Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

FedSTGCN: a novel federated spatiotemporal graph learning-based network intrusion detection method for the Internet of Things#

Yalu WANG[1], Jie LI[2], Zhijie HAN‡[3], Pu CHENG[3], Roshan KUMAR[4]

[1]*School of Computer and Information Engineering, Henan University, Kaifeng 475004, China*
[2]*School of Computer Science, Zhengzhou University of Aeronautics, Zhengzhou 450046, China*
[3]*School of Software, Henan University, Kaifeng 475004, China*
[4]*Miami College of Henan University, Kaifeng 475004, China*

E-mail: 104752200075@henu.edu.cn; jsjt9@henu.edu.cn; hanzj@henu.edu.cn; chengpul@henu.edu.cn; roshan.iit123@gmail.com

Received Oct. 20, 2024; Revision accepted Mar. 25, 2025; Crosschecked June 27, 2025

Abstract: The rapid growth and increasing complexity of Internet of Things (IoT) devices have made network intrusion detection a critical challenge, especially in edge computing environments where data privacy is a primary concern. Machine learning-based intrusion detection techniques enhance IoT network security but often require centralized network data, posing significant risks to data privacy and security. Although federated learning (FL)-based network intrusion detection methods have emerged in recent years to address privacy concerns, they have not fully leveraged the advantages of graph neural networks (GNNs) for intrusion detection. To address this issue, we propose a federated spatiotemporal graph convolutional network (FedSTGCN) model, which integrates the capabilities of spatiotemporal GNNs (STGNNs) and federated learning. This framework enables collaborative model training across distributed IoT devices without requiring the sharing of raw data, thereby improving network intrusion detection accuracy while preserving data privacy. Extensive experiments are conducted on two widely used IoT intrusion detection datasets to evaluate the effectiveness of the proposed approach. The results demonstrate that FedSTGCN outperforms other methods in both binary and multiclass classification tasks, achieving over 97% accuracy in binary classification tasks and over 92% weighted F1-score in multiclass classification tasks.

Key words: Internet of Things (IoT); Network intrusion detection; Spatiotemporal graph neural network (STGNN); Federated learning (FL); Data privacy
https://doi.org/10.1631/FITEE.2400932 **CLC number:** TP39

1 Introduction

With the rapid growth of the number and complexity of Internet of Things (IoT) devices, network intrusion detection has become increasingly important, particularly for edge computing devices. Devices in edge networks such as smart televisions, network cameras, smart curtains, and wireless sensors, which are highly convenient for both life and work, have become potential targets for cyberattacks (Ghasempour, 2019). With the exponential increase in the number and frequency of attacks, ensuring the security of these smart devices has become a key issue driving the continuous development of the IoT.

To ensure smart device security, researchers have proposed various network intrusion detection methods.

‡ Corresponding author
# Electronic supplementary materials: The online version of this article (https://doi.org/10.1631/FITEE.2400932) contains supplementary materials, which are available to authorized users
ORCID: Yalu WANG, https://orcid.org/0000-0003-3823-6230; Zhijie HAN, https://orcid.org/0000-0002-7362-7520

Among them, machine learning-based intrusion detection algorithms are particularly popular because they can automatically learn and identify potential threat patterns by analyzing data flows. In recent years, a subfield of deep learning—graph neural network (GNN)—has been applied to network intrusion detection due to its advantages in learning from graph-structured data (Caville et al., 2022; Lo et al., 2022; Zhou et al., 2022; Hu et al., 2023). Although GNN-based intrusion detection methods address some challenges in network intrusion detection, they rely on centralized deep learning approaches. To train a generalized network intrusion detection model, raw data must be transmitted from edge devices to a central server for training, which exposes the data to potential attacks and raises privacy concerns. Therefore, this paper integrates spatiotemporal GNNs (STGNNs) with federated learning (FL) and proposes a federated STGNN (FSTGNN) model. When applied to network intrusion detection, this approach not only improves detection accuracy but also preserves data privacy.

Two key challenges must be overcome to apply the FSTGNN to network intrusion detection systems. The first challenge is how to construct local subgraphs from the network intrusion detection datasets on each client and then build a global graph across all clients. The second challenge is guaranteeing that the aggregation process maintains data privacy, since GNNs are used as the training model for each client in the FSTGNN architecture, which necessitates collecting information from surrounding nodes for training GNNs. To tackle the first challenge, this paper uses the graph construction method in N-STGAT proposed by Wang et al. (2023) and presents an improved approach for constructing a global graph from subgraphs. For the second challenge, this paper employs the graph convolutional aggregation method mentioned in FedCog (Lei et al., 2023) to prevent privacy leakage during the aggregation process.

This paper's contributions to the field of network intrusion detection are as follows:

1. To the best of our knowledge, this is the first time that a federated spatiotemporal graph learning model has been used to address the problem of network intrusion detection. This is also the first attempt to perform distributed training using a spatiotemporal graph model.

2. A method for constructing edges between subgraphs on different clients is proposed, ensuring both proper edge construction and the protection of data privacy.

3. Experiments were conducted on the latest datasets, and the proposed method was compared with several recent FL-based approaches. The results demonstrate that the proposed method outperforms other methods in all respects.

2 Related work

In recent years, many scholars have contributed significantly to intrusion detection systems, proposing numerous methods for detecting network attacks. These methods can be categorized primarily as traditional machine learning, GNNs, and federated graph learning (FGL).

2.1 Traditional machine learning

Li XM et al. (2023) designed a tensor train support vector machine (TT-SVM) model based on dimensionality reduction data processing, which can comprehensively optimize the big data management (BDM) platform and has been compared and evaluated against other models. Aburomman and Reaz (2016) used an SVM and *k*-nearest neighbors (kNN) to train their model but created an ensemble classifier with better accuracy using weights generated by particle swarm optimization for intrusion detection. Almogren (2020) proposed a deep belief network (DBN) based on advanced intrusion detection methods, which addressed the issue of intrusion detection in distributed environments. In early applications of machine learning to network intrusion detection, Sangkatsanee et al. (2011) applied decision tree techniques, reporting good results with their designed real-time network intrusion detection system (RT-IDS). Dina et al. (2023) proposed a focal loss function to address the issue of data imbalance and applied it to two well-known machine learning methods (feed-forward neural networks (FNNs) and convolutional neural networks (CNNs)).

2.2 Graph neural networks

Since GNNs primarily learn from graph structures, they have a natural advantage when handling inherently graph-based network systems. Researchers have

recently applied GNNs to intrusion detection systems. Lo et al. (2022) proposed the E-GraphSAGE architecture based on graph sample and aggregate (GraphSAGE), which can capture edge features and topological information of graphs for IoT network intrusion detection. This approach has provided a solid foundation for subsequent researchers to apply GNNs to network intrusion detection. Caville et al. (2022) introduced a self-supervised network intrusion detection method called Anomal-E, which was built based on E-GraphSAGE. The authors claimed that this method was the first successful and practical network intrusion detection approach. Considering the challenges of limited and imbalanced attack data in existing intrusion detection methods, Zhou et al. (2022) proposed a novel hierarchical adversarial attack (HAA) generation method to realize the level-aware black-box adversarial attack strategy, targeting the GNN-based intrusion detection in IoT systems with a limited budget. To address the complexity of existing graph-based intrusion detection methods, which require large amounts of training data, Hu et al. (2023) proposed an early and accurate network intrusion detection method based on graph embedding techniques, named Graph2vec+RF.

2.2 Federated graph learning

Recently, researchers have integrated GNNs with FL, applying FGL to network intrusion detection. To address the issues such as label dependency, detection delay, privacy leakage, and imbalanced attack scenario distribution, Mao et al. (2025) proposed a method called FeCoGraph. It is a label-aware federated graph contrastive learning framework designed for intrusion detection in limited scenarios, and its performance has been validated through experiments. Wu et al. (2024) proposed an attention-based graph neural network for detecting cross-level and cross-department network attacks. This method not only supports collaborative model training but also protects data privacy on distributed devices. To tackle model dependency and privacy concerns, Mansour Bahar et al. (2024) integrated FL into a GNN-based intrusion detection system architecture; compared with state-of-the-art provenance-based intrusion detection system (PIDS), this approach has shown promising results in reducing false positive rates.

Although these FL-based approaches incorporate GNNs, they consider only the spatial characteristics of network attacks. In reality, network attacks exhibit temporal characteristics, as data flows within the same time period tend to have similar properties. Therefore, considering temporal characteristics in network intrusion detection is essential.

3 Background

Since this paper adopts federated spatiotemporal graph learning to address the network intrusion problem, it uses two core methods: N-STGAT and FedCog (an FL model based on a graph convolutional network (GCN)). Therefore, it is necessary to provide a brief introduction to N-STGAT and FL.

3.1 N-STGAT

N-STGAT is a centralized network intrusion detection method based on GNNs, which is part of our previous research (Wang et al., 2023). The previous research compared the different states of nodes during abnormal and normal traffic, as well as the temporal relationships in detecting abnormal traffic. The method combines graph attention network (GAT, a derivative of GNNs) (Veličković et al., 2018) with long short-term memory (LSTM) for model training. The model's structure is depicted in Fig. 1. The main process involves first using GAT to aggregate neighbor information for the current node on the generated graph, producing node embeddings of data flows incorporating temporal information. Then, LSTM is used to perform the final classification based on these node embeddings. This paper uses N-STGAT extensively, particularly in the graph construction process. Interested readers are encouraged to refer to Wang et al. (2023) for further details.

3.2 Federated learning

FL is a decentralized machine learning method designed to train models across multiple devices or servers without transferring the original data to a central location. This approach helps address data privacy and security concerns while reducing the need for data transmission. In recent years, FL has been used predominantly in fields such as the IoT (Nguyen et al., 2021),

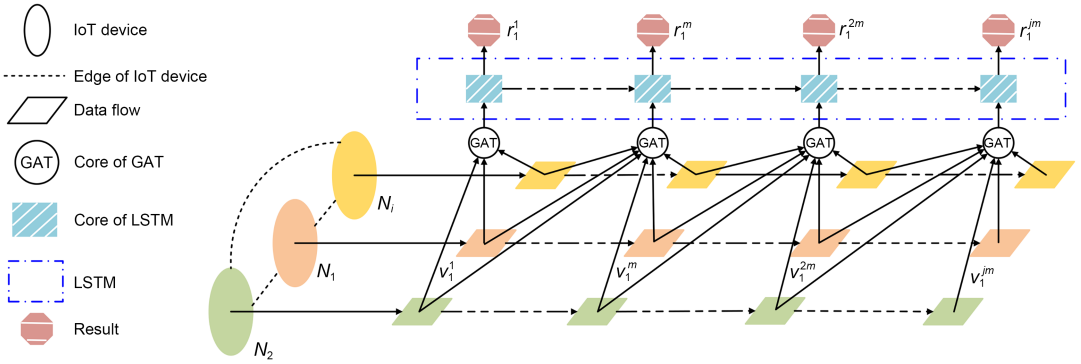

Fig. 1 Model structure of N-STGAT. N_1 represents the data sending node 1, v_1^m represents the m^{th} data flow of node 1, and r_1^m represents the attack type of the m^{th} data flow of node 1. Reprinted from Wang et al. (2023), Copyright 2023, with permission from the authors, licensed under CC BY 4.0

healthcare (Nguyen et al., 2022), finance (Chatterjee et al., 2024), mobile devices, and edge computing (Lim et al., 2020). FL allows clients to collaboratively train a shared model while keeping personal data on their devices; its typical architecture is illustrated in Fig. 2. FL has various categories, such as vertical FL and horizontal FL; interested readers can refer to Yang et al. (2019).

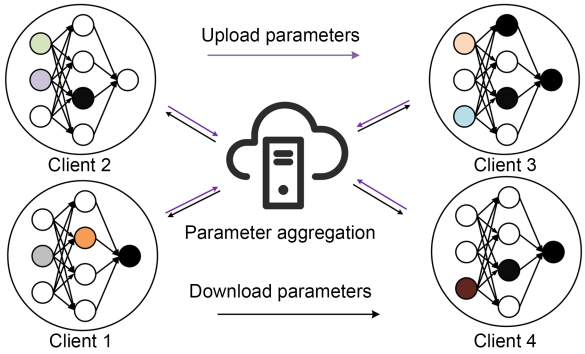

Fig. 2 FL architecture diagram. References to color refer to the online version of this figure

4 Method description

4.1 Problem definition

To protect data privacy while training a high-accuracy model that covers a wide range of attack types, it is necessary to abstractly define the real-world environment. The results are shown in Fig. 3. The problem studied in this paper can be defined from the structure shown in Fig. 3. Let there be a set of clients $C=\{C_1, C_2, C_3, \cdots, C_m\}$, each with n IoT nodes, where each client has a private dataset $d_i \in D$ related to network intrusion detection, which is not shared among the clients. Within each client, all data flows are sorted according to nodes and time series, defined as $u_i^{jq}(i\in\{1, 2, \cdots, m\}, q\in\{1, 2, \cdots, \infty\}, j\in\{1, 2, \cdots, n\})$. The goal of this paper is to use deep learning methods to train a network intrusion detection model to identify the attack types r_i^{jq} in the data flow u_i^{jq}. To protect the data privacy of each client, this paper employs an FGL approach to achieve this objective.

4.2 Federated graph construction

The structure of the FSTGNN used to train classification models is shown in Fig. 4, from which it can be seen that the first step involves generating a local graph G_i^1 on client C_i and creating edge e between G_i^1 (represented by black edges in the global graph of Fig. 4). This forms the global graph G used for training, as illustrated in Fig. 5. Then, each client trains its own data using STGNN and aggregates the parameters from each client on the FL server. These two processes present two challenges: first, there is a risk of privacy leakage when generating edge e between local graphs; second, during STGNN training, the need to aggregate neighbor node information poses privacy leakage concerns. Next, we discuss how to address the first issue.

Fig. 5 shows that constructing edges within a local graph is relatively straightforward and can be easily achieved using the methods provided by N-STGAT. However, when generating edge e between local graphs, using the methods provided by N-STGAT may lead to privacy leakage issues. This is because

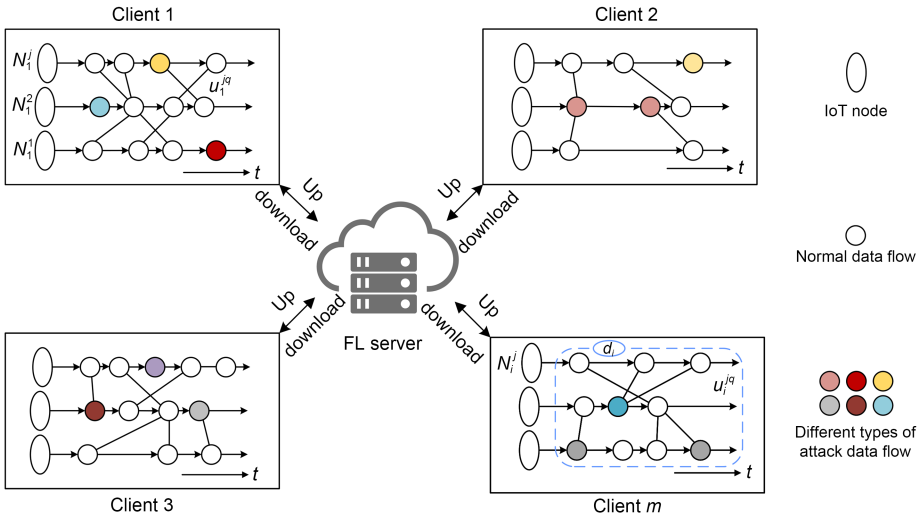

Fig. 3 Structure of the problem definition. N_i^j represents the data sending node, and u_i^{jq} represents data flow. References to color refer to the online version of this figure

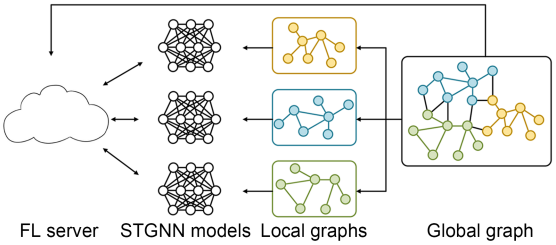

Fig. 4 General structure of FL on distributed graph data (Lei et al., 2023). The FL server has a global graph, while each client has a local graph, which is a subgraph of the global graph. References to color refer to the online version of this figure

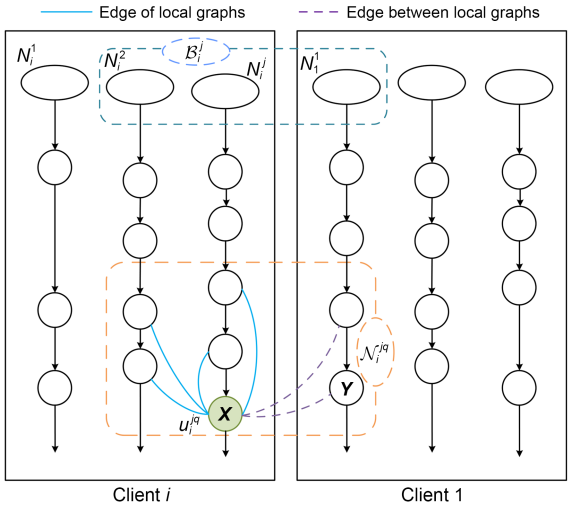

Fig. 5 Results of the graph construction. $\mathcal{B}_i^j$ represents the neighbors of the j^{th} node N_i^j in client i, $\mathcal{N}_i^{jq}$ represents the neighbors of the j^{th} data flow u_i^{jq} of node N_i^j, and X and Y represent data flows

N-STGAT employs cosine similarity measurement to obtain neighbor nodes $\mathcal{B}_i$; directly using cosine similarity measurement requires the other party to know the current node's information, which is not permissible in FL. Therefore, it is necessary to restructure the cosine similarity measurement to ensure that the cosine value can still be calculated while preserving privacy.

The cosine similarity formula can be decomposed into three parts—A, B, and C—as shown in Eq. (1). The formula shows that as long as A, B, and C can be computed separately, the final cosine similarity can be determined.

$$\begin{aligned}\cos(\theta) &= \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}},\\ A &= \sqrt{\sum_{i=1}^{n} x_i^2},\\ B &= \sqrt{\sum_{i=1}^{n} y_i^2},\\ C &= \sum_{i=1}^{n} x_i y_i.\end{aligned} \tag{1}$$

Assuming that two data flows belonging to different clients are $\boldsymbol{X}$ and $\boldsymbol{Y}$, with feature values $\boldsymbol{X}=(x_1, x_2, \cdots, x_n)$ and $\boldsymbol{Y}=(y_1, y_2, \cdots, y_n)$, the process of

cosine similarity measurement can be divided into the following six steps:

Step 1: data flow $\boldsymbol{X}$ calculates A using formula $\sqrt{\sum_{i=1}^{n} x_i^2}$, while data flow $\boldsymbol{Y}$ calculates B using formula $\sqrt{\sum_{i=1}^{n} y_i^2}$. Then, data flow $\boldsymbol{Y}$ sends B to data flow $\boldsymbol{X}$.

Step 2: data flow $\boldsymbol{X}$ generates a random number $\boldsymbol{P}=(p_1, p_2, \cdots, p_n)$ and sends it to the FL server.

Step 3: data flow $\boldsymbol{X}$ generates an intermediate state $\mathbf{XP}=(\mathrm{xp}_1, \mathrm{xp}_2, \cdots, \mathrm{xp}_n)$ using formula $\mathrm{xp}_i=x_i p_i$ and sends **XP** to data flow $\boldsymbol{Y}$.

Step 4: data flow $\boldsymbol{Y}$ generates an intermediate state $\mathbf{XPY}=(\mathrm{xpy}_1, \mathrm{xpy}_2, \cdots, \mathrm{xpy}_n)$ using formula $\mathrm{xpy}_i=\mathrm{xp}_i y_i$ and sends **XPY** to the FL server.

Step 5: the FL server obtains C using formula $\sum_{i=1}^{n}(\mathrm{xpy}_i/p_i)$ and sends C to data flow $\boldsymbol{X}$.

Step 6: data flow $\boldsymbol{X}$ calculates the cosine value $\cos(XY)=\frac{C}{AB}$.

To clarify the steps mentioned above, a process flow chart is provided in Fig. 6. Furthermore, to emphasize the details of privacy protection, all fields must be normalized before cosine similarity measurement using the aforementioned steps. This prevents extreme values of one party from being used by the other party to infer the range of its field values. The specific normalization method involves finding the maximum value $\boldsymbol{X}^{\max}$ and minimum value $\boldsymbol{X}^{\min}$ of the feature values for all data flows within the client of data flow $\boldsymbol{X}$, as well as the maximum value $\boldsymbol{Y}^{\max}$ and minimum value $\boldsymbol{Y}^{\min}$ of the feature values for all data flows within the client of data flow $\boldsymbol{Y}$. Their specific values are shown in Eq. (2).

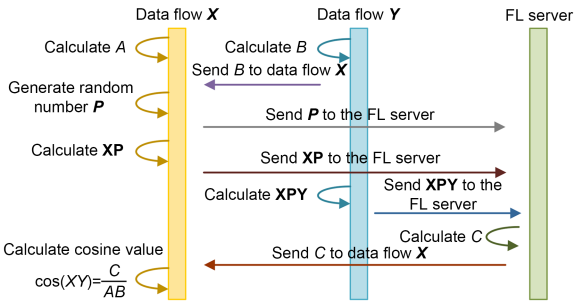

Fig. 6 Process steps for generating edges between subgraphs. References to color refer to the online version of this figure

$$
\begin{aligned}
&\boldsymbol{X}^{\max}=\left(x_1^{\max}, x_2^{\max}, \cdots, x_n^{\max}\right),\\
&\boldsymbol{X}^{\min}=\left(x_1^{\min}, x_2^{\min}, \cdots, x_n^{\min}\right),\\
&\boldsymbol{Y}^{\max}=\left(y_1^{\max}, y_2^{\max}, \cdots, y_n^{\max}\right),\\
&\boldsymbol{Y}^{\min}=\left(y_1^{\min}, y_2^{\min}, \cdots, y_n^{\min}\right),\\
&\mathbf{XY}_i^{\max}=\max\left(x_i^{\max}, y_i^{\max}\right), i\in\{1, 2, \cdots, n\},\\
&\mathbf{XY}_i^{\min}=\min\left(x_i^{\min}, y_i^{\min}\right), i\in\{1, 2, \cdots, n\}.
\end{aligned}
\tag{2}
$$

Here, $x_i^{\max}$ and $x_i^{\min}$ represent the maximum and minimum values of the i^{th} dimension feature for all nodes of the client to which dataflow $\boldsymbol{X}$ belongs, respectively. Similarly, $y_i^{\max}$ and $y_i^{\min}$ represent the maximum and minimum values of the i^{th} dimension feature for all nodes of the client to which data flow $\boldsymbol{Y}$ belongs. With these parameters for normalization, we can easily normalize the nodes of each client without affecting cosine similarity measurement while effectively preserving privacy.

In addition, to ensure the privacy of the cosine similarity measurement process, this paper introduces differential privacy by adding controlled noise to the random value $\boldsymbol{P}$. Specifically, Laplace noise is added before the intermediate states **XP** and **XPY** are shared with the FL server. This ensures that even if an attacker gains access to these intermediate values, they cannot accurately infer the original data.

4.3 FedSTGCN model

The global graph and local subgraphs have been constructed. The next step is model training. The main process is as follows: N-STGAT is used to perform local model training on the generated graphs, and then the trained models are sent to the federated server for aggregation. However, since the first step in N-STGAT model training is to use GAT to aggregate the neighbor information of the current data flow, it can be observed from Fig. 6 that the neighbors of the current data flow may include data flows from other clients. Therefore, directly using GAT to aggregate neighbor information may lead to privacy leakage.

FedCog primarily adopts an improved GCN (Kipf and Welling, 2017) to aggregate the neighbor information of nodes. Therefore, in this paper, N-STGAT is improved by replacing its GNN core (as shown in Fig. 1) with the improved GCN from FedCog. This training approach is referred to as federated spatiotemporal GCN (FedSTGCN) in this paper. A more detailed explanation of the FedSTGCN model is provided below.

Since its core improvement is based on GCN, the GCN content is explained in detail here. The GCN primarily converts a graph into an adjacency matrix and then uses convolution operations to aggregate node information. The core formula is as follows: suppose that the graph is represented by $\mathcal{G}=(V, E, \boldsymbol{F})$, where V denotes the set of all the nodes in the graph, E represents the set of all the edges, and $\boldsymbol{F}$ denotes the features of all the nodes. In Eq. (3), u represents the current node being calculated, l represents the GNN layer number, $\mathcal{N}_u$ denotes all the neighboring nodes of node u, $\boldsymbol{h}_u$ represents the node embedding of u, $\boldsymbol{W}$ is the learnable weight matrix, σ represents the activation function, and $c_{uv}=1\Big/\sqrt{(1+d_u)(1+d_v)}$ represents the degree calculation between nodes u and v.

$$\boldsymbol{h}_u^{(l+1)}=\sigma\left(\sum_{v\in\mathcal{N}_u} c_{uv}\boldsymbol{h}_v^{(l)}\boldsymbol{W}^{(l)}\right), u\in V. \tag{3}$$

In FedCog, the primary method for addressing privacy concerns in GNNs is graph decoupling, where the graph $\mathcal{G}$ is decoupled into an internal graph $\mathcal{G}^{\mathrm{I}}$ and an external graph $\mathcal{G}^{\mathrm{E}}$. As shown in Fig. 6, the nodes connected by blue line form the internal graph $\mathcal{G}^{\mathrm{I}}$, while those connected by purple line form the external graph $\mathcal{G}^{\mathrm{E}}$. The aggregation of nodes is then divided into two processes: the aggregation of nodes in the internal graph $\mathcal{G}^{\mathrm{I}}$ and the aggregation of nodes in the external graph $\mathcal{G}^{\mathrm{E}}$. The external graph aggregation method is shown in Eq. (4), and the internal graph node aggregation method is shown in Eq. (5). After completing the aggregation of both external and internal graphs, Eq. (6) is used to merge the two results, thus computing the node embedding of u at layer $l+1$. The detailed proof is provided in FedCog, and interested readers can refer to Kipf and Welling (2017) for more information.

$$\hat{\boldsymbol{h}}_u^{(l+1)}=\sigma\left(\sum\nolimits_{v\in\mathcal{N}_u^{\mathrm{E}}}\gamma_{uv}\hat{\boldsymbol{h}}_v^{\left(l+\frac{1}{2}\right)}\right), \tag{4}$$

$$\hat{\boldsymbol{h}}_{u_j}^{(l+\frac{1}{2})}=\sigma\left(\sum\nolimits_{v\in\mathcal{N}_{u_j}^{\mathrm{I}}}\beta_{u_jv}\hat{\boldsymbol{h}}_v^{(l)}\boldsymbol{W}^{(l)}\right), \tag{5}$$

$$\begin{aligned}\hat{\boldsymbol{h}}_u^{(l+1)}&=\sigma\left(\gamma_{uu}\hat{\boldsymbol{h}}_u^{\left(l+\frac{1}{2}\right)}+\sum\nolimits_{j\in B_u}\gamma_{uu_j}\hat{\boldsymbol{h}}_{u_j}^{\left(l+\frac{1}{2}\right)}\right)\\&=\sigma\left(\sum\nolimits_{j\in B_u}\sum\nolimits_{v\in\mathcal{N}_{u_j}^{\mathrm{I}}}c_{uv}\hat{\boldsymbol{h}}_v^{(l)}\boldsymbol{W}^{(l)}\right),\end{aligned} \tag{6}$$

where $\beta_{uv}=1\Big/\sqrt{(1+d_v)}$, $\gamma_{uv}=1\Big/\sqrt{(1+d_u)}$, $\mathcal{N}_u^{\mathrm{I}}$ represents the neighbors of node u in the internal graph $\mathcal{G}^{\mathrm{I}}$, B_u represents the set of parties that hold node u as an external border node, and $\mathcal{N}_u^{\mathrm{E}}$ represents the neighbors of node u in the external graph $\mathcal{G}^{\mathrm{E}}$.

The FedSTGCN model training process has been clearly explained. The model's pseudocode is shown in Algorithm 1.

Algorithm 1 Pseudocode of FedSTGCN

Input: Local graph $\left\{\mathcal{G}_i(V,E,\boldsymbol{F})\right\}_{i=1}^{m}$ for m FL clients, l-layer GCN model $f(\mathcal{G};\theta)$, $\theta=\left\{\boldsymbol{W}^{(0)},\boldsymbol{W}^{(1)},\cdots,\boldsymbol{W}^{(l-1)}\right\}$, and LSTMcore$(u,\boldsymbol{W}_{\mathrm{lstm}})$

1 /* Graph decoupling */
2 **for** i=1 to m **do** /* Execution on client i */
3 $\mathcal{G}_i^{\mathrm{I}}, \mathcal{G}_i^{\mathrm{E}}\leftarrow$GraphDecoupling$(\mathcal{G}_i)$
4 **end for**
5 /* Internal and external graph converge */
6 **for** l=0 to $L-1$ **do**
7 **for** i=1 to m **do** /* Execution on client i */
8 $\hat{\boldsymbol{h}}_u^{(l+\frac{1}{2})}=\sum_{v\in\mathcal{N}_u^{\mathrm{I}}}\beta_{uv}\hat{\boldsymbol{h}}_v^{(l)}\boldsymbol{W}^{(l)}$
9 Receive $\left\{\hat{\boldsymbol{h}}_{uv}^{(l+\frac{1}{2})}, v\in V_j\right\}$ from client j
10 $\hat{\boldsymbol{h}}_u^{(l+1)}\leftarrow\sigma\left(\sum_{j\in B_u}\sum_{v\in\mathcal{N}_{u_j}^{\mathrm{I}}}c_{uv}\hat{\boldsymbol{h}}_v^{(l)}\boldsymbol{W}^{(l)}\right)$
11 **end for**
12 **end for**
13 $r_u\leftarrow$LSTMcore$(\hat{\boldsymbol{h}}_u^{(l+1)},\boldsymbol{W}_{\mathrm{lstm}})$
14 /* Federated training */
15 **for** t=1 to k **do**
16 Sample a set of parties $\mathcal{C}_t$
17 **for** $i\in\mathcal{C}_t$ **do** /* Execution on client i */
18 $\theta_i\leftarrow\theta$
19 $\mathcal{G}_i\leftarrow$GradientDescent(θ_i, r_u)
20 **end for**
21 $\bar{g}\leftarrow\sum_{i\in\mathcal{C}_t}p_i^{(t)}g_i$ /* Execution on FL server */
22 $\theta\leftarrow\theta-\eta\bar{g}$
23 **end for**

In Algorithm 1, lines 2–4 describe the FedCog graph decoupling process. Each client generates an internal graph $\mathcal{G}^{\mathrm{I}}$ and an external graph $\mathcal{G}^{\mathrm{E}}$. Lines 6–12 explain how GCN performs node aggregation between the internal and external graphs, a key step in addressing the privacy issue. Since STGNNs with temporal correlations are required to train the nodes, LSTM is introduced in line 13 to process the node embeddings

generated by GCN. Lines 15–23 detail the FL training method, where FedAvg (McMahan et al., 2017) is used to aggregate the parameters uploaded by each client.

5 Experimental evaluation

5.1 Experimental setup

1. Experimental environment. To evaluate the proposed method, a lightweight Python framework, PyTorch, was used to build the FL framework on an Ubuntu 22.04.4 LTS system. The hardware used in the experiments includes a dual Intel Xeon E5-2696 v4 CPU with an NVIDIA Tesla A10 GPU (24 GB), 256 GB of RAM, and 21 TB of storage.

2. Datasets. The two datasets used in the experiments are those mentioned in N-STGAT and are derived from the original datasets NF-BoT-IoT-v2, NF-ToN-IoT-v2 (Sarhan et al., 2022), BoT-IoT (Koroniotis et al., 2019), and TON_IoT (Moustafa, 2021). These datasets contain status information about nodes during data flow transmission and include 14 node information (NI) fields and 43 flow information (FI) fields. Sample instances of the datasets can be found in the original N-STGAT paper (Wang et al., 2023). They are widely applied in the field of network intrusion detection, as referenced in the literature (Zeng et al., 2022; Nitish et al., 2023; Nuaimi et al., 2023). Due to the large size of the experimental datasets, 20% data of each dataset was used for the experiments. The experimental dataset was then divided into two parts, with 70% for training and 30% for test. In addition, this paper adopts the dataset preprocessing process from the original N-STGAT paper (Wang et al., 2023).

3. Baseline studies. This paper compares the proposed method with the FL-based network intrusion detection methods introduced by Chen et al. (2020), Li BB et al. (2021), and Huang et al. (2023). These methods claimed to achieve satisfactory results in their respective studies.

4. Experimental setting. The datasets were normalized, and fields that were not useful for training the model, such as IPV4_SRC_ADDR and L4_SRC_PORT, were removed. The invalid data were set to zero. The number of clients m was set to 20, and the number of rounds R for aggregating FL parameters using the FedAvg method was set to 100.

5. Metrics for measuring performance. To evaluate the performance of the proposed method, the performance metrics shown in Table 1 were used. These metrics are commonly employed to assess deep learning models. Many studies (Aljuhani et al., 2023; Altaf et al., 2023; Khan et al., 2023; Ma et al., 2023) used these metrics for performance comparison, including the baseline methods compared in this paper.

Table 1 Metrics for measuring performance

Metric	Definition
Recall	$\frac{TP}{TP + FN}$
Precision	$\frac{TP}{TP + FP}$
F1-score	$2 \times \frac{Recall \times Precision}{Recall + Precision}$
Accuracy	$\frac{TP + TN}{TP + FP + FN + TN}$

TP: true positive; TN: true negative; FP: false positive; FN: false negative

5.2 Results

This paper presents the experimental results of various methods for both binary and multiclass classifications. The training dataset is evenly divided into m parts, denoted as $TRD_i, i \in \{1, 2, \cdots, m\}$, where m represents the number of clients. The test dataset is evenly divided into m parts, denoted as $TED_i, i \in \{1, 2, \cdots, m\}$.

5.2.1 Local test dataset

In this subsection, TRD_i and TED_i are provided to client i to simulate a scenario in which each client has its own training and test datasets. In this experiment, the number of clients is set to 5, i.e., $m = 5$. The experimental results for both binary and multiclass classifications are presented below.

1. Binary classification results

We conducted 100 rounds of training and communication on two datasets, calculating the accuracy, precision, recall, and F1-score every 10 rounds, as shown in Tables 2 and 3. The results show that after 100 rounds, the proposed method achieved accuracies of 0.9694 and 0.9748, significantly outperforming other methods. Specifically, the accuracy is improved by 0.0277–0.0701, the precision is improved by 0.0286–0.0771, the recall is improved by 0.0283–0.0891, and the

Table 2 Results of four methods for binary classification on NF-BoT-IoT-v2

R	Accuracy				Precision			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.8758	0.9147	0.9084	0.9294	0.9048	0.9214	0.8749	0.9014
20	0.9020	0.9095	0.9219	0.9268	0.9177	0.9334	0.9213	0.9232
30	0.9324	0.9301	0.9226	0.9305	0.9229	0.9392	0.9329	0.9189
40	0.9401	0.9299	0.9219	0.9464	0.9220	0.9414	0.9344	0.8890
50	0.9400	0.9306	0.9256	0.9619	0.9327	0.9416	0.9344	0.9422
60	0.9412	0.9316	0.9267	0.9646	0.9328	0.9422	0.9344	0.9584
70	0.9415	0.9316	0.9265	0.9650	0.9328	0.9422	0.9345	0.9467
80	0.9415	0.9316	0.9269	0.9681	0.9345	0.9424	0.9345	0.9616
90	0.9415	0.9316	0.9268	0.9682	0.9345	0.9424	0.9345	0.9623
100	0.9417	0.9318	0.9274	0.9694	0.9347	0.9426	0.9347	0.9712
R	Recall				F1-score			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.8547	0.9014	0.8874	0.9218	0.8941	0.8941	0.9064	0.9314
20	0.8925	0.9065	0.8603	0.9381	0.9020	0.9026	0.9062	0.9582
30	0.9184	0.9084	0.8566	0.9409	0.9021	0.9178	0.9072	0.9699
40	0.9259	0.9125	0.9134	0.9477	0.9126	0.9193	0.9021	0.9732
50	0.9261	0.9124	0.9212	0.9494	0.9216	0.9205	0.9121	0.9732
60	0.9261	0.9115	0.9231	0.9518	0.9394	0.9202	0.9118	0.9726
70	0.9262	0.9093	0.9219	0.9522	0.9387	0.9193	0.9120	0.9725
80	0.9262	0.9126	0.9222	0.9533	0.9482	0.9181	0.9127	0.9731
90	0.9262	0.9128	0.9232	0.9528	0.9491	0.9201	0.9131	0.9732
100	0.9264	0.9147	0.9245	0.9547	0.9496	0.9214	0.9142	0.9749

Table 3 Results of four methods for binary classification on NF-ToN-IoT-v2

R	Accuracy				Precision			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.7418	0.6708	0.8715	0.7448	0.8417	0.7297	0.8268	0.7548
20	0.7909	0.7161	0.8835	0.8859	0.9066	0.8411	0.8716	0.8801
30	0.8020	0.8584	0.9020	0.9563	0.9122	0.9096	0.8726	0.8982
40	0.8816	0.9144	0.9017	0.9588	0.9142	0.9117	0.8763	0.9351
50	0.8990	0.9223	0.9020	0.9699	0.9142	0.9077	0.8768	0.9408
60	0.9048	0.9236	0.9021	0.9666	0.9144	0.9052	0.8781	0.9461
70	0.9244	0.9293	0.9045	0.9602	0.9143	0.9158	0.8779	0.9545
80	0.9250	0.9299	0.9045	0.9707	0.9145	0.9171	0.8779	0.9554
90	0.9247	0.9304	0.9045	0.9703	0.9145	0.9171	0.8781	0.9554
100	0.9267	0.9311	0.9047	0.9748	0.9145	0.9175	0.8785	0.9556
R	Recall				F1-score			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.8048	0.7419	0.8493	0.7818	0.7914	0.6917	0.8269	0.7158
20	0.8280	0.9242	0.8287	0.8304	0.8784	0.8458	0.8438	0.8136
30	0.8790	0.9326	0.8685	0.9470	0.9076	0.8484	0.8579	0.9572
40	0.8810	0.9317	0.8747	0.9390	0.9116	0.8082	0.8629	0.9585
50	0.8613	0.9316	0.8783	0.9243	0.9146	0.8947	0.8629	0.9543
60	0.8773	0.9335	0.8784	0.9416	0.9153	0.9147	0.8651	0.9653
70	0.8972	0.9339	0.8829	0.9487	0.9153	0.9194	0.8639	0.9657
80	0.9017	0.9339	0.8824	0.9717	0.9153	0.9187	0.8658	0.9652
90	0.9020	0.9341	0.8842	0.9734	0.9155	0.9211	0.8666	0.9659
100	0.9039	0.9343	0.8847	0.9738	0.9155	0.9217	0.8671	0.9671

F1-score is improved by 0.0253–0.1000. In addition, the accuracy, precision, recall, and F1-score all demonstrated significant improvements compared to those of other methods, proving the strong advantages of this approach. Fig. 7 shows that after 40 rounds, the accuracy stabilized approximately 0.95, further validating its effectiveness in real-world applications. These results indicate that the FL-based intrusion detection method achieves over a 10% performance improvement in binary classification tasks, demonstrating its potential and superiority in network intrusion detection.

2. Multiclass classification results

We conducted 100 rounds of training and communication on two datasets, calculating the weighted recall and weighted F1-score every 10 rounds, as shown in Tables 4 and 5. The results show that after 100 rounds, the proposed method achieved weighted recalls of 0.9015 and 0.9159 and weighted F1-scores of 0.9214 and 0.9237, significantly outperforming other methods. Specifically, the weighted recall is improved by 0.0942–0.1990, and the weighted F1-score is improved by 0.0946–0.2293. Fig. 8 illustrates that the proposed method outperformed other methods after just 30 rounds. Ultimately, the weighted recall exceeds 0.9 on the two datasets. These results indicate a significant performance improvement in multiclass classification tasks.

Additionally, the recall of each attack on the two datasets was collected, as shown in Table 6. It can be observed that the proposed method achieved a high recall in most attacks, with the recall of all attacks surpassing those of the other methods, directly demonstrating the effectiveness of the proposed method. Notably, while the proposed method achieved a recall of cross-site scripting (XSS) attacks as high as 0.9997, the recall of distribution denial-of-service (DDoS)

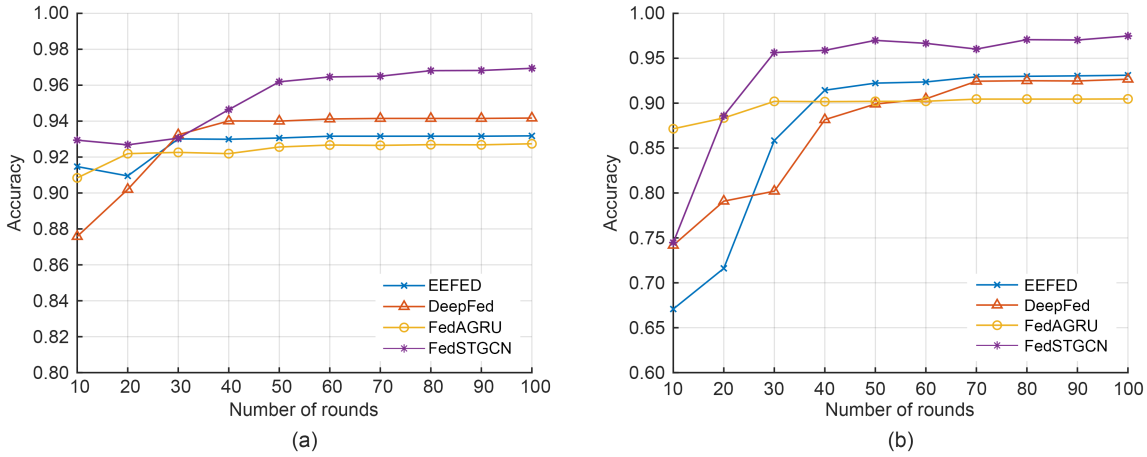

Fig. 7 Changes in binary classification accuracy with the number of rounds for different methods on the two datasets: (a) NF-BoT-IoT-v2; (b) NF-ToN-IoT-v2

Table 4 Results of four methods on multiclass classification on NF-BoT-IoT-v2

R	Weighted recall				Weighted F1-score			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.4859	0.5716	0.4612	0.6812	0.4348	0.6257	0.4586	0.7214
20	0.5476	0.6462	0.5967	0.8462	0.6240	0.7490	0.4748	0.9205
30	0.5853	0.6650	0.6294	0.8661	0.6583	0.7703	0.6428	0.9205
40	0.5061	0.7095	0.6838	0.8812	0.6825	0.7788	0.6471	0.9199
50	0.6002	0.7801	0.6893	0.8831	0.7317	0.7577	0.6305	0.9201
60	0.7192	0.8003	0.6802	0.9000	0.7384	0.8220	0.6828	0.9210
70	0.7153	0.8001	0.6990	0.8991	0.7422	0.8221	0.6882	0.9210
80	0.7247	0.8012	0.7021	0.9011	0.7419	0.8255	0.6883	0.9208
90	0.7248	0.8012	0.7024	0.9013	0.7448	0.8249	0.6847	0.9210
100	0.7249	0.8014	0.7025	0.9015	0.7458	0.8268	0.6921	0.9214

Table 5 Results of four methods on multiclass classification on NF-ToN-IoT-v2

R	Weighted recall				Weighted F1-score			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
10	0.6819	0.7513	0.6217	0.6817	0.6175	0.7025	0.5914	0.7249
20	0.7678	0.7553	0.6190	0.7056	0.6872	0.7760	0.6734	0.5700
30	0.7790	0.7605	0.6390	0.7834	0.7324	0.7787	0.7647	0.6748
40	0.7884	0.8144	0.6630	0.8399	0.7361	0.8009	0.7792	0.9101
50	0.7914	0.8214	0.6993	0.8767	0.7420	0.8010	0.7796	0.9125
60	0.7917	0.8214	0.7028	0.8644	0.7419	0.8011	0.7800	0.9147
70	0.7919	0.8214	0.7226	0.9081	0.7479	0.8011	0.7806	0.9189
80	0.7919	0.8215	0.7498	0.9106	0.7494	0.8011	0.7809	0.9235
90	0.7924	0.8215	0.7503	0.9079	0.7494	0.8011	0.7809	0.9235
100	0.7926	0.8217	0.7514	0.9159	0.7517	0.8014	0.7812	0.9237

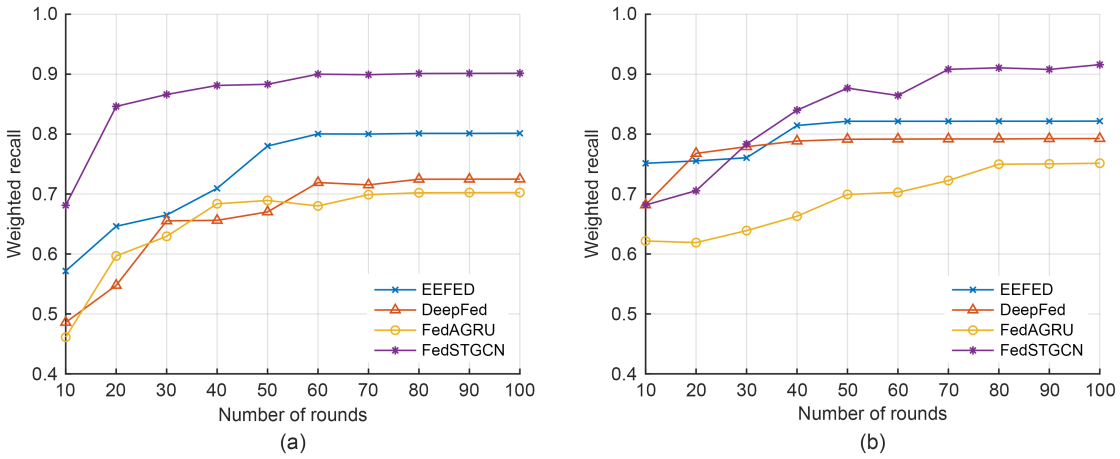

Fig. 8 Changes in multiclass classification weighted recall with the number of rounds for different methods on the two datasets: (a) NF-BoT-IoT-v2; (b) NF-ToN-IoT-v2

Table 6 Recall of different attacks for the four methods on the two datasets

Dataset	Attack	Recall			
		DeepFed	EEFED	FedAGRU	FedSTGCN
NF-BoT-IoT-v2	Benign	0.7914	0.8419	0.8016	0.9516
	Reconnaissance	0.8136	0.8103	0.7219	0.9743
	DDoS	0.7049	0.7915	0.6483	0.8671
	DoS	0.6901	0.7496	0.7814	0.8819
	Theft	0.8492	0.8617	0.8049	0.9416
NF-ToN-IoT-v2	Benign	0.8216	0.8318	0.7913	0.9314
	Reconnaissance	0.7946	0.8613	0.7419	0.9813
	DDoS	0.7297	0.7619	0.7743	0.8719
	DoS	0.7525	0.7945	0.7581	0.8934
	Backdoor	0.8216	0.8016	0.6913	0.9916
	Injection	0.6785	0.7916	0.7459	0.9763
	MitM	0.7509	0.7148	0.8216	0.9417
	Password	0.9016	0.8043	0.8015	0.9518
	Scanning	0.8116	0.7319	0.6649	0.9743
	XSS	0.8052	0.8419	0.8216	0.9997

MitM: man-in-the-middle

attacks only reached 0.8671. One primary reason for this discrepancy is the high similarity between certain types of attacks, such as DoS and DDoS. This paper argues that better distinguishing between these similar attacks is crucial for improving recall.

Since the results of both binary and multiclass classification tests were very good, multiple tests were conducted to verify the stability of the proposed method. The confidence intervals of the method in binary and multiclass classifications were statistically analyzed, and the results can be found in the supplementary materials.

5.2.2 Multi-participant

In Section 5.2.1, model training was conducted with five clients; however, the number of clients in FL is variable. For instance, in a smart community within a city, the number of smart communities may increase over time, necessitating that the method retains its advantages with different client counts. Therefore, this subsection conducts experiments with varying numbers of clients m (where m=1, 2, ⋯, 20) and records the training results of both multiclass and binary classifications, as shown in Figs. 9 and 10 and Tables 7 and 8. It is evident from the figures that even with changes in the number of clients, as the number of rounds increases, the final binary classification accuracy and multiclass classification weighted recall consistently exceed those of the other methods. Tables 7 and 8 show that as the number of clients increases, the classification accuracy and weighted recall gradually decrease and eventually stabilize. The initial gradual decrease is due to the distribution of training data across

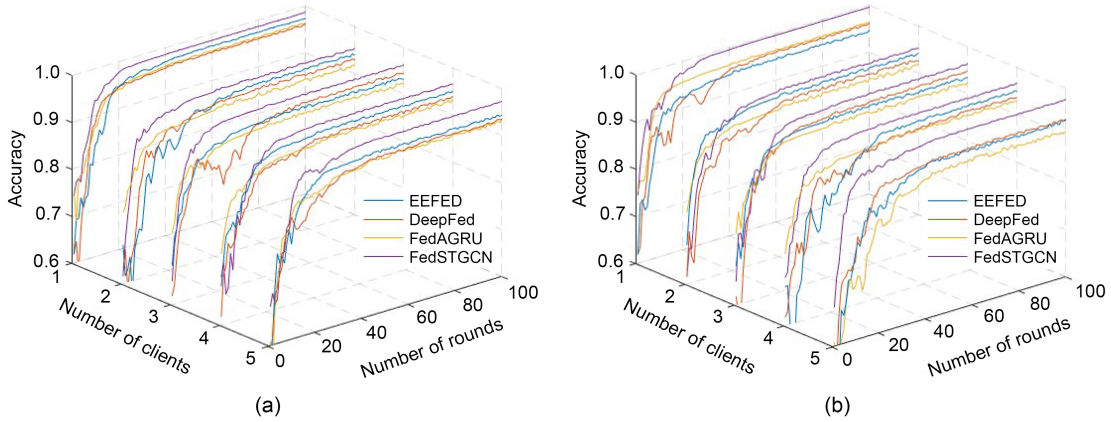

Fig. 9 Changes in binary classification accuracy with the number of rounds for different methods across different numbers of clients (only the first 5 clients are shown, e.g., m=1, 2, 3, 4, 5) on the two datasets: (a) NF-BoT-IoT-v2; (b) NF-ToN-IoT-v2. References to color refer to the online version of this figure

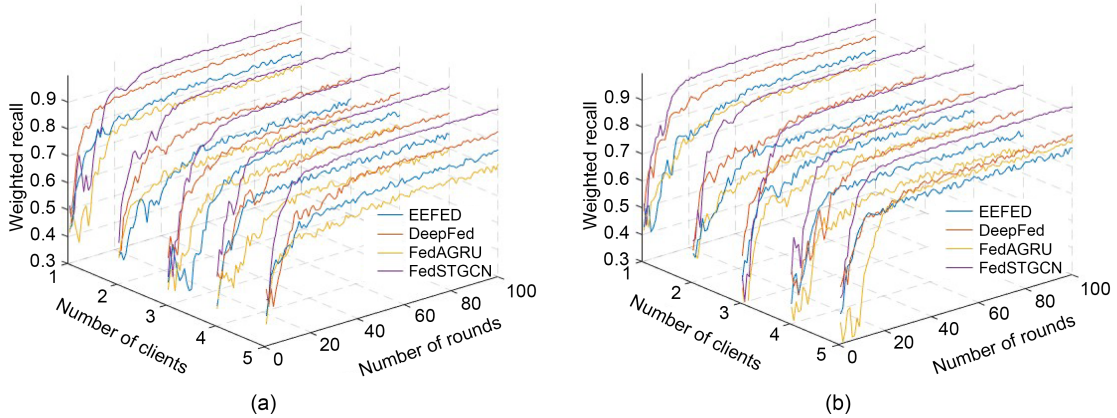

Fig. 10 Changes in multiclass classification weighted recall with the number of rounds for different methods across different numbers of clients (only the first 5 clients are shown, e.g., m=1, 2, 3, 4, 5) on the two datasets: (a) NF-BoT-IoT-v2; (b) NF-ToN-IoT-v2. References to color refer to the online version of this figure

Table 7 Training results (accuracy) of binary classification with different numbers of clients

Client number	Accuracy							
	NF-BoT-IoT-v2				NF-ToN-IoT-v2			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
1	0.9512	0.9586	0.9523	0.9788	0.9521	0.9434	0.9535	0.9794
2	0.9501	0.9574	0.9513	0.9781	0.9501	0.9422	0.9493	0.9801
3	0.9461	0.9515	0.9412	0.9726	0.9322	0.9386	0.9301	0.9787
4	0.9432	0.9422	0.9355	0.9702	0.9358	0.9312	0.9152	0.9763
5	0.9317	0.9418	0.9274	0.9694	0.9267	0.9311	0.9047	0.9748
8	0.9345	0.9395	0.9276	0.9699	0.9218	0.9299	0.9085	0.9732
11	0.9351	0.9387	0.9257	0.9664	0.9201	0.9286	0.9043	0.9754
15	0.9318	0.9391	0.9247	0.9685	0.9216	0.9257	0.9068	0.9736
18	0.9322	0.9402	0.9288	0.9697	0.9224	0.9266	0.9062	0.9749
20	0.9341	0.9398	0.9274	0.9675	0.9234	0.9247	0.9054	0.9752

Table 8 Training results (weighted recall) of multiclass classification with different numbers of clients

Client number	Weighted recall							
	NF-BoT-IoT-v2				NF-ToN-IoT-v2			
	DeepFed	EEFED	FedAGRU	FedSTGCN	DeepFed	EEFED	FedAGRU	FedSTGCN
1	0.7513	0.8714	0.7217	0.9324	0.8264	0.8512	0.7842	0.9288
2	0.7321	0.8526	0.7189	0.9301	0.8154	0.8394	0.7715	0.9213
3	0.7258	0.8114	0.7113	0.9152	0.8112	0.8315	0.7599	0.9199
4	0.7294	0.8094	0.7088	0.9082	0.8039	0.8254	0.7536	0.9164
5	0.7249	0.8143	0.7025	0.9045	0.7996	0.8217	0.7514	0.9159
8	0.7215	0.8122	0.7064	0.9072	0.8001	0.8263	0.7548	0.9167
11	0.7264	0.8099	0.7059	0.9077	0.7981	0.8178	0.7532	0.9188
15	0.7252	0.8135	0.7034	0.9068	0.8023	0.8247	0.7513	0.9157
18	0.7269	0.8101	0.7052	0.9052	0.8137	0.8211	0.7495	0.9149
20	0.7233	0.8164	0.7039	0.9032	0.7986	0.8197	0.7515	0.9166

different clients, which reduces the amount of training data available on each client and affects the results. As the variation in the training data across clients decreases, the training performance stabilizes. This finding indicates that changes in the number of clients have a smaller impact on the results. Additionally, the advantages of the method proposed in this paper over other methods do not change with the number of clients. Notably, the training results are the best for both binary and multiclass classifications when m=1. This is because, in this scenario, all the data from the FL system are trained on a single client, essentially reflecting the ideal model of the entire system.

5.2.3 Robustness

In real-world complex scenarios, the variety of attack types and the speed of sample collection and labeling often do not meet the demands of the situation. Each client's network flow data may not contain the same attack types, making it difficult to identify unknown attacks locally. Therefore, the ability to recognize unknown attacks is crucial for such methods. In addition to the aforementioned experiments, we conducted binary classification experiments for recognizing unknown attacks. Each attack type was labeled with a numerical identifier, and datasets containing different attack types were distributed among m clients (where m=5). Binary classification training was then performed, and the recognition results were recorded, as illustrated on the right side of Fig. 11. For instance, during model training with the NF-BoT-IoT-v2 dataset, benign flows were labeled as 0, while the other four types of attack flows were labeled as 1–4. The training dataset for Client 1 included only

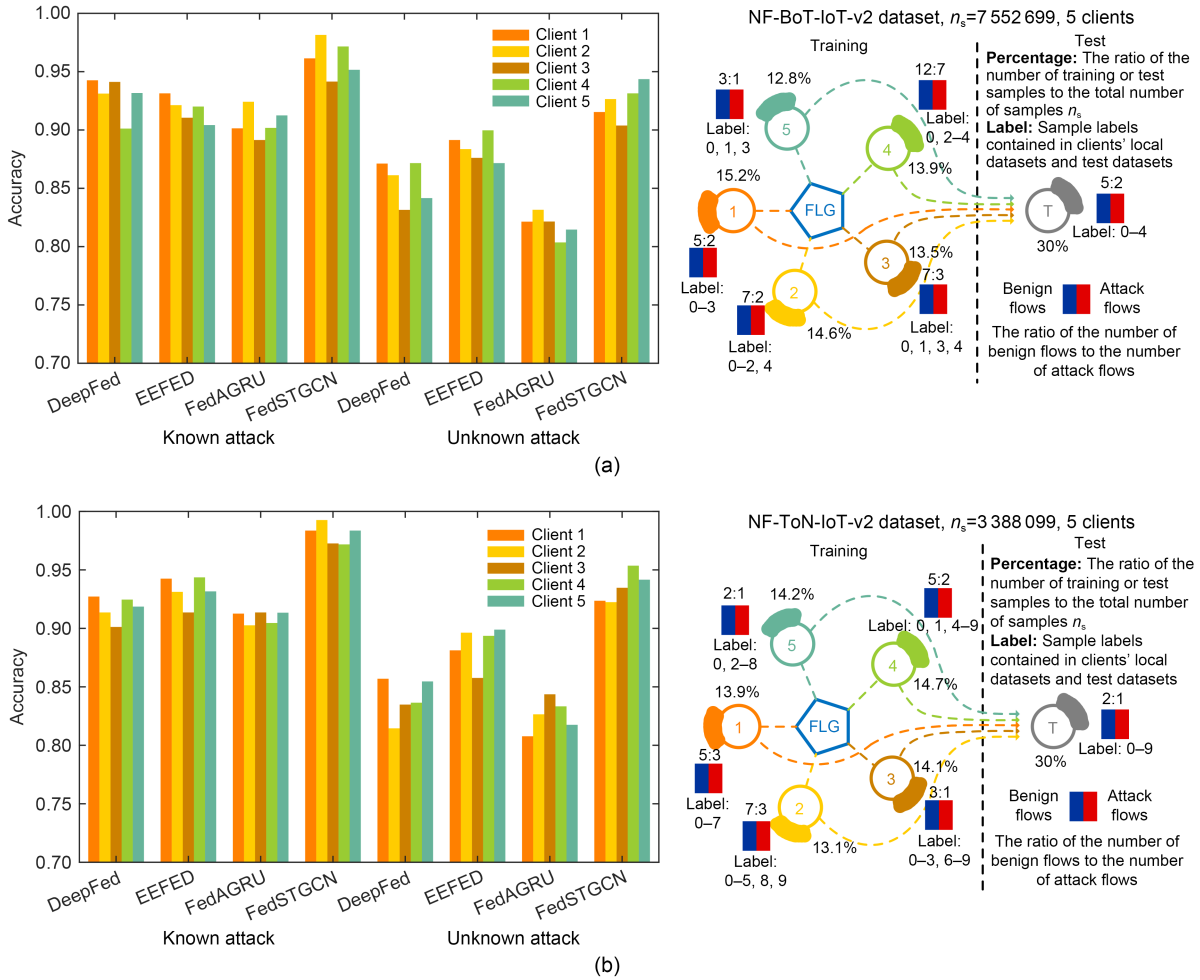

Fig. 11 Recognition accuracy of different attacks in binary classification for each method on the two datasets: (a) NF-BoT-IoT-v2; (b) NF-ToN-IoT-v2. References to color refer to the online version of this figure

classifications labeled 0–3, while the training dataset for Client 2 included classifications labeled 0–2 and 4. Similarly, other clients excluded their respective classifications. After model training, a test set containing all labels was used to validate each client, which comprised 30% of the entire dataset. The recognition results for known and unknown attacks for each client across all methods are presented on the left side of Fig. 11.

As shown in Fig. 11, on the NF-BoT-IoT-v2 dataset, the proposed method achieves an average recognition accuracy of 0.9382 for unknown attacks across 5 clients, with a maximum improvement of 0.1213 compared to other methods. On the NF-ToN-IoT-v2 dataset, the proposed method shows an average recognition accuracy of 0.9276 for unknown attacks across 5 clients, representing a maximum improvement of 0.1057 over other methods. These results strongly indicate that the proposed method is the most effective for identifying unknown attacks, with each client demonstrating a greater responsiveness to unknown attacks than other methods do. This further proves that the proposed method exhibits greater robustness compared to other methods.

In addition to the above experiments, this paper analyzes each method's computational cost during the training process. The results can be found in the supplementary materials.

6 Conclusions

To address the issue of training distributed network intrusion detection models, this paper proposes an

advanced model called FedSTGCN, which combines spatiotemporal GNNs with FL. Solutions to the privacy issues arising during the graph construction and node information aggregation processes are also proposed. In the graph construction process, the cosine similarity function is split into several parts, effectively using its relationships to compute the cosine similarity between two nodes, generating a graph for GNN training. In the node information aggregation process, the model combines N-STGAT with FedCog to address privacy protection issues. Finally, classification accuracy experiments, client number variation experiments, unknown attack recognition experiments, and computational cost experiments are conducted on two recent datasets. The results consistently validate the effectiveness of the proposed method.

Notably, the methods and experiments presented in this paper are based on independent and identically distributed (IID) datasets, where FL generally achieves high performance. However, in real-world network scenarios, the datasets collected from different clients are often non-IID, which poses significant challenges for FGL. Therefore, future research should focus on distributed network intrusion detection in non-IID environments.

Contributors

Yalu WANG and Zhijie HAN designed the research. Yalu WANG, Jie LI, and Zhijie HAN processed the data. Yalu WANG drafted the paper. Pu CHENG helped organize the paper. Yalu WANG and Roshan KUMAR revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Aburomman AA, Reaz MBI, 2016. A novel SVM-kNN-PSO ensemble method for intrusion detection system. *Appl Soft Comput*, 38:360-372. https://doi.org/10.1016/j.asoc.2015.10.011

Aljuhani A, Kumar P, Kumar R, et al., 2023. Fog intelligence for secure smart villages: architecture and future challenges. *IEEE Consum Electron Mag*, 12(5):12-21. https://doi.org/10.1109/MCE.2022.3193268

Almogren AS, 2020. Intrusion detection in Edge-of-Things computing. *J Parall Distrib Comput*, 137:259-265. https://doi.org/10.1016/j.jpdc.2019.12.008

Altaf T, Wang X, Ni W, et al., 2023. A new concatenated multigraph neural network for IoT intrusion detection. *Int Things*, 22:100818. https://doi.org/10.1016/j.iot.2023.100818

Caville E, Lo WW, Layeghy S, et al., 2022. Anomal-E: a self-supervised network intrusion detection system based on graph neural networks. *Knowl-Based Syst*, 258:110030. https://doi.org/10.1016/j.knosys.2022.110030

Chatterjee P, Das D, Rawat DB, 2024. Federated learning empowered recommendation model for financial consumer services. *IEEE Trans Consum Electron*, 70(1):2508-2516. https://doi.org/10.1109/TCE.2023.3339702

Chen Z, Lv N, Liu PF, et al., 2020. Intrusion detection for wireless edge networks based on federated learning. *IEEE Access*, 8:217463-217472. https://doi.org/10.1109/ACCESS.2020.3041793

Dina AS, Siddique AB, Manivannan D, 2023. A deep learning approach for intrusion detection in Internet of Things using focal loss function. *Int Things*, 22:100699. https://doi.org/10.1016/j.iot.2023.100699

Ghasempour A, 2019. Internet of Things in smart grid: architecture, applications, services, key technologies, and challenges. *Inventions*, 4(1):22. https://doi.org/10.3390/inventions4010022

Hu XY, Gao WJ, Cheng G, et al., 2023. Toward early and accurate network intrusion detection using graph embedding. *IEEE Trans Inform Forens Secur*, 18:5817-5831. https://doi.org/10.1109/TIFS.2023.3318960

Huang XT, Liu J, Lai YX, et al., 2023. EEFED: personalized federated learning of execution & evaluation dual network for CPS intrusion detection. *IEEE Trans Inform Forens Secur*, 18:41-56. https://doi.org/10.1109/TIFS.2022.3214723

Khan NW, Alshehri MS, Khan MA, et al., 2023. A hybrid deep learning-based intrusion detection system for IoT networks. *Math Biosci Eng*, 20(8):13491-13520. https://doi.org/10.3934/mbe.2023602

Kipf TN, Welling M, 2017. Semi-supervised classification with graph convolutional networks. 5th Int Conf on Learning Representations, p.1-14.

Koroniotis N, Moustafa N, Sitnikova E, et al., 2019. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Fut Gener Comput Syst*, 100:779-796. https://doi.org/10.1016/j.future.2019.05.041

Lei RZ, Wang PH, Zhao JZ, et al., 2023. Federated learning over coupled graphs. *IEEE Trans Parall Distrib Syst*, 34(4): 1159-1172. https://doi.org/10.1109/TPDS.2023.3240527

Li BB, Wu YH, Song JR, et al., 2021. DeepFed: federated deep learning for intrusion detection in industrial cyber–physical systems. *IEEE Trans Ind Inform*, 17(8):5615-5624. https://doi.org/10.1109/TII.2020.3023430

Li XM, Zhang D, Zheng Y, et al., 2023. Evolutionary computation-based machine learning for smart city high-dimensional big data analytics. *Appl Soft Comput*, 133:109955. https://doi.org/10.1016/j.asoc.2022.109955

Lim WYB, Luong NC, Hoang DT, et al., 2020. Federated learning in mobile edge networks: a comprehensive survey.

IEEE Commun Surv Tuto, 22(3):2031-2063. https://doi.org/10.1109/COMST.2020.2986024

Lo WW, Layeghy S, Sarhan M, et al., 2022. E-GraphSAGE: a graph neural network based intrusion detection system for IoT. IEEE/IFIP Network Operations and Management Symp, p.1-9. https://doi.org/10.1109/NOMS54207.2022.9789878

Ma ZC, Liu L, Meng WZ, et al., 2023. ADCL: toward an adaptive network intrusion detection system using collaborative learning in IoT networks. *IEEE Int Things J*, 10(14): 12521-12536. https://doi.org/10.1109/JIOT.2023.3248259

Mansour Bahar AA, Ferrahi KS, Messai ML, et al., 2024. FedHE-Graph: federated learning with hybrid encryption on graph neural networks for advanced persistent threat detection. Proc 19th Int Conf on Availability, Reliability and Security, Article 119. https://doi.org/10.1145/3664476.3670466

Mao QH, Lin X, Xu WC, et al., 2025. FeCoGraph: label-aware federated graph contrastive learning for few-shot network intrusion detection. *IEEE Trans Inform Forens Secur*, 20: 2266-2280. https://doi.org/10.1109/TIFS.2025.3541890

McMahan B, Moore E, Ramage D, et al., 2017. Communication-efficient learning of deep networks from decentralized data. Proc 20th Int Conf on Artificial Intelligence and Statistics, p.1273-1282.

Moustafa N, 2021. A new distributed architecture for evaluating AI-based security systems at the edge: network TON_IoT datasets. *Sustain Cities Soc*, 72:102994. https://doi.org/10.1016/j.scs.2021.102994

Nguyen DC, Ding M, Pathirana PN, et al., 2021. Federated learning for Internet of Things: a comprehensive survey. *IEEE Commun Surv Tutor*, 23(3):1622-1658. https://doi.org/10.1109/COMST.2021.3075439

Nguyen DC, Pham QV, Pathirana PN, et al., 2022. Federated learning for smart healthcare: a survey. *ACM Comput Surv*, 55(3):60. https://doi.org/10.1145/3501296

Nitish A, Hanumanthappa J, Shiva Prakash SP, et al., 2023. On-device context-aware misuse detection framework for heterogeneous IoT edge. *Appl Intell*, 53(12):14792-14818. https://doi.org/10.1007/s10489-022-04039-5

Nuaimi M, Fourati LC, Hamed BB, 2023. Intelligent approaches toward intrusion detection systems for Industrial Internet of Things: a systematic comprehensive review. *J Netw Comput Appl*, 215:103637. https://doi.org/10.1016/J.JNCA.2023.103637

Sangkatsanee P, Wattanapongsakorn N, Charnsripinyo C, 2011. Practical real-time intrusion detection using machine learning approaches. *Comput Commun*, 34(18):2227-2235. https://doi.org/10.1016/j.comcom.2011.07.001

Sarhan M, Layeghy S, Portmann M, 2022. Towards a standard feature set for network intrusion detection system datasets. *Mob Netw Appl*, 27(1):357-370. https://doi.org/10.1007/s11036-021-01843-0

Veličković P, Cucurull G, Casanova A, et al., 2018. Graph attention networks. 6th Int Conf on Learning Representations, p.1-12.

Wang YL, Li J, Zhao W, et al., 2023. N-STGAT: spatio-temporal graph neural network based network intrusion detection for near-Earth remote sensing. *Remote Sens*, 15(14):3611. https://doi.org/10.3390/rs15143611

Wu JP, Qiu GQ, Wu CM, et al., 2024. Federated learning for network attack detection using attention-based graph neural networks. *Sci Rep*, 14(1):19088. https://doi.org/10.1038/s41598-024-70032-2

Yang Q, Liu Y, Chen TJ, et al., 2019. Federated machine learning: concept and applications. *ACM Trans Intell Syst Technol*, 10(2):12. https://doi.org/10.1145/3298981

Zeng ZR, Peng W, Zeng DT, 2022. Improving the stability of intrusion detection with causal deep learning. *IEEE Trans Netw Serv Manag*, 19(4):4750-4763. https://doi.org/10.1109/TNSM.2022.3193099

Zhou XK, Liang W, Li WM, et al., 2022. Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system. *IEEE Int Things J*, 9(12): 9310-9319. https://doi.org/10.1109/JIOT.2021.3130434

List of supplementary materials

1 Supplementary experimental results

Table S1 Confidence intervals and *p*-values for binary classification on the NF-BoT-IoT-v2 dataset

Table S2 Confidence intervals and *p*-values for binary classification on the NF-ToN-IoT-v2 dataset

Table S3 Confidence intervals and *p*-values for multiclass classification on two datasets

Table S4 Various overhead information in the experiments