FEDGCN: COMMUNICATION TRADE OFFS IN FEDERATED TRAINING OF GRAPH CONVOLUTIONAL NETWORKS

Md. Shakib Khan
40236443
https://github.com/Shakib-IO/FedGCN

Abstract

Federated GCN training is a distributed approach that enables multiple clients to collaboratively train a GCN model without sharing their raw data. Training models on graph data across multiple clients has recently gained popularity due to the size of these graphs as well as regulations on keeping data where it is generated. However, graph structure on multiple clients forms a cross-edge connection between clients. But given the cross-client edges connection, it is impossible to partition a connected graph onto multiple distributed clients in a way that leaves them disjoint. Thus, distributed methods for training a model on a single graph incur significant communication overhead between clients or a loss of available information to the training. We introduce the **F**ederated **G**raph **C**onvolutional **N**etwork (**FedGCN**) algorithm, which is a new approach that uses federated learning to train GCN models for identifying fraud in large graphs with limited communication. Unlike previous methods that require frequent communication among clients during training, FedGCN only requires communication with the central server in one pre-training step, which significantly reduces communication costs. We also provide a theoretical analysis of the trade-offs between communication cost and data distribution in FedGCN.

Keywords: Fraud Detection; Federated Learning; Graph Convolution Network; Deep Learning.

1 Introduction

Fraud detection has received increasing attention in fields such as social network, e-commerce and advertising [1]. While this fields certainly makes people's life easier and more convenient, it also indirectly creates a market for malicious users. One can earn huge profits by selling fake followers on Instagram and Twitter, or fake reviews on Yelp [2] and Amazon [3]. Some malicious service providers could also help disseminating information such as ads or fake news by manipulating botnets on social networks. It turns out that these behaviours have a negative impact on our society; fake news could have tremendous effects on political activities; false reviews constantly undermine customers' ability to make fair judgements; and phony followers will cause fake popularity, giving false credentials and breaking a competitive market. Due to these behaviours, a lot of awareness is gained to detecting fraud users from the network. Therefore, various efforts have been made to address the problem of fraud detection, including graph leraning based methods such as FRAUDAR [4], DOMINANT [5] and CARE-GNN [6].

The advancement of Graph Neural Networks (GNNs) enables the effective representation learning for a variety of areas, including bioinformatics, chemoinformatics, social networks, natural language processing [7], social events [8], recommender system [9], spatial-temporal traffic [10], computer vision and physics where graphs are primarily the denotation. GNN models are proven to reach the performance target over massive datasets – citation networks, biochemical networks, knowledge graphs [11], etc – on different tasks, such as node classification, node clustering, link prediction, graph classification, etc. Recently, numerous GNN-based fraud detectors have been presented to detect fraud, financial fraud, mobile fraud, and cybercriminal. This is due to the development of Graph Neural Networks (GNNs), such as GCN [12], GAT [13], GraphSAGE [14], and FdGars [15]. The traditional GNN-based methods aggregate neighborhood information to learn the representation of a centre node using neural modules. GNN-based techniques can be trained in an end-to-end, semi-supervised manner, which reduces the cost of feature engineering and data annotation. Furthermore, Graph Convolution Network (GCN) is a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. GCNs have been widely used for applications ranging from fraud news detection in social networks to anomaly detection in sensor networks.

However, the data from social networks and online platforms can be too large to store on a single server, and it is often privacy-sensitive. For example, records of billions of users' website visits may contain sensitive information that users do not want to reveal, such as the websites they have visited. Furthermore, all of the existing approaches for fraud detection have been evaluated using centralized data, which requires collecting a large amount of user data. This can be expensive and may not be feasible if users are unwilling to provide their personal information. To address these issues, federated learning (FedAvg) [16] has been shown to be effective at preserving user privacy while training accurate models.

Some authors have proposed federated training of GCNs [17], FedVGCN [18]. These methods typically follow a framework where each client computes local updates to a semi-supervised model on their local subgraphs, which are periodically aggregated at a central server. However, applying federated learning to GCN training on a single large graph is challenging because disjoint partitions across clients are not possible. In Figure 1, for example, we can see that when nodes are partitioned among clients, some edges will cross different clients. These edges are called "cross-client edges".

However, GCNs require information about a nodes neighbors to be aggregated in order to construct an embedding of each node that is used to accomplish tasks such as fraud node detection, classification and link prediction. However, many federated graph training algorithms [19] simply ignore information from neighbors located on other clients, which can result in less accurate models because of the loss of information. On the other hand, sending the features of neighboring nodes to other clients can cause significant communication overhead and reveal private node information to other clients.

Instead, in this study, we understand that the knowledge needed to train a GCN only needs to be shared once, prior to training. This means that each node at a given client only needs to be aware of the total amount of information about its neighbors at other clients. Based on this insight, we introduce the **FedGCN** method for distributed GCN training. FedGCN greatly reduces communication costs without information loss, compared with existing distributed settings that reduce cross-client information loss with communication in each training round [20] [21].

Graph based federated learning meets both the communication and statistical difficulties. Federated learning relies on stochastic gradient descent (SGD) [16], which is widely used to train deep networks with good empirical performance [22]. It is vital to use IID sampling in the training data to ensure that the stochastic gradient is an unbiased estimate of the full gradient. However, in reality, it is unlikely that the local data on each device will always be IID. As a result, it is important to consider non-IID settings where each client has different personal data. Figure 2 demonstrate the i.i.d (IID) and non-i.i.d (non-IID) scenario.

In this work, we examine the use of our proposed FedGCN method in scenarios where the nodes in each client have balanced types of classes (i.e., IID data). In such cases, there will be many edges across clients and our method may require a significant amount of communication. However, if the nodes are concentrated at the same clients (as is often the case with non-IID data in federated learning), then ignoring cross-client edges will not result in a significant loss of information. We also provide an analysis of the FedGCN method in terms of the amount of communication required under both IID and non-IID client data.

The main contributions of this paper can be summarized as follows:

- We provide FedGCN, a framework that is effective for federated training of GCNs to complete fraud node-level prediction tasks with little communication and information loss.

- Our experiments on real-world datasets demonstrate that FedGCN outperforms existing distributed GCN training methods in most cases with higher accuracy.

- In our proposed FedGCN framework, we use cross-client edges to obtain information about the neighbors of each node on other clients. This approach allows us to share information only once, greatly speeding up communication compared to existing methods that require ongoing communication between clients. By leveraging cross-client edges in this way, we are able to reduce communication costs and avoid information loss, leading to more accurate models for node-level prediction tasks.

The remaining parts of this paper are organized as follows. In Section 2, we discussed the related study on graph networks, federated learning, and, specifically, fraud detection tasks done with the graph approach. Section 3 demonstrates the methods and tools used in our work. In Section 4, experiments are performed on datasets to find out the effectiveness of the proposed approach in comparison with baseline methods and we discussed our contribution and findings. Finally, concluded in Section 5.

2 Related Works

Traditional methods for detecting fraudulent transactions often treat each transaction as a standalone entity and train supervised models using statistical features based on transaction-related data. For example, in [23], the authors used support vector machines, random forests, and logistic regression to detect credit card fraud. However, these methods rely on careful feature engineering and can struggle to effectively capture changes in user behavior.

Recently, graph neural networks have received extensive attention, and many representative works have emerged. As pioneering work, proposed a scalable approach for semi-supervised learning on graph-structured data, called graph convolutional networks (GCN) [12]. This presented a general inductive framework that leveraged node feature information to efficiently generate node embeddings for previously unseen data. Authors proposed graph attention networks (GAT) [13], which leveraged masked self-attentional layers to learn specifying different weights to different nodes in a neighborhood. Graph neural networks have also been widely applied in various fields, such as text classification [24], machine translation [25] [2], anomaly detection [26] and etc. For example, Authors [27] proposed to model user profiling based on heterogeneous graph learning in a semi-supervised manner. In this work, we apply graph convolutional networks for fraud transactions detection. Graph-based methods have shown their superiority in fraud detection. For example, SemiGNN [28] utilize a GNN-supported hierarchical attention mechanism to detect fraudsters on Alipay. GraphConsis [29] and CARE-GNN [6] filter dissimilar neighbors before aggregation to find out camouflage fraudsters. PC-GNN [30] and AO-GNN [31] solve the label imbalance issue by node resampling and edge pruning, respectively. AMG-DP [32] utilize attention mechanisms to tell the essential factors for the fraud. FRAUDRE [4] unifies four modules into a GNN to tackle the graph inconsistency and imbalance issues. H2-F Detector [33] considers the homophilic and heterophilic connections simul-

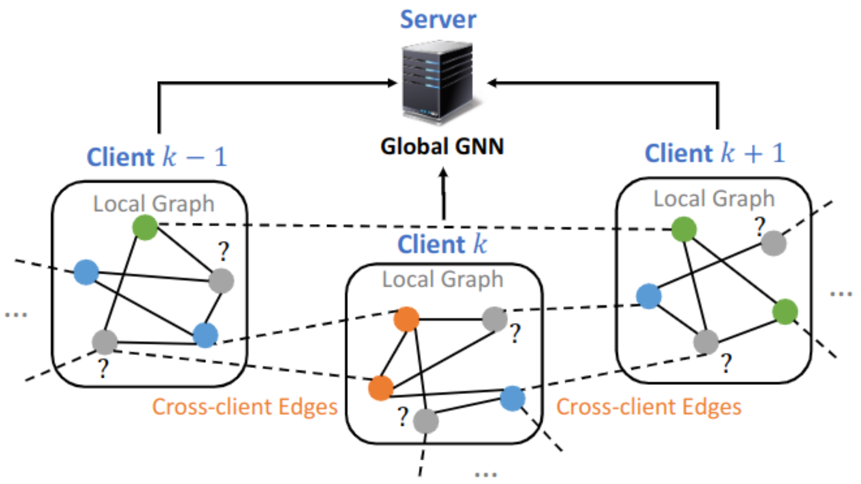

Figure 1: Federated GCN training scheme for fraud detection, nodes in the graph are distributed among different clients, as indicated by the different colors. The dotted lines show the edges between nodes belonging to different clients, and the arrows indicate that each client can exchange updates with the central server during training. The task is to predict the unknown labels of the grey nodes on each client, while the labels of the other nodes are known.

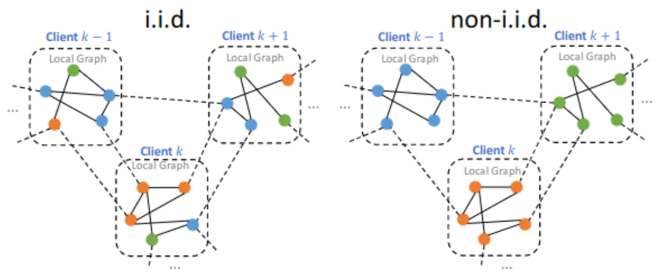

Figure 2: Cross-client graph with i.i.d (left) and non-i.i.d. (right) data distribution. Node colour represents the class of the node. Data distribution affects the number of cross-client edges.

taneously. Recently, the inherent explainability of GNNs [34] has received much attention. Self-explainable GNNs like ProtGNN [35] proposed to give predictions and explanations.

A recent area of study is federated learning on graph neural networks [36]. ASFGNN [37] is a Bayesian optimization strategy to automatically tune the hyper-parameters of all clients for separated graphs, whereas GraphFL [38] is a model-agnostic meta learning approach to learn tasks from multiple graphs at different clients (such as graph classification and picture classification).

In this work, our FedGCN method instead considers semi-supervised tasks on a single large graph (e.g., for node classification), for which existing methods generally ignore cross-client edges. CNFGNN [39] uses the central server to deal with spatial dependencies between nodes, but this method affects the privacy of users by revealing information about their data to the server. Distributed GNN [20] proposes a training algorithm communicating the neighbor features and intermediate outputs of GNN layers among clients with expensive communication costs. BDS-GCN [40] then proposes to sample cross-client neighbors. These methods may violate client privacy by revealing per node information to other clients. FedSage+ [41] recovers missing neighbors for the input graph based on the node embedding, which requires fine-tuning a linear model of neighbor generation and may not fully recover the cross-client information. It is further vulnerable to the data reconstruction attack, compromising privacy.

The works discussed above typically require communication at every training round, which can introduce a high communication overhead. In contrast, FedGCN enables the private recovery of cross-client neighbor information with a single, pre-training communication round, greatly reducing the communication overhead compared to these existing methods. This makes FedGCN a more efficient and effective approach for distributed GCN training in federated learning settings.

3 Method

This section presents the main components, methodologies, and algorithmic approaches of the proposed task on a single graph. We also introduce the federated setting in which we aim to solve this problem, highlighting the challenges and benefits of using a distributed approach. We describe the key components of our proposed method, including the graph convolutional network (GCN) architecture, the use of cross-client edges to obtain neighbor information, and the single-round communication scheme that enables efficient and effective training in a federated setting. Finally, we provide an overview of the algorithms and techniques used in our approach, including the training and inference procedures for the GCN model.

3.1 Federated Semi-Supervised Node Classification

We consider a graph $G = (V, E)$, where $V = 1, ..., N$ is the set of N nodes and E is a set of edges between them. The graph can be equivalently described by a weighted adjacency matrix $A \varepsilon \mathbb{R}^{N \times N}$, where each entry A_{ij} indicates the weight of an edge from node i to node j (if the edge does not exist, the weight is zero). Every node $i \varepsilon V_i$ is associated with a feature vector $x_i \varepsilon \mathbb{R}^d$. Each node i in a subset $V^{train} \subset V$ is associated with a corresponding label y_i, which is used in the training process. Semi-supervised node classification aims to assign labels to nodes in the remaining set V/V^{train}, based on their feature vectors and connections to other nodes. We train a GCN model to do so.

GCNs [12] consist of multiple convolutional layers, each of which constructs a node embedding by aggregating the features of its neighboring nodes. Typically, the node embedding matrix $H(l)$ for each layer $l = 1, 2, ..., L$ is initialized to $H^{(0)} = X$, the matrix of features for each node (i.e., each row of X corresponds to the features for one node), and follows the propagation rule $H^{(l+1)} = \phi(AH^{(l)}W^{(l)})$. Here the $W(l)$ are parameters to be learned, A is the weighted adjacency matrix, and ϕ is an activation function. Typically, ϕ is chosen as the softmax function in the last layer, so that the output can be interpreted as the probabilities of a node lying in each class, with ReLU activations in the preceding layers. The embedding of each node $i \epsilon V$ at layer $l + 1$ is then

$$h_i^{l+1} = \phi \left(\sum_{j \varepsilon N_i} A_{ij} \, h_j^l \, W^l \right) \tag{1}$$

which can be computed from the previous layer's embedding $h_{(l)}^j$ for each neighbor j and the weights A_{ij} on edges originating at node i. For a GCN with L layers in this form, the output for node i will depend on neighbors up to L steps away (i.e., there exists a path of no more than L edges to node i). We denote this set by N_i^L (note that $i \varepsilon N_i^L$) and refer to these nodes as L-hop neighbors of i.

To solve the node classification problem in federated settings, as described in Figure 1, we consider, as usual in federated learning, a central server with K clients. The graph $G = (V, E)$ is separated across the K clients, each of which has a subgraph $G_k = (V_k, E_k)$. The nodes are disjointly partitioned across clients. The features of nodes in the set V_k can then be represented as the matrix X_k. The cross-client edges of client k, E_k^c, for which the nodes connected by the edge are at different clients, are known to the client k. We use $V_k^{train} \subset V_k$ to denote the set of training nodes with associated labels y_i. The task of federated semi-supervised node classification is then to assign labels to nodes in the remaining set V_k/V_{train}^k for each client k.

Applying GCNs in the federated setting immediately raises a challenge. As seen from 1, in order to find the embedding of the $i - th$ node in the $l - th$ layer, we need the previous layer's embedding $h_j^{(l)}$ for all neighbors of node i. In the federated setting, however, some of these neighbors may be located at other clients, and thus their embeddings must be iteratively sent to the client that contains node i for each layer at every training round. Authors [17] ignore these neighbors, considering only G_k and E_k in training the model, while distributed models [20]; require such communication, which may lead to high overhead and privacy costs. FedGCN provides a communication-efficient method to account for these neighbors.

3.2 Federating Graph Convolutional Networks

In this section, we first introduce our federated training method with communication at the initial step and then outline the corresponding training algorithm.

In the federated learning setting, let $c(i)$ denote the index of the client that contains node i and $W_{c(i)}^{(l)}$ denote the weight matrix of the $l - th$ GCN layer of client $c(i)$. The embedding of node i at layer $l + 1$ is then

$$h_i^{l+1} = \phi \left(\sum_{j \varepsilon N_i} A_{ij} \, h_j^l \, W_{c(i)}^l \right) \tag{2}$$

Note that the weights $W_{c(i)}^{(l)}$ may differ from client to client, due to the local training in federated learning. In practice, GCNs often require only two or three layers for node-level prediction tasks to have sufficient performance. We can then write the computation of a 2-layer federated GCN as

$$\mathbf{\hat{y}_i} = \phi \left(\sum_{j \varepsilon N_i} A_{ij} \phi \left(\sum_{m \varepsilon N_j} A_{jm} \, x_m^T \, W_{c(i)}^{(1)} \right) W_{c(i)}^{(2)} \right) \tag{3}$$

We thus see that, to evaluate this model, it suffices for the client $k = c(i)$ to receive the following messages

$$\sum_{j \varepsilon N_i} A_{ij} x_j, \text{ and } \left\{ \sum_{m \varepsilon N_i} A_{jm} x_j \right\}_{j \epsilon N_{i/i}} \tag{4}$$

which are the feature aggregation of 1-hop and 2-hop neighbors of node i respectively. Note that this information does not change over the course of the model training, as it simply depends on the (fixed) adjacency matrix A and node features x. The client also naturally knows $\{A_{ij}\}_{\forall j \epsilon N_i}$. One way to obtain the above information is to receive the following message from clients z that contain at least one two-hop neighbor of k:

$$\sum_{j \varepsilon N_i} \mathbb{I}_z(c(j)) A_{ij} x_j, \text{ and } \forall_j \epsilon N_i, \sum_{m \varepsilon N_i} \mathbb{I}_z(c(m)) \cdot A_{jm} x_m \tag{5}$$

Here, the indicator $\mathbb{I}_z(c(j))$ is 1 if $z = c(m)$ and zero otherwise. More generally, for a L-layer GCN, each layer requires the following information:

$$\forall_j \varepsilon N_i^L / N_i^{L-1}, \sum_{j \varepsilon N_i} \mathbb{I}_z(c(m)) \cdot A_{jm} x_m \tag{6}$$

Further, the set of edges up to L-1 hops away from the node i, is needed for normalization of A. However, this method requires communication among multiple pairs of clients and suffers privacy leakage in a case when there is only one neighbor node in the client.

To avoid this overhead, we can instead send the aggregation of each client to the central server; the server then calculates the sum of neighbor features of the node i

$$\sum_{j \varepsilon N_i} A_{ij} x_j = \sum_{k=1}^{K} \sum_{j \varepsilon N_i} \mathbb{I}_k(c(j)) \cdot A_{ij} x_j \tag{7}$$

The server can then send the required feature aggregation in 4 back to each client k. Thus, we only need to send the accumulated features of each node's (possibly multi-hop) neighbors, in order to evaluate the GCN model. If there are multiple neighbors stored in other clients, this accumulation serves to protect their individual privacy. For the computation of all nodes V_k stored in client k with an L-layer GCN, the client needs to receive $\left\{\sum_{j \varepsilon N_i} A_{ij} x_j\right\}_{j \epsilon N_{V_k}^L}$ where $N_{V_k}^L$ is the set of L-hop neighbors of nodes V_k.

FedGCN is based on the insight that GCNs require only the accumulated information of the L-hop neighbors of each node at each client, which may be communicated in advance of the training. In practice, however, even this limited communication may be infeasible. If L is too large, then the L-hop neighbors may actually consist of almost the entire graph (many social network graphs have diameters < 10), which might introduce prohibitive storage and communication requirements when there are many clients. Thus, FedGCN is to accommodate three types of communication approximations, according to the most appropriate choice for a given application:

- **No communication (0-hop)**:In some cases, e.g., any communication might reveal private information. In this case, each client simply trains on G_k and ignores cross-client edges, as in prior work.
- **One-hop communication**:If some communication is permissible, we may use accumulation of feature information from nodes one-hop neighbors,$\left\{\sum_{j \varepsilon N_i} A_{ij} x_j\right\}_{i \epsilon i_{V_k}}$ where $N_{V_k}^L$ to approximate the computation of the GCN. Since we only include one-hop neighbors of each node, this is unlikely to introduce significant memory or communication overhead as long as the graph is sufficiently sparse (as is often the case in practice, e.g., in social networks).
- **Two-hop communication**: Many GCNs used in practice for node classification only contain two layers. Thus, communicating the information, $\left\{\sum_{j \varepsilon N_i} A_{ij} x_j\right\}_{i \epsilon i_{V_k}}$ can perfectly recover all neighboring nodes information (i.e., there is no information loss). This choice requires more communication than one-hop communication as there are more two-hop neighbors than one-hop neighbors

3.3 Algorithm

Based on the insights in the previous section, we introduce the FedGCN training algorithm shown in Algorithm 1. The algorithm requires communication between clients and the central server at the initial communication round. Clients first send the encrypted accumulations of local node features to the server. The server then accumulates the neighbor features for each node, as described above. Each client receives and decrypts the feature aggregation of its one-hop or two-hop neighbors. After communication, FedGCN uses the standard FedAvg algorithm [16] to train the models. Specifically, each client computes τ gradient descent steps for local updates. Here we use $w_k^{(t,e)}$, to denote the concatenation of the weights $W_k^{(l)}$ across the L GCN layers, for client k in global training round t and local training step e, and f_k to denote the local loss function, e.g., the cross entropy of the classification estimates. After τ local steps, the local model updates at clients are sent to the central server for the global model update, and the new global model is pushed back to all clients to begin the next training round. The process repeats for T global rounds. Note that this procedure can easily be replaced with other federated learning methods [42], aggregation [43] methods or local update procedures.

Algorithm 1 Federated Training for Graph Convolutional Network

```
1:  for $i \epsilon V$ parallelly do
2:      $\left[\sum_{j \varepsilon N_i} A_{ij} x_j\right] = \sum_{k=1}^{K} \left[\sum_{j \varepsilon N_i} \mathbb{I}_k(c(j)) \cdot A_{ij} x_j\right]$
3:  end for
4:  for each client $k \epsilon K$ parallelly do
5:      if 1-hop then
6:          Receive $\left[\sum_{j \varepsilon N_i} A_{ij} x_j\right]_{i \, \varepsilon \, V_k}$
7:      end if
8:      if 2-hop then
9:          Receive $\left[\sum_{j \varepsilon N_i} A_{ij} x_j\right]_{i \, \varepsilon \, N_{V_k}}$
10:     end if
11: end for
12: // Training Round
13: for $t = 1, ..., T$ do
14:     for each client $k \varepsilon K$ parallelly do
15:         Receive $[[w^t]]$ and decrypt it
16:         Set $w_k^{(t,1)} = w^k$,
17:         for $e = 1, ...., \tau$ do
18:             $w_k^{(t,e+1)} = w_k^{(t,e)} - \eta \, g_{w_k}^{(t,e)}$ // Update Parameters
19:         end for
20:         $\Delta_{w_k}^{(t,\pi)} = w_k^{(t,\pi+1)} - w_k^{(t,1)}$
21:         Send $\left[\left[\Delta_{w_k}^{(t,\pi)}\right]\right]$ to the server
22:     end for
23:     // Update Global Models
24:     $[[w^{(t+1)}]] = [[w^{(t)}]] - [[\Delta_w^t]]$ and broadcast to local clients
25: end for
26: //Communication Round
27: for each client $k \epsilon K$ parallelly do
28:     $\left\{\sum_{j \varepsilon N_i} \mathbb{I}_z(c(j)) \cdot A_{ij} x_j\right\}_{i \epsilon V_k}$
29: end for
```

3.4 Data Distribution across Clients

We assume the number of node label classes is equal to the number of clients, K.

3.4.1 i.i.d.

The data distribution is i.i.d. when nodes are uniformly and randomly assigned to clients with label distribution $[\frac{1}{K}, ..., \frac{1}{K}]^K$.

3.4.2 Non-i.i.d.

The data distribution is non-i.i.d when nodes at a given client have the same class $[1, 0, ..., 0]^K$.

3.4.3 Partial i.i.d.

The data distribution is partial i.i.d when the node label distribution at a given client is a mixture of i.i.d. and non-i.i.d. data, e.g., $[1 - p + \frac{p}{K}, \frac{p}{K} ..., \frac{p}{K}]^K$, where $p \varepsilon [0, 1]$ is the i.i.d control parameter.

4 Experiments

In this section, we present the results of experiments conducted to evaluate the performance of the FedGCN approach on the task of fraud node detection. We describe the experimental setup, including the datasets, evaluation metrics, and comparison methods used in our experiments. We then present the results of our experiments, including a detailed analysis of the performance of our proposed method compared to existing approaches. We also discuss the implications of our results and suggest directions for future work. The source code to reproduce the experimental results will be made publicly available on GitHub[1].

4.1 Experimental Setup

Dataset details. We chose the Amazon [3] dataset to validate FedGCN's performance. The Amazon dataset contains more than 34,000 consumer reviews, from which 11,944 product reviews under the musical instrument category. The statistical information of the dataset is shown in Table 1. Amazon dataset, is composed of users and their comments on musical instruments. Users are modeled as nodes and there are three different types of relations among these nodes. Specifically, under the U-P-U relation, users who have commented on one or more of the same products are linked. The U-S-U relation links users who issued at least one rating (1-5 stars) that was the same within 7 days. Under the last relation, U-V-U, users who have comments ranked in the top 5% in terms of TF-IDF [44] similarity are linked. In addition, all topologies under different relations are merged together to form the relation ALL in Table 1. Users with more than 80% or less than 20% helpful votes are labeled as normal users and fraudsters, respectively.

[1]https://github.com/Shakib-IO/FF-GCN

Table 1: Dataset and graph statistics.

Dataset	Nodes	Fraudsters%	Relations(T)	#Edges
Amazon	11,944	9.5%	U-P-U	175,608
			U-S-U	3,566,479
			U-V-U	1,036,737
			ALL	4,398,392

Implementation Details. For this work, we implement FedGCN with Pytorch [45]. All models were run on a Python 3.8.0 environment, using an NVIDIA GTX 3080 with 10GB GDDR6X RAM and a 2.90GHz Intel Core i7-10700 Linux desktop. We use a two-layer GCN with ReLU activation for the first and Softmax for the second layer. There are 16 hidden units. A dropout layer between the two GCN layers has a dropout rate of 0.5. We use 300 training rounds with the SGD optimizer for all settings with a learning rate of 0.5, L2 regularization 5×10^4 and 3 local steps per round for federated settings. The adjacency matrix is normalized by row $\tilde{A} = D^{-1}A$ to provide equal gradient updates. We set the number of clients to equal the number of classes.

Baselines We compare the performance of five training methods: Centralized GCN assumes a single client has access to the entire graph. It has neither information loss nor communication; Distributed GCN [20] trains GCN in distributed clients which requires communicating node features and hidden states of each layer. FedGCN (0-hop) is equivalent to federated training without communication (FedGraphnn) [17]. BDS-GCN [40] randomly samples cross-client edges in each global training round. It is thus an approximation of FedGCN(1-hop), which communicates the 1-hop neighbors' information across clients to reduce information loss with less communication. FedGCN eliminates information loss for two-layer GCNs by communicating the two-hop neighbors'information across clients.

We also make comparisons of the performance with only graph based models such as GCN [12]; this spectral GNN represents nodes via extending the graph convolution in the spectral domain. GAT [13]; This method employs an attention mechanism and builds a graph attention neural network to enhance embedding performance. GraphSAGE [14]; This is a spatial GNN method that represents nodes inductively based on the nodes' ego features and the aggregated neighbourhood features. GEM [46]; This detector is built from a GCN and is dedicated to malicious account detection. FdGars [15]: Similar to GEM, FdGars is also GCN-based but aimed at spam detection. GraphConsis [29]; This GNN-based fraud detector considers graph inconsistency and is achieved by filtering dissimilar neighbors for nodes based on a fixed threshold. CARE-GNN [6]; This method is camouflage resistant. It filters neighbors for nodes by an adaptive threshold to enhance the graph convolution process. FRAUDRE [4]; which has dual resistance to graph inconsistency and imbalance.

Evaluation Metrics. We adopted evaluation metrics that included the Precision, Recall, F1 Score. To calculate all of the

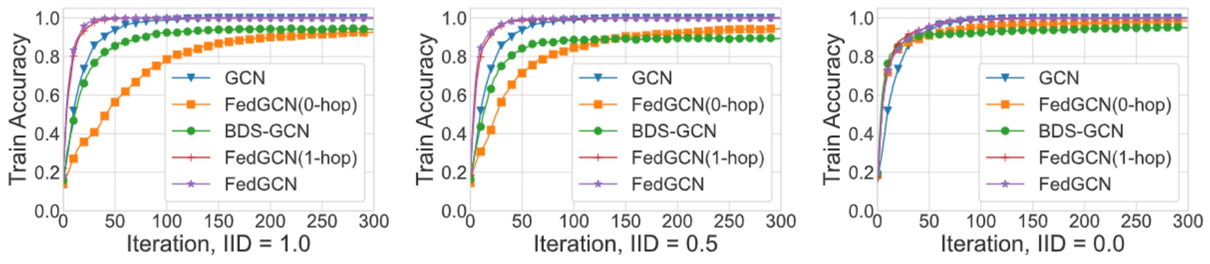

Figure 3: Training accuracy of different algorithms.

metrics, need to find the TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative) values. True Positive indicates that the output of the real class is yes, and the output of the predicted class is also yes, whereas True Negative indicates that the value of the real class is no, and the value of the anticipated class is no. False Negative means the true class is yes but the expected class is no, and False Positive means the true class is no but the predicted class is yes.

The accuracy metrics are the ratio of correct predictions over the total number of predictions evaluated.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{8}$$

Recall or true positive rate (TPR) tells the proportion of actual positive samples that got predicted as positive. To get the value recall using this equation

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

The average of Precision and Recall is denoted as F1 Score. Here is the equation given below

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

4.2 Results and Analysis

In this section, we evaluate and compare the performance of the FedGCN approach. We present the results of experiments on real datasets, using different evaluation metrics.

Effect of Cross-Client Communication We first evaluate our methods under i.i.d., non-i.i.d., and partial (50%) i.i.d. data distributions on the dataset to illustrate FedGCN's performance relative to the centralized and BDS-GCN baselines under different levels of communication. As shown in Figure 3, FedGCN has higher test and training accuracy compared to BDS-GCN in all settings. Similarly to FedGCN(1-hop)'s model accuracy shown in Figure 3. The FedGCN(0-hop) performs worst in the i.i.d. and partial i.i.d. settings, due to information loss from cross-client edges. Under the extreme non-i.i.d.setting, the information loss is small, as there are few cross-client edges. The performance of all algorithms is thus similar, indicating that FedGCN (0-hop) has sufficient information to train a good model.

From table 2, we can see that FedGCN performs best on i.i.d. data, where FFCN (0-hop) has the most information loss.

Table 2: Test accuracy for i.i.d., partial (50% i.i.d.) non-i.i.d., data.

Method	i.i.d	partial	non-i.i.d.
Central	0.764	0.652	0.026
BDS-GCN	0.092	0.298	0.115
FedGCN (0-hop)	0.791	0.365	0.095
FedGCN (1-hop)	0.824	0.683	0.579
FedGCN	0.894	0.562	0.442

FedGCN (1-hop) performs better when the data becomes less i.i.d. Also, all the other methods consistently outperform the BDS-GCN. Additionally, FedGCN obtains its best accuracy at roughly 0.894 when the data is i.i.d. The highest score is achieved by FedGCN(1-hop) for partial and non-i.i.d. data. Our FedGCN approach loses accuracy with increasing non-i.i.d of the data. Overall, FedGCN(1-hop) performs well for parial and non-i.i.d, while FedGCN has demonstrated good performance for i.i.d.

Table 3 reports the performance of our method and all baselines on Amazon dataset. Here, to measure the performance, we are initiating most feasible performance metrics like F1 and Recall percentage. Among the methods, GCN, GAT and GraphSAGE showed almost similar outcomes. 'Recall' score for all 3 approaches is 50% on each and 'F1' score is approximately 47.50% on each of the methods from these three. GEM method showed considerably better performance than the previous three methods in terms of 'Recall' value, as its 'Recall' value is a bit higher (70.66%). But its 'F1' score is a little lower than the previous three approaches (47.42%). FdGars method brought better performance than the previously stated procedures. It's 'Recall' score came as 72.82% and 'F1' score came as 55.14% which are higher than earlier mentioned ones. So, the FdGars method is in a higher-class performance group. Furthermore, if we dig more into the table 3, we can see GraphConsis, CARE-GNN and FRAUDRE, these approaches are also in higher performing groups, as they showed a reasonably better performance. GraphConsis method's 'Recall' value is 85.10%, and 'F1' score is 77.98%, where CARE-GNN's 'Recall' percentage (83.90%) is little lower than GraphConsis, but its 'F1' percentage (88.36%) came out way better than GraphConsis method. FRAUDERE approach is the best among the mentioned approaches according to its resulted performance metrics. This method shows

Table 3: Comparison of various methods and performance(%) on AMAZON Dataset. RECALL, F1

Method	Recall	F1
GCN [12]	50.00	47.51
GAT [13]	50.00	47.50
GraphSAGE [14]	50.00	47.50
GEM [46]	70.66	47.42
FdGars [15]	72.82	55.14
GraphConsis [29]	85.10	77.98
CARE-GNN [6]	83.90	88.36
FRAUDRE [4]	88.61	91.10
FedGCN (*ours*)	**81.23**	**83.94**

88.61% of 'Recall' value, and its 'F1' score came out as 83.94%, which can be said performed way better than the other methods on AMAZON data. Our initiated approach FedGCN showed strong outcomes in terms of performance. FedGCN carried out its performance on AMAZON data based on the structured outcomes mentioned in our model analysis section. We have found 'Recall' value as 81.23%, which is better performing than GCN, GAT, GraphSAGE, GEM and FDGars, and 'F1' score came as 83.94% which is better performing than GCN, GAT, GraphSAGE, GEM, FDGars and GraphConsis. In terms of 'Recall' value GraphConsis, CARE-GNN and FRAUDRE performed better than our proposed FedGCN method, because these models filter dissimilarity between neighbors node before aggregation and resolved the imbalance issue respectively. 'F1' value of our approach (83.94%) is also demonstrating lower performance than CARE-GNN and FRAUDRE methods, but we got a better outcome than GraphConsis method which was better performing in terms of 'Recall' score. So, FedGCN is also can be considered in high-performance class structure for the task. Overall, our proposed FedGCN model is exhibited as a consolidated structure for the task on the AMAZON dataset.

5 Conclusion

We present FedGCN, a framework for training graph convolutional networks in a federated setting for semi-supervised node classification. The FedGCN algorithm is based on the observation that cross-client edges, which are typically ignored in distributed GCN training, can actually contain useful information for the model. Furthermore, FedGCN only requires a single round of communication before training begins, which allows for flexibility in terms of privacy and overhead concerns. We show that FedGCN achieves high accuracy on real-world datasets with significantly less communication than previous algorithms, and that One-hop FedGCN is effective at balancing communication costs. Our analytical framework suggests that new algorithms that selectively communicate edge information may be able to further optimize the trade-off between communication and accuracy.

References

[1] Q. Li, Y. He, C. Xu, F. Wu, J. Gao, and Z. Li, "Dual-augment graph neural network for fraud detection," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 4188–4192, 2022.

[2] S. Rayana and L. Akoglu, "Collective opinion spam detection: Bridging review networks and metadata," in *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*, pp. 985–994, 2015.

[3] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 897–908, 2013.

[4] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "Fraudar: Bounding graph fraud in the face of camouflage," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 895–904, 2016.

[5] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 594–602, SIAM, 2019.

[6] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 315–324, 2020.

[7] H. Peng, J. Li, Y. Song, R. Yang, R. Ranjan, P. S. Yu, and L. He, "Streaming social event detection and evolution discovery in heterogeneous information networks," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 5, pp. 1–33, 2021.

[8] H. Peng, J. Li, Q. Gong, Y. Song, Y. Ning, K. Lai, and P. S. Yu, "Fine-grained event categorization with heterogeneous graph convolutional networks," *arXiv preprint arXiv:1906.04580*, 2019.

[9] R. Qiu, Z. Huang, J. Li, and H. Yin, "Exploiting cross-session information for session-based recommendation with graph neural networks," *ACM Transactions on Information Systems (TOIS)*, vol. 38, no. 3, pp. 1–23, 2020.

[10] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, *et al.*, "Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting," *Information Sciences*, vol. 521, pp. 277–290, 2020.

[11] Y. Liu, Y. Wan, L. He, H. Peng, and S. Y. Philip, "Kg-bart: Knowledge graph-augmented bart for generative commonsense reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 6418–6425, 2021.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[15] J. Wang, R. Wen, C. Wu, Y. Huang, and J. Xiong, "Fdgars: Fraudster detection via graph convolutional networks in online app review system," in *Companion proceedings of the 2019 World Wide Web conference*, pp. 310–316, 2019.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[17] C. He, K. Balasubramanian, E. Ceyani, C. Yang, H. Xie, L. Sun, L. He, L. Yang, P. S. Yu, Y. Rong, *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *arXiv preprint arXiv:2104.07145*, 2021.

[18] X. Ni, X. Xu, L. Lyu, C. Meng, and W. Wang, "A vertical federated learning framework for graph convolutional network," *arXiv preprint arXiv:2106.11593*, 2021.

[19] C. He, E. Ceyani, K. Balasubramanian, M. Annavaram, and S. Avestimehr, "Spreadgnn: Serverless multi-task federated learning for graph neural networks," *arXiv preprint arXiv:2106.02743*, 2021.

[20] S. Scardapane, I. Spinelli, and P. Di Lorenzo, "Distributed training of graph convolutional networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 87–100, 2020.

[21] C. Wan, Y. Li, A. Li, N. S. Kim, and Y. Lin, "Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 673–693, 2022.

[22] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[23] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision support systems*, vol. 50, no. 3, pp. 602–613, 2011.

[24] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 7370–7377, 2019.

[25] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," *arXiv preprint arXiv:1704.04675*, 2017.

[26] A. Li, Z. Qin, R. Liu, Y. Yang, and D. Li, "Spam review detection with graph convolutional networks," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2703–2711, 2019.

[27] W. Chen, Y. Gu, Z. Ren, X. He, H. Xie, T. Guo, D. Yin, and Y. Zhang, "Semi-supervised user profiling with heterogeneous graph attention networks.," in *IJCAI*, vol. 19, pp. 2116–2122, 2019.

[28] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 598–607, IEEE, 2019.

[29] Z. Liu, Y. Dou, P. S. Yu, Y. Deng, and H. Peng, "Alleviating the inconsistency problem of applying graph neural network to fraud detection," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pp. 1569–1572, 2020.

[30] Y. Liu, X. Ao, Z. Qin, J. Chi, J. Feng, H. Yang, and Q. He, "Pick and choose: a gnn-based imbalanced learning approach for fraud detection," in *Proceedings of the Web Conference 2021*, pp. 3168–3177, 2021.

[31] M. Huang, Y. Liu, X. Ao, K. Li, J. Chi, J. Feng, H. Yang, and Q. He, "Auc-oriented graph neural network for fraud detection," in *Proceedings of the ACM Web Conference 2022*, pp. 1311–1321, 2022.

[32] B. Hu, Z. Zhang, J. Zhou, J. Fang, Q. Jia, Y. Fang, Q. Yu, and Y. Qi, "Loan default analysis with multiplex graph learning," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2525–2532, 2020.

[33] F. Shi, Y. Cao, Y. Shang, Y. Zhou, C. Zhou, and J. Wu, "H2-fdetector: A gnn-based fraud detector with homophilic and heterophilic connections," in *Proceedings of the ACM Web Conference 2022*, pp. 1486–1494, 2022.

[34] Z. Qin, Y. Liu, Q. He, and X. Ao, "Explainable graph-based fraud detection via neural meta-graph search," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 4414–4418, 2022.

[35] Z. Zhang, Q. Liu, H. Wang, C. Lu, and C. Lee, "Protgnn: Towards self-explaining graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 9127–9135, 2022.

[36] X. Fu, B. Zhang, Y. Dong, C. Chen, and J. Li, "Federated graph machine learning: A survey of concepts, techniques, and applications," *arXiv preprint arXiv:2207.11812*, 2022.

[37] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "Asfgnn: Automated separated-federated graph neural network," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1692–1704, 2021.

[38] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," *arXiv preprint arXiv:2012.04187*, 2020.

[39] C. Meng, S. Rambhatla, and Y. Liu, "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, 2021.

[40] C. Wan, Y. Li, N. S. Kim, and Y. Lin, "Bds-gcn: Efficient full-graph training of graph convolutional nets with partition-parallelism and boundary sampling," *ICLR 2021 Conference*, 2020.

[41] K. Zhang, C. Yang, X. Li, L. Sun, and S. M. Yiu, "Subgraph federated learning with missing neighbor generation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6671–6682, 2021.

[42] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Advances in neural information processing systems," *30th Annual Conference on Neural Information Processing Systems 2016*, 2020.

[43] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[44] B. Das and S. Chakraborty, "An improved text sentiment classification model using tf-idf and next word negation," *arXiv preprint arXiv:1806.06407*, 2018.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[46] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2077–2085, 2018.