Vertically Federated Graph Neural Network for Privacy-Preserving Node Classification

Jun Zhou
Ant Group
Beijing, China
jun.zhoujun@antfin.com

Chaochao Chen*
Ant Group
Hangzhou, China
chaochao.ccc@antfin.com

Longfei Zheng
Ant Group
Beijing, China
zlf206411@antfin.com

Huiwen Wu
Ant Group
Hangzhou, China
huiwen.whw@antfin.com

Jia Wu
Macquarie University
Sydney, Australia
jia.wu@mq.edu.au

Xiaolin Zheng
Zhejiang University
Hangzhou, China
xlzheng@zju.edu.cn

Bingzhe Wu
Peking University
Beijing, China
wubingzhe@pku.edu.cn

Ziqi Liu
Ant Group
Hangzhou, China
ziqiliu@antfin.com

Li Wang
Ant Group
Hangzhou, China
raymond.wangl@antfin.com

ABSTRACT

Recently, Graph Neural Network (GNN) has achieved remarkable progresses in various real-world tasks on graph data, consisting of node features and the adjacent information between different nodes. High-performance GNN models always depend on both rich features and complete edge information in graph. However, such information could possibly be isolated by different data holders in practice, which is the so-called data isolation problem. To solve this problem, in this paper, we propose VFGNN, a federated GNN learning paradigm for privacy-preserving node classification task under data vertically partitioned setting, which can be generalized to existing GNN models. Specifically, we split the computation graph into two parts. We leave the private data (i.e., features, edges, and labels) related computations on data holders, and delegate the rest of computations to a semi-honest server. We also propose to apply differential privacy to prevent potential information leakage from the server. We conduct experiments on three benchmarks and the results demonstrate the effectiveness of VFGNN.

KEYWORDS

Privacy-preserving, secret sharing, graph neural network, federated learning, differential privacy

ACM Reference Format:
Jun Zhou, Chaochao Chen, Longfei Zheng, Huiwen Wu, Jia Wu, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2021. Vertically Federated Graph Neural Network for Privacy-Preserving Node Classification. In *Conference, Date, Place.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

*Chaochao Chen is the corresponding author.

Conference, Date, Place

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

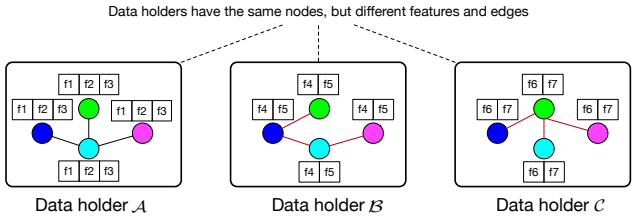


Figure 1: Problem description of vertically federated GNN.

1 INTRODUCTION

Graph Neural Network (GNN) has gained increasing attentions from both academy and industry due to its ability to model high-dimensional feature information and high-order adjacent information on both homogeneous and heterogeneous graphs [39]. GNN has a wide range of practical applications in multi-media area, e.g., computer vision [37], text classification [41], fraud detection [25], question answering [27], and social recommendation [12].

An important ingredient for high-performance GNN models is high-quality graph data including rich node features and complete adjacent information. However, in practice, such information could possibly be isolated by different data holders, which is the so-called *data isolation* problem. Such a data isolation problem presents a serious challenge for the development of Artificial Intelligence, which becomes a hot research topic recently.

Problem. Figure 1 shows a privacy-preserving node classification problem under vertically partitioned data setting. Here, for simplification, we assume there are three data holders ($\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$) and they have four same nodes. The node features are vertically split, i.e., $\mathcal{A}$ has $f1$, $f2$, and $f3$, $\mathcal{B}$ has $f4$ and $f5$, and $\mathcal{C}$ has $f6$ and $f7$. Meanwhile, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ may have their own edges. For example, $\mathcal{A}$ has social relation between nodes while $\mathcal{B}$ and $\mathcal{C}$ have payment relation between nodes. We also assume $\mathcal{A}$ is the party who has

the node labels. Note that one can slightly modify Figure 1 to the horizontally partitioned setting [40], i.e., data holders have the same features and edge types but different nodes. The problem is to build federated GNN models by using the graph data of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ cooperatively. In this paper, we take vertically partitioned setting for example, and present how to build federated GNN models.

Existing work. To date, kinds of privacy-preserving machine learning models have been designed for the data isolation problem, e.g., decision tree [22], linear regression [13], logistic regression [2], recommender system [6], and neural network [36, 44]. There are also several work on studying the privacy issues in GNN, e.g., graph publishing [30], GNN inference [17], and federated GNN when data are horizontally partitioned [7, 38, 45]. However, few research has studied the problem of GNN when data are vertically partitioned, which popularly exists in practice. Unlike previous privacy-preserving machine learning models that assume only samples (nodes) are held by different parties and these samples have no relationship, our task is more challenging because GNN relies on the relationships between samples, which are also kept by different data holders.

Naive solution. A direct way to build privacy-preserving GNN is employing advanced cryptographic algorithms, such as homomorphic encryption (HE) and secure multi party computation (MPC) [10, 26], to build the training/inference of GNN models. Such a pure cryptographic way can provide high security guarantees, however, it suffers high computation and communication costs, which limits their efficiency, as analyzed in [28].

Our solution. Instead, we propose VFGNN, a federated GNN learning paradigm for privacy-preserving node classification task under data vertically partitioned setting. Motivated by the existing work in split learning [14, 28, 35], we split the computation graph of GNN into two parts for privacy and efficiency concern, i.e., the private data related computations carried out by data holders and non-private data related computations conducted by a semi-honest server. Here, private data refers to node features, edges, and node labels, while the non-private data refers to the encoded hidden representations.

Specifically, data holders first apply secure Multi-Party Computation (MPC) techniques to compute the initial layer of the GNN using private node feature information collaboratively, which acts as the feature extractor module, and then perform neighborhood aggregation using private edge information individually, similar as the existing GNNs [34], and finally get the local node embeddings. Next, we propose different combination strategies for a semi-honest server to combine local node embeddings from data holders and generate global node embeddings, based on which the server can conduct the successive non-private data related computations, e.g., the non-linear operations in deep network structures that are time-consuming for MPC techniques. Finally, the server returns the final hidden layer to the party who has labels to compute prediction and loss. Data holders and the server perform forward and back propagations to complete model training and prediction, during which the private data (i.e., features, edges, and labels) are always kept by data holders themselves. Here we assume data holders are honest-but-curious, and the server does not collude with data holders. We argue that this is a reasonable assumption since the server can be played by authorities such as governments or replaced by trusted execution environment [9]. Moreover, we propose to apply differential privacy on the local node embeddings when the server generates global node embeddings, to further protect potential information leakage.

Results. Experiments on three benchmark datasets show that (1) VFGNN significantly improves the GNN models that are trained on the isolated dataset, and achieves comparable performance with the insecure model trained on the plaintext mixed data; (2) Our proposed combination strategies are effective under different situations; (3) VFGNN outperforms the GNN models that are built using purely MPC techniques, in terms of both accuracy and efficiency.

Contributions. We summarize our main contributions as:

- We propose a novel VFGNN learning paradigm, which not only can be generalized to most existing GNNs, but also enjoy good accuracy and efficiency.
- We propose different combination strategies for the server to combine local node embeddings from data holders.
- We evaluate our proposals on three real-world datasets, and the results demonstrate the effectiveness of VFGNN.

2 PRELIMINARIES

In this section, we present some preliminary techniques of our proposal, including security model, GNN, a popular MPC technique, i.e., secret sharing, and differential privacy.

2.1 Security Model

In literature, there are two commonly used security model, i.e., honest-but-curious (semi-honest) model and malicious model. The former has better efficiency while the later has better security degree. Our proposed model is secure against honest-but-curious (semi-honest) adversaries. That is, data holders and the server strictly follow the protocol, but they also use all intermediate computation results to infer as much information as possible. We also assume that the server does not collude with any data holders. This security setting is the same as most existing works [16, 26].

2.2 Graph Neural Network

GNN extended existing neural networks for processing the data represented in graph domains. The key of GNN is to learn a function f to generate node embeddings $\mathbf{h}_v$ for each node v in a graph based on its own features $\mathbf{x}_v$ and neighbors' features $\mathbf{x}_u, u \in \mathcal{N}(v)$, with $\mathcal{N}(v)$ denoting the neighborhood function in a graph [39]. After this, with the generated node embeddings, one can do further tasks such as classification [18], link prediction [43], and recommendation [42] using deep neural networks. In this paper, we focus on node classification task. The key to this task is the design of *aggregator function*, which learns to aggregate feature information from the node's local neighborhood [46]. To date, different types of aggregator functions have been proposed, e.g., convolution based [15], gated mechanism based [21], and attention based [24, 34].

2.3 Additive Secret Sharing

Our proposal depends on Additive Secret Sharing [32]. To additively share $\mathbf{Shr}(\cdot)$ an ℓ-bit value a for party $i \in \mathcal{P} = \{1, ..., I\}$, party i generates $\{a_j \in \mathbb{Z}_{2^\ell}, j \in \mathcal{P} \text{ and } j \neq i\}$ uniformly at random, sends

a_j to party j, and keeps $a_i = a - \sum_j a_j \bmod 2^\ell$. We use $\langle a \rangle_i = a_i$ to denote the share of party i. To reconstruct **Rec**$(\cdot,\cdot)$ a shared value $\langle a \rangle$, each party i sends $\langle a \rangle_i$ to one who computes $\sum_i a_i \bmod 2^\ell$. For simplification, we denote additive sharing by $\langle \cdot \rangle$ in this paper.

Multiplication and Vectorization. Existing research on secret sharing multiplication protocol is mainly based on Beaver's triplet technique [4]. Specifically, to multiply two secretly shared values $\langle a \rangle$ and $\langle b \rangle$ between two parties $\mathcal{P}_0$ and $\mathcal{P}_1$, they first need to collaboratively choose a shared triple $\langle u \rangle$, $\langle w \rangle$, and $\langle z \rangle$, where u, w are uniformly random values in $\mathbb{Z}_{2^\ell}$ and $z = uw \bmod 2^\ell$. They then locally compute $\langle e \rangle_i = \langle a \rangle_i - \langle u \rangle_i$ and $\langle f \rangle_i = \langle b \rangle_i - \langle w \rangle_i$, where $i \in \{0, 1\}$. Next, they run **Rec**$(\langle e \rangle_0, \langle e \rangle_1)$ and **Rec**$(\langle f \rangle_0, \langle f \rangle_1)$. Finally, $\mathcal{P}_i$ gets $\langle c \rangle_i = -i \cdot e \cdot f + f \cdot \langle a \rangle_i + e \cdot \langle b \rangle_i + \langle z \rangle_i$ as a share of the multiplication result, such that $\langle a \rangle \langle b \rangle = \langle c \rangle_0 + \langle c \rangle_1$. It is easy to vectorize the addition and multiplication protocols under secret sharing setting. The above protocols work in finite field, and we adopt fixed-point representation to approximate decimal arithmetics efficiently [26].

2.4 Differential Privacy

Definition 2.1. (Differential Privacy [11]). A randomized algorithm $\mathcal{M}$ that takes as input a dataset consisting of individuals is $(\epsilon, \delta)-$differentially private (DP) if for any pair of neighbouring data x, y that differ in a single entry, and any event E,

$$P[\mathcal{M}(x) \in E] \leq \exp(\epsilon) P[\mathcal{M}(y) \in E] + \delta, \tag{1}$$

and if $\delta = 0$, we say that $\mathcal{M}$ is $\epsilon-$differentially private.

In [11], the authors pointed out that the ℓ_2-sensitivity of a function f measures the magnitude by which a single individual's data can change the function in the worst case. In formal sense, it gives an upper bound on how much one must perturb its output to preserve privacy. Based on ℓ_2-sensitivity, the noise distributed according to Gaussian distribution naturally leads to (ϵ, δ)-DP.

Definition 2.2. (ℓ_2-sensitivity [11]). Suppose x and y are neighbouring inputs different in one entry. The ℓ_2-sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^d$ is:

$$\Delta_2 f = \max_{x, y \in \mathcal{D}, \|x-y\|=1} \|f(x) - f(y)\|_2.$$

Definition 2.3. (The Gaussian Mechanism [11]) Given a function $f : \mathcal{D} \to \mathbb{R}^d$ over a dataset $\mathcal{D}$, the Gaussian mechanism is defined as:

$$\mathcal{M}_G(x, f(\cdot), \epsilon) = f(x) + (Y_1, \cdots, Y_k)$$

where Y_i are *i.i.d.* random variables drawn from $\mathcal{N}(0, \sigma^2 \Delta_2 f^2)$ and $\sigma = \frac{\sqrt{2 \ln(1.25/\delta)}}{\epsilon}$.

Theorem 2.4. *[11] The Gaussian mechanism defined in Definition 2.3 preserves $(\epsilon, \delta)-$differential privacy.*

3 THE PROPOSED MODEL

In this section, we first give an overview of the proposed privacy-preserving GNN learning framework, which divides the computational graph of GNN into three sub-graphs. We then present its three important modules, i.e., generating initial node embeddings, generating local node embeddings, and generating global node embeddings on server. Finally, we present how to learn the proposed model.

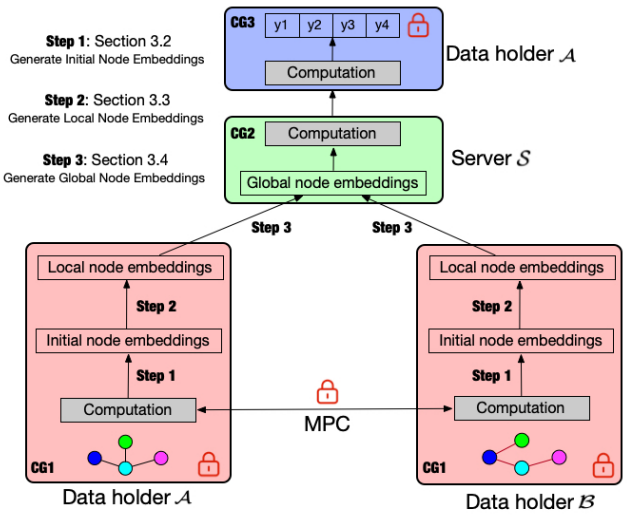


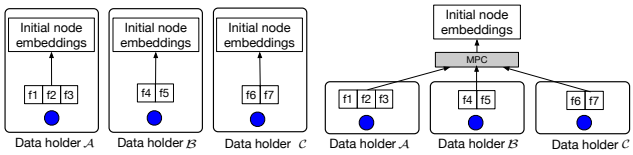
Figure 2: The proposed VFGNN, which is divided into three Computational Graphs (CG), i.e., private feature and edge related computations (CG1, shown in red), non-private data related computations (CG2, shown in green), and private label related computations on data holder (CG3, shown in blue). Besides, VFGNN has three key steps, i.e., generate initial node embeddings (Step 1), generate local node embeddings (Step 2), and generate global node embeddings (Step 3).

3.1 Overview of VFGNN

In practice, the data isolation problem can be mainly divided into two types based on data split forms, i.e., horizontally partitioned data and vertically partitioned data. Take GNN for example, the former indicates the isolated data holders have different nodes, but the same feature schema and edge type. While the later denotes the isolated data holders have the same nodes, but different feature schemes and edge types, as shown in Figure 1. Currently, users are always active on different platforms at the same time, e.g., social platform and e-commerce platform, which makes them have different feature schemes and edge types. Therefore, vertically data split setting is more common in practice, and we focus on this setting in this paper.

The first step in private machine learning under vertically data split setting is secure entity alignment, also known as Private Set Intersection (PSI). That is, data holders align nodes without exposing those that do not overlap with each other. PSI has been researched extensively in the prior work[29]. In this paper, we assume data holders have aligned their nodes and are ready for performing privacy-preserving GNN training and inference.

As described in Section 1, for the sake of privacy and efficiency, we design a vertically federated GNN (VFGNN) learning paradigm by *splitting the computational graph of GNN* into two parts. That is, we keep the *private data related computations* to data holders for privacy concern, and delegate the *non-private data related computations* to a semi-honest server for efficiency concern. In the context of GNN, the private data refers to node features, labels, and edges (node relations). To be specific, we divide the computational



(a) Individually (b) Collaboratively

Figure 3: Two methods of generating initial node embeddings.

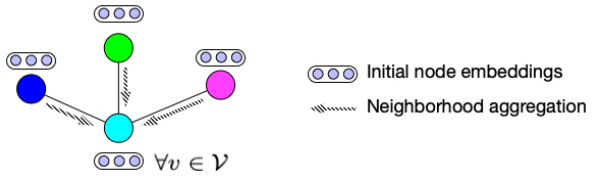


Figure 4: Generate local node embeddings by neighborhood aggregation.

graph into the following three sub-Computational Graphs (CG), as is shown in Figure 2.

CG1: private feature and edge related computations. The first step of GNN is generating initial node embeddings using node's private features, e.g., user features in social networks. In vertically data split setting, each data holder has partial node features, as shown in Figure 1. We will present how data holders collaboratively learn initial node embeddings in Section 3.2. In the next step, data holders generate local node embeddings by aggregating multi-hop neighbors' information using different *aggregator functions*, which we will describe in Section 3.3.

CG2: non-private data related computations. We delegate non-private data related computations to a semi-honest server for efficiency concern. First, the server combines the local node embeddings from data holders with different COMBINE strategies, and obtains the global node embeddings, which we will further describe in details in Section 3.4. Next, the server can perform the successive computations using plaintext data. Note that this part has many non-linear computations such as max-pooling and activation functions, which are not cryptographically friendly. For example, existing works approximate the non-linear activations by using either piece-wise functions that need secure comparison protocols [26] or high-order polynomials [16]. Therefore, their accuracy and efficiency are limited. Delegating these plaintext computations to server will not only improve our model accuracy, but also significantly improve our model efficiency, as we will present in experiments. After this, the server gets a final hidden layer ($\mathbf{z}_L$) and sends it back to the data holder who has label to calculate prediction, where L is the total number of layers of the deep neural network.

CG3: private label related computations on data holder. The data holder who has label can compute prediction using the final hidden layer it receives from the server. For node classification task, the Softmax activation function is used for the output layer [18], which is defined as $\text{softmax}(z_c) = \frac{1}{Z}\exp(z_c)$ with $c \in C$ be the node class and $Z = \sum_c \exp(z_c)$.

In the following subsections, we will describe three important components of VFGNN, i.e., initial node embeddings generation in **CG1**, local node embeddings generation in **CG2**, and global node embedding generation in **CG3**.

3.2 Generate Initial Node Embeddings

Initial node embeddings are generated by using node features. Under vertically partitioned data setting, each data holder has partial node features. Under such setting, there are two methods for data holders to generate initial node embeddings, i.e., *individually* and *collaboratively*, as is shown in Figure 3.

The '*individually*' method means that data holders generate initial node embeddings using their own node features, individually. For data holder $i \in \mathcal{P}$, this can be done by $\mathbf{h}_0^i = (\mathbf{x}^i)^T \cdot \mathbf{W}^i$, where $\mathbf{x}^i$ and $\mathbf{W}^i$ are node features and weight matrix of data holder i. Although this method is simple and data holders do not need to communicate with each other, it cannot capture the relationship between features of data holders and thus causes information loss. As the example in Figure 3, data holder $\mathcal{B}$ generates its inital node embeddings using features $f4$ and $f5$, while data holder C generates its inital node embeddings using features $f6$ and $f7$. That is, it assumes the features of $\mathcal{B}$ and C are independent, and thus cannot capture the relation between them.

To solve the shortcoming of '*individually*' method, we propose the '*collaboratively*' method. It indicates that data holders generate initial node embeddings using their node features, collaboratively, and meanwhile keep their private features secure. Technically, this can be done by using cryptographic methods such as secret sharing and homomorphic encryption [1]. In this paper, we choose additive secret sharing (described in Section 2.3) due to its high efficiency. We summarize the protocol in Algorithm 1. Traditionally, the initial node embeddings can be generated by $\mathbf{h}_0 = \mathbf{x}^T \cdot \mathbf{W}$, where $\mathbf{x}$ is node features and $\mathbf{W}$ is weight matrix. When features are vertically partitioned, we calculate initial node embeddings as follows. First, we secretly share $\mathbf{x}$ among data holders (Lines 3-4). Then, data holders concat their received shares in order (Line 5). After it, we calculate $\mathbf{x}^T \cdot \mathbf{W}$ following distributive law (Lines 6, 8, and 9). Take two data holders for example, $\mathbf{x}^T \cdot \mathbf{W} = (\langle\mathbf{x}\rangle_1 + \langle\mathbf{x}\rangle_2) \cdot (\langle\mathbf{W}\rangle_1 + \langle\mathbf{W}\rangle_2) = \langle\mathbf{x}\rangle_1 \cdot \langle\mathbf{W}\rangle_1 + \langle\mathbf{x}\rangle_1 \cdot \langle\mathbf{W}\rangle_2 + \langle\mathbf{x}\rangle_2 \cdot \langle\mathbf{W}\rangle_1 + \langle\mathbf{x}\rangle_2 \cdot \langle\mathbf{W}\rangle_2$. Finally, data holders reconstruct $\mathbf{x}^T \cdot \mathbf{W}$ by summing over all the shares (Lines 13-15). The security can be proved by following the *real world* and *ideal world* simulation based approach [23], similar as SecureML [26].

3.3 Generate Local Node Embeddings

We generate local node embeddings by using multi-hop neighborhood aggregation on graphs, based on inital node embeddings. Note that, unlike existing GNNs that perform neighborhood aggregation using the centralized data, under data isolated setting, *neighborhood aggregation should be done by data holders separately*, rather than cooperatively, to protect the private edge information. This is because one may infer the neighboorhoold information of v given the neighborhood aggregation results of k-hop ($\mathbf{h}_v^k(i)$) and $k+1$-hop ($\mathbf{h}_v^{k+1}(i)$), if neighborhood aggregation is done by data holders together. A special case is $\mathbf{h}_v^k(i) = \mathbf{h}_v^{k+1}(i)$, where it is likely that v is

Algorithm 1: Data holders $\mathcal{P}$ securely generate the initial node embeddings using secret sharing

Input: $\{\mathbf{x}_v^i \in \mathbb{Z}_{2^\ell}, \forall v \in \mathcal{V}, \forall i \in \mathcal{P}\}$, and $\mathbf{x}^i$ for short
Output: The share of initial node embeddings for each data holder i $\{\mathbf{h}_v^0(i), i \in \mathcal{P}, \forall v \in \mathcal{V}\}$

1 **for** $\mathcal{P}_i \in \mathcal{P}$ `in parallel` **do**
2 $\quad$ $\mathcal{P}_i$ randomly initializes $\langle \mathbf{W} \rangle_i$
3 $\quad$ $\mathcal{P}_i$ locally generates $\left\{ \left\langle \mathbf{x}^i \right\rangle_j \right\}_{j \in \mathcal{P}}$
4 $\quad$ $\mathcal{P}_i$ distributes $\left\{ \left\langle \mathbf{x}^i \right\rangle_j \right\}_{j \neq i}$ to others
5 $\quad$ $\mathcal{P}_i$ concats $\left\{ \left\langle \mathbf{x}^k \right\rangle_j \right\}_{j \in \mathcal{P}}$ and get $\langle \mathbf{x} \rangle_i$
6 $\quad$ $\mathcal{P}_i$ locally calculates $\langle \mathbf{x} \rangle_i^T \cdot \langle \mathbf{W} \rangle_i$ as i-share
7 **for** $\mathcal{P}_j \in \mathcal{P}$ *and* $j \neq i$ **do**
8 $\quad$ $\mathcal{P}_i$ and $\mathcal{P}_j$ calculate i-share $\left\langle \langle \mathbf{x} \rangle_i^T \cdot \langle \mathbf{W} \rangle_j \right\rangle_i$ and j-share $\left\langle \langle \mathbf{x} \rangle_i^T \cdot \langle \mathbf{W} \rangle_j \right\rangle_j$ using secret sharing
9 $\quad$ $\mathcal{P}_i$ and $\mathcal{P}_j$ calculate i-share $\left\langle \langle \mathbf{x} \rangle_j^T \cdot \langle \mathbf{W} \rangle_i \right\rangle_i$ and j-share $\left\langle \langle \mathbf{x} \rangle_j^T \cdot \langle \mathbf{W} \rangle_i \right\rangle_j$ using secret sharing
10 **for** $\mathcal{P}_i \in \mathcal{P}$ `in parallel` **do**
11 $\quad$ $\mathcal{P}_i$ locally calculates the summation of all i-shares, denoted as $\left\langle \mathbf{h}_v^0 \right\rangle_i = \left\langle \mathbf{x}^T \cdot \mathbf{w} \right\rangle_i$
12 $\quad$ $\mathcal{P}_i$ sends $\left\langle \mathbf{h}_v^0 \right\rangle_i$ to $\mathcal{P}_j, \forall j \in \mathcal{P}, j \neq i$
13 $\quad$ $\mathcal{P}_i$ reconstructs $\mathbf{h}_v^0$, denote as $\mathbf{h}_v^0(i)$
14 **return** $\mathbf{h}_v^0(i)$ for each data holder $i \in \mathcal{P}$

an isolated node in the graph of data holder i. Therefore, to protect graph privacy, we let data holders perform multi-hop neighborhood aggregation separately using their own graphs.

For $\forall v \in \mathcal{V}$ at each data holder, neighborhood aggregation is the same as the traditional GNN, as is shown in Figure 4. Take GraphSAGE [15], a general inductive framework, for example, it introduces aggregator functions to update hidden embeddings by sampling and aggregating features from a node's local neighborhood:

$$\begin{aligned} \mathbf{h}_{\mathcal{N}(v)}^k &\leftarrow \mathrm{AGG}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}), \\ \mathbf{h}_v^k &\leftarrow (\mathbf{W}^k \cdot \mathrm{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)), \end{aligned} \tag{2}$$

where we follow the same notations as GraphSAGE, and the aggregator functions AGG are of three types, i.e., Mean, LSTM, and Pooling. Since our proposed VFGNN learning framework is suitable for any existing GNN models, without loss of generality, in this paper, we will take GraphSAGE as a typical GNN case and present how to make it secure in data isolated setting. After it, data holders send local node embeddings to a semi-honest server for combination and further non-private data related computations.

3.4 Generate Global Node Embeddings

The server combines the local node embeddings from data holders and gets global node embeddings. The combination strategy (COMBINE) would be trainable and maintaining high representational capacity. We design three combination strategies.

Concat. The concat operator can fully preserve local node embeddings learnt from different data holders. That is, Line 11 in Algorithm 2 becomes

$$\mathbf{h}_v^K \leftarrow \mathrm{CONCAT}(\mathbf{h}_v^K(1), \mathbf{h}_v^K(2), ..., \mathbf{h}_v^K(I)). \tag{3}$$

Mean. The mean operator takes the elementwise mean of the vectors in $(\{\mathbf{h}_v^K(i), \forall i \in \mathcal{P}\})$, assuming data holders contribute equally to the global node embeddings, i.e.,

$$\mathbf{h}_v^K \leftarrow \mathrm{MEAN}(\mathbf{h}_v^K(1) \cup \mathbf{h}_v^K(2) \cup ... \cup \mathbf{h}_v^K(I)). \tag{4}$$

Regression. The above two strategies treat data holders equally. In reality, the local node embeddings from different data holder may contribute diversely to the global node embeddings. We propose a Regression strategy to handle this kind of situation. The regression operator aims to combine the embedding elements from data holders through a regression model, whose parameters are learnt intelligently during training. Let $\boldsymbol{\omega_i}$ be the weight vector of local node embeddings from data holder $i \in \mathcal{P}$, then

$$\mathbf{h}_v^K \leftarrow \boldsymbol{\omega_1} \odot \mathbf{h}_v^K(1) + \boldsymbol{\omega_2} \odot \mathbf{h}_v^K(2) ... + \boldsymbol{\omega_I} \odot \mathbf{h}_v^K(I), \tag{5}$$

where $\odot$ is element-wise multiplication. Regression can handle the situation where the data quality and quantity (feature and edge size) of data holders are different from each other.

These different combination operators can utilize local node embeddings in diverse ways, and we will empirically study their effects on model performances in experiments.

3.5 Enhancing Privacy by Adopting Differential Privacy

Directly sending (publishing) the local node embeddings to the server may cause potential information leakage, and we propose to apply differential privacy to further enhance privacy. In this section, we introduce two mechanisms to publish local node embeddings differentially private, such that with a single entry modification in the local node embedding, there is a large probability that the server cannot distinguish the difference of the local node embedding before or after the modification.

The two mechanisms, i.e., Gaussian Mechanism and James-Stein Estimator, are presented in Algorithm 3. As we have described in Section 2.4, Gaussian mechanism can provide (ϵ, δ)-DP by adding noise subject to Gaussian distribution with scale of ℓ_2 sensitivity and noise multiplier. Compared with Gaussian mechanism, James-Stein Estimator [3] provides more accurate estimator of local node embeddings $\mathbf{h}^K$ while keeping (ϵ, δ)-differential privacy. Assume that the local node embedding has the prior distribution $\mathcal{N}(0, w^2 I)$, one can design an adaptive estimator without knowing w^2 explicitly. The James-Stein estimator has been proved to always improve Maximum Likelihood Estimation (MLE) when the embedding dimension $d \geq 3$ by [33].

Algorithm 2: Privacy-preserving GraphSAGE for node label prediction (forward propagation)

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}^i)$ and node features $\{\mathbf{x}_v^i, \forall v \in \mathcal{V}\}$ on data holder $i, i \in \mathcal{P} = \{1, ..., I\}$; depth K; aggregator functions $\text{AGG}_k, \forall k \in \{1, ..., K\}$; weight matrices $\mathbf{W}_i^k, \forall i \in \mathcal{P}, \forall k \in \{1, ..., K\}$; max layer L; weight matrices $\mathbf{W}_l, \forall l \in \{0, ..., L\}$; non-linearity σ; neighborhood functions $\mathcal{N}^i : v \rightarrow 2^{\mathcal{V}}, \forall i \in \mathcal{P}$; node labels on data holder $p \in \mathcal{P}$ and $c \in C$

Output: Node label predictions $\{\hat{y}_{vc}, \forall v \in \mathcal{V}, \forall c \in C\}$ on data holder p

1 # **CG1:** private feature and edge related computations by data holders
2 **Data holders**: collaboratively calculate the initial node embeddings $\mathbf{h}_v^0(i) \leftarrow \mathbf{x}_v^i, \forall i \in \mathcal{P}, \forall v \in \mathcal{V}$, using Algo. 1
3 **for** $i \in \mathcal{P}$ `in parallel` **do**
4 **for** $k = 1$ *to* K **do**
5 **for** $v \in \mathcal{V}$ **do**
6 **Data holder**: calculates $\mathbf{h}_{\mathcal{N}(v)}^k(i) \leftarrow \text{AGG}_k(\{\mathbf{h}_u^{k-1}(i), \forall u \in \mathcal{N}^i(v)\})$
7 **Data holder**: calculates $\mathbf{h}_v^k(i) \leftarrow \sigma(\mathbf{W}_i^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}(i), \mathbf{h}_{\mathcal{N}(v)}^k(i))$
8 **Data holder**: calculates local node embeddings $\mathbf{h}_v^K(i) \leftarrow \mathbf{h}_v^K(i)/||\mathbf{h}_v^K(i)||_2, \forall v \in \mathcal{V}$ and sends (publishs) it to server using differential privacy
9 # **CG2:** non-private data related computations by server
10 **for** $v \in \mathcal{V}$ **do**
11 **Server**: combines the local node embeddings from data holders $\mathbf{h}_v^K = \text{COMBINE}\ (\{\mathbf{h}_v^K(i), \forall i \in \mathcal{P}\})$
12 **Server**: forward propagation based on the global node embeddings $\mathbf{z}_L = \sigma(\mathbf{W}_{L-1} \cdot \sigma(...\sigma(\mathbf{W}_0 \cdot \mathbf{h}_v^K)))$
13 **Server**: sends $\mathbf{z}_L$ to data holder p
14 # **CG3:** private label related computations by data holder who has label
15 **Data holder** p: makes prediction by $\hat{y}_{vc} \leftarrow \text{softmax}(\mathbf{W}_L \cdot \mathbf{z}_L), \forall v \in \mathcal{V}, \forall c \in C$

Algorithm 3: Publishing local node embeddings using differential privacy

Input: Local node embeddings $\mathbf{h}^K$, dimension of node embeddings d, noise multiplier σ, clipping value C.

Output: Differentially private node embeddings.

1 Scale embedding $\bar{\mathbf{h}}^K = \min(1, C/\|\mathbf{h}^K\|)\mathbf{h}^K$;
2 Draw i.i.d. samples from $\mathcal{N}(0, \sigma^2 C^2)$, which forms a d-dimension noise vector $\mathbf{n}$;
3 # **Gaussian Mechanism**
4 Add noise $\hat{\mathbf{h}}^K = \mathbf{h}^K + \mathbf{n}$;
5 # **James-Stein Estimator**
6 Compute James Estimator $\tilde{\mathbf{h}}_{\text{JS}}^K = \left(1 - \frac{(d-2)\sigma^2 C^2}{\|\hat{\mathbf{h}}^K\|^2}\right)\hat{\mathbf{h}}^K$
7 **return** Guassian perturbed node embeddings $\hat{\mathbf{h}}^K$ or James Estimator of Gaussian perturbed node embeddings $\tilde{\mathbf{h}}_{\text{JS}}^K$.

Theorem 3.1. *(James-Stein Estimator and its adaptivity [3]). Suppose d is the dimension of a local node embedding $\boldsymbol{h}^K$. When $d \geq 3$, substituting w in $\tilde{\mathbf{h}}_{\text{Bayes}}^K$ with its maximum likelihood estimate under*

$$\mathbf{h}^K \sim \mathcal{N}(0, w^2 I), \quad \hat{\mathbf{h}}^K | \mathbf{h}^K \sim \mathcal{N}(\mathbf{h}^K, \sigma^2 C^2 I),$$
$$\tilde{\mathbf{h}}_{\text{Bayes}}^K = argmin_{\tilde{\mathbf{h}}^K} \|\tilde{\mathbf{h}}^K - \mathbf{h}^K\|^2,$$

produces James-Stein Estimator

$$\tilde{\mathbf{h}}_{\text{JS}}^K = \left(1 - \frac{(d-2)\sigma^2 C^2}{\|\hat{\mathbf{h}}^K\|^2}\right)\hat{\mathbf{h}}^K.$$

Moreover, it has an Mean Squared Error (MSE)

$$E[\|\tilde{\mathbf{h}}_{\text{JS}}^K - \mathbf{h}^K\|^2] = d\sigma^2 \left(1 - \frac{(d-2)^2}{d^2} \frac{\sigma^2 C^2}{w^2 + \sigma^2 C^2}\right).$$

The MSE of Gaussian Mechanism $\hat{\mathbf{h}}^K$ to exact embedding $\mathbf{h}^K$ is $E\|\hat{\mathbf{h}}^K - \mathbf{h}^K\|^2 = \sigma^2 C^2 d$. The MSE of James-Stein estimator is reduced with a factor of $(1 - \frac{(d-2)^2}{d^2} \frac{\sigma^2 C^2}{w^2 + \sigma^2 C^2})$ compared to Gaussian Mechanism $\hat{\mathbf{h}}^K$ for any w^2. We observe that the larger the dimension d or the noise magnitude σ^2, the more significant noise reduction we obtain from James-Stein estimator. To conclude, the smaller privacy budget (ϵ, δ) we set, the larger magnitude of σ, the more reduction we can expect via James-Stein estimator.

Both methods preserve (ϵ, δ)-DP while James-Stein estimator shows reductions in MSE, thus improves utility. Replacing $\mathbf{h}^K$ with $\hat{\mathbf{h}}^K$ or $\tilde{\mathbf{h}}_{\text{JS}}^K$ in Algorithm 3 will protect the local node embeddings privately. We will empirically study the effects of differential privacy on model performance in Section 4.

3.6 Putting together

By combining thee sub-computational graphs (**CG1**-**CG3**) and the key components described above, we complete the forward propagation of VFGNN. To describe the procedures in details, we take GraphSAGE [15], a general inductive GNN framework, for example and present its forward propagation process in Algorithm 2. In it, Lines 2-8 denote **CG1** on data holders, and particularly, Line 2 (Algorithm 1) shows how data holders generate initial node embeddings using node feature information collaboratively (Section 3.2), and Lines 3-8 describe how data holders generate local node embeddings separately (Section 3.3), i.e., perform multi-hop neighborhood aggregation using edge information. Note that Lines 3-8 can be replaced with the aggregation strategies of other existing GNNs, e.g., [15, 21, 34], to adapt them to our VFGNN learning paradigm. Lines 10-13 are **CG2** on server, where Line 11 shows how to generate global node embeddings on server (Section 3.4). Line 12 shows how the server makes forward propagation to get the last hidden layer. Line 15 shows how data holder conduct **CG3** using the last hidden layer.

Table 1: Dataset statistics.

Dataset	#Node	#Edge	#Features	#Classes
Cora	2,708	5,429	1,433	7
Pubmed	19,717	44,338	500	3
Citeseer	3,327	4,732	3,703	6

3.7 Model learning

VFGNN can be learnt by gradient descent through minimizing the cross-entropy error over all labeled training examples

$$\mathcal{L} = -\sum_{v \in \mathcal{Y}_v} \sum_{c \in C} y_{vc} \ln(\hat{y}_{vc}), \tag{6}$$

where $\mathcal{Y}_v$ is the set of training nodes that have labels.

Specifically, the model weights of VFGNN are in four parts. The *weights for the initial node embeddings*, i.e., $\langle \mathbf{W} \rangle_i, \forall i \in \mathcal{P}$, that are secretly shared by data holders, the *weights for neighborhood aggregation on graphs*, i.e., $\mathbf{W}_i^k$, that are also kept by data holders, the *weights for the hidden layers of deep neural networks*, i.e., $\mathbf{W}_l, 0 \leq l < L$, that are hold by server, and the *weights for the output layer of deep neural networks*, i.e., $\mathbf{W}_L$, that are hold by the data holder who has label. These weights are trained using gradient descent. In this work, we perform batch gradient descent, i.e., we use the full dataset in each iteration, which is a viable option as long as dataset fits in memory. We leave memory-efficient extensions with mini-batch gradient descent for future work. As can be seen, in VFGNN, both private data and model are hold by data holders themselves, thus the data privacy is guaranteed.

4 EXPERIMENTS

We conduct experiments to answer the following questions:

- **Q1:** whether VFGNN outperforms the GNN models that are trained on the isolated data.
- **Q2:** how does VFGNN behave comparing with the traditional insecure model trained on the plaintext mixed data.
- **Q3:** how does VFGNN perform comparing with the naive solution in Section 1.
- **Q4:** are our proposed combination strategies effective to VFGNN.
- **Q5:** what is the effect of the number of data holders on VFGNN.
- **Q6:** what is the effect of differential privacy on VFGNN.

4.1 Experimental Setup

We first describe the datasets and comparison methods we use in our experiments.

Datasets. We use three benchmark datasets, i.e., Cora, Pubmed, and Citeseer [31]. They are popularly used to evaluate the performance of GNN. We use exactly the same dataset partition of training, validate, and test following the prior work [18]. Besides, in data isolated GNN setting, both node features and edges are hold by different parties. For all the experiments, we split the datasets randomly, use **five-fold cross validation** and adopt accuracy as the evaluation metric.

Comparison methods. We compare VFGNN with GraphSAGE models [15] that are trained using isolated data and mixed plaintext data to answer **Q1** and **Q2**. We also compare VFGNN with the naive solution described in Section 1 to answer **Q3**. To answer **Q4**, we vary the proportion of data (features and edges) hold by $\mathcal{A}$ and $\mathcal{B}$, and change VFGNN with different combination strategies. We vary the number of data holders in VFGNN to answer **Q5**, and vary the parameters of differential privacy to answer **Q6**. For all these models, we choose Mean operator as the aggregator function.

Table 2: Accuracy comparison on three datasets (Q1 and Q2).

Dataset	Cora	Pubmed	Citeseer
GraphSAGE$_{\mathcal{A}}$	0.6110	0.6720	0.5410
GraphSAGE$_{\mathcal{B}}$	0.6060	0.7030	0.4570
VFGNN_C	0.7900	0.7740	0.6850
VFGNN_M	0.8090	0.7810	0.6950
VFGNN_R	0.8020	0.7820	0.6930
GraphSAGE$_{\mathcal{A}+\mathcal{B}}$	0.8150	0.7910	0.7001

Parameter settings. For all the models, we use TanH as the active function of neighbor propagation, and Sigmoid as the active function of hidden layers. For the deep neural network on server, we set the dropout rate to 0.5 and network structure as $(d, d, |C|)$, where $d \in \{32, 64, 128\}$ is the dimension of node embeddings and $|C|$ the nubmer of classes. We vary $\epsilon \in \{1, 2, 4, 8, 16, 32, 64, \infty\}$, set $\delta = 1e^{-4}$ and the clip value $C = 1$ to study the effects of differential privacy on our model. Since we have many comparison and ablation models, and they achieve the best performance with different parameters, we cannot report all the best parameters. Instead, we report the range of the best parameters. We vary the propagation depth $K \in \{2, 3, 4, 5\}$, L2 regularization in $\{10^{-2} - 10^{-4}\}$, and learning rate in $\{10^{-2} - 10^{-3}\}$. We tune parameters based on the validate dataset and evaluate model performance on the test dataset during five-fold cross validation.

4.2 Comparison Results and Analysis

To answer **Q1**-**Q3**, we assume there are two data holders ($\mathcal{A}$ and $\mathcal{B}$) who have equal number of node features and edges, i.e., the proportion of data held by $\mathcal{A}$ and $\mathcal{B}$ is 5:5, and compare our models with GraphSAGEs that are trained on isolated data individually and on mixed plaintext data. We also set $\epsilon = \infty$ during comparison and will study its effects later. We summarize the results in Table 2, where VFGNN_C, VFGNN_M, and VFGNN_R denote VFGNN with Concat, Mean, and Regression combination strategies.

Result1: answer to Q1. We first compare VFGNNs with the GraphSAGEs that are trained on isolated feature and edge data, i.e., GraphSAGE$_{\mathcal{A}}$ and GraphSAGE$_{\mathcal{B}}$. From Table 2, we find that, VFGNNs with different combination strategies significantly outperforms GraphSAGE$_{\mathcal{A}}$ and GraphSAGE$_{\mathcal{B}}$ on all the three datasets. Take Citeseer for example, our VFGNN_R improves GraphSAGE$_{\mathcal{A}}$ and GraphSAGE$_{\mathcal{B}}$ by as high as 28.10% and 51.64%, in terms of accuracy.

Analysis of Result1. The reason of result1 is straightforward. GraphSAGE$_{\mathcal{A}}$ and GraphSAGE$_{\mathcal{B}}$ can only use partial feature and edge information hold by $\mathcal{A}$ and $\mathcal{B}$, respectively. In contrast, VFGNNs provide a solution for $\mathcal{A}$ and $\mathcal{B}$ to train GNNs collaboratively without compromising their own data. By doing this, VFGNNs can use the information from the data of both $\mathcal{A}$ and $\mathcal{B}$ simultaneously, and therefore achieve better performance.

Result2: answer to Q2. We then compare VFGNNs with GraphSAGE that is trained on the mixed plaintext data, i.e., GraphSAGE$_{\mathcal{A}+\mathcal{B}}$. It can be seen from Table 2 that VFGNNs have comparable performance with GraphSAGE$_{\mathcal{A}+\mathcal{B}}$, e.g., 0.8090 vs. 0.8150 on Cora dataset and 0.6950 vs. 0.7001 on Citeseer dataset.

Analysis of Result2. It is not hard to explain why our proposal has comparable performance with the model that are trained on the mixed plaintext data. First, we propose a secret sharing based protocol for $\mathcal{A}$ and $\mathcal{B}$ securely generate the initial node embeddings from their node features, as described in Algorithm 2, which are the same as those generated by using mixed plaintext features. Second, although $\mathcal{A}$ and $\mathcal{B}$ generate local node embeddings by using their own edge data to do neighbor aggregation separately (for security concern), we propose different combination strategies to combine their local node embeddings. Eventually, the edge information from both $\mathcal{A}$ and $\mathcal{B}$ is used for training the classification model. Therefore, VFGNN have comparable performance with GraphSAGE$_{\mathcal{A}+\mathcal{B}}$.

Result3: answer to Q3. In VFGNN, we delegate the non-private data related computations to server. One would be curious about what if these computations are also performed by data holders using existing secure neural network protocols, i.e., SecureML [26]. To answer this question, we compare VFGNN with the secure GNN model that is implemented using SecureML, which we call as SecureGNN, where we use 3-degree Taylor expansion to approximate TanH and Sigmoid. The accuracy and running time per epoch (in seconds) of VFGNN vs. SecureGNN on Pubmed are 0.8090 vs. 0.7970 and 18.65 vs. 166.81, respectively, where we use local area network.

Analysis of Result3. From the above comparison results, we find that our proposed VFGNN learning paradigm not only achieves better accuracy, but also has much better efficiency. This is because the non-private data related computations involve many non-linear functions that are not cryptographically friendly, which have to be approximately calculated using time-consuming MPC techniques in SecureML.

4.3 Ablation Study

We now study the effects of different combination operators and different number of data holders on VFGNN.

Result4: answer to Q4. Different combination operators can utilize local node embeddings in diverse ways and make our proposed VFGNN adaptable to different scenarios, we study this by varying the proportion (Prop.) of data (node features and edges) hold by $\mathcal{A}$ and $\mathcal{B}$ in $\{9:1, 8:2, 7:3\}$. The results on Cora dataset are shown in Table 3.

Analysis of Result4. From Table 3, we find that with the proportion of data hold by $\mathcal{A}$ and $\mathcal{B}$ being even, i.e., from $\{9:1\}$ to $\{7:3\}$, the performances of most strategies tend to decrease. This is because the neighbor aggregation is done by data holders individually, and with a bigger proportion of data hold by a single holder, it is more easier for this party to generate better local node embeddings. Moreover, we also find that Mean operator works well when data are evenly split, and Regression operator is good at handling the situations where data holders have different quatity of data, since it treats the local node embeddings from each data holder differently, and assigns weights to them intelligently.

Table 3: Comparison of combination operators on Cora by varying the proportion of data hold by $\mathcal{A}$ and $\mathcal{B}$ (Q4).

Model	VFGNN_C	VFGNN_M	VFGNN_R
Prop.=9:1	0.8085	0.8050	0.8090
Prop.=8:2	0.8015	0.7960	0.8070
Prop.=7:3	0.7925	0.7925	0.8030

Table 4: Comparison results on Cora by varying the number of data holders (Q5).

No. of DH	VFGNN_C	VFGNN_M	VFGNN_R
2	0.7900	0.8090	0.8020
3	0.7490	0.7740	0.7600
4	0.7120	0.7330	0.7220

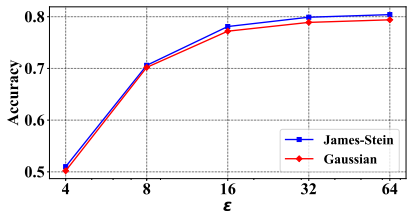


Figure 5: Effect of DP on VFGNN using Cora dataset (Q6).

Result5: answer to Q5. We vary the number of data holders in {2, 3, 4} and study the performance of VFGNN. We report the results in Table 4, where we use the Cora dataset and assume data holders have even feature and edge data.

Analysis of Result5. From Table 4, we find that, as the number of data holders increases, the accuracy of all the models decreases. This is because the neighborhood aggregation in VFGNN is done by each holder individually for privacy concern, and each data holder will have less edge data when there are more data holders, since they split the original edge information evenly. Therefore, when more participants are involved, more information are lost during the neighborhood aggregation procedure.

Result6: answer to Q6. We vary ϵ and set $\delta = 1e^{-4}$ to study the effects of Differential Privacy (DP) on our model. We report the results in Figure 5, where we use Cora dataset and assume data holders have even feature and edge data.

Analysis of Result6. From Figure 5, we can see that the accuracy of VFGNN increase with ϵ. In other words, there is a trade-off between accuracy and privacy. The smaller ϵ, the bigger noises are added into the local node embeddings, which causes stronger privacy guarantee but lower accuracy. We also find James-Stein estimator consistently works better than Gaussian mechanism, since it can reduce MSE, as we have analyzed in Section 3.5.

5 RELATED WORK

We review three kinds of existing Privacy-Preserving Neural Network (PPNN) models, including GNN models.

PPNN based on cryptographic methods. These methods mainly use cryptographic techniques, e.g., secret sharing and homomorphic encryption, to build approximated neural networks models

[26, 36], since the nonlinear active functions are not cryptographically computable. Cryptograph based neural network models are difficult to scale to deep networks and large datasets due to its high communication and computation complexities. In this paper, we use cryptographic techniques for data holders to calculate the initial node embeddings securely.

PPNN based on federated learning. The privacy issues in machine learning has boosted the development of federated learning. To date, federated neural networks have been extensively studied [5, 19] and applied into real-world scenarios [20]. There are also several work on federated GNN when data are horizontally partitioned [7, 38, 45]. However, a mature solution for federated GNN models under vertically partitioned data is still missing. In this paper, we fill this gap by proposing VFGNN.

PPNN based on split computation graph. These methods split the computation graph of neural networks into two parts, and let data holders calculate the private data related computations individually and get a hidden layer, and then let a server makes the rest computations [8, 14, 28, 35]. Our model differs from them in mainly two aspects. First, we train a GNN rather than a neural network. Seconds, we use cryptographic techniques for data holders to calculate the initial node embeddings collaboratively rather than compute them based on their plaintext data individually.

6 CONCLUSION AND FUTURE WORK

We proposed VFGNN,a vertically federated GNN learning paradigm for privacy-preserving node classification task. We did this by splitting the computation graph of GNN. We left the private data related computations on data holders and delegated the rest computations to a server. Experiments on real world datasets demonstrate that our model significantly outperforms the GNNs by using the isolated data and has comparable performance with the traditional GNN by using the mixed plaintext data insecurely. In future, we would like to further strength the security of VFGNN besides publishing local node embeddings using differential privacy.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 79.

[2] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM, 142–144.

[3] Borja Balle and Yu-Xiang Wang. 2018. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. *arXiv preprint arXiv:1805.06530* (2018).

[4] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*. Springer, 420–432.

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019).

[6] Chaochao Chen, Liang Li, Bingzhe Wu, Cheng Hong, Li Wang, and Jun Zhou. 2020. Secure social recommendation based on secret sharing. In *ECAI*. 506–512.

[7] Mingyang Chen, Wen Zhang, Zonggang Yuan, Yantao Jia, and Huajun Chen. 2020. FedE: Embedding Knowledge Graphs in Federated Setting. *arXiv preprint arXiv:2010.12882* (2020).

[8] Jianfeng Chi, Emmanuel Owusu, Xuwang Yin, Tong Yu, William Chan, Patrick Tague, and Yuan Tian. 2018. Privacy Partitioning: Protecting User Data During the Deep Learning Inference Phase. *arXiv preprint arXiv:1812.02863* (2018).

[9] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[10] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[11] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[12] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*. 417–426.

[13] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies* 2017, 4 (2017), 345–364.

[14] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Securing input data of deep learning inference systems via partitioned enclave execution. *arXiv preprint arXiv:1807.00969* (2018).

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[16] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[17] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. 2020. Stealing Links from Graph Neural Networks. *arXiv preprint arXiv:2005.02131* (2020).

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

[19] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[20] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. 2020. A review of applications in federated learning. *Computers & Industrial Engineering* (2020), 106854.

[21] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[22] Yehida Lindell. 2005. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*. IGI Global, 1005–1009.

[23] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*. Springer, 277–346.

[24] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4424–4431.

[25] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2077–2085.

[26] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.

[27] Medhini Narasimhan, Svetlana Lazebnik, and Alexander G Schwing. 2018. Out of the box: reasoning with graph convolution nets for factual visual question answering. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2659–2670.

[28] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Kleomenis Katevas, Sina Sajadmanesh, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2019. A hybrid deep learning architecture for privacy-preserving mobile analytics. *arXiv preprint arXiv:1703.02952* (2019).

[29] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX Security Symposium*. 797–812.

[30] Sina Sajadmanesh and Daniel Gatica-Perez. 2020. Locally Private Graph Neural Networks. *CoRR abs/2006.05535* 12 (2020).

[31] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[32] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[33] Charles Stein. 2020. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Contribution to the Theory of Statistics*. University of California Press, 197–206.

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

[35] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018).

[36] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proceedings on Privacy Enhancing Technologies* 1 (2019), 24.

[37] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2018. Dynamic graph cnn for learning on point clouds. *arXiv*

preprint arXiv:1801.07829 (2018).

[38] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. FedGNN: Federated Graph Neural Network for Privacy-Preserving Recommendation. *arXiv preprint arXiv:2102.04925* (2021).

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[40] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 12.

[41] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7370–7377.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 974–983.

[43] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. 5165–5175.

[44] Longfei Zheng, Chaochao Chen, Yingting Liu, Bingzhe Wu, Xibin Wu, Li Wang, Lei Wang, Jun Zhou, and Shuang Yang. 2020. Industrial Scale Privacy Preserving Deep Neural Network. *arXiv preprint arXiv:2003.05198* (2020).

[45] 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* (2021), 1–13.

[46] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).