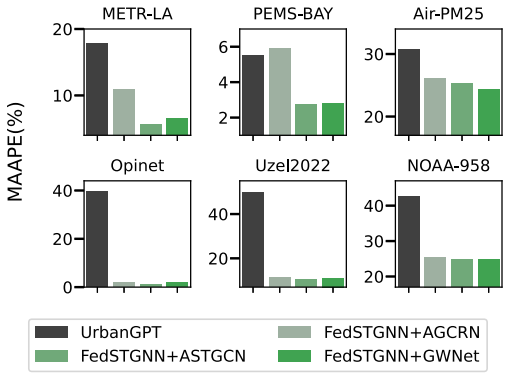

Fig. 14. Prediction error of LLM-based model (i.e., UrbanGPT) and our method, measured in MAAPE (unit: %).

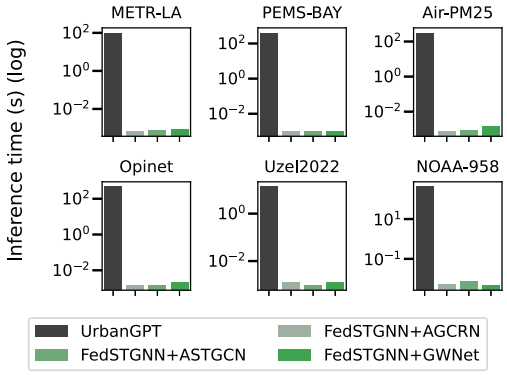

Fig. 15. Inference time of LLM-based model (i.e., UrbanGPT) and our method, measured in seconds and shown as a log scale.

5.7. Comparison with alternative methods

This section evaluates FedSTGNN against the methods following different approaches. Here, we consider two approaches: the FL approach combined with GNN, which does not explicitly target spatio-temporal data, and a large language model (LLM)-based spatio-temporal prediction approach.

We compare FedSTGNN with methods that integrate FL with typical GNNs, which are not specialized for spatio-temporal data. Specifically, we select FedGCN (Yao et al., 2023), which combines simple graph convolution with FedAvg. We used the default settings provided in the paper and its official source code. In Fig. 13, FedGCN exhibits significantly higher prediction errors than FedSTGNN across all datasets. Notably, for traffic datasets such as METR-LA and PEMS-BAY, and the neural dataset Uzel2022, FedGCN's errors are more than twice those of FedSTGNN. This performance gap seems to originate from the inability of FedGCN's simple graph convolution to capture spatio-temporal dependencies effectively. In contrast, FedSTGNN is powered by single-node STGNN models designed to detect spatio-temporal dependencies. In other words, FedSTGNN can preserve single-node STGNNs' performance well while expanding the model into FL.

We compare FedSTGNN with a recently proposed Large Language Model (LLM)-based spatio-temporal model. UrbanGPT (Li et al., 2024) integrates spatio-temporal embeddings with the Vicuna LLM model (Chiang et al., 2023) and offers the advantage of zero-shot learning that enables inference without training on new datasets. Due to limited experimental resources, we perform inference only using the pretrained UrbanGPT model. As shown in Fig. 14, UrbanGPT exhibits significantly higher prediction errors than FedSTGNN across all datasets, with huge error gaps on datasets such as Opinet and Uzel2022. We also measure the inference time required for all sensors, presented in Fig. 15. UrbanGPT typically requires hundreds of seconds for inference, whereas FedSTGNN achieves millisecond-level inference speeds. One of the reasons is that UrbanGPT cannot process multivariate time series simultaneously, but processes individual prompts on each sensor, increasing computation time, so its runtime scales poorly as the number of vertices increases. Therefore, LLM-based approaches significantly underperform compared to FedSTGNN in both prediction accuracy and computational efficiency.

5.8. Framework robustness

This section investigates the robustness of the proposed FedSTGNN framework under various challenging conditions, including sparse graph environment, long-term forecasting, and dynamic edge server participation, across multiple datasets and models.

We investigate FedSTGNN's performance in highly sparse graph environments. To control graph density, we vary the threshold value k of the Gaussian kernel defined in Eq. (1), using k= {0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0}. As shown in the first row of Fig. 16, increasing the threshold value significantly reduces graph density, especially in the Air-PM25, Opinet, and NOAA-958. For prediction performance, extremely sparse conditions can increase errors in some datasets (METR-LA, PEMS-BAY, Uzel2022) or dense conditions can also increase errors (Air-PM25, Opinet, NOAA-958). Although the impact of graph density varies depending on the dataset and the base model for FedSTGNN, performance is mainly sustained across most density levels, with a few cases of degradation in specific density cases. The proposed method can ensure consistent performance across various graph densities.

We also assess FedSTGNN's applicability for long-term forecasting by extending the output time series length from ×1 to ×5 of the original length while maintaining the input time series length. Fig. 17 shows that prediction error gradually rises as the output length increases across all cases, but the increment is typically within 5–10%p. For the NOAA-958 climate dataset, where one step equals one day, the error for a 14-step (2-week) forecast is approximately 25%, rising to about 30% for a 70-step (10-week, approximately 2-month) forecast, showing relatively limited error increase in long-term predictions. Similar patterns are discovered in other datasets. For instance, the METR-LA dataset, where one step equals 5 min, shows an error increase from approximately 5% for a 12-step (1-hour) forecast to about 10% for a 60-step (5-hour) forecast in FedSTGNN and ASTGCN combination, which is a 5%p rise. Likewise, the PEMS-BAY dataset exhibits an error increase from about 3% to 6% (+3%p). These results imply that FedSTGNN can support both short-term and long-term forecasting tasks.

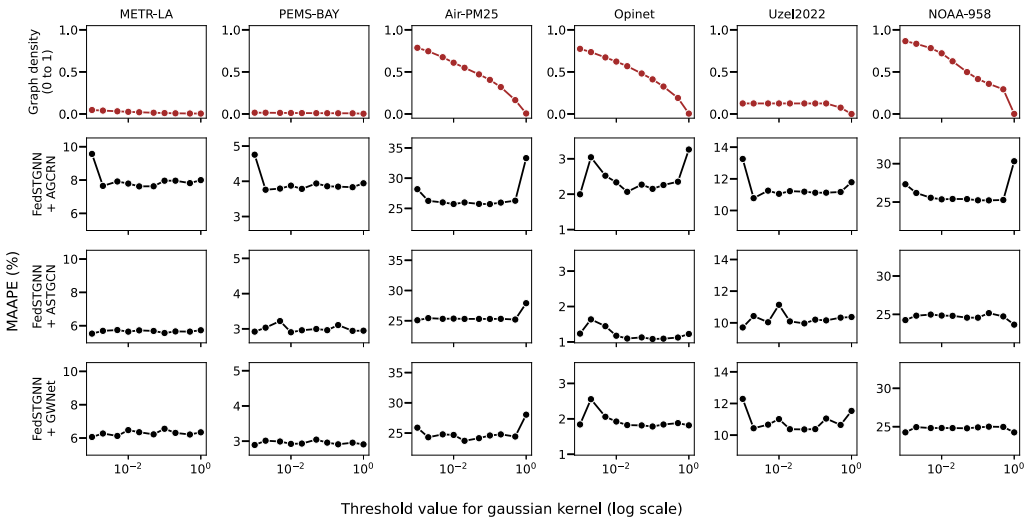

Fig. 16. Graph density and prediction error, depending on the threshold value k of the Gaussian kernel from Eq. (1). Each column of plots is a dataset. The first row (brown line plots) of plots is the graph density. The second to fourth rows (black line plots) mean the performance of FedSTGNN combined with each STGNN model. Each plot's horizontal axis is the threshold value k on a log scale, the vertical axis of plots in the first row is 0 to 1 value without unit, and the vertical axis of plots in the second to fourth rows is error (MAAPE, unit: %).

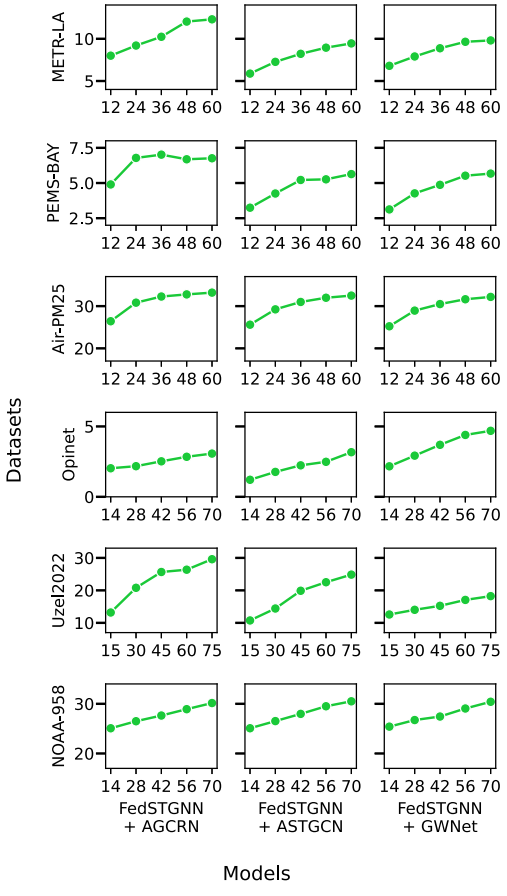

Fig. 17. Prediction error depending on the number of output timepoints, measured in MAAPE (unit: %).

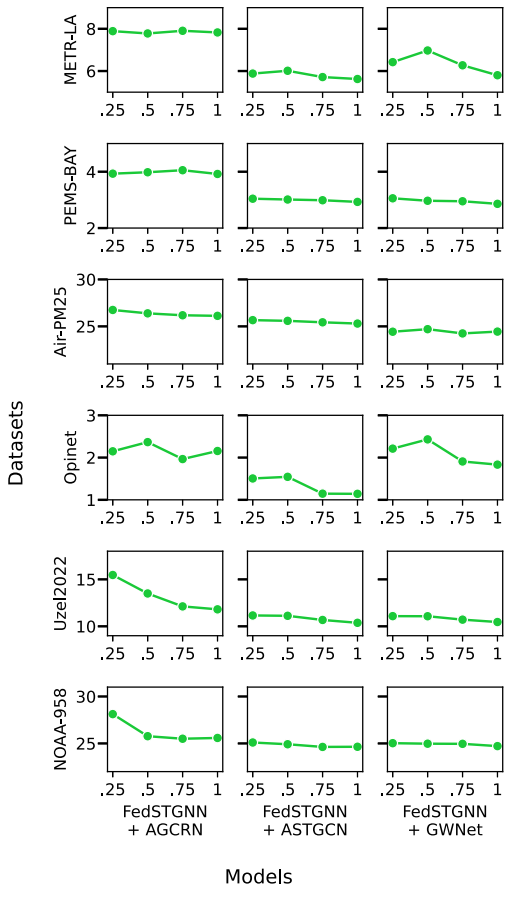

Fig. 18. Prediction error depending on the edge server's participation probabilities, measured in MAAPE (unit: %).

Finally, we evaluate FedSTGNN's robustness in dynamic environments with frequent edge server failures. To simulate this, we set the participation ratio of each edge server as 25%, 50%, 75%, and 100% for all training rounds. As shown in Fig. 18, higher participation probabilities slightly improve performance, but prediction performance remains

stable across different participation rates. We note that, FedSTGNN maintains its performance even at a 25% participation probability, which means only one out of four edge servers is active on average. The result proves that FedSTGNN is robust to edge server failure.

6. Drawbacks of FedSTGNN

Although FedSTGNN demonstrates its practical and versatile features by extending centralized STGNNs to federated learning environments, it is not universally adaptable to all types of models and scenarios that remain in future research. Our framework currently cannot handle several types of spatio-temporal graphs. For example, a multi-attribute vertex is not yet supported, such as when temperature, humidity, and illumination are jointly used in meteorological prediction. Also, heterogeneous graphs with multiple types of vertices (e.g., predicting gas price using the relationship of oil drilling facilities, refineries, and gas stations) or knowledge graphs containing words, not numerical values, fall outside the current scope. These cases require further extensions beyond the present framework.

Second, it cannot be directly applied to industry-level settings. One of the cases is that the framework does not support an edge computing environment if the base STGNN model requires a large amount of GPU memory or if edge servers have minimal resources, such as mobile or embedded systems. Furthermore, the current framework does not incorporate online graph update algorithms or dynamic data structures if sensors are frequently added or removed, such as in a real-time scenario. However, this type of optimization is relatively close to the platform-specific problems, as the solution depends on specific hardware and network specifications.

Third, privacy preservation is another drawback. Our work primarily targets converting centralized STGNNs into federated learning, and therefore is merely concerned about privacy. However, we can consider some security techniques for federated learning, such as differential privacy (Wei et al., 2020) or secure aggregation (Bonawitz et al., 2016). Since these techniques are well-established and compatible, we expect that integrating them would not be difficult, and we plan to investigate this direction in future research. Addressing privacy concerns will further enhance the applicability of FedSTGNN in sensitive domains such as healthcare and smart city infrastructures.

7. Conclusion

We proposed FedSTGNN, a federated learning (FL) framework for arbitrary STGNNs. We introduced the basic concepts of FL and outlined the general training process of STGNNs regardless of the specific model. These concepts were combined into FedSTGNN, enabling the extension of arbitrary STGNN models to an FL setting. In order to address missing sensor values in the edge-computing environment, our framework adapted graph-based imputation methods according to the range of vertices for aggregating values. We demonstrated the effectiveness of our framework in several aspects. Our analysis shows that FedSTGNN sacrifices a slight amount of accuracy while achieving substantial gains in time and network efficiency. Although our framework is designed as a general approach, it can suppress the prediction error increase to an average of $\times 1.24$ and $\times 1.25$ compared to single-node models and a competing FL-specific model, respectively. Regarding efficiency, FedSTGNN required only 42% of the training time and utilized merely 0.74% of the network bandwidth compared to the baseline model. Ablation studies confirmed that both FL and graph-based imputation are critical for performance. Our graph-based imputation significantly reduced prediction errors, outperforming non-graph-based methods. We demonstrated superior performance compared to the LLM-based prediction method. The framework also proved robust in challenging scenarios, maintaining stable performance with sparse graphs for long-term forecasting and under dynamic server participation. Consequently, the results indicate that FedSTGNN is a practical, robust, and universal solution for extending existing single-node STGNN models to real-world federated learning environments.

CRediT authorship contribution statement

Heeyong Yoon: Writing – review & editing, Writing – original draft, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Kang-Wook Chon:** Writing – review & editing, Writing – original draft, Supervision, Funding acquisition, Formal analysis, Conceptualization. **Min-Soo Kim:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

A preliminary version of this paper has been presented in the Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC), held in Jeju, Korea, in October 2024. We have significantly extended this version with new techniques, algorithms, and extensive experiments.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00347471), the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01267, GPU-based Ultrafast Multi-type Graph Database Engine SW)), and the National Research Council of Science & Technology (NST) grant by the Korea government (MSIT) (No. GTL24031-000).

Data availability

Data will be made available on request.

References

Bai, L., Yao, L., Li, C., Wang, X., Wang, C., 2020. Adaptive graph convolutional recurrent network for traffic forecasting. Adv. Neural Inf. Process. Syst. 33, 17804–17815.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K., 2016. Practical secure aggregation for federated learning on user-held data. arXiv preprint arXiv:1611.04482.

California Department of Transportation, 2018. The freeway performance measurement (PeMS). Website, URL: https://pems.dot.ca.gov/.

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al., 2020. Spectral temporal graph neural network for multivariate time-series forecasting. Adv. Neural Inf. Process. Syst. 33, 17766–17778.

Chang, H., Hari, A., Mukherjee, S., Lakshman, T., 2014. Bringing the cloud to the edge. In: 2014 IEEE Conference on Computer Communications Workshops. INFOCOM WKSHPS, IEEE, pp. 346–351.

Chaolong, L., Zhen, C., Wenming, Z., Chunyan, X., Jian, Y., 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. Proc. AAAI Conf. Artif. Intell. 32 (1), http://dx.doi.org/10.1609/aaai.v32i1.12328, URL: https://ojs.aaai.org/index.php/AAAI/article/view/12328.

Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H., Zhang, Q., 2018. Edge computing in IoT-based manufacturing. IEEE Commun. Mag. 56 (9), 103–109.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J.E., et al., 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. 2, (3), p. 6, See https://vicuna.lmsys.org (Accessed 14 April 2023).

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Cirincione, G., Verma, D., 2019. Federated machine learning for multi-domain operations at the tactical edge. In: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications. vol. 11006, SPIE, pp. 29–48.

Davidson, M., Moodley, D., 2022. ST-GNNs for weather prediction in South Africa. In: Southern African Conference for Artificial Intelligence Research. Springer, pp. 93–107.

Deng, S., Wang, S., Rangwala, H., Wang, L., Ning, Y., 2020. Cola-GNN: Cross-location attention based graph neural networks for long-term ILI prediction. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. pp. 245–254.

Diao, Y., Li, Q., He, B., 2022. Towards addressing label skews in one-shot federated learning. In: The Eleventh International Conference on Learning Representations. pp. 1–22.

Dong, E., Du, H., Gardner, L., 2020. An interactive web-based dashboard to track COVID-19 in real time. Lancet Infect. Dis. 20 (5), 533–534.

Douglas, H.J., 1994. Time Series Analysis. Princeton University Press, Princeton, NJ, USA.

Fang, Z., Long, Q., Song, G., Xie, K., 2021. Spatial-temporal graph ode networks for traffic flow forecasting. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 364–373.

Guo, S., Lin, Y., Feng, N., Song, C., Wan, H., 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, (01), pp. 922–929.

Guo, S., Lin, Y., Wan, H., Li, X., Cong, G., 2021. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. IEEE Trans. Knowl. Data Eng. 34 (11), 5415–5428.

Hällman, L., 2017. The rolling window method: Precisions of financial forecasting. Dissertation.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D., 2018. Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604.

He, J., Khushi, M., Nguyen, T.A., Tran, N.H., 2023. Fed-mssa: A federated approach for spatio-temporal data modeling using multivariate singular spectrum analysis. In: 2023 IEEE International Conference on Data Mining. ICDM, IEEE, pp. 1055–1060.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Hou, Z., Liu, X., Cen, Y., Dong, Y., Yang, H., Wang, C., Tang, J., 2022. Graphmae: Self-supervised masked graph autoencoders. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 594–604.

Jiang, W., Luo, J., 2022. Graph neural network for traffic forecasting: A survey. Expert Syst. Appl. 207, 117921.

Jiang, R., Yin, D., Wang, Z., Wang, Y., Deng, J., Liu, H., Cai, Z., Deng, J., Song, X., Shibasaki, R., 2021. Dl-traff: Survey and benchmark of deep learning models for urban traffic prediction. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 4515–4525.

Jin, G., Liang, Y., Fang, Y., Huang, J., Zhang, J., Zheng, Y., 2023. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. arXiv preprint arXiv:2303.14483.

Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T., 2020. Scaffold: Stochastic controlled averaging for federated learning. In: International Conference on Machine Learning. PMLR, pp. 5132–5143.

Kim, S., Kim, H., 2016. A new metric of absolute percentage error for intermittent demand forecasts. Int. J. Forecast. 32 (3), 669–679.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Korea National Oil Corporation, 2024. Opinet. Website, URL: https://www.opinet.co.kr/.

Kreuzberger, D., Kühl, N., Hirschl, S., 2023. Machine learning operations (mlops): Overview, definition, and architecture. IEEE Access.

Lai, Q., Chen, P., 2024. LEISN: A long explicit–implicit spatio-temporal network for traffic flow forecasting. Expert Syst. Appl. 245, 123139. http://dx.doi.org/10.1016/j.eswa.2024.123139, URL: https://www.sciencedirect.com/science/article/pii/S0957417424000046.

Lan, S., Ma, Y., Huang, W., Wang, W., Yang, H., Li, P., 2022. Dstagnn: Dynamic spatial-temporal aware graph neural network for traffic flow forecasting. In: International Conference on Machine Learning. PMLR, pp. 11906–11917.

Leetaru, K., Schrodt, P.A., 2013. Gdelt: Global data on events, location, and tone, 1979–2012. In: ISA Annual Convention. vol. 2, (4), Citeseer, pp. 1–49.

Li, F., Feng, J., Yan, H., Jin, G., Yang, F., Sun, F., Jin, D., Li, Y., 2023. Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. ACM Trans. Knowl. Discov. Data 17 (1), 1–21.

Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V., 2020. Federated optimization in heterogeneous networks. Proc. Mach. Learn. Syst. 2, 429–450.

Li, Z., Xia, L., Tang, J., Xu, Y., Shi, L., Xia, L., Yin, D., Huang, C., 2024. Urbangpt: Spatio-temporal large language models. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 5351–5362.

Li, Y., Yu, R., Shahabi, C., Liu, Y., 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. URL: https://openreview.net/forum?id=SJiHXGWAZ.

Liang, Y., Ke, S., Zhang, J., Yi, X., Zheng, Y., 2018. Geoman: Multi-level attention networks for geo-sensory time series prediction. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, pp. 3428–3434. http://dx.doi.org/10.24963/ijcai.2018/476.

Liu, S., Liu, L., Tang, J., Yu, B., Wang, Y., Shi, W., 2019. Edge computing for autonomous driving: Opportunities and challenges. Proc. IEEE 107 (8), 1697–1716.

Liu, Q., Sun, S., Liang, Y., Liu, M., Xue, J., 2025. Personalized federated learning for spatio-temporal forecasting: A dual semantic alignment-based contrastive approach. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, (11), pp. 12192–12200.

Liu, Q., Sun, S., Liang, Y., Xu, X., Liu, M., Bilal, M., Wang, Y., Li, X., Zheng, Y., 2024a. REFOL: Resource-efficient federated online learning for traffic flow forecasting. IEEE Trans. Intell. Transp. Syst..

Liu, Q., Sun, S., Liu, M., Wang, Y., Gao, B., 2024b. Online spatio-temporal correlation-based federated learning for traffic flow forecasting. IEEE Trans. Intell. Transp. Syst..

Long, G., Tan, Y., Jiang, J., Zhang, C., 2020. Federated learning for open banking. In: Federated Learning: Privacy and Incentive. Springer, pp. 240–254.

Luo, X., Zhu, C., Zhang, D., Li, Q., 2023. STG4traffic: A survey and benchmark of spatial-temporal graph neural networks for traffic prediction. arXiv preprint arXiv:2307.00495.

Mao, J., Lin, H., Tian, X., Pan, Y., Liu, J., 2023. Fedgst: Federated graph spatio-temporal framework for brain functional disease prediction. In: 2023 IEEE International Conference on Bioinformatics and Biomedicine. BIBM, IEEE, pp. 1356–1361.

McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A., 2017. Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. PMLR, pp. 1273–1282.

Mehmood, M.Y., Oad, A., Abrar, M., Munir, H.M., Hasan, S.F., Muqeet, H.A.u., Golilarz, N.A., 2021. Edge computing for IoT-enabled smart grid. Secur. Commun. Netw. 2021 (1), 5524025.

Meng, C., Rambhatla, S., Liu, Y., 2021. Cross-node federated graph neural network for spatio-temporal data modeling. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 1202–1211.

Mohri, M., Sivek, G., Suresh, A.T., 2019. Agnostic federated learning. In: International Conference on Machine Learning. PMLR, pp. 4615–4625.

Moritz, S., Sardá, A., Bartz-Beielstein, T., Zaefferer, M., Stork, J., 2015. Comparison of different methods for univariate time series imputation in r. arXiv preprint arXiv:1510.03924.

Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Ray, P.P., Dash, D., De, D., 2019. Edge computing for internet of things: A survey, e-healthcare case study and future direction. J. Netw. Comput. Appl. 140, 1–22.

Rossi, E., Kenlay, H., Gorinova, M.I., Chamberlain, B.P., Dong, X., Bronstein, M.M., 2022. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. In: Learning on Graphs Conference. 198, PMLR, pp. 11:1–11:16.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1985. Learning Internal Representations by Error Propagation. Institute for Cognitive Science, University of California, San Diego, LA, USA.

Sahili, Z.A., Awad, M., 2023. Spatio-temporal graph neural networks: A survey. arXiv preprint arXiv:2301.10569.

Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50 (1), 30–39.

Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Trans. Neural Netw. 20 (1), 61–80.

Smith, A., Lott, N., Vose, R., 2011. The integrated surface database: Recent developments and partnerships. Bull. Am. Meteorol. Soc. 92 (6), 704–708.

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks. Adv. Neural Inf. Process. Syst. 27.

Thapa, C., Arachchige, P.C.M., Camtepe, S., Sun, L., 2022. Splitfed: When federated learning meets split learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, (8), pp. 8485–8493.

Tian, Y., Luo, J., Liu, Z., Li, S., Qu, Y., 2023. M3FGM: A node masking and multi-granularity message passing-based federated graph model for spatial-temporal data prediction. In: International Conference on Neural Information Processing. Springer, pp. 551–566.

Uzel, K., Kato, S., Zimmer, M., 2022. A set of hub neurons and non-local connectivity features support global brain dynamics in c. elegans. Curr. Biol. 32 (16), 3443–3459.

Vaid, A., Jaladanki, S.K., Xu, J., Teng, S., Kumar, A., Lee, S., Somani, S., Paranjpe, I., De Freitas, J.K., Wanyan, T., et al., 2020. Federated learning of electronic health records improves mortality prediction in patients hospitalized with COVID-19. MedRxiv.

Vaswani, A., 2017. Attention is all you need. Adv. Neural Inf. Process. Syst..

Wang, S., Li, X., Liao, G., Liu, J., Liao, C., Liu, M., Liao, J., Liu, L., 2024. A spatio-temporal graph neural network for fall prediction with inertial sensors. Knowl.-Based Syst. 293, 111709. http://dx.doi.org/10.1016/j.knosys.2024.111709, URL: https://www.sciencedirect.com/science/article/pii/S0950705124003447.

Wang, T., Xu, N., Chen, K., Lin, W., 2021. End-to-end video instance segmentation via spatial-temporal graph neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10797–10806.

Wei, K., Li, J., Ding, M., Ma, C., Yang, H.H., Farokhi, F., Jin, S., Quek, T.Q., Poor, H.V., 2020. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Trans. Inf. Forensics Secur. 15, 3454–3469.

Wen, J., Zhang, Z., Lan, Y., Cui, Z., Cai, J., Zhang, W., 2023. A survey on federated learning: challenges and applications. Int. J. Mach. Learn. Cybern. 14 (2), 513–535.

Wikipedia contributors, 2024. Stochastic matrix — Wikipedia, the free encyclopedia. Online, https://en.wikipedia.org/w/index.php?title=Stochastic_matrix&oldid=1259743512, (Accessed 10 December 2024).

Williams, B.M., Hoel, L.A., 2003. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. J. Transp. Eng. 129 (6), 664–672.

Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C., 2019. Graph wavenet for deep spatial-temporal graph modeling. arXiv preprint arXiv:1906.00121.

Xia, Z., Zhang, Y., Yang, J., Xie, L., 2024. Dynamic spatial–temporal graph convolutional recurrent networks for traffic flow forecasting. Expert Syst. Appl. 240, 122381. http://dx.doi.org/10.1016/j.eswa.2023.122381, URL: https://www.sciencedirect.com/science/article/pii/S095741742302883X.

Yang, L., Chen, W., He, X., Wei, S., Xu, Y., Zhou, Z., Tong, Y., 2024. Fedgtp: exploiting inter-client spatial dependency in federated graph-based traffic prediction. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 6105–6116.

Yao, Y., Jin, W., Ravi, S., Joe-Wong, C., 2023. FedGCN: Convergence-communication tradeoffs in federated training of graph convolutional networks. Adv. Neural Inf. Process. Syst. 36, 79748–79760.

Yu, B., Yin, H., Zhu, Z., 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875.

Yu, B., Yin, H., Zhu, Z., 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, pp. 3634–3640. http://dx.doi.org/10.24963/ijcai.2018/505.

Yuan, X., Chen, J., Yang, J., Zhang, N., Yang, T., Han, T., Taherkordi, A., 2022. Fedstn: Graph representation driven federated learning for edge computing enabled urban traffic flow prediction. IEEE Trans. Intell. Transp. Syst. 24 (8), 8738–8748.

Zhang, Q., Chang, J., Meng, G., Xiang, S., Pan, C., 2020. Spatio-temporal graph structure learning for traffic forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, (01), pp. 1177–1185.

Zhang, T., Liu, Y., Shen, Z., Xu, R., Chen, X., Huang, X., Zheng, X., 2023. An adaptive federated relevance framework for spatial–temporal graph learning. IEEE Trans. Artif. Intell. 5 (5), 2227–2240.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., Li, H., 2019. T-gcn: A temporal graph convolutional network for traffic prediction. IEEE Trans. Intell. Transp. Syst. 21 (9), 3848–3858.

Zheng, Y., Yi, X., Li, M., Li, R., Shan, Z., Chang, E., Li, T., 2015. Forecasting fine-grained air quality based on big data. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 2267–2276.