FedGCN: Convergence-Communication Tradeoffs in Federated Training of Graph Convolutional Networks

Yuhang Yao
Carnegie Mellon University
yuhangya@andrew.cmu.edu

Weizhao Jin
University of Southern California
weizhaoj@usc.edu

Srivatsan Ravi
University of Southern California
sravi@isi.edu

Carlee Joe-Wong
Carnegie Mellon University
cjoewong@andrew.cmu.edu

Abstract

Methods for training models on graphs distributed across multiple clients have recently grown in popularity, due to the size of these graphs as well as regulations on keeping data where it is generated. However, the cross-client edges naturally exist among clients. Thus, distributed methods for training a model on a single graph incur either significant communication overhead between clients or a loss of available information to the training. We introduce the Federated Graph Convolutional Network (FedGCN) algorithm, which uses federated learning to train GCN models for semi-supervised node classification with fast convergence and little communication. Compared to prior methods that require extra communication among clients at each training round, FedGCN clients only communicate with the central server in one pre-training step, greatly reducing communication costs and allowing the use of homomorphic encryption to further enhance privacy. We theoretically analyze the tradeoff between FedGCN's convergence rate and communication cost under different data distributions. Experimental results show that our FedGCN algorithm achieves better model accuracy with 51.7% faster convergence on average and at least 100× less communication compared to prior work[1].

1 Introduction

Graph convolutional networks (GCNs) have been widely used for applications ranging from fake news detection in social networks to anomaly detection in sensor networks (Benamira et al., 2019; Zhang et al., 2020). This data, however, can be too large to be trained on a single server, e.g., records of billions of users' website visits. Strict data protection regulations such as the General Data Protection Regulation (GDPR) in Europe and Payment Aggregators and Payment Gateways (PAPG) in India also require that private data only be stored in local clients. In non-graph settings, federated learning has recently shown promise for training models on data that is kept at multiple clients (Zhao et al., 2018; Yang et al., 2021). Some papers have proposed federated training of GCNs (He et al., 2021a; Zhang et al., 2021). Typically, these consider a framework in which each client has access to a subset of a large graph, and clients iteratively compute local updates to a semi-supervised model on their local subgraphs, which are occasionally aggregated at a central server. Figure 1(left) illustrates the federated node classification task of predicting unknown labels of local nodes in each client.

[1] Code in https://github.com/yh-yao/FedGCN

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

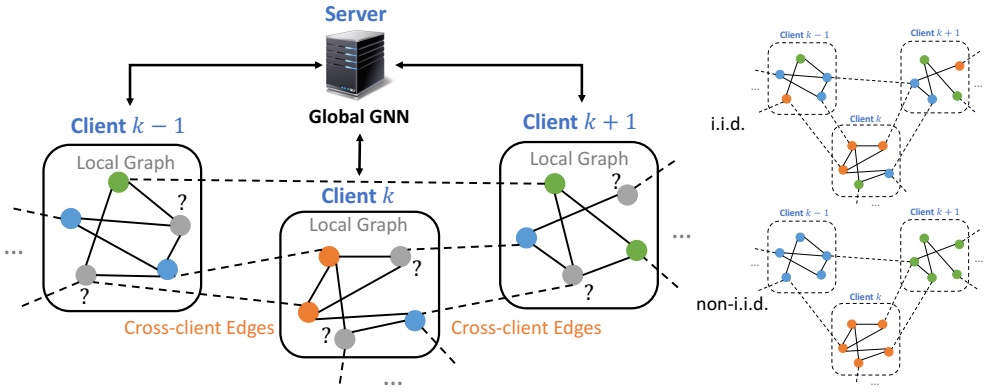

Figure 1: Federated GCN training schematic for node classification, with colors indicating the known labels of some nodes. Nodes in a graph (shown as circles) are distributed across clients, and dashed lines show cross-client edges between nodes at different clients. Arrows in the left figure indicate that each client can exchange updates with a central server during the training process to predict the unknown labels of the grey nodes in each client. At right, we show a graph with i.i.d (top) and non-i.i.d. (bottom) data distribution across clients, which affects the number of cross-client edges.

The main challenge of applying federated learning to GCN training tasks involving a single large graph is that cross-client edges exist among clients. In Figure 1, for example, we see that some edges will connect nodes in different clients. We refer to these as "cross-client edges".

Such cross-client edges typically are stored in both clients. Intuitively, this is due to the fact that edges are generated when nodes at clients interact with each other. Thus, the interaction record, though not personal node characteristics, is then naturally stored at both nodes, i.e., in both clients. For example, a graph may represent buying behaviors (edges) that exist between users (nodes) in two countries (clients). Users in one country want to buy items in another country. The records of these transactions between users in different countries (i.e., the cross-client edges) are then stored in both clients. Due to the General Data Protection Regulation, however, sensitive user information (node features including name, zip code, gender, birthday, credit card number, email address, etc.) cannot be stored in another country. Yet these cross-client edges cannot be ignored: including cross-country transactions (cross-client edges) is key for training models that detect international money laundering and fraud. Another example is online social applications like Facebook and LinkedIn. Users in different countries can build connections with each other (e.g., a person in the United States becoming Facebook friends with a person in China). The users in both the U.S. and China would then have a record of this friendship link, while the personal user information cannot be shared across countries.

However, GCNs require information about a node's neighbors to be aggregated in order to construct an embedding of each node that is used to accomplish tasks such as node classification and link prediction. Ignoring the information from neighbors located at another client, as in prior federated graph training algorithms (Wang et al., 2020a; He et al., 2021b), may then result in less accurate models due to loss of information from nodes at other clients.

Prior works on federated or distributed graph training reduce cross-client information loss by communicating information about nodes' neighbors at other clients in each training round (Scardapane et al., 2020; Wan et al., 2022; Zhang et al., 2021), which can introduce significant communication overhead and reveal private node information to other clients. We instead realize that the information needed to train a GCN only needs to be communicated once, before training. This insight allows us to further alleviate the privacy challenges of communicating node information between clients (Zhang et al., 2021). Specifically, we leverage Homomorphic Encryption (HE), which can preserve client privacy in federated learning but introduces significant overhead for each communication round; with only one communication round, this overhead is greatly reduced. Further, in practice each client may contain several node neighbors, e.g., clients might represent social network data in different countries, which cannot leave the country due to privacy regulations. Each client would then receive aggregated feature information about all of a node's neighbors in a different country, which itself can help preserve privacy through accumulation across multiple nodes. In the extreme case when nodes only have one cross-client neighbor, we can further integrate differential privacy techniques (Wei et al.,

2020). We propose the **FedGCN algorithm** for distributed GCN training based on these insights. FedGCN greatly reduces communication costs and speeds up convergence without information loss, compared with existing distributed settings (Scardapane et al., 2020; Wan et al., 2022; Zhang et al., 2021)

In some settings, we can further reduce FedGCN's required communication without compromising the trained model's accuracy. First, GCN models for node classification rely on the fact that nodes of the same class will have more edges connecting them, as shown in Figure 1(right). If nodes with each class tend to be concentrated at a single client, a version of the non-i.i.d. (non-independent and identically distributed) data often considered in federated learning, then ignoring cross-client edges discards little information, and FedGCN's communication round may be unnecessary. The model, however, may not converge, as federated learning may converge poorly when client data is non-i.i.d. (Zhao et al., 2018). Second, GCN models with multiple layers require accumulated information from nodes that are multiple hops away from each other, introducing greater communication overhead. However, such multi-hop information may not be needed in practice.

We analytically quantify the convergence rate of FedGCN with various degrees of communication, under both i.i.d. and non-i.i.d. client data. To the best of our knowledge, we are the first to analytically illustrate the resulting tradeoff between a fast convergence rate (which intuitively requires more information from cross-client edges) and low communication cost, which we do by considering a stochastic block model (Lei and Rinaldo, 2015; Keriven et al., 2020) of the graph topology. We thus quantify when FedGCN's communication significantly accelerates the GCN's convergence.

In summary, our work has the following **contributions:**

- We introduce FedGCN, an efficient framework for federated training of GCNs to solve node-level prediction tasks where distributed training incurs high communication cost and privacy leakage, which also leverages Fully Homomorphic Encryption for enhanced privacy guarantees.
- We theoretically analyze the convergence rate and communication cost of FedGCN compared to prior methods, as well as its dependence on the data distribution. We can thus quantify the usefulness of communicating different amounts of cross-client information.
- Our experiments on both synthetic and real-world datasets demonstrate that FedGCN outperforms existing distributed GCN training methods in most cases with exact model computation, higher accuracy, and orders-of-magnitude (e.g. $100\times$) less communication cost.

We outline related works in Section 2 before introducing the problem of node classification in graphs in Section 3. We then introduce FedGCN in Section 4 and analyze its performance theoretically (Section 5) and experimentally (Section 6) before concluding in Section 7.

2 Related Work

Graph neural networks aim to learn representations of graph-structured data (Bronstein et al., 2017). GCNs (Kipf and Welling, 2016), GraphSage (Hamilton et al., 2017), and GAT (Veličković et al., 2017) perform well on node classification and link prediction. Several works provide a theoretical analysis of GNNs based on the Stochastic Block Model (Zhou and Amini, 2019; Lei and Rinaldo, 2015; Keriven et al., 2020). We similarly adopt the SBM to quantify FedGCN's performance.

Federated learning was first proposed in McMahan et al. (2017)'s widely adopted FedAvg algorithm, which allows clients to train a model via coordination with a central server while keeping training data at local clients. However, FedAvg may not converge if data from different clients is non-i.i.d. (Zhao et al., 2018; Li et al., 2019a; Yang et al., 2021). We show similar results for federated graph training.

Federated learning on graph neural networks is a topic of recent interest (He et al., 2021a). Unlike learning tasks in which multiple graphs each constitute a separate data sample and are distributed across clients (e.g., graph classification (Zhang et al., 2018), image classification (Li et al., 2019b), and link prediction (Yao et al., 2023), FedGCN instead considers semi-supervised tasks on a single large graph (e.g., for node classification). Existing methods for such tasks generally ignore the resulting cross-client edges (He et al., 2021a). Scardapane et al. (2020)'s distributed GNN proposes a training algorithm communicating the neighbor features and intermediate outputs of GNN layers among clients with expensive communication costs. BDS-GCN (Wan et al., 2022) proposes to sample

cross-client neighbors. These methods may violate client privacy by revealing per-node information to other clients. FedSage+ (Zhang et al., 2021) 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.

All of the above works further require communication at every training round, while FedGCN enables the private recovery of cross-client neighbor information with a single, pre-training communication round that utilizes HE. We also provide theoretical bounds on FedGCN's convergence.

3 Federated Semi-Supervised Node Classification

In this section, we formalize the problem of node classification on a single graph and introduce the federated setting in which we aim to solve this problem.

We consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = [N]$ is the set of N nodes and $\mathcal{E}$ is the set of edges. The graph is equivalent to a weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, where $\boldsymbol{A}_{ij}$ indicates the weight of an edge from node i to node j (if the edge does not exist, the weight is zero). Every node $i \in \mathcal{V}$ has a feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$, where d represents the number of input features. Each node i in a subset $\mathcal{V}^{train} \subset \mathcal{V}$ has a corresponding label y_i used during training. Semi-supervised node classification aims to assign labels to nodes in the remaining set $\mathcal{V} \backslash \mathcal{V}^{train}$, based on their feature vectors and edges to other nodes. We train a GCN model to do so.

GCNs (Kipf and Welling, 2016) 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 $\boldsymbol{H}^{(l)}$ for each layer $l = 1, 2, \ldots, L$ is initialized to $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, the matrix of features for each node (i.e., each row of $\boldsymbol{X}$ corresponds to the features for one node), and follows the propagation rule $\boldsymbol{H}^{(l+1)} = \phi(\boldsymbol{A}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$. Here $\boldsymbol{W}^{(l)}$ are parameters to be learned, $\boldsymbol{A}$ 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 $i \in \mathcal{V}$ at layer $l + 1$ is then

$$\boldsymbol{h}_i^{(l+1)} = \phi\left(\sum_{j \in \mathcal{N}_i} \boldsymbol{A}_{ij} \boldsymbol{h}_j^{(l)} \boldsymbol{W}^{(l)}\right), \tag{1}$$

which can be computed from the previous layer's embedding $\boldsymbol{h}_j^{(l)}$ for each neighbor j and the weight $\boldsymbol{A}_{ij}$ on edges from node i to node j. 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 $\mathcal{N}_i^L$ (note that $i \in \mathcal{N}_i^L$) and refer to these nodes as L-hop neighbors of i.

To solve the node classification problem in **federated settings** (Figure 1), we consider, as usual in federated learning, a central server with K clients. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is separated across the K clients, each of which has a sub-graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$. Here $\bigcup_{k=1}^K \mathcal{V}_k = \mathcal{V}$ and $\mathcal{V}_i \bigcap \mathcal{V}_j = \varnothing$ for $\forall i \neq j \in [K]$, i.e., the nodes are disjointly partitioned across clients. The features of nodes in the set $\mathcal{V}_k$ can then be represented as the matrix $\boldsymbol{X}_k$. The cross-client edges of client k, $\mathcal{E}_k^c$, for which the nodes connected by the edge are at different clients, are known to the client k. We use $\mathcal{V}_k^{train} \subset \mathcal{V}_k$ 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 $\mathcal{V}_k \backslash \mathcal{V}_k^{train}$ for each client k.

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 $\boldsymbol{h}_j^{(l)}$ 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. He et al. (2021a) ignore these neighbors, considering only $\mathcal{G}_k$ and $\mathcal{E}_k$ in training the model, while Scardapane et al. (2020); Wan et al. (2022); Zhang et al. (2021) require such communication, which may lead to high overhead and privacy costs. FedGCN provides a communication-efficient method to account for these neighbors.

4 Federated Graph Convolutional Network

In order to overcome the challenges outlined in Section 3, we propose our Federated Graph Convolutional Network (FedGCN) algorithm. In this section, we first introduce our federated training method with communication at the initial step and then outline the corresponding training algorithm.

Federating Graph Convolutional Networks. In the federated learning setting, let $c(i)$ denote the index of the client that contains node i and $\boldsymbol{W}_{c(i)}^{(l)}$ 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 $\boldsymbol{h}_i^{(l+1)} = \phi\left(\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{h}_j^{(l)}\boldsymbol{W}_{c(i)}^{(l)}\right)$.

Note that the weights $\boldsymbol{W}_{c(i)}^{(l)}$ may differ from client to client, due to the independent local training in federated learning. For example, we can then write the computation of a 2-layer federated GCN as $\hat{\boldsymbol{y}}_i = \phi\left(\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\phi\left(\sum_{m\in\mathcal{N}_j} \boldsymbol{A}_{jm}\boldsymbol{x}_m^T\boldsymbol{W}_{c(i)}^{(1)}\right)\boldsymbol{W}_{c(i)}^{(2)}\right)$. To evaluate this 2-layer model, it then suffices for the client $k=c(i)$ to receive the message $\sum_{m\in\mathcal{N}_j} \boldsymbol{A}_{jm}\boldsymbol{x}_m^T$. We can write these messages as

$$\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j, \text{ and } \left\{\sum_{m\in\mathcal{N}_j} \boldsymbol{A}_{jm}\boldsymbol{x}_m\right\}_{j\in\mathcal{N}_i/i}, \tag{2}$$

which are the feature aggregations of 1- and 2-hop neighbors of node i respectively. This information does not change with the model training, as it simply depends on the (fixed) adjacency matrix $\boldsymbol{A}$ and node features $\boldsymbol{x}$. The client also naturally knows $\{\boldsymbol{A}_{ij}\}_{\forall j\in\mathcal{N}_i}$, which is included in $\mathcal{E}_k \bigcup \mathcal{E}_k^c$.

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:

$$\sum_{j\in\mathcal{N}_i} \mathbb{I}_z(c(j))\boldsymbol{A}_{ij}\boldsymbol{x}_j, \text{ and } \forall j\in\mathcal{N}_i, \sum_{m\in\mathcal{N}_j} \mathbb{I}_z(c(m))\cdot\boldsymbol{A}_{jm}\boldsymbol{x}_m. \tag{3}$$

Here the indicator $\mathbb{I}_z(c(m))$ is 1 if $z=c(m)$ and zero otherwise. More generally, for a L-layer GCN, each layer requires $\forall j\in\mathcal{N}_i^L/\mathcal{N}_i^{L-1}, \sum_{m\in\mathcal{N}_j}\mathbb{I}_z(c(m))\cdot\boldsymbol{A}_{jm}\boldsymbol{x}_m$. Further, $\mathcal{E}_i^{L-1}$, i.e., the set of edges up to $L-1$ hops away from node i, is needed for normalization of A.

To avoid the overhead of communicating between multiple pairs of clients, which can also induce privacy leakage when there is only one neighbor node in the client, we can instead send the aggregation of each client to the central server. In the 2-layer GCN example, the server then calculates the sum of neighbor features of node i as $\sum_{j\in\mathcal{N}_i}\boldsymbol{A}_{ij}\boldsymbol{x}_j = \sum_{k=1}^K\sum_{j\in\mathcal{N}_i}\mathbb{I}_k(c(j))\cdot\boldsymbol{A}_{ij}\boldsymbol{x}_j$. The server can then send the required feature aggregation in (2) 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[2]. For the computation of all nodes $\mathcal{V}_k$ stored in client k with an L-layer GCN, the client needs to receive $\{\sum_{j\in\mathcal{N}_i}\boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{N}_{\mathcal{V}_k}^L}$, where $\mathcal{N}_{\mathcal{V}_k}^L$ is the set of L-hop neighbors of nodes $\mathcal{V}_k$.

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

- **No communication (0-hop):** If any communication is unacceptable, e.g., due to overhead, each client simply trains on $\mathcal{G}_k$ and ignores cross-client edges, as in prior work.
- **One-hop communication:** If some communication is permissible, we may use the accumulation of feature information from nodes' 1-hop neighbors, $\{\sum_{j\in\mathcal{N}_i}\boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{V}_k}$, to approximate the GCN computation. 1-hop neighbors are unlikely to introduce significant memory or communication overhead as long as the graph is sparse, e.g. social networks.

[2]In the extreme case when the node only has one neighbor stored in other clients, we can drop the neighbor, which likely has minimal effect on model performance, or add differential privacy to the communicated data.

- **Two-hop communication:** To further improve model performance, we can communicate the information from 2-hop neighbors, $\{\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{N}_{\mathcal{V}_k}}$ and perform the aggregation of L-layer GCNs. As shown in Figure 2, the 2-hop approximation does not decrease model accuracy in practice compared to L-hop communication for L-hop GCNs, up to $L \leq 10$.

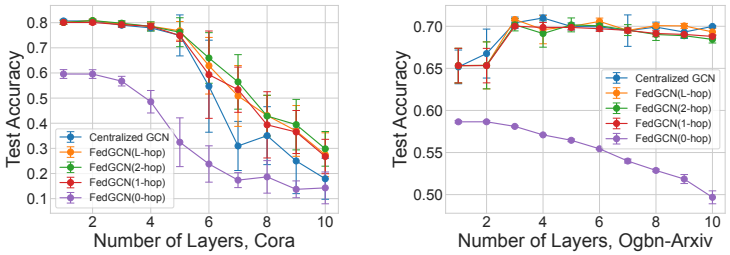

Figure 2: Test Accuracy of GCNs with different numbers of layers, using centralized and FedGCN training on Cora (left) and Ogbn-Arxiv (right) datasets with 10 clients and non-i.i.d partition. Communicating 2-hop information in FedGCN is sufficient for up to 10-layer GCNs. In Ogbn-Arxiv, BatchNorm1d is added between GCN layers to ensure consistent performance (Hu et al., 2020).

Why Not L-hop Communication? Although FedGCN supports L-hop communication, communicating L-hop neighbors across clients requires knowledge of the $L-1$ hop neighborhood graph structures, which may incur privacy leakage. If L is large enough, L-hop neighbors may also cover the entire graph, incurring prohibitive communication and computation costs. Thus, in practice we restrict ourselves to 2-hop communication, which only requires knowledge of 1-hop neighbors. Indeed, Figure 2 shows that even on the Ogbn-Arxiv dataset, which has more than one million edges, adding more layers and k-hop communication does not increase model accuracy for $L \geq 3$ and $k \geq 2$. Thus, it is reasonable to use 0, 1, or 2-hop communication with $L \leq 3$-layer GCNs in practice.

Secure Neighbor Feature Aggregation. To guarantee privacy during the aggregation process of accumulated features, we leverage Homomorphic Encryption (HE) to construct a secure neighbor feature aggregation function. HE (Brakerski et al., 2014; Cheon et al., 2017) allows a computing party to perform computation over ciphertext without decrypting it, thus preserving the plaintext data.

The key steps of the process can be summarized as follows: (i) all clients agree on and initialize a HE keypair, (ii) each client encrypts the local neighbor feature array and sends it to the server, and (iii) upon receiving all encrypted neighbor feature arrays from clients, the server performs secure neighbor feature aggregation $\left[\!\left[\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j\right]\!\right] = \sum_{k=1}^{K}\left[\!\left[\sum_{j\in\mathcal{N}_i} \mathbb{I}_k(c(j))\cdot\boldsymbol{A}_{ij}\boldsymbol{x}_j\right]\!\right]$, where $[\![\cdot]\!]$ represents the encryption function. The server then distributes the aggregated neighbor feature array to each client, and (iv) upon receiving the aggregated neighbor feature array, each client decrypts it and moves on to the model training phase. We can also use HE for a secure model gradient aggregation function during the model's training rounds, which provides extra privacy guarantees.

Since model parameters are often floating point numbers and node features can be binary (e.g., one-hot indicators), a naïve HE scheme would use CKKS (Cheon et al., 2017) for parameters and integer schemes such as BGV (Brakerski et al., 2014) for features. To avoid the resulting separate cryptographic setups, we adopt CKKS with a rounding procedure for integers and also propose an efficient HE file optimization, Boolean Packing, that packs arrays of boolean values into integers and optimizes the cryptographic communication overhead. The encrypted features then only require twice the communication cost of the raw data, compared to 20x overhead with general encryption.

Training Algorithm. Based on the insights in the previous section, we introduce the FedGCN training algorithm (details are provided in Appendix A):

- **Pretraining Communication Round** The algorithm requires communication between clients and the central server at the initial communication round.
  1. Each client k sends its encrypted accumulations of local node features, $[\![\{\sum_{j\in\mathcal{N}_i} \mathbb{I}_k(c(j))\cdot\boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{V}_k}]\!]$, to the server.
  2. The server then accumulates the neighbor features for each node i, $[\![\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j]\!] = \sum_{k=1}^{K}[\![\sum_{j\in\mathcal{N}_i} \mathbb{I}_k(c(j))\cdot\boldsymbol{A}_{ij}\boldsymbol{x}_j]\!]$.

3. Each client receives and decrypts the feature aggregation of its one-hop, $[\![\{\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{V}_k}]\!]$, and if needed two-hop, neighbors $[\![\{\sum_{j\in\mathcal{N}_i} \boldsymbol{A}_{ij}\boldsymbol{x}_j\}_{i\in\mathcal{N}_{\mathcal{V}_k}}]\!]$.

- **Federated Aggregation** After communication, FedGCN uses the standard FedAvg algorithm McMahan et al. (2017) to train the models. Other federated learning methods, e.g., as proposed by Reddi et al. (2020); Fallah et al. (2020), can easily replace this procedure.

5 FedGCN Convergence and Communication Analysis

In this section, we theoretically analyze the convergence rate and communication cost of FedGCN for i.i.d. and non-i.i.d. data with 0-, 1-, and 2-hop communication.

We first give a formal definition of the i.i.d. and non-i.i.d. data distributions, using distribution-based label imbalance (Hsu et al., 2019; Li et al., 2022). Figure 3 visualizes eight example data distributions across clients. For simplicity, we assume the number of clients K exceeds the number of node label classes C, though Section 6's experiments support any number of clients. We also assume that each class contains the same number of nodes and that each client has the same number of nodes.

Definition 5.1. Each client k's *label distribution* is defined as $[p_1 \quad p_2 \quad \dots \quad p_C] \in \mathbb{R}^C$, where p_c denotes the fraction of nodes of class c at client k and $\sum_c p_c = 1$.

Definition 5.2. Clients' data distributions are *i.i.d.* when nodes are uniformly and randomly assigned to clients, i.e., each client's label distribution is $[1/C \quad \dots \quad 1/C]$. Otherwise, they are *non i.i.d.*

Non-i.i.d. distributions include that of McMahan et al. (2017) in which $p_i = 1 - p + \frac{p}{C}$ for some $i = 1, 2, \dots, C$ and $\frac{p}{C}$ otherwise, where $p \in [0, 1]$ is a control parameter; or the Dirichlet distribution (Hsu et al., 2019) $Dir(\beta/C)$, where $\beta \geq 0$ is a control parameter. With these distributions, each client has one dominant class. If $p = 0$ or $\beta \to \infty$, all nodes at a client have the same class.

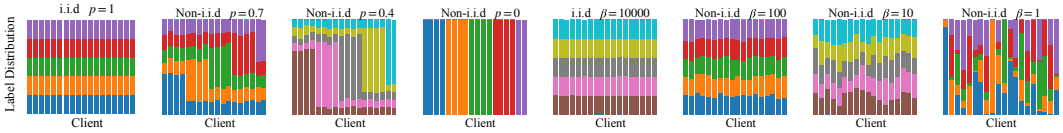

Figure 3: Example client data distributions. Different colors represent different node label classes. Client data is generated from the data distribution with different parameters p and β respectively.

We use $\|\boldsymbol{x}\|$ to denote the ℓ_2 norm if $\boldsymbol{x}$ is a vector, and the Frobenius norm if $\boldsymbol{x}$ is a matrix. Given model parameters $\boldsymbol{w}$ and K clients, we define the local loss function $f_k(\boldsymbol{w})$ for each client k, and the global loss function $f(\boldsymbol{w}) = \sum_{k=1}^{K} f_k(\boldsymbol{w})$, which has minimum value f^*.

Assumption 5.3. (λ-Lipschitz Continuous Gradient) There exists a constant $\lambda > 0$, such that $\|\nabla f_k(\boldsymbol{w}) - \nabla f_k(\boldsymbol{v})\| \leq \lambda\|\boldsymbol{w} - \boldsymbol{v}\|$, $\forall \boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$, and $k \in [K]$.

Assumption 5.4. (Bounded Global Variability) There exists a constant $\sigma_G \geq 0$, such that the global variability of the local gradients of the cost function $\|\nabla f_k(\boldsymbol{w_t}) - \nabla f(\boldsymbol{w_t})\| \leq \sigma_G$, $\forall k \in [K]$, $\forall t$[3]

Assumption 5.5. (Bounded Gradient) There exists a constant $G \geq 0$, such that the local gradients of the cost function $\|\nabla f_k(\boldsymbol{w_t})\| \leq G$, $\forall k \in [K]$, $\forall t$.

Assumptions 5.3, 5.4 and 5.5 are standard in the federated learning literature (Li et al., 2019a; Yu et al., 2019; Yang et al., 2021). We consider a two-layer GCN, though our analysis can be extended to more layers. We work from Yu et al. (2019)'s convergence result for FedAvg[4] to find:

Theorem 5.6. *(Convergence Rate for FedGCN) Under the above assumptions, while training with K clients, T global training rounds, E local updates per round, and a learning rate $\eta \leq \frac{1}{\lambda}$, we have*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\nabla f\left(\boldsymbol{w}_{t-1}\right)\right\|^2\right] \leq \frac{2}{\eta T}\left(f\left(\boldsymbol{w}_0\right) - f^*\right) + \frac{\lambda}{K}\eta\|I_{local} - I_{glob}\|^2 + 4\eta^2E^2G^2\lambda^2, \quad (4)$$

where f^* is the minimum value of f, $I_{local} = K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{A}_k\boldsymbol{X}_k$, $I_{glob} = \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}\boldsymbol{X}$.

[3]Clients with i.i.d. label distributions may still have global variability $\sigma_G > 0$ due to having finite samples.

[4]Our analysis applies to any federated training algorithm with bounded global variability (Yang et al., 2021).

	Non-i.i.d.	i.i.d.
0-hop	$(1-\frac{1}{K^4})\frac{N^5}{C^5}\|B^4\|+(1-\frac{1}{C})^{\frac{5}{2}}(1-p)^5$	$(1-\frac{1}{K^4})\frac{N^5}{C^5}\|B^4\|$
1-hop	$(1-\frac{1}{K^4}(1+c_\alpha p+c_\mu)^2)\frac{N^5}{C^5}\|B^4\|+(1-\frac{1}{C})^{\frac{5}{2}}(1-p)^5$	$(1-\frac{1}{K^4}(1+c_\alpha+c_\mu)^2)\frac{N^5}{C^5}\|B^4\|$
2-hop	$(1-\frac{1}{K^4}(1+c_\alpha p+c_\mu)^6)\frac{N^5}{C^5}\|B^4\|+(1-\frac{1}{C})^{\frac{5}{2}}(1-p)^5$	$(1-\frac{1}{K^4}(1+c_\alpha+c_\mu)^6)\frac{N^5}{C^5}\|B^4\|$

Table 1: Convergence rate bounds of FedGCN with the Stochastic Block Model and data distribution from McMahan et al. (2017) We define $c_\alpha = \frac{(1-\mu)\alpha N(K-1)}{CK}$ and $c_\mu = \frac{\mu\alpha N(K-1)}{K}$ for the SBM; αN is a constant, $c_\alpha \gg c_\mu$ and $c_\mu \simeq 0$. More hops speed up the convergence from the order of 2 to 6 (highlighted as purple and green). Communication helps more when data is more i.i.d with factor $c_\alpha p$ (highlighted as blue). Non-i.i.d. data implies a longer convergence time with factor $(1-p)^5$.

The convergence rate is thus bounded by the difference of the information provided by local and global graphs $\|I_{local} - I_{glob}\|$, which upper bounds the global variability $\|\nabla f_k(\boldsymbol{w}) - \nabla f(\boldsymbol{w})\|$. By 1- and 2-hop communication, the local graph $\boldsymbol{A}_k$ is closer to $\boldsymbol{A}$, resulting in faster convergence.

Table 1 bounds $\|I_{local} - I_{glob}\|$ for FedGCN trained on an SBM (stochastic block model) graph, in which we assume N nodes with C classes. Nodes in the same (different) class have an edge between them with probability α ($\mu\alpha$). Appendix B.3 details the full SBM. Appendix G derives Table 1's results; the SBM structure allows us to develop intuitions about FedGCN's convergence with different hops, simply by knowing the node features and graph adjacency matrix (i.e., without knowing the model). Appendix H derives corresponding expressions for the required communication costs. We validate these results in experiments and Appendix E.1, and make the following **key observations**:

- **Faster convergence with more communication hops**: 1-hop and 2-hop communication accelerate the convergence with factors $(1+c_\alpha p+c_\mu)^2$ and $(1+c_\alpha p+c_\mu)^6$, respectively. When the i.i.d control parameter p increases, the difference among no (0-hop), 1-hop, and 2-hop communication increases: communication helps more when data is more i.i.d.
- **More hops are needed for cross-device FL**: When the number of clients K increases, as in cross-device federated learning (FL), the convergence takes longer by a factor $\frac{1}{K^4}$, but 2-hop communication can recover more edges in other clients to speed up the training.
- **One-hop is sufficient for cross-silo FL**: If the number of clients K is small, approximation methods via one-hop communication can balance the convergence rate and communication overhead. We experimentally validate this intuition in the next section.

6 Experimental Validation

We experimentally show that FedGCN converges to a more accurate model with less communication compared to previously proposed algorithms. We further validate Section 5's theoretical observations.

6.1 Experiment Settings

We use the Cora (2708 nodes, 5429 edges), Citeseer (3327 nodes, 4732 edges), Ogbn-ArXiv (169343 nodes, 1166243 edges), and Ogbn-Products (2449029 nodes, 61859140 edges) datasets to predict document labels and Amazon product categories (Wang et al., 2020b; Hu et al., 2020).

Methods Compared. Centralized GCN assumes a single client has access to the entire graph. **Distributed GCN** (Scardapane et al., 2020) trains GCN in distributed clients, which requires communicating node features and hidden states of each layer. **FedGCN (0-hop)** (Section 4) is equivalent to federated training without communication (**FedGraphnn**) (Wang et al., 2020a; Zheng et al., 2021; He et al., 2021a). **BDS-GCN** (Wan et al., 2022) randomly samples cross-client edges in each global training round, while **FedSage+** (Zhang et al., 2021) recovers missing neighbors by learning a linear predictor based on the node embedding, using cross-client information in each training round. It is thus an approximation of **FedGCN (1-hop)**, which communicates the 1-hop neighbors' information across clients. **FedGCN (2-hop)** communicates the two-hop neighbors' information across clients.

We consider the Dirichlet label distribution with parameter β, as shown in Figure 3. For Cora and Citeseer, we use a 2-layer GCN with Kipf and Welling (2016)'s hyper-parameters. For Ogbn-Arxiv and Ogbn-Products, we respectively use a 3-layer GCN and a 2-layer GraphSage with Hu et al.

(2020)'s hyper-parameters. We average our results over 10 experiment runs. Detailed experimental setups and extended results, including an evaluation of the HE overhead, are in Appendix E.2 and C.

6.2 Effect of Cross-Client Communication

We first evaluate our methods under i.i.d. ($\beta = 10000$) and non-i.i.d. ($\beta = 100, 1$) Dirichlet data distributions on the four datasets to illustrate FedGCN's performance relative to the centralized, BDS-GCN, and FedSage+ baselines under different levels of communication.

	Cora, 10 clients			Citeseer, 10 clients		
Centralized GCN	0.8069±0.0065			0.6914±0.0051		
	β = 1	β = 100	β = 10000	β = 1	β = 100	β = 10000
FedGCN(0-hop)	0.6502±0.0127	0.5958±0.0176	0.5992±0.0226	0.617±0.0118	0.5841±0.0168	0.5841±0.0138
BDS-GCN	0.7598±0.0143	0.7467±0.0117	0.7479±0.018	0.6709±0.0184	0.6596±0.0128	0.6582±0.01
FedSage+	0.8026±0.0054	0.7942±0.0075	0.796±0.0075	0.6977±0.0097	0.6856±0.0121	0.688±0.0086
FedGCN(1-hop)	**0.81±0.0066**	0.8009±0.007	0.8009±0.0077	**0.7006±0.0071**	0.6891±0.0067	0.693±0.0069
FedGCN(2-hop)	0.8064±0.0043	**0.8084±0.0051**	**0.8087±0.0061**	0.6933±0.0067	**0.6953±0.0069**	**0.6948±0.0032**
	Ogbn-Arxiv, 10 clients			Ogbn-Products, 5 clients		
Centralized GCN	0.7±0.0082			0.7058±0.0008		
	β = 1	β = 100	β = 10000	β = 1	β = 100	β = 10000
FedGCN(0-hop)	0.5981±0.0094	0.5809±0.0017	0.5804±0.0015	0.6789±0.0031	0.658±0.0008	0.658±0.0008
BDS-GCN	0.6769±0.0086	0.6689±0.0024	0.6688±0.0015	0.6996±0.0019	0.6952±0.0012	0.6952±0.0009
FedSage+	0.7053±0.0073	0.6921±0.0014	0.6918±0.0024	0.7044±0.0017	0.7047±0.0009	0.7051±0.0006
FedGCN(1-hop)	0.7101±0.0078	**0.6989±0.0038**	0.7004±0.0031	0.7049±0.0016	**0.7057±0.0003**	**0.7057±0.0004**
FedGCN(2-hop)	**0.712±0.0075**	0.6972±0.0075	**0.7017±0.0081**	**0.7053±0.002**	**0.7057±0.0009**	0.7055±0.0006

Table 2: Test Accuracy on four datasets, for i.i.d.($\beta = 10000$) and non-i.i.d. ($\beta = 100, 1$) data. FedGCN (1-hop,2-hop) performs best on i.i.d. and non-i.i.d. data, and FedGCN (0-hop) has the most information loss. We assume FedSage+'s linear approximator perfectly learns neighbor information.

As shown in Table 2, FedGCN(1-, 2-hop) has higher test accuracy than FedSage+ and BDS-GCN, reaching the same test accuracy as centralized training in all settings. FedGCN(1-, 2-hop) is able to converge faster to reach the same accuracy with the optimal number of hops depending on the data distribution. In such a cross-silo setting (10 clients), FedGCN(1-hop) achieves a good tradeoff between communication and model accuracy. FedGCN (0-hop) performs worst in the i.i.d. and non-i.i.d. settings, due to information loss from cross-client edges.

Why Non-i.i.d Has Better Accuracy in 0-hop? In Table 2 with 10 clients, 0-hop has better accuracy since non-i.i.d. has fewer cross-client edges (less information loss) than i.i.d as in theoretical analysis.

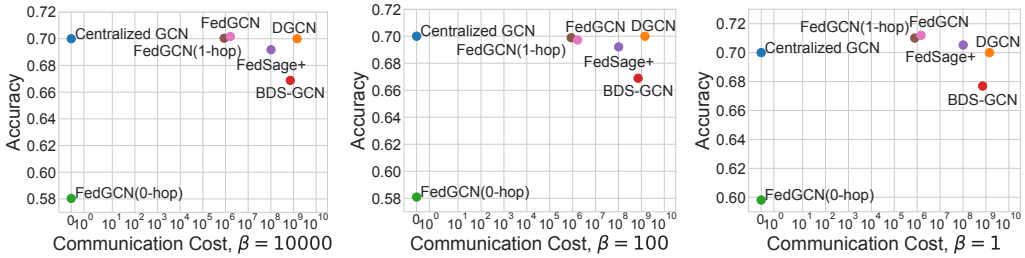

Figure 4: Test accuracy vs. communication cost until convergence of algorithms in the i.i.d., partial-i.i.d. and non-i.i.d. settings for the OGBN-ArXiv dataset. FedGCN uses orders of magnitude less communication (at least 100×) than BDS-GCN and FedSage+, while achieving higher test accuracy.

Communication Cost vs. Accuracy. Figure 4 shows the communication cost and test accuracy of different methods on the OGBN-ArXiv dataset. FedGCN (0-, 1-, and 2-hop) requires little communication with high accuracy, while Distributed GCN, BDS-GCN and FedSage+ require communication at every round, incurring over 100× the communication cost of any of FedGCN's variants. FedGCN (0-hop) requires much less communication than 1- and 2-hop FedGCN, but has lower accuracy due to information loss, displaying a convergence-communication tradeoff. Both 1- and 2-hop FedGCN achieve similar accuracy as centralized GCN, indicating that a 1-hop approximation of cross-client edges is sufficient in practice to achieve an accurate model.

Cross-Silo and Cross-Device Training. As shown in Figure 5 (left), 1-hop and 2-hop communication achieve similar test accuracy in the cross-silo setting with few clients, though 1-hop communication

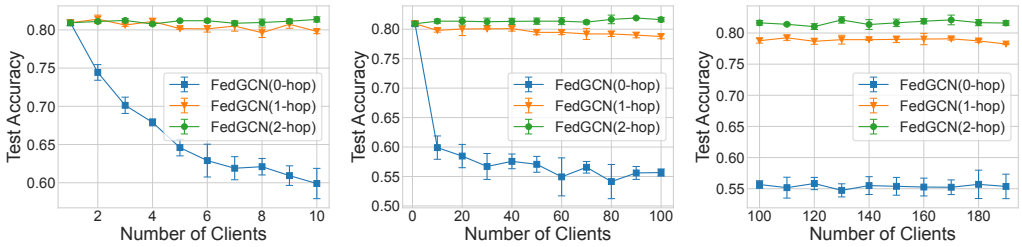

Figure 5: Test accuracy with the number of clients on the Cora dataset.

has lower communication costs. However, in the cross-device setting with many clients (Figure 5 right), the 1-hop test accuracy drops with more clients, indicating that 2-hop communication may be necessary to maintain high model accuracy, as suggested by our theoretical analysis.

7 Conclusion

We propose FedGCN, a framework for federated training of graph convolutional networks for semi-supervised node classification. The FedGCN training algorithm is based on the insight that, although distributed GCN training typically ignores cross-client edges, these edges often contain information useful to the model, which can be sent in a single round of communication before training begins. FedGCN allows for different levels of communication to accommodate different privacy and overhead concerns, with more communication generally implying less information loss and faster convergence, and further integrates HE for privacy protection. We quantify FedGCN's convergence under different levels of communication and different degrees of non-i.i.d. data across clients and show that FedGCN achieves high accuracy on real datasets, with orders of magnitude less communication than previous algorithms.

8 Applications of FedGCN

In this section, we discuss three important yet challenging applications of federated graph learning. Compared with prior works, our FedGCN can overcome the challenges of each application setting without sacrificing accuracy.

8.1 Multi-step Distributed Training with $1000\times$ Less Communication Cost

In distributed training, the main focus is to train a model with fast computation time and high accuracy, utilizing the resources of multiple computing servers. Privacy is not a concern in this case. FedGCN requires much less communication cost compared with distributed training methods (e.g., BDS-GCN Wan et al. (2022)) since FedGCN only requires pre-training communication. Moreover, FedGCN suggests that non-i.i.d. data distributions can further reduce communication, since non-i.i.d. partition results in fewer cross-client edges. FedGCN can first partition the graph to non-i.i.d. and perform precomputation to minimize the communication cost while maintaining model accuracy.

8.2 A Few Clients with Large Graphs in Federated Learning

Due to privacy regulations (e.g., GDPR in Europe) across countries, some data, e.g., that collected by Internet services like social networks, needs to be localized in each country or region. Here, each country represents a client, and "nodes" might be citizens of that country who use a particular service, g., users of a social network. Users in different countries (at different clients) may then interact, creating a cross-client edge. In this case, a single large country might be able to train models with sufficient performance using only its users' information, but some small countries might find it hard to train a good model as they have fewer constituent nodes. Federated learning for training a global model across countries, i.e., cross-silo federated learning, can help to train models in this setting, and there are relatively few cross-client edges compared to in-client edges. For example, in social networks, edges represent the connections between users, and users are more closely connected within a country. However, the small amount of cross-country (cross-client) edges might seriously affect the model performance since these edges can be more important for decision-making than the

in-client edges. For example, for anomaly detection on payment records, cross-country transactions are the key to detecting international money laundering and fraud, and ignoring these edges makes it impossible to detect these behaviors. FedGCN can take these edges into account and the trainers can decide if they want these edges based on the edge utility for their tasks.

FedGCN only communicates accumulated and homomorphically encrypted neighbor information at the initial round with better privacy guarantees and can also add differential privacy to better fit privacy regulations.

8.3 Millions of Clients with Small Graphs in Federated Learning

With the development of IoT (Internet-of-Things) devices, many people own several devices that can collect, process, and communicate data (e.g., Mobile phones, smart watches, or computers). Recently, smart home devices (e.g. cameras, light bulbs, and smart speakers) have also been adopted by millions of users, e.g., to monitor home security, the health care of senior citizens and infants, and package delivery. These mobile devices, and the applications that run on them, can also have interactions and connect with these IoT devices, e.g., unlocking the front door triggers the living room camera.

All these devices and applications are connected and form a graph, in which the devices/applications are nodes and their interactions are edges. The information of local devices or applications (node features) can then be very privacy sensitive, e.g., it may include video recording, accurate user location, and other information about users' personal habits. Federated training keeps the data localized to maintain privacy, while cross-client edges affect the federated training performance with privacy leakage. Here we take a "client" to mean a single user, who may own several devices or applications (i.e., nodes in the graph). Millions of devices across multiple users can then be connected, although each user (client) owns only a few devices, which means there are many edges across clients. The huge amount of cross-client edges can then seriously affect federated training's performance. The condition becomes more serious with the co-optimization of heterogeneous data distribution with limited local data.

Our FedGCN can significantly improve both the convergence time and model accuracy since it does not have information loss regardless of the number of clients. Since the number of cross-client edges increases with the number of clients, prior methods that ignore these edges or communicate (some of) their information in every round respectively face more information loss and communication costs.

9 Future Directions

Although the FedGCN can overcome the challenges mentioned above, it mainly works on training accumulation-based models like GCN and GraphSage. Our first theoretical analysis on federated graph learning with cross-client edges can be further improved. Several open problems in federated graph learning also need to be explored.

Federated Training of Attention-based GNNs Attention-based GNNs like GAT (Graph attention network) require calculating the attention weights of edges during neighbor feature aggregation, where the attention weights are based on the node features on both sides of edges and attention parameters. The attention parameters are updated at every training iteration and cannot be simply fixed at the initial round. How to train attention-based GNNs in a federated way with high performance and privacy guarantees is an open challenge and promising direction.

Neighbor Node and Feature Selection to Optimize System Performance General federated graph learning optimizes the system by only sharing local models, without utilizing cross-device graph edge information, which leads to less accurate global models. On the other hand, communicating massive additional graph data among devices introduces communication overhead and potential privacy leakage. To save the communication cost without affecting the model performance, one can select key neighbors and neighbor features to reduce communication costs and remove redundant information. For privacy guarantee, if there is one neighbor node, it can be simply dropped to avoid private data communication. FedGCN can be extended by using selective communication in its pre-training communication round.

Integration with L-hop Linear GNN Approximation methods To speed up the local computation speed, L-hop Linear GNN Approximation methods use precomputation to reduce the training computations by running a simplified GCN ($\boldsymbol{A}^L\boldsymbol{X}\boldsymbol{W}$ in SGC Wu et al. (2019), $[\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}, \boldsymbol{A}^2\boldsymbol{X}\boldsymbol{W}, \ldots, \boldsymbol{A}^L\boldsymbol{X}\boldsymbol{W}]$ in SIGN Frasca et al. (2020), and $\boldsymbol{\Pi}\boldsymbol{X}\boldsymbol{W}$ in PPRGo Bojchevski et al. (2020) where $\boldsymbol{\Pi}$ is the pre-computed personalized pagerank), but the communication cost is not reduced if we perform these methods alone. They are thus a complementary approach for efficient GNN training. FedGCN (2-hop, 1-hop) changes the model input ($\boldsymbol{A}$ and $\boldsymbol{X}$) to reduce communication in the FL setting, but the GCN model itself is not simplified. FedGCN can incorporate these methods to speed up the local computation, especially in constrained edge devices.

10 Acknowledgement

This research was supported by the National Science Foundation under grant CNS-1909306, Cloudbank support through CNS-1751075, and the Lee-Stanziale Ohana Fellowship by the ECE department at Carnegie Mellon University. The authors would like to thank Jiayu Chang, Cynthia (Xinyi) Fan, and Shoba Arunasalam for helping with the coding.

References

Adrien Benamira, Benjamin Devillers, Etienne Lesot, Ayush K Ray, Manal Saadi, and Fragkiskos D Malliaros. 2019. Semi-supervised learning and graph neural networks for fake news detection. In 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, 568–569.

Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2464–2473.

Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6, 3 (2014), 1–36.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine 34, 4 (2017), 18–42.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In International conference on the theory and application of cryptology and information security. Springer, 409–437.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. Advances in Neural Information Processing Systems 33 (2020), 3557–3568.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. arXiv preprint arXiv:2004.11198 (2020).

William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 1025–1035.

Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. 2021a. Fedgraphnn: A federated learning system and benchmark for graph neural networks. arXiv preprint arXiv:2104.07145 (2021).

Chaoyang He, Emir Ceyani, Keshav Balasubramanian, Murali Annavaram, and Salman Avestimehr. 2021b. Spreadgnn: Serverless multi-task federated learning for graph neural networks. arXiv preprint arXiv:2106.02743 (2021).

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the effects of non-identical data distribution for federated visual classification. arXiv preprint arXiv:1909.06335 (2019).

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. Advances in neural information processing systems 33 (2020), 22118–22133.

Nicolas Keriven, Alberto Bietti, and Samuel Vaiter. 2020. Convergence and stability of graph convolutional networks on large random graphs. arXiv preprint arXiv:2006.01868 (2020).

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).

Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. 2022. Sharper convergence guarantees for asynchronous sgd for distributed and federated learning. Advances in Neural Information Processing Systems 35 (2022), 17202–17215.

Jing Lei and Alessandro Rinaldo. 2015. Consistency of spectral clustering in stochastic block models. The Annals of Statistics 43, 1 (2015), 215–237.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019b. Deepgcns: Can gcns go as deep as cnns?. In Proceedings of the IEEE/CVF international conference on computer vision. 9267–9276.

Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2022. Federated learning on non-iid data silos: An experimental study. In 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 965–978.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019a. On the convergence of fedavg on non-iid data. arXiv preprint arXiv:1907.02189 (2019).

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

PALISADE. 2020. PALISADE Release. `https://gitlab.com/palisade/palisade-release`

Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Mahmut T Kandemir, and Anand Sivasubramaniam. 2021. Learn locally, correct globally: A distributed algorithm for training graph neural networks. arXiv preprint arXiv:2111.08202 (2021).

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. arXiv preprint arXiv:2003.00295 (2020).

Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo. 2020. Distributed Graph Convolutional Networks. arXiv preprint arXiv:2007.06281 (2020).

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. arXiv preprint arXiv:1710.10903 (2017).

Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. Proceedings of Machine Learning and Systems 4 (2022), 673–693.

Binghui Wang, Ang Li, Hai Li, and Yiran Chen. 2020a. Graphfl: A federated learning framework for semi-supervised node classification on graphs. arXiv preprint arXiv:2012.04187 (2020).

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020b. Microsoft academic graph: When experts are not enough. Quantitative Science Studies 1, 1 (2020), 396–413.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Transactions on Information Forensics and Security 15 (2020), 3454–3469.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In International conference on machine learning. PMLR, 6861–6871.

Haibo Yang, Minghong Fang, and Jia Liu. 2021. Achieving Linear Speedup with Partial Worker Participation in Non-IID Federated Learning. arXiv preprint arXiv:2101.11203 (2021).

Yuhang Yao, Mohammad Mahdi Kamani, Zhongwei Cheng, Lin Chen, Carlee Joe-Wong, and Tianqiang Liu. 2023. FedRule: Federated Rule Recommendation System with Graph Neural Networks. In Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation. 197–208.

Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33. 5693–5700.

Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. Advances in Neural Information Processing Systems 34 (2021).

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In Thirty-second AAAI conference on artificial intelligence.

Weijia Zhang, Hao Liu, Yanchi Liu, Jingbo Zhou, and Hui Xiong. 2020. Semi-supervised hierarchical recurrent graph neural network for city-wide parking availability prediction. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 1186–1193.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. arXiv preprint arXiv:1806.00582 (2018).

Longfei Zheng, Jun Zhou, Chaochao Chen, Bingzhe Wu, Li Wang, and Benyu Zhang. 2021. ASFGNN: Automated separated-federated graph neural network. Peer-to-Peer Networking and Applications 14, 3 (2021), 1692–1704.

Zhixin Zhou and Arash A Amini. 2019. Analysis of spectral clustering algorithms for community detection: the general bipartite setting. The Journal of Machine Learning Research 20, 1 (2019), 1774–1820.

A FedGCN Training Algorithm

Algorithm 1 FedGCN Federated Training for Graph Convolutional Network

```
// Pre-Training Communication Round
for each client k ∈ [K] do in parallel
    Send ⟦{∑_{j∈N_i} I_k(c(j)) · A_ij x_j}_{i∈V_k}⟧ to the server
end
// Server Operation
for i ∈ V do in parallel
    ⟦∑_{j∈N_i} A_ij x_j⟧ = ∑_{d=1}^{C} ⟦∑_{j∈N_i} I_k(c(j)) · A_ij x_j⟧
end
for each client k ∈ [K] do in parallel
    if 1-hop then
        Receive ⟦{∑_{j∈N_i} A_ij x_j}_{i∈V_k}⟧ and decrypt it
    end
    if 2-hop then
        Receive ⟦{∑_{j∈N_i} A_ij x_j}_{i∈N_{V_k}}⟧ and decrypt it
    end
end
// Training Rounds
for t = 1, ..., T do
    for each client k ∈ [K] do in parallel
        Receive ⟦w^(t)⟧ and decrypt it
        Set w_k^(t,0) = w^(t),
        for e = 1, ..., E do
            Set g_{w_k}^(t,e) = ∇_{w_k} f_k(w_k^(t,e−1); G_k)
                w_k^(t,e) = w_k^(t,e−1) − η g_{w_k}^(t,e) // Update Parameters
        end
        Send [w_k^(t,E)⟧ to the server
    end
    // Server Operations
    ⟦w^(t+1)⟧ = (1/K) ∑_{d=1}^{C} ⟦w_k^(t,E)⟧ // Update Global Models
    Broadcast ⟦w^(t+1)⟧ to local clients
end
```

B Background and Preliminaries

B.1 Federated Learning

Federated learning was first proposed by McMahan et al. (2017), who build decentralized machine learning models while keeping personal data on clients. Instead of uploading data to the server for centralized training, clients process their local data and occasionally share model updates with the server. Weights from a large population of clients are aggregated by the server and combined to create an improved global model.

The FedAvg algorithm McMahan et al. (2017) is used on the server to combine client updates and produce a new global model. At training round t, a global model $\boldsymbol{w}^{(t)}$ is sent to K client devices.

At each local iteration e, every client k computes the gradient, $\boldsymbol{g}_{\boldsymbol{w}_k}^{(t,e)}$, on its local data by using the current model $\boldsymbol{w}_k^{(t,e-1)}$. For a client learning rate η, the local client update at the e-th local iteration, $\boldsymbol{w}_k^{(t,e)}$, is given by

$$\boldsymbol{w}_k^{(t,e)} \leftarrow \boldsymbol{w}_k^{(t,e-1)} - \eta \boldsymbol{g}_{\boldsymbol{w}_k}^{(t,e)}. \tag{5}$$

After E local iterations, the server then does an aggregation of clients' local models to obtain a new global model,

$$\boldsymbol{w}^{(t+1)} = \frac{1}{K}\sum_{d=1}^{C} \boldsymbol{w}_k^{(t,E)}. \tag{6}$$

The process then advances to the next training round, $t+1$.

B.2 Graph Convolutional Network

A multi-layer Graph Convolutional Network (GCN) (Kipf and Welling, 2016) has the layer-wise propagation rule

$$\boldsymbol{H}^{(l+1)} = \phi(\boldsymbol{A}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}). \tag{7}$$

The weight adjacency matrix $\boldsymbol{A}$ can be normalized or non-normalized given the original graph, and $\boldsymbol{W}^{(l)}$ is a layer-specific trainable weight matrix. The activation function is ϕ, typically ReLU (rectified linear units), with a softmax in the last layer for node classification. The node embedding matrix in the l-th layer is $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N\times d}$, which contains high-level representations of the graph nodes transformed from the initial features; $\boldsymbol{H}^{(0)} = \boldsymbol{X}$.

In general, for a GCN with L layers of the form 7, the output for node i will depend on neighbors up to L steps away. We denote this set by $\mathcal{N}_i^L$ as L-hop neighbors of i. Based on this idea, the clients can first communicate the information of nodes. After the communication of information, we can then train the model.

B.3 Stochastic Block Model

For positive integers C and N, a probability vector $\boldsymbol{p} \in [0,1]^C$, and a symmetric connectivity matrix $\boldsymbol{B} \in [0,1]^{C\times C}$, the SBM defines a random graph with N nodes split into C classes. The goal of a prediction method for the SBM is to correctly divide nodes into their corresponding classes, based on the graph structure. Each node is independently and randomly assigned a class in $\{1,...,C\}$ according to the distribution $\boldsymbol{p}$; we can then say that a node is a "member" of this class. Undirected edges are independently created between any pair of nodes in classes c and d with probability $\boldsymbol{B}_{cd}$, where the (c,d) entry of $\boldsymbol{B}$ is

$$\boldsymbol{B}_{cd} = \begin{cases} \alpha, & c = d \\ \mu\alpha, & c \neq d, \end{cases} \tag{8}$$

for $\alpha \in (0,1)$ and $\mu \in (0,1)$, implying that the probability of an edge forming between nodes in the same class is α (which is the same for each class) and the edge formation probability between nodes in different classes is $\mu\alpha$.

Let $\boldsymbol{Y} \in \{0,1\}^{N\times C}$ denotes the matrix representing the nodes' class memberships, where $\boldsymbol{Y}_{ic} = 1$ indicates that node i belongs to the c-th class, and is 0 otherwise. We use $\boldsymbol{A} \in \{0,1\}^{N\times N}$ to denote the (symmetric) adjacency matrix of the graph, where $\boldsymbol{A}_{ij}$ indicates whether there is a connection (edge) between node i and node j. From our node connectivity model, we find that given $\boldsymbol{Y}$, for $i < j$, we have

$$\boldsymbol{A}_{ij} | \{\boldsymbol{Y}_{ic} = 1, \boldsymbol{Y}_{jd} = 1\} \backsim \text{Ber}(\boldsymbol{B}_{cd}), \tag{9}$$

where $\text{Ber}(p)$ indicates a Bernoulli random variable with parameter p. Since all edges are undirected, $\boldsymbol{A}_{ij} = \boldsymbol{A}_{ji}$. We further define the connection probability matrix $\boldsymbol{P} = \boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T \in [0,1]^{N\times N}$, where $\boldsymbol{P}_{ij}$ is the connection probability of node i and node j and $\mathbb{E}[\boldsymbol{A}] = \boldsymbol{P}$.

C Training Configuration

C.1 Statistics of Datasets

C.2 Experiment Hyperparameters

For Cora and Citeseer, we use a two-layer GCN with ReLU activation for the first and Softmax for the second layer, as in Kipf and Welling (2016). 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. For the OGBN-Arxiv dataset, we instead use a 3-layer GCN with 256 hidden units

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Ogbn-Arxiv	169,343	1,166,243	128	40
Ogbn-Products	2,449,029	61,859,140	100	47

Table 3: Statistics of datasets.

and 600 training rounds. For the OGBN-Products dataset, we use 2-layer GraphSage, 256 hidden units, and 450 training rounds. All settings are the same as the papers Kipf and Welling (2016); Hu et al. (2020). The local adjacency matrix is normalized by $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ when using GCN. We evaluate the local test accuracy given the local graph $\mathcal{G}_k$ and get the average test accuracy of all clients as the global test accuracy. We set the number of clients to 10 and averaged over 10 experiment runs.

C.3 Computation Resource

Experiments are done in a p3d.16xlarge instance with 8 GPUs (32GB memory for each GPU) and 10 g4dn.xlarge instances (16GB GPU memory in each instance). One run of the OGBN-Products experiment can take 20 minutes due to full-batch graph training.

D Communication Cost and Tradeoffs

In this appendix, we examine the communication cost, and the resulting convergence-communication tradeoff, of FedGCN. As for the convergence analysis in Section 5, we derive communication costs for general graphs and then more interpretable results for the SBM model.

Proposition D.1. *(Communication Cost for FedGCN) For L-hop communication of GCNs with a number of layers $\geq L$, the size of messages from K clients to the server in a generic graph is*

$$\sum_{i\in\mathcal{V}} |c(\mathcal{N}_i)|d + \sum_{k=1}^{K} |\mathcal{N}_k^{L-1}|d, \tag{10}$$

where $c(\mathcal{N}_i)$ denotes the set of clients storing the neighbors of node i.

Data Distribution	0-hop	1-hop	2-hop
Generic Graph	0	$\sum_{i\in\mathcal{V}} \|c(\mathcal{N}_i)\|d + Nd$	$\sum_{i\in\mathcal{V}} \|c(\mathcal{N}_i)\|d + \sum_{k=1}^{K} \|\mathcal{N}_{\mathcal{V}_k}\|d$
Non-i.i.d. (SBM)	0	$(c_\mu + 2)Nd$	$2(c_\mu + 1)Nd$
Partial-i.i.d. (SBM)	0	$(c_\alpha p + c_\mu + 2)Nd$	$2(c_\alpha p + c_\mu + 1)Nd$
i.i.d. (SBM)	0	$(c_\alpha + c_\mu + 2)Nd$	$2(c_\alpha + c_\mu + 1)Nd$

Table 4: Communication costs. $|.|$ denotes the size of the set and $\sum_{i\in\mathcal{V}} |c(\mathcal{N}_i)|d$ is the cost of the message that the server received from all clients, where $c(\mathcal{N}_i)$ denotes the set of clients storing the neighbors of node i. Communication cost increases with the i.i.d control parameter $\boldsymbol{p}$. 2-hop communication has around twice the cost of 1-hop communication.

For a better understanding of the above form, Table 4 gives the approximated (assuming $\alpha, \mu << 1$) size of messages between clients for i.i.d. and non-i.i.d. data, for generic graphs and an SBM with N nodes and d-dimensional node features. Half the partial i.i.d. nodes are chosen in the i.i.d. and half the non-i.i.d. settings.

Appendix H proves this result. In the non-i.i.d. setting, most nodes with the same labels are stored in the same client, which means there are much fewer edges linked to nodes in the other clients than in the i.i.d. setting, incurring much less communication cost (specifically, $c_\alpha Nd$ fewer communications) for 1- and 2-hop FedGCN. Note that communication costs vary with N but not K, the number of clients, as clients communicate directly with the server and not with each other.

Combining Table 1's and Table 4's results, we observe i.i.d. data reduces the gradient variance but increases the communication cost, while the non-i.i.d. setting does the opposite. Approximation methods via one-hop communication then might be able to balance the convergence rate and communication. We experimentally validate this intuition in Section 6's results, as well as the next appendix section.

E Additional Experimental Results

E.1 Validation of Theoretical Analysis on Cora Dataset

We validate the qualitative results in the main theory and D.1 on the Cora dataset. As shown in Figure 6, 0-hop FedGCN does not need to communicate but requires a high convergence time. One- and 2-hop FedGCN have similar convergence times, but 1-hop FedGCN needs much less communication. The right graph in Figure 6 shows Table 1's gradient norm bound for the Cora dataset. We expect

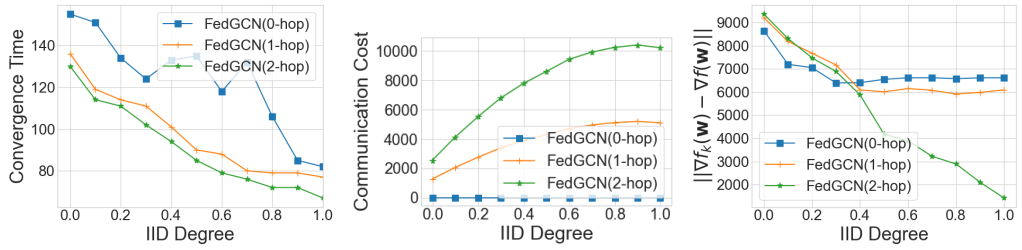

Figure 6: Convergence time (left), communication cost (middle) on Cora, and theoretical convergence upper bound (right, Table 1). FedGCN (1-hop) balances convergence and communication.

these to qualitatively follow the same trends as we increase the fraction of i.i.d. data, since from Theorem 5.6 the convergence time increases with $\|\nabla f_k(\boldsymbol{w}_k) - \nabla f(\boldsymbol{w})\|$. FedGCN (2-hop) and FedGCN (0-hop), as we would intuitively expect, respectively decrease and increase: as the data becomes more i.i.d., FedGCN (0-hop) has more information loss, while FedGCN (2-hop) gains more useful information from cross-client edges. Federated learning also converges faster for i.i.d. data, and we observe that FedGCN (0-hop)'s increase in convergence time levels off for $> 80\%$ i.i.d. data.

E.2 Homomorphic Encryption Microbenchmarking

Scheme	**Cheon-Kim-Kim-Song (CKKS)**
ring dimension	4096
security level	HEStd_128_classic
multi depth	1
scale factor bits	30

Table 5: HE Scheme Parameter Configuration On PALISADE. Multi depth is configured to be 1 for optimal (minimum) maximum possible multiplicative depth in our evaluation.

We implement our HE module using the HE library PALISADE (v1.10.5) PALISADE (2020) with the cryptocontext parameters configuration as in Table 5. In our paper, we evaluate the real-number HE scheme, i.e., the Cheon-Kim-Kim-Song (CKKS) scheme Cheon et al. (2017).

Array Size	Plaintext (Bool)	Plaintext (Long & Double)	CKKS	CKKS (Boolean Packing)
1k	1 kB	8 kB	266 kB	266 kB
10k	10 kB	80 kB	798 kB	266 kB
100k	100 kB	800 kB	7 MB	1 MB
1M	1 MB	8 MB	70 MB	8 MB
100M	100 MB	800 MB	7 GB	793 MB
1B	1 GB	8 GB	70 GB	8 GB

Table 6: Communication Cost Comparison between Plaintext and Encryption: Plaintext files are numpy arrays with pickle and ciphertext files are generated under CKKS.

In our framework, neighboring features (long integers, int64) are securely aggregated under the BGV scheme and local model parameters (double-precision floating-point, float64) are securely aggregated under the CKKS scheme. The microbenchmark results of additional communication overhead can

be found in Table 6. In general, secure computation using HE yields a nearly 15-fold increase in communicational cost compared to insecure communication in a complete view of plaintexts. However, with our Boolean Packing technique, the communication overhead only **doubles** for a large-size array.

F Assumptions of Proof

F.1 Standard Assumptions

Lipschitz Continuous Gradient, Bounded Global Variability, and Bounded Gradient are standard assumptions for FL analysis Li et al. (2019a); Yu et al. (2019); Yang et al. (2021); Ramezani et al. (2021); Koloskova et al. (2022). For example, Ramezani et al. (2021) incorporates them as assumption 1, assumption 2, and Eqn. 14 of Appendix B.2 in their paper,

$$\mathbb{E}[\mathcal{L}(\bar{\theta}^{t+1})] \leq \mathbb{E}[\mathcal{L}(\bar{\theta}^{t})] + \mathbb{E}[\langle \nabla \mathcal{L}(\bar{\theta}^{t}), \bar{\theta}^{t+1} - \bar{\theta}^{t} \rangle] + \frac{L}{2}\mathbb{E}[\|\bar{\theta}^{t+1} - \bar{\theta}^{t}\|^2].$$

They are "very standard non-convex, non-iid FL assumptions", which has been the state-of-the-art FL convergence analysis.

However, we also agree that these non-convex assumptions are still relatively strict, since how to provide FL analysis without such assumptions is still an open problem.

Bounded Global Variability and Bounded Gradient allow the convergence analysis to accommodate data distribution heterogeneity, a core challenge of FL.

The bounded gradient assumption in particular holds for certain activation functions, e.g., sigmoid functions, and bounded input features. Lipschitz Continuous Gradient is a technical condition on the shape of the loss function that is standard for non-convex analysis. It in fact relaxes the assumption of (strongly) convex loss functions that were previously common in analyzing FL and stochastic gradient descent. Our theory is also based on [2]'s convergence result for FedAvg. We hope our theory can open the area of theoretical analysis of federated graph learning.

F.2 Are these assumptions still valid for the graph?

For Assumption 5.3 (λ-Lipschitz Continuous Gradient),

$$\|\nabla f_k(w) - \nabla f_k(v)\| \leq \lambda \|w - v\|, \forall w, v \in \mathbb{R}^d,$$

w and v in the statement of this assumption represent two arbitrary sets of parameter values of the model. In a graph neural network, w represents the concatenation of parameters of each layer, i.e., $[W_1, W_2, ..., W_L]$ where W_l is the vectorized weight matrix of the l-th layer. For example, w can be the model parameters at the first training iteration, and v can be the model parameters after subsequent training iterations.

The assumption is general for arbitrary w and v. In other words, it means that changing the model parameters from w to v will not change the value of the loss function ∇f_k by more than a constant factor, multiplied by the norm of $\|w - v\|$. The correlation between w and v will not affect the bound. Intuitively, one might in fact expect a correlation between w and v to make the Lipschitz property more likely to hold, since the loss value is less likely to change much if the new parameter values (v) are correlated with the old parameter values (w).

Assumption 5.4, $\|\nabla f_k(w_t) - \nabla f(w_t)\| \leq \sigma_G$, follows from Assumption 5.5. If the local gradient is bounded, then since the global gradient is the sum of local gradients, it is also bounded. Thus, the difference between local and global gradients will also be bounded.

For Assumption 5.5, $\|\nabla f_k(w_t)\| \leq G$, we agree that the bounded gradient assumption may not always hold. However, it can be shown that this assumption holds for certain activation functions, e.g., sigmoid functions, and bounded input features.

F.3 Why do we adopt such assumptions?

Our analysis is based on (Yu et al., 2019). To the best of our knowledge, all state-of-the-art FL papers, such as Li et al. (2019a); Yu et al. (2019); Yang et al. (2021); Ramezani et al. (2021); Koloskova et al. (2022), make very similar assumptions. Since the main purpose of our paper is not to advance the convergence analysis of FL in general, but rather to show how this analysis applies to federated graph training, we follow these papers' assumptions. We believe that if better FL theory papers emerge that remove one or all assumptions, we can extend our work to this more advanced theory by analyzing the difference between the local gradient and global gradient in the graph setting.

We further believe that, while our exact quantitative convergence bounds may not hold in practice given that some of the theoretical assumptions may be violated, the qualitative insights derived from those bounds may still be valuable. In Figure 5, for example, we empirically validate our qualitative observations on how FedGCN's convergence varies with the number of clients and number of hops. We will further emphasize in the paper that the convergence analysis suggests qualitative insights about FedGCN's performance, even if the exact mathematical expressions do not always hold.

G Convergence Proof

We first give an example of a 1-layer GCN, then we mainly analyze a 2-layer GCN, which is the most common architecture for graph neural networks. The intuition of the theory is bounding the difference between the local gradient and global gradient in non-i.i.d settings. Our analysis also fits any layers of GCN and GraphSage.

G.1 Convergence Analysis of 1-layer GCNs

We first get the gradient of 1-layer GCNs in centralized, 0-hop, and 1-hop cases. Then we provide bounds to approximate the difference between local (0,1-hop) and global gradients.

G.1.1 Gradient of Centralized GCN (Global Gradient)

In centralized training, we consider a graph $\mathcal{G}$ with N nodes and d-dim feature for each node. The graph can be also represented as the adjacency matrix $\boldsymbol{A}$ and the feature matrix $\boldsymbol{X}$. Each nodes belongs to a specific class c, where the node label matrix can be represented as $\boldsymbol{Y}$. We then consider a 1-layer graph convolutional network with model parameter $\boldsymbol{W}$ and softmax activation ϕ, which has the following form

$$\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{X}\boldsymbol{W}. \tag{11}$$

We then pass it to the softmax activation

$$\boldsymbol{Q} = \phi(\boldsymbol{Z}), \tag{12}$$

where

$$\boldsymbol{Q}_{ic} = \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{c=1}^{C} e^{\boldsymbol{Z}_{ic}}}. \tag{13}$$

$\boldsymbol{Q}_{ic}$ is then the model prediction result for node i with specific class c.

Let $f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}, \boldsymbol{Y})$ represent the output of the cross-entropy loss, we have

$$f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}, \boldsymbol{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} \boldsymbol{Y}_{ic} \log \boldsymbol{Q}_{ic}. \tag{14}$$

Equation 1 Gradient to the input of softmax layer $\frac{\partial f}{\partial \boldsymbol{Z}} = \frac{1}{N}(\boldsymbol{Q} - \boldsymbol{Y})$

Proof. At first, we calculate the gradient of f given the element $\boldsymbol{Z}_{ic}$ of the matrix $\boldsymbol{Z}$, $\frac{\partial f}{\partial \boldsymbol{Z}_{ic}}$,

$$
\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{Z}_{ic}} &= \frac{\partial(-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\boldsymbol{Y}_{ic}\log \boldsymbol{Q}_{ic})}{\partial \boldsymbol{Z}_{ic}} \\
&= \frac{\partial(-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\boldsymbol{Y}_{ic}\log \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}})}{\partial \boldsymbol{Z}_{ic}} \\
&= \frac{\partial(-\frac{1}{N}\sum_{c=1}^{C}\boldsymbol{Y}_{ic}\log \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}})}{\partial \boldsymbol{Z}_{ic}} \\
&= -\frac{1}{N}\frac{\partial(\sum_{c=1}^{C}\boldsymbol{Y}_{ic}\log \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}})}{\partial \boldsymbol{Z}_{ic}} \\
&= -\frac{1}{N}\frac{\partial(\sum_{c=1}^{C}(\boldsymbol{Y}_{ic}\boldsymbol{Z}_{ic} - \boldsymbol{Y}_{ic}\log \sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}))}{\partial \boldsymbol{Z}_{ic}} \\
&= -\frac{1}{N}(\boldsymbol{Y}_{ic} - \frac{\partial(\sum_{c=1}^{C}(\boldsymbol{Y}_{ic}\log \sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}))}{\partial \boldsymbol{Z}_{ic}}) \\
&= -\frac{1}{N}(\boldsymbol{Y}_{ic} - \frac{\partial(\log \sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}))}{\partial \boldsymbol{Z}_{ic}}) \\
&= -\frac{1}{N}(\boldsymbol{Y}_{ic} - \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}}) \\
&= \frac{1}{N}(\frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}} - \boldsymbol{Y}_{ic}) \\
&= \frac{1}{N}(\boldsymbol{Q}_{ic} - \boldsymbol{Y}_{ic})
\end{aligned}
\tag{15}
$$

Given the property of the matrix, we have

$$
\frac{\partial f}{\partial \boldsymbol{Z}} = \frac{1}{N}(\boldsymbol{Q} - \boldsymbol{Y}).
$$

□

Lemma 1 If $\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{X}\boldsymbol{B}$,

$$
\frac{\partial f}{\partial \boldsymbol{X}} = \boldsymbol{A}^T \frac{\partial f}{\partial \boldsymbol{Z}} \boldsymbol{B}^T.
$$

Equation 2 The gradient over the weights of GCN

$$
\frac{\partial f}{\partial \boldsymbol{W}} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{A}^T(\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}) - \boldsymbol{Y}). \tag{16}
$$

Proof.

$$
\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{W}} &= (\boldsymbol{A}\boldsymbol{X})^T \frac{\partial f}{\partial \boldsymbol{Z}} \\
&= \boldsymbol{X}^T\boldsymbol{A}^T \frac{\partial f}{\partial \boldsymbol{Z}} \\
&= \frac{1}{N}\boldsymbol{X}^T\boldsymbol{A}^T(\boldsymbol{Q} - \boldsymbol{Y}) \\
&= \frac{1}{N}\boldsymbol{X}^T\boldsymbol{A}^T(\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}) - \boldsymbol{Y})
\end{aligned}
\tag{17}
$$

□

G.1.2 Gradients of local models with 0-hop communication

We then consider the federated setting. Let $\boldsymbol{A}^{N\times N}$ denote the adjacency matrix of all nodes and $\boldsymbol{A}_k^{N_k\times N_k}$ denotes the adjacency matrix of the nodes in client k. Let f_k represent the local loss function (without communication) of client k. Then the local gradient given model parameter $\boldsymbol{W}$ is

$$\frac{\partial f_k}{\partial \boldsymbol{W}} = \frac{1}{N_k}\boldsymbol{X}_k^T\boldsymbol{A}_k^T(\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{Y}_k) \tag{18}$$

G.1.3 Gradients of local models with 1-hop communication

With 1-hop communication, let $\dot{\boldsymbol{A}}_k^{N_k\times|\mathcal{N}_k|}$ denotes the adjacency matrix of the nodes in client k and their 1-hop neighbors ($\mathcal{N}_k$ also includes the current nodes). The output of GCN with 1-hop communication (recovering 1-hop neighbor information) is

$$\phi(\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k\boldsymbol{W}). \tag{19}$$

The local gradient with 1-hop communication given model parameter $\boldsymbol{W}$ is then

$$\frac{\partial \dot{f}_k}{\partial \boldsymbol{W}} = \frac{1}{N_k}\dot{\boldsymbol{X}}_k^T\dot{\boldsymbol{A}}_k^T(\phi(\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k\boldsymbol{W}) - \boldsymbol{Y}_k) \tag{20}$$

G.1.4 Bound the difference of local gradient and global gradient

Assuming each client has an equal number of nodes, we have $N_k = \frac{N}{K}$. The local gradient of 0-hop communication is then

$$\frac{\partial f_k}{\partial \boldsymbol{W}} = \frac{K}{N}\boldsymbol{X}_k^T\boldsymbol{A}_k^T(\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{Y}_k) \tag{21}$$

The difference between the local gradient (0-hop) and the global gradient is then

$$\begin{aligned}
&\|\frac{\partial f_k}{\partial \boldsymbol{W}} - \frac{\partial f}{\partial \boldsymbol{W}}\| \\
&= \|\frac{K}{N}\boldsymbol{X}_k^T\boldsymbol{A}_k^T(\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{Y}_k) - \frac{1}{N}\boldsymbol{X}^T\boldsymbol{A}^T(\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}) - \boldsymbol{Y})\| \\
&= \frac{1}{N}\|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T(\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{Y}_k) - \boldsymbol{X}^T\boldsymbol{A}^T(\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}) - \boldsymbol{Y})\| \\
&= \frac{1}{N}\|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{Y}_k - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}) + \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{Y}\| \\
&\leq \frac{1}{N}(\|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})\| + \|\boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{Y} - K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{Y}_k\|) \\
&= \frac{1}{N}(\|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})\| + \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{Y}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{Y}\|) \\
&\lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})\| + \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{Y}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{Y}\|
\end{aligned} \tag{22}$$

Since model training is to make the model output $\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})$ close to label matrix $\boldsymbol{Y}$, we provide an upper bound

$$\|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{Y}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{Y}\| \lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})\| \tag{23}$$

Based on Equation 22 and Equation 23 , we then have

$$\|\frac{\partial f_k}{\partial \boldsymbol{W}} - \frac{\partial f}{\partial \boldsymbol{W}}\| \lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\phi(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}) - \boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})\| \tag{24}$$

By assuming the function $\boldsymbol{X}^T\boldsymbol{A}^T\phi(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W})$ is λ-smooth w.r.t $\boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}$, we have

$$\begin{aligned}
\|\frac{\partial f_k}{\partial \boldsymbol{W}} - \frac{\partial f}{\partial \boldsymbol{W}}\| &\lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W} - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}\| \\
&\lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{X}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}\|
\end{aligned} \tag{25}$$

We can then provide the following bound to compare the local gradient with 0-hop communication and the global gradient given the same model parameter $\boldsymbol{w}$

$$\|\frac{\partial f_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{X}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}\|, \tag{26}$$

where $\boldsymbol{w}$ is the vectorization of model parameters $\boldsymbol{W}$.

Similarly, let $\dot{f}_k$ represent the loss function with 1-hop communication, the difference between the local gradient with 1-hop communication and the global gradient is

$$\|\frac{\partial \dot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \lesssim \lambda\|K\dot{\boldsymbol{X}}_k^T\dot{\boldsymbol{A}}_k^T\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}\|. \tag{27}$$

G.2 Convergence Analysis of 2-layer GCNs

Based on the same idea in 1-layer GCNs, we provide the convergence analysis of 2-layer GCNs. We first derive the gradient of 2-layer GCNs in centralized, 0-hop, 1-hop, and 2-hop cases. Then we provide bounds to approximate the difference between local (0, 1, 2-hop) and global gradients. Based on the Stochastic Block Model, we are able to quantify the difference.

G.2.1 Gradient of Centralized GCN (Global Gradient)

Based on the analysis of 1-layer GCNs, for graph $\mathcal{G}$ with adjacency matrix $\boldsymbol{A}$ and feature matrix $\boldsymbol{X}$ in clients, we consider a 2-layer graph convolutional network with ReLU activation for the first layer, Softmax activation for the second layer, and cross-entropy loss, which has the following form

$$\boldsymbol{Z} = \boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2, \tag{28}$$

$$\boldsymbol{Q} = \phi_2(\boldsymbol{Z}), \tag{29}$$

where

$$\boldsymbol{Q}_{ic} = \frac{e^{\boldsymbol{Z}_{ic}}}{\sum_{d=1}^{C} e^{\boldsymbol{Z}_{id}}} \tag{30}$$

The objective function is

$$f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\boldsymbol{Y}_{ic}\log\boldsymbol{Q}_{ic}. \tag{31}$$

We then show how to calculate the gradient $\nabla f(\mathbf{w}) = [\frac{\partial f}{\partial \boldsymbol{W}_1}, \frac{\partial f}{\partial \boldsymbol{W}_2}]$.

Equation 1 $\frac{\partial f}{\partial \boldsymbol{Z}} = \frac{1}{N}(\boldsymbol{Q} - \boldsymbol{Y})$

Equation 2 The gradient over the weights of the second layer

$$\frac{\partial f}{\partial \boldsymbol{W}_2} = \frac{1}{N}(\phi_1(\boldsymbol{W}_1^T\boldsymbol{X}^T\boldsymbol{A}^T))\boldsymbol{A}^T(\phi_2(\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y}) \tag{32}$$

Proof.

$$\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{W}_2} &= (\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1))^T\frac{\partial f}{\partial \boldsymbol{Z}} \\
&= (\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1))^T\boldsymbol{A}^T\frac{\partial f}{\partial \boldsymbol{Z}} \\
&= \frac{1}{N}(\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1))^T\boldsymbol{A}^T(\boldsymbol{Q} - \boldsymbol{Y}) \\
&= \frac{1}{N}(\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1))^T\boldsymbol{A}^T(\phi_2(\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y}) \\
&= \frac{1}{N}(\phi_1(\boldsymbol{W}_1^T\boldsymbol{X}^T\boldsymbol{A}^T))\boldsymbol{A}^T(\phi_2(\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y})
\end{aligned} \tag{33}$$

□

Equation 3 The gradient over the weights of the first layer.

$$\frac{\partial f}{\partial \boldsymbol{W}_1} = \frac{1}{N}(\boldsymbol{A}\phi_1'(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{A}\boldsymbol{X})^T(\phi_2(\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y})\boldsymbol{W}_2^T \tag{34}$$

Proof.

$$\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{W}_1} &= (\boldsymbol{A}\phi_1'(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{A}\boldsymbol{X})^T \frac{\partial f}{\partial \boldsymbol{Z}}\boldsymbol{W}_2^T \\
&= (\boldsymbol{A}\phi_1'(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{A}\boldsymbol{X})^T \frac{\partial f}{\partial \boldsymbol{Z}}\boldsymbol{W}_2^T \\
&= \frac{1}{N}(\boldsymbol{A}\phi_1'(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{A}\boldsymbol{X})^T(\boldsymbol{Q} - \boldsymbol{Y})\boldsymbol{W}_2^T \\
&= \frac{1}{N}(\boldsymbol{A}\phi_1'(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{A}\boldsymbol{X})^T(\phi_2(\boldsymbol{A}\phi_1(\boldsymbol{A}\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y})\boldsymbol{W}_2^T
\end{aligned} \tag{35}$$

□

G.2.2 Gradient of local models (0, 1, 2-hop)

For client k with local adjacency matrix $\boldsymbol{A}_k$, let $\dot{\boldsymbol{A}}_k^{n\times|\mathcal{N}_k|}$ denotes the adjacency matrix of the current nodes with complete edge information form their 1-hop neighbors ($\mathcal{N}_k$ also includes the current nodes), and $\ddot{\boldsymbol{A}}_k^{|\mathcal{N}_k|\times|\mathcal{N}_k^2|}$ denotes the adjacency matrix of nodes with complete edge information form their 2-hop neighbors ($\mathcal{N}_k^2$ also includes the current nodes and 1-hop neighbors).

The output of GCN without communication is

$$\phi_2(\boldsymbol{A}_k\phi_1(\boldsymbol{A}_k\boldsymbol{X}_k\boldsymbol{W}_1)\boldsymbol{W}_2). \tag{36}$$

The output of GCN with 1-hop communication is

$$\phi_2(\boldsymbol{A}_k\phi_1(\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k\boldsymbol{W}_1)\boldsymbol{W}_2). \tag{37}$$

The output of GCN with 2-hop communication is

$$\phi_2(\dot{\boldsymbol{A}}_k\phi_1(\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k\boldsymbol{W}_1)\boldsymbol{W}_2). \tag{38}$$

For 2-layer GCNs, output with 2-hop communication is the same as the centralized model.

The gradient of GCNs with 2-hop communication (recover the 2-hop neighbor information) over the weights of the first layer is then

$$\frac{\partial \ddot{f}_k}{\partial \boldsymbol{W}_1} = \frac{1}{N_k}(\dot{\boldsymbol{A}}_k\phi_1'(\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k\boldsymbol{W}_1)\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k)^T(\phi_2(\dot{\boldsymbol{A}}_k\phi_1(\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k\boldsymbol{W}_1)\boldsymbol{W}_2) - \boldsymbol{Y}_k)\boldsymbol{W}_2^T. \tag{39}$$

G.2.3 Bound the difference of local gradient and global gradient

Assuming each client has an equal number of nodes, we have $N_k = \frac{N}{K}$. Based on the same process in 1-layer case, we can then provide the following approximations between the local model and the global model.

The difference between the local gradient without communication and the global gradient is

$$\|\frac{\partial f_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \lesssim \|K\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{A}_k\boldsymbol{X}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}\boldsymbol{X}\|. \tag{40}$$

The difference between the local gradient with 1-hop communication and the global gradient is

$$\|\frac{\partial \dot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \lesssim \|K\dot{\boldsymbol{X}}_k^T\dot{\boldsymbol{A}}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}\boldsymbol{X}\|. \tag{41}$$

The difference between the local gradient with 2-hop communication and the global gradient is

$$\|\frac{\partial \ddot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \lesssim \|K\ddot{\boldsymbol{X}}_k^T\ddot{\boldsymbol{A}}_k^T\dot{\boldsymbol{A}}_k^T\dot{\boldsymbol{A}}_k\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k - \boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}\boldsymbol{X}\|. \tag{42}$$

With more communication, the local gradient gets closer to the global gradient.

G.3 Analysis on Stochastic Block Model with Node Features

To better quantify the difference, we can analyze it on generated graphs, the Stochastic Block Model.

G.3.1 Preliminaries

Assume the node feature vector $\boldsymbol{x}$ follows the Gaussian distribution with linear projection $\boldsymbol{H}$ of node label $\boldsymbol{y}$,

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{H}\boldsymbol{y}, \sigma), \tag{43}$$

we then have the expectation of the feature matrix

$$E(\boldsymbol{X}) = E(\boldsymbol{Y}\boldsymbol{H}^T). \tag{44}$$

According to the Stochastic Block Model, we have

$$E(\boldsymbol{A}) = \boldsymbol{P} = \boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T. \tag{45}$$

G.3.2 Quantify the gradient difference

Based on above results, the expectation of the global gradient given the label matrix $\boldsymbol{Y}$ is

$$E(\boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{A}\boldsymbol{X}|\boldsymbol{Y}) = \boldsymbol{H}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{H}^T. \tag{46}$$

Notice that $\boldsymbol{Y}^T\boldsymbol{Y}$ is counting the number of nodes belonging to each class. Based on this observation, we can better analyze the data distribution.

For adjacency matrix without communication

$$E(\boldsymbol{A}_k) = \boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T. \tag{47}$$

The expectation of the former gradient given the label matrix $\boldsymbol{Y}$ is then

$$E(\boldsymbol{X}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\boldsymbol{A}_k\boldsymbol{X}_k|\boldsymbol{Y}) = \boldsymbol{H}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{H}^T. \tag{48}$$

For adjacency matrix with 1-hop communication

$$E(\dot{\boldsymbol{A}}_k) = \boldsymbol{Y}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T \tag{49}$$

The expectation of the former gradient with 1-hop communication given the label matrix $\boldsymbol{Y}$ is then

$$\begin{aligned} &E(\dot{\boldsymbol{X}}_k^T\dot{\boldsymbol{A}}_k^T\boldsymbol{A}_k^T\boldsymbol{A}_k\dot{\boldsymbol{A}}_k\dot{\boldsymbol{X}}_k|\boldsymbol{Y}) \\ &\qquad = \boldsymbol{H}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{H}^T. \end{aligned} \tag{50}$$

For adjacency matrix with 2-hop communication

$$E(\ddot{\boldsymbol{A}}) = \dot{\boldsymbol{Y}}\boldsymbol{B}\ddot{\boldsymbol{Y}}^T. \tag{51}$$

The expectation of the former gradient with 1-hop communication given the label matrix $\boldsymbol{Y}$ is then

$$\begin{aligned} &E(\ddot{\boldsymbol{X}}_k^T\ddot{\boldsymbol{A}}_k^T\dot{\boldsymbol{A}}_k^T\dot{\boldsymbol{A}}_k\ddot{\boldsymbol{A}}_k\ddot{\boldsymbol{X}}_k|\boldsymbol{Y}) \\ &\qquad = \boldsymbol{H}\ddot{\boldsymbol{Y}}_k^T\ddot{\boldsymbol{Y}}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{B}\ddot{\boldsymbol{Y}}_k^T\ddot{\boldsymbol{Y}}_k\boldsymbol{H}. \end{aligned} \tag{52}$$

The difference in gradient can then be written as

$$\begin{aligned} \|\frac{\partial \ddot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| \leq \lambda\|K&\ddot{\boldsymbol{Y}}_k^T\ddot{\boldsymbol{Y}}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{B}\boldsymbol{Y}_k^T\boldsymbol{Y}_k\boldsymbol{B}\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k\boldsymbol{B}\ddot{\boldsymbol{Y}}_k^T\ddot{\boldsymbol{Y}}_k \\ &-\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{B}\boldsymbol{Y}^T\boldsymbol{Y}\|\end{aligned} \tag{53}$$

Notice that $\boldsymbol{Y}_k^T\boldsymbol{Y}_k$ is counting the number of nodes in client k belonging to each class, $\dot{\boldsymbol{Y}}_k^T\dot{\boldsymbol{Y}}_k$ and $\ddot{\boldsymbol{Y}}_k^T\ddot{\boldsymbol{Y}}_k$ are respectively counting the number of 1-hop and 2-hop neighbors of nodes in client k belonging to each class. It can be decomposed as

$$\boldsymbol{Y}_k^T\boldsymbol{Y}_k = N_k\boldsymbol{p}_k, \tag{54}$$

We then have

$$\begin{aligned}\|\frac{\partial \ddot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| &\lesssim \|K\mathcal{N}^2_{\mathcal{V}_k}\boldsymbol{p}_k\mathcal{N}^1_{\mathcal{V}_k}\boldsymbol{p}_kN_k\boldsymbol{p}_k\mathcal{N}^1_{\mathcal{V}_k}\boldsymbol{p}_k\mathcal{N}^2_{\mathcal{V}_k}\boldsymbol{p}_k - N^5\boldsymbol{p}^5\|\|\boldsymbol{B}^4\| \\ &\lesssim \|KN_k(\mathcal{N}^1_{\mathcal{V}_k})^2(\mathcal{N}^2_{\mathcal{V}_k})^2(\boldsymbol{p}_k)^5 - N^5\boldsymbol{p}^5\| \\ &\lesssim \|(KN_k(\mathcal{N}^1_{\mathcal{V}_k})^2(\mathcal{N}^2_{\mathcal{V}_k})^2 - N^5)(\boldsymbol{p}_k)^5\| + N^5\|(\boldsymbol{p}_k)^5 - \boldsymbol{p}^5\|\end{aligned} \tag{55}$$

$(KN_k(\mathcal{N}^1_{\mathcal{V}_k})^2(\mathcal{N}^2_{\mathcal{V}_k})^2 - N^5)$ evaluates the difference between the number of nodes with communication in local client and the number of nodes in total. $\|(\boldsymbol{p}_k)^5 - \boldsymbol{p}^5\|$ evaluates the difference between local distribution and global distribution. The second term can be bounded by

$$N^5\|(\boldsymbol{p}_k)^5 - \boldsymbol{p}^5\| \leq N^5(1-\frac{1}{C})^{\frac{5}{2}}(1-p)^5. \tag{56}$$

We then work on bounding the first term.

G.4 Number of 1-hop and 2-hop neighbors for clients

We need to get the number of 1-hop neighbors $\mathcal{N}^1_{\mathcal{V}_k}$ and 2-hop neighbors $\mathcal{N}^2_{\mathcal{V}_k}$ in both i.i.d and non-i.i.d cases.

G.4.1 Number of 1-hop and 2-hop neighbors in i.i.d

Recall the the definition of SBM B.3, the edge between two nodes is independent of other edges. For node i in other clients, the probability that it has at least one edge with the nodes in client i

$$1-(1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}} \tag{57}$$

The expectation of 1-hop neighbor (including nodes in local client)

$$\begin{aligned}\frac{N}{K} + \frac{K-1}{K}N(1-(1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}}) &\approx \frac{N}{K} + \frac{K-1}{K}N(1-(1-\alpha\frac{N}{CK})(1-\mu\alpha\frac{(C-1)N}{CK})) \\ &\approx \frac{N}{K} + \frac{K-1}{K}N(1-(1-\alpha\frac{N}{CK}-\mu\alpha\frac{(C-1)N}{CK})) \\ &= \frac{N}{K} + \frac{K-1}{K}N(\alpha\frac{N}{CK}+\mu\alpha\frac{(C-1)N}{CK}) \\ &= \frac{N}{K}(1+(K-1)(\alpha\frac{N}{CK}+\mu\alpha\frac{(C-1)N}{CK}))\end{aligned} \tag{58}$$

Notice that it is $(1+(K-1)(\alpha\frac{N}{CK}+\mu\alpha\frac{(C-1)N}{CK})$ propotional to the number of local nodes.

Similarly, approximated expectation of 2-hop neighbor (including nodes in local client). This approximation is provided based on that in expectation there is no label distribution shift between 2-hop nodes and 1-hop nodes.

$$\frac{N}{K}(1+(K-1)(\alpha\frac{N}{CK}+\mu\alpha\frac{(C-1)N}{CK}))^2 \tag{59}$$

G.4.2 Number of 1-hop and 2-hop neighbors in non-i.i.d.

The expectation of 1-hop neighbor (including nodes in local client)

$$\begin{aligned}\frac{N}{K}+\frac{K-1}{K}N(1-(1-\mu\alpha)^{\frac{N}{K}}) \\ \approx \frac{N}{K}(1+\mu\alpha\frac{K-1}{K}N)\end{aligned} \tag{60}$$

Approximated expectation of 2-hop neighbor (Including nodes in local client).

$$\frac{N}{K}(1+\mu\alpha\frac{K-1}{K}N)^2 \tag{61}$$

G.4.3 Number of 1-hop and 2-hop neighbors in non-i.i.d.

The expectation of 1-hop neighbor (including nodes in local client)

$$\begin{aligned}\frac{N}{K}+\frac{K-1}{K}N(1-(1-\alpha)^{\frac{Np}{CK}}(1-\mu\alpha)^{\frac{N(C-p)}{CK}}) &\approx \frac{N}{K}+\frac{K-1}{K}N(\alpha\frac{N}{CK}((1-\mu)p+\mu C)) \\ &= \frac{N}{K}+\frac{K-1}{K}N(\alpha\frac{N}{CK}(1-\mu)p+\alpha\frac{N}{CK}\mu C) \\ &= \frac{N}{K}(1+(K-1)(\alpha\frac{N}{CK}(1-\mu)p+\mu\alpha\frac{N}{K}))\end{aligned} \tag{62}$$

Approximated expectation of 2-hop neighbor (including nodes in local client)

$$\frac{N}{K}(1+(K-1)(\alpha\frac{N}{CK}(1-\mu)p+\mu\alpha\frac{N}{K}))^2 \tag{63}$$

G.5 Data Distribution with Labels

We assume each label has the same number of nodes. Each client k has the same number of nodes $N_k=\frac{N}{K}$.

For global label distribution, we have

$$\boldsymbol{p}=diag(\frac{1}{C},...,\frac{1}{C}) \tag{64}$$

G.5.1 i.i.d

The local label distribution is the same as the global distribution in the i.i.d condition.

$$\boldsymbol{p}_k=diag(\frac{1}{C},...,\frac{1}{C}) \tag{65}$$

For local gradient without communication and global gradient,

$$\begin{aligned}\|\frac{\partial f_k}{\partial \boldsymbol{w}}-\frac{\partial f}{\partial \boldsymbol{w}}\| &\lesssim \|(K(N_k)^5-N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\ &\lesssim (1-K\frac{(N_k)^5}{N^5})N^5\|diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\ &\lesssim (1-K\frac{(N_k)^5}{N^5})\frac{N^5}{C^5}\|\boldsymbol{B}^4\| \\ &\lesssim (1-\frac{1}{K^4})\frac{N^5}{C^5}\|\boldsymbol{B}^4\|\end{aligned} \tag{66}$$

For local gradient with 1-hop communication and global gradient,

$$
\begin{aligned}
\|\frac{\partial \dot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| &\lesssim \|(K(N_k)^3|\mathcal{N}_k|^2 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim \|(K\frac{N^3}{K^3}(\frac{N}{C} + \frac{C-1}{C}N(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^2 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim \|(C\frac{N^5}{C^5}(1 + (C-1)(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^2 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim (1 - \frac{1}{C^4}(1 + (C-1)(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^2)\frac{N^5}{C^5}\|\boldsymbol{B}^4\|
\end{aligned} \tag{67}
$$

For local gradient with 2-hop communication and global gradient,

$$
\begin{aligned}
\|\frac{\partial \ddot{f}_k}{\partial \boldsymbol{w}} - \frac{\partial f}{\partial \boldsymbol{w}}\| &\lesssim \|(K(N_k)|\mathcal{N}_k|^2|\mathcal{N}_k^2|^2 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim \|(K\frac{N}{C}(\frac{N}{C} + \frac{C-1}{C}N(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^2 \\
&\quad ((\frac{N}{C} + \frac{C-1}{C}N(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^2)^2 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim \|(K\frac{N^5}{C^5}(1 + (C-1)(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^6 - N^5)diag(\frac{1}{C},...,\frac{1}{C})^5\boldsymbol{B}^4\| \\
&\lesssim (1 - \frac{1}{C^4}(1 + (C-1)(\alpha\frac{N}{C} + \mu\alpha\frac{(C-1)N}{CK}))^6)\frac{N^5}{C^5}\|\boldsymbol{B}^4\|
\end{aligned} \tag{68}
$$

For non-i.i.d, we can simply replace the number of 1-hop and 2-hop neighbors.

H Communication Cost under SBM

Assume the number of clients K is equal to the number of label types in the graph G. d represents the dimension of the node feature and N represents the number of nodes. Table 7 shows the communication cost of FedGCN and BDS-GCN Wan et al. (2022). Distributed training methods like BDS-GCN requires communication per local update, which makes the communication cost increase linearly with the number of global training round T and number of local updates E. FedGCN only requires low communication cost at the initial step.

Methods	1-hop	L-hop	BDS-GCN
Generic Graph	$C_1 + Nd$	$C_1 + \sum_{k=1}^{K} \|\mathcal{N}_k^{L-1}\|d$	$LTE\rho d\sum_{k=1}^{K} \|\mathcal{N}_k^1/\mathcal{V}_k\|$

Table 7: Communication costs of FedGCN and BDS-GCN on the generic graph. BDS-GCN requires communication at every local update.

H.1 Server Aggregation

We consider the communication cost of node i in client $c(i)$. For node i, the server needs to receive messages from $c(i)$ (note that $c(i)$ needs to send the local neighbor aggregation) and other clients containing the neighbors of node i.

H.1.1 Non-i.i.d.

The possibility that there is no connected node in client j for node i is

$$
(1 - \mu\alpha)^{\frac{N}{K}}. \tag{69}
$$

The possibility that there is at least one connected node in client j for node i is

$$
1 - (1 - \mu\alpha)^{\frac{N}{K}}. \tag{70}
$$

The number of clients that node i needs to communicate with is

$$1 + (K-1)(1 - (1-\mu\alpha)^{\frac{N}{K}}). \tag{71}$$

The communication cost of N nodes is

$$N(1 + (K-1)(1 - (1-\mu\alpha)^{\frac{N}{K}}))d. \tag{72}$$

1-order Approximation To better understand the communication cost, we can expand the form to provide a 1-order approximation

$$(1-\mu\alpha)^{\frac{N}{K}} \approx 1 - \mu\alpha\frac{N}{K} \tag{73}$$

Possibility that there is no connected node in client j for node i is

$$1 - (1-\mu\alpha)^{\frac{N}{K}} \approx 1 - 1 + \mu\alpha\frac{N}{K} = \mu\alpha\frac{N}{K}. \tag{74}$$

The number of clients that node i needs to communicate with is then

$$1 + (K-1)(1 - (1-\mu\alpha)^{\frac{N}{K}}) \approx 1 + (K-1)\mu\alpha\frac{N}{K}. \tag{75}$$

H.1.2 i.i.d.

The possibility that there is no connected node in client j for node i is

$$(1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}}. \tag{76}$$

The possibility that there is at least one connected node in client j for node i is

$$1 - (1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}}. \tag{77}$$

The number of clients that node i needs to communicate with are

$$1 + (C-1)(1 - (1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}}). \tag{78}$$

Node i needs to communicate with more clients in i.i.d. than the case in non-i.i.d.

The communication cost of N nodes is

$$N(1 + (C-1)(1 - (1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}}))d. \tag{79}$$

1-order Approximation

The number of clients that node i needs to communicate with is then

$$\begin{aligned}
1 - (1-\alpha)^{\frac{N}{CK}}(1-\mu\alpha)^{\frac{(C-1)N}{CK}} &\approx 1 - (1 - \alpha\frac{N}{CK})(1 - \mu\alpha\frac{(C-1)N}{CK}) \\
&= 1 - (1 - \alpha\frac{N}{CK} - \mu\alpha\frac{(C-1)N}{CK} + \alpha\frac{N}{CK}\mu\alpha\frac{(C-1)N}{CK}) \\
&= \alpha\frac{N}{CK} + \mu\alpha\frac{(C-1)N}{CK} - \alpha\frac{N}{CK}\mu\alpha\frac{(C-1)N}{CK} \\
&\approx \alpha\frac{N}{CK} + \mu\alpha\frac{(C-1)N}{CK}.
\end{aligned} \tag{80}$$

Therefore, the expression estimates the number of clients that node i needs to communicate with and its own client gives

$$(1 + (C-1)(\alpha\frac{N}{CK} + \mu\alpha\frac{(C-1)N}{CK})). \tag{81}$$

H.1.3 Non-i.i.d.

Similarly, let $\boldsymbol{p}$ denote the percent of i.i.d., we then have the communication cost

$$N(1+(C-1)(1-(1-\alpha)^{\frac{Np}{CK}}(1-\mu\alpha)^{\frac{N(C-p)}{CK}}))d. \tag{82}$$

1-order Approximation

The number of clients that node i needs to communicate with is then

$$\begin{aligned}
1-(1-\alpha)^{\frac{Np}{CK}}(1-\mu\alpha)^{\frac{N(C-p)}{CK}} &\approx 1-(1-\alpha\frac{Np}{CK})(1-\mu\alpha\frac{N(C-p)}{CK}) \\
&= 1-(1-\alpha\frac{Np}{CK}-\mu\alpha\frac{N(C-p)}{CK}+\alpha\frac{Np}{CK}\mu\alpha\frac{N(C-p)}{CK} \\
&= \alpha\frac{Np}{CK}+\mu\alpha\frac{N(C-p)}{CK}-\alpha\frac{Np}{CK}\mu\alpha\frac{N(C-p)}{CK} \\
&\approx \alpha\frac{Np}{CK}+\mu\alpha\frac{N(C-p)}{CK} \\
&= \alpha\frac{N}{CK}(p+\mu(C-p)) \\
&= \alpha\frac{N}{CK}(p-\mu p+\mu C) \\
&= \alpha\frac{N}{CK}((1-\mu)p+\mu C).
\end{aligned} \tag{83}$$

The communication cost of all nodes is then

$$\begin{aligned}
N(1+(C-1)\alpha\frac{N}{CK}((1-\mu)p+\mu C))d. &= (((1-\mu)p+\mu C)\frac{\alpha N(C-1)}{CK}+1)Nd. \\
&= (((1-\mu)p+\mu C)\frac{\alpha N(C-1)}{CK}+1)Nd. \\
&= (\frac{(1-\mu)\alpha N(C-1)}{CK}p+\frac{\mu\alpha N(C-1)}{C}+1)Nd.
\end{aligned} \tag{84}$$

H.2 Server sends to clients

Since the aggregations of neighbor features have been calculated in the server, it then needs to send the aggregations back to clients.

For 1-hop communication, each client requires the aggregations of neighbors (1-hop) of its local nodes. The communication cost equals to the number of local nodes times the size of the node feature is given by

$$\sum_{k=1}^{K}|\mathcal{V}_k|d = Nd. \tag{85}$$

For 2-hop communication, each client requires the aggregations of 2-hop neighbors of its local nodes, in which the cost equals to the number of 1-hop neighbors times the size of the node feature,

$$\sum_{k=1}^{K}|\mathcal{N}_{\mathcal{V}_k}|d \tag{86}$$

The number of neighbors in partial i.i.d setting for client k account for the number of local nodes, 1-hop, 2-hop neighbors and taking into account of parameters α and μ is

$$\begin{aligned}
\frac{N}{C}+\frac{C-1}{C}N(1-(1-\alpha)^{\frac{Np}{CK}}(1-\mu\alpha)^{\frac{N(C-p)}{CK}}) &\approx \frac{N}{C}+\frac{C-1}{C}N(\alpha\frac{N}{CK}((1-\mu)p+\mu C)) \\
&= \frac{N}{C}+\frac{C-1}{C}N(\alpha\frac{N}{CK}(1-\mu)p+\alpha\frac{N}{C}\mu)
\end{aligned} \tag{87}$$

Then the number of neighbors in partial i.i.id for all clients is given by the sum over all clients

$$N + (C-1)N(1-(1-\alpha)^{\frac{Np}{CK}}(1-\mu\alpha)^{\frac{N(C-p)}{CK}}) \approx N + (C-1)N(\alpha\frac{N}{CK}(1-\mu)p + \alpha\frac{N}{C}\mu) \tag{88}$$

The communication cost considering all local nodes and neighbors is then

$$(N + (C-1)N(\alpha\frac{N}{CK}(1-\mu)p + \alpha\frac{N}{CK}\mu C))d = (1 + (C-1)(\alpha\frac{N}{CK}(1-\mu)p + \alpha\frac{N}{C}\mu))Nd$$
$$= (1 + (C-1)\alpha\frac{N}{CK}(1-\mu)p + \mu\alpha(C-1)\frac{N}{C})Nd \tag{89}$$

For L-hop communication, each client requires the aggregations of L-hop neighbors of its local nodes, which equals to the number of $(L-1)$-hop neighbors times the size of the node feature,

$$\sum_{k=1}^{K} |\mathcal{N}_{\mathcal{V}_k}^{L-1}|d. \tag{90}$$

I Negative Social Impacts of the Work

We believe that our work overall may have a positive social impact, as it helps to protect user privacy during federated training of GCNs for node-level prediction problems. However, by enabling such training to occur without compromising privacy, there is a chance that we could enable improved training of models with negative social impact. For example, models might more accurately classify users in social networks due to their ability to leverage a larger, cross-client dataset of users in the training. Depending on the model being trained, these results could be used against such users, e.g., targeting dissidents under an authoritarian regime. We believe that such negative impacts are no more likely than positive impacts from improved training, e.g., allowing an advertising company to send better products to users through improved predictions of what they will like. This work itself is agnostic to the specific machine learning model being trained.