SecGNN: Privacy-Preserving Graph Neural Network Training and Inference as a Cloud Service

Songlei Wang, Yifeng Zheng, and Xiaohua Jia, *Fellow, IEEE*

***Abstract*—Graphs are widely used to model the complex relationships among entities. As a powerful tool for graph analytics, graph neural networks (GNNs) have recently gained wide attention due to its end-to-end processing capabilities. With the proliferation of cloud computing, it is increasingly popular to deploy the services of complex and resource-intensive model training and inference in the cloud due to its prominent benefits. However, GNN training and inference services, if deployed in the cloud, will raise critical privacy concerns about the information-rich and proprietary graph data (and the resulting model). While there has been some work on secure neural network training and inference, they all focus on convolutional neural networks handling images and text rather than complex graph data with rich structural information. In this article, we design, implement, and evaluate SecGNN, the first system supporting privacy-preserving GNN training and inference services in the cloud. SecGNN is built from a synergy of insights on lightweight cryptography and machine learning techniques. We deeply examine the procedure of GNN training and inference, and devise a series of corresponding secure customized protocols to support the holistic computation. Extensive experiments demonstrate that SecGNN achieves comparable plaintext training and inference accuracy, with promising performance.**

***Index Terms*—Cloud computing services, graph neural networks, model training and inference services, privacy preservation.**

I. Introduction

GRAPHS have been widely used to model and manage data in various real-world applications, including recommendation systems [1], social networks [2] and webpage networks [3]. Graph data, however, is highly complex and inherently sparse, making graph analytics challenging [4]. With the rapid advancements in deep learning, Graph Neural Networks (GNNs) [5] have recently gained a lot of traction as a powerful tool for graph analytics due to its end-to-end processing capabilities. GNNs can empower a variety of graph-centric applications such as node classification [6], edge classification [7] and link prediction [8]. With the widespread adoption of cloud computing, it is increasingly popular to deploy machine learning training and inference services in the cloud [9], [10], due to the well-understood benefits [11], [12]. However, GNN training and inference, if deployed in the public cloud, will raise critical severe privacy concerns. Graph data is information-rich and can reveal a considerable amount of sensitive information. For example, in a social network graph, the connections between nodes represent users' circles of friends and each node's features represent each user's preferences. Meanwhile, the graph data as well as the trained GNN model are the proprietary to the data owner, so revealing them may easily harm the business model. Therefore, security must be embedded in outsourcing GNN training and inference to the cloud.

In the literature, privacy-preserving machine learning has received great attention in recent years, especially the design of secure protocols for neural network-based applications. A number of research efforts have been proposed for secure neural network inference and training. Most of existing works [13], [14], [15], [16], [17], [18], [19], [20], [21] are focused on designing specialized protocols for secure inference, and only a few works [22], [23], [24], [25], [26] study secure training which is more sophisticated and resource-intensive. However, prior works are all focused on the support for Convolutional Neural Networks (CNNs) handling unstructured data like images and text. How to achieve secure in-the-cloud training and inference of GNNs that handle complex graph data remains unexplored.

Supporting secure training and inference of GNNs in the cloud, however, faces unique challenges and require delicate treatments due to the complex structured nature of graphs. There are various kinds of structural information in graphs: 1) relationships between nodes (i.e., edges), 2) edge weights, and 3) number of neighboring nodes (i.e., degrees of nodes). Designing solutions for securing GNN training and inference thus demands protection for not only numerical information (e.g., the values of features associated with nodes) but also the rich structural information unique to different graphs.

In light of the above, in this article, we present the first research endeavor towards privacy-preserving training and inference of GNNs in the cloud. We design, implement, and evaluate a new system SecGNN, which allows a data owner to send encrypted graph data to the cloud, which can then effectively train a GNN model without seeing the graph data as well as provide secure inference once an encrypted GNN model is trained. Targeting privacy assurance as well as high efficiency, SecGNN builds

Manuscript received 3 February 2022; revised 20 January 2023; accepted 30 January 2023. Date of publication 2 February 2023; date of current version 8 August 2023. This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grants 2021A1515110027 and 2023A1515010714, and in part by the Shenzhen Science and Technology Program under Grants RCBS20210609103056041 and JCYJ20220531095416037. Recommended for acceptance by E. Damiani. *(Corresponding author: Yifeng Zheng.)*

Songlei Wang and Yifeng Zheng are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China (e-mail: songlei.wang@outlook.com; yifeng.zheng@hit.edu.cn).

Xiaohua Jia is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China, and also with the Department of Computer Science, City University of Hong Kong, Hong Kong, China (e-mail: csjia@cityu.edu.hk).

Digital Object Identifier 10.1109/TSC.2023.3241615

on only lightweight cryptographic techniques (mainly additive secret sharing) for efficient graph data encryption at the data owner as well as secure training and inference at the cloud side. To be compatible with the working paradigm of additive secret sharing, SecGNN employs a multi-server and decentralized trust setting where the power of the cloud is split into three cloud servers that are hosted by independent cloud service providers. The adoption of such a multi-server model to facilitate security applications in various contexts has gained increasing traction in prior works [24], [26], [27] as well as in industry [28], [29]. SecGNN leverages the above trend and contributes a new design point of secure GNN training and inference in the cloud through highly customized cryptographic protocols.

We start with considering how to appropriately encrypt the graph data in SecGNN so that it can still be effectively used at the cloud for secure training and inference. As mentioned above, graphs contain not only numerical information (i.e., feature vectors associated with the nodes) but also structural information connecting the nodes, all demanding strong protection. The challenge here is to how to encrypt the structural information in an effective and efficient manner. One may try to directly encrypt the adjacency matrix of the graph of size $O(N^2)$, where N is the number of nodes, with additive secret sharing.

Such a simple method, however, is neither efficient nor necessary. First, there can be tens of thousands or even millions of nodes in a graph for practical applications, leading to the adjacency matrix being of very large size. Directly encrypting the adjacency matrix would incur significant overheads. Second, graphs are usually sparse, leading to the adjacency matrix being sparse and filled with many zeros. Encrypting all the zeros in the adjacency matrix would result in unnecessary cost as well. To tackle this challenge, our insight is to devise a set of customized data structures to appropriately store and represent the structural information, and so the encryption is performed over these data structures rather than the original (big) adjacency matrix. With our customized data structures, the complexity of encryption significantly reduces to $O(N \cdot d_{max})$, where d_{max} is the maximum degree of nodes in the graph, and far less than the number of nodes (e.g., only 0.87% to 6.2% as practically observed in our experiments over several popular real-world graph datasets).

Subsequently, we consider how to securely perform training and inference at the cloud over the delicately-encrypted graph data in SecGNN. Through an in-depth examination on the computation required in GNN training and inference, we decompose the holistic computation into a series of functions and devise corresponding tailored secure constructions with the lightweight additive secret sharing technique. Specifically, we manage to decompose the whole procedure into secure feature normalization, secure neighboring states aggregation, secure activation functions, and secure model convergence evaluation.

SecGNN supports secure feature normalization through realizing secure division with effective approximation mechanisms. For secure neighboring states aggregation in SecGNN, our insight is to transform the problem into secure array access over encrypted arrays and indexes. We design a secure array access protocol building on the state-of-the-art yet achieving much improved efficiency, through customized mechanisms. To support the secure evaluation of activation functions (ReLU and Softmax), SecGNN mainly leverages insights from digital circuit design and provides tailored protocols in the secret sharing domain, rather than relying on expensive garbled circuits as in prior work [22].

Last but not least, SecGNN provides the first mechanism for secure convergence evaluation, allowing fine-grained control on the secure training process. This is in substantial contrast to prior work on secure (CNN) training which simply sets a fixed number for the training epochs and thus may not necessarily meet convergence. The synergy of these customized secure and efficient components leads to SecGNN, the first system supporting secure GNN training and inference in the cloud. The security of SecGNN is formally analyzed. We implement SecGNN and conduct extensive experiments over multiple real-world graph datasets. The evaluation results demonstrate that SecGNN, while providing privacy protection in training and inference, achieves comparable plaintext accuracy, with promising performance.

We highlight our contributions below:

- We present SecGNN, the *first* system supporting privacy-preserving GNN training and inference as a cloud service, through a delicate synergy of lightweight cryptography and machine learning.
- We devise customized data structures to facilitate efficient and effective graph data encryption, and thoroughly propose a series of customized secure protocols to support the essential components required by secure GNN training and inference.
- Among others, notably SecGNN provides a secure array access protocol with much improved efficiency over the state-of-the-art as well as the first secure fine-grained convergence evaluation protocol, which can be of independent interests.
- We make a full-fledged implementation of SecGNN and conduct an extensive evaluation over a variety of real-world graph datasets. The experiment results demonstrate the performance efficiency of SecGNN.

The rest of this article is organized as follows. Section II discusses the related work. Section III introduces preliminaries. Section IV presents the problem statement. Section V gives the design of SecGNN. The security analysis is presented in Section VI, followed by the experiments in Section VII. Finally, we conclude this article in Section VIII.

II. Related Work

A. Graph Neural Networks in Plaintext Domain

Graphs can characterize the complex inter-dependency among data and are widely used in many applications, such as citation networks, social media networks, webpage networks [4]. GNN models have the strong ability of capturing the dependence of graphs through message passing between the nodes of graphs, and have shown impressive performance in graph processing tasks. The first GNN model was proposed in the seminal work of Scarselli et al. [3]. Since then, many advanced GNN models targeting different applications and with varying capabilities have been put forward. In general, GNN models can be divided into three categories: Gated Graph Neural Networks (GGNN)

[30], Graph Convolutional Networks(GCN) [31], and Graph ATtention networks (GAT) [32]. GGNN models are proposed to accommodate applications that require to output sequences about a graph such as drug discovery [33]. GCN models are variants of CNNs which operate directly on graphs, and it is typically used for graphs with relatively stable nodes such as recommendation systems [6]. GAT models introduce an attention-based architecture to calculate the weight of neighboring nodes, so that the whole network information can be obtained without knowing the structure of the whole graph, which is also commonly used in recommendation systems [34]. Although the above GNN models can achieve excellent performance on graph-structured data, they are trained and work in the plaintext domain without considering privacy protection.

B. Secure Neural Network Training and Inference

There has been a surge of interests on developing methods for secure neural network training and inference in recent years. Most of existing works [13], [14], [15], [16], [17], [18], [19], [20], [21] are focused on secure inference, and operated under different settings. Some works [13], [14], [15], [16], [19], [21] consider a 2-party setting where a model owner and a client directly engage in tailored cryptographic protocols for secure inference. Their security goal is that through the interactions the model owner learns no information while the client only learns the inference result. In contrast, some works [17], [18], [20] consider an outsourced setting where a set of cloud servers are employed to perform secure inference over encrypted neural networks and inputs. Throughout the procedure, the cloud servers learn no information about the models, inputs, and inference results. The cryptographic techniques adopted by the above works in different settings usually include homomorphic encryption, garbled circuits, and secret sharing. In comparison with secret sharing, homomorphic encryption and garbled circuits are relatively expensive and usually incur large performance overheads.

In contrast with secure inference, secure training of neural networks is much more challenging because more complex operations would be required, and a large dataset needs to be processed in the ciphertext domain. In the literature, only a few works study the problem of secure neural network training. Mohassel et al. [22] propose the first secure training method for shallow neural networks under a two-server setting, based on secret sharing (for linear operations) and garbled circuits (for approximated activation functions). Subsequently, several works [23], [24], [25], [26] achieve better performance in accuracy and efficiency by devising customized secure training protocols in a three-server setting. Despite being useful, existing works on secure neural network are focused on CNN models that do not support the processing of graph data. In light of this gap, in this article we present the first research endeavor towards privacy-preserving training and inference of GNNs outsourced to the cloud, providing techniques for adequately encrypting graph data and securely supporting the essential operations required in GNN training and inference. Following the trend as in prior work, our design adopts a similar three-server architecture, and only make use of lightweight cryptographic techniques in devising our secure protocol highly customized for secure GNN training and inference.

C. Federated Learning-Based Private GNN Training

There are some works [35], [36], [37], [38], [39], [40] focusing on privacy-preserving training of GNNs under the federated learning paradigm, where GNNs are trained across multiple clients holding local graph datasets in such a way that the graph datasets stay local. Specifically, the work [35] focuses on GNNs over decentralized spatio-temporal data, and has the clients exchange model updates with the central server in plaintext. In contrast, the works [36], [37] focus on distributed graph datasets where each client only holds a subgraph and design privacy-preserving mechanisms to protect the individual model updates. Different from [36], [37], the work [38] focuses on vertically federated GNN, where all clients hold the same graph nodes, but different node features and edges. The work [39] considers federated dynamic GNN, which learns the representations of the objects at each timestamp by capturing the structural and patterns in the dynamic graph sequence. Pei et al. [40] focus on decentralized federated GNN, which allows multiple clients to train a GNN model without a centralized server and introduces the Diffie-Hellman key exchange method [41] to achieve secure model aggregation between clients. These federated learning-based works all target system models that are substantially different from ours. SecGNN targets an outsourced setting where the graph data owner can send its encrypted graph data to the cloud for secure training and can simply offline offline during the training process. In the meantime, SecGNN readily supports secure GNN inference over encrypted GNNs and inputs as well, while those works can only deal with private training.

D. Other Related Work

There are some other works [42], [43] focusing on making the node features and edges differentially private [44] when clients share their graph data to the central server or other clients during GNN models training. Specifically, the work [42] considers that a server holds a graph, whose nodes, which correspond to real users, have some private features that the server wishes to utilize for training a GNN model on the graph. The work [43] considers a distributed scenario where each client has all nodes but only partial private edges for a graph, and the clients wish to collaboratively train a GNN model on the distributed graph. These works [42], [43] protect graph data privacy at the cost of notable accuracy degradation and rely on delicate parameter tuning for balancing accuracy and privacy. In independent work, Wang et al. [45] propose a privacy-preserving representation learning framework on graphs from the mutual information perspective. The framework considers a *centralized* GNN training scenario and focuses on preventing the trained GNN models from leaking the training data by bounding the node features, node label, and link status during training GNN models.

III. Preliminaries

A. Graph Neural Networks

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of *nodes* $\mathcal{V}$ and connections between nodes, i.e., *edges* $\mathcal{E}$. Two nodes connected by an edge are *neighboring nodes*. The neighboring nodes of each node $v_i \in \mathcal{V}$ is denoted by $\{ne_{i,j}\}_{j\in 1,d_i}$, where d_i is node v_i's *degree*. GNNs deal with graph-structured data, where each node in the graph is associated with a *feature* vector and some of the nodes are *labeled nodes*, each of which carries a *classification label*. Formally, we define the graph-structured data in GNNs as $\mathcal{D} = \{\mathbf{A}, \mathbf{F}, \mathbf{T}\}$. Here, $\mathbf{A}$ is the *adjacency matrix* of the graph, where $\mathbf{A}_{i,j}$ is an element in $\mathbf{A}$: If there exists an edge between node v_i and node v_j, then $\mathbf{A}_{i,j} = 1$ (binary graph) or $\mathbf{A}_{i,j} = w_{i,j}$ (weighted graph), and otherwise $\mathbf{A}_{i,j} = 0$. In addition, Each row of $\mathbf{F}$ (denoted as $\mathbf{F}_{v_i}$) is node v_i's feature vector, and each row of $\mathbf{T}$ (denoted as $\mathbf{T}_{v_i}$) is the classification label vector (one-hot encoding [46]) of labeled node $v_i \in \mathcal{T}$, where $\mathcal{T}$ is the set of labeled nodes.

Utilizing the graph-structured data $\mathcal{D}$, a GNN model can be trained to perform graph analytic tasks. In this article, we focus on GCN as the first instantiation, which is well-established and the most representative GNN model [31]. With a trained GCN model, the classification labels of the unlabeled nodes can be inferred. At a high level, this proceeds as follows. Given an unlabeled node v_i, the trained GCN model infers its *state* vector $\mathbf{x}_{v_i}^{(k)}$ (row vector) in the k_{th}-layer of the GCN. The dimension of the state vector decreases along with the layer propagation in the GCN. The last layer state vector $\mathbf{x}_{v_i}^{(K)}$ is the inference result of node v_i, which is usually a probability vector with length C and C is the number of possible classification labels. Finally, the node v_i is labeled with the class having the maximum probability.

Without loss of generality and to facilitate the presentation, we elaborate on a representative two-layer GCN model [31] as follows, and will use it to illustrate the design of our SecGNN afterwards. The GCN's propagation model is:

$$\mathbf{Z} = \text{Softmax}(\hat{\mathbf{A}}\text{ReLU}(\hat{\mathbf{A}}\mathbf{F}\mathbf{M}^{(1)})\mathbf{M}^{(2)}), \tag{1}$$

where $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(2)}$ are two trainable weight matrices. $\hat{\mathbf{A}}$ is a symmetric normalized adjacency matrix $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with self-connection added ($\mathbf{I}$ is the identity matrix). $\tilde{\mathbf{D}}$ is a diagonal matrix:

$$\tilde{\mathbf{D}}_{i,i} = sw_{v_i} = \sum_{j\in[1,N]} \tilde{\mathbf{A}}_{i,j} = 1 + \sum_{j\in[1,d_i]} w_{i,j}, \tag{2}$$

where N is the number of nodes in the graph, d_i is the degree of node v_i and sw_{v_i} is the sum of v_i's edge weights. Namely, $\tilde{\mathbf{D}}_{i,i}$ is the sum of node v_i's edge weights with self-connection added (i.e, $w_{i,i} = 1$). In particular, for a binary graph we have $\tilde{\mathbf{D}}_{i,i} = d_i + 1$. The activation function $\text{ReLU}(x)$ is defined as [47]:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0, \end{cases} \tag{3}$$

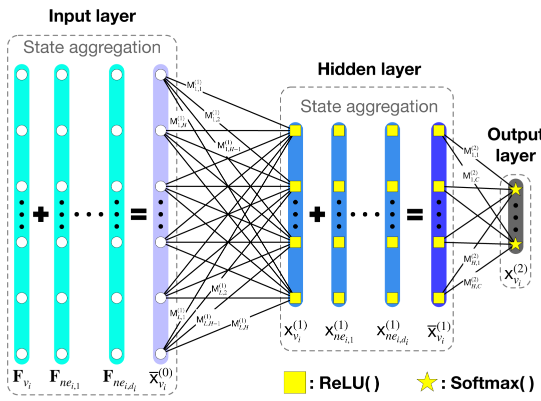


Fig. 1. Illustration of the two-layer GCN model.

and the activation function $\text{Softmax}(\mathbf{x})$ is defined as [48]:

$$z_i = \frac{e^{x_i}}{\sum_{j\in[1,C]} e^{x_j}}, i \in [1, C], \tag{4}$$

where C is the number of possible classification labels.

To *train* the GCN model, the *forward propagation* (i.e., (1)) is performed for each labeled node, and then the two trainable weight matrices $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(2)}$ are updated based on the difference between each labeled node's inference result and label vector through backward propagation. Fig. 1 illustrates the process of performing the forward propagation for a node v_i:

1) The 0_{th}-layer *aggregate state* $\overline{\mathbf{x}}_{v_i}^{(0)}$ of node v_i is the weighted sum of the states of its neighbors and its own:

$$\overline{\mathbf{x}}_{v_i}^{(0)} = (\hat{\mathbf{A}}\mathbf{F})_{v_i} = \hat{\mathbf{A}}_{v_i,v_i}\mathbf{F}_{v_i} + \sum_{j\in[1,d_i]} \hat{\mathbf{A}}_{v_i,ne_{i,j}}\mathbf{F}_{ne_{i,j}}, \tag{5}$$

where $(\hat{\mathbf{A}}\mathbf{F})_{v_i}$ is the vector in row v_i of matrix $\hat{\mathbf{A}}\mathbf{F}$, $\{\mathbf{F}_{ne_{i,j}}\}_{j\in[1,d_i]}$ are the feature vectors of v_i's neighbors and $\hat{\mathbf{A}}_{v_i,ne_{i,j}}$ is the element in row v_i and column $ne_{i,j}$ of matrix $\hat{\mathbf{A}}$.

2) The 1_{st}-layer state $\mathbf{x}_{v_i}^{(1)}$ of node v_i is

$$\mathbf{x}_{v_i}^{(1)} = \text{ReLU}(\overline{\mathbf{x}}_{v_i}^{(0)}\mathbf{M}^{(1)}). \tag{6}$$

3) The 1_{st}-layer aggregate state of node v_i is

$$\overline{\mathbf{x}}_{v_i}^{(1)} = (\hat{\mathbf{A}}\mathbf{X}^{(1)})_{v_i} = \hat{\mathbf{A}}_{v_i,v_i}\mathbf{x}_{v_i}^{(1)} + \sum_{j\in[1,d_i]} \hat{\mathbf{A}}_{v_i,ne_{i,j}}\mathbf{x}_{ne_{i,j}}^{(1)}, \tag{7}$$

where $\mathbf{X}^{(1)}$ is all nodes' 1_{st}-layer states.

4) The 2_{nd}-layer state of node v_i is

$$\mathbf{Z}_{v_i} = \mathbf{x}_{v_i}^{(2)} = \text{Softmax}(\overline{\mathbf{x}}_{v_i}^{(1)}\mathbf{M}^{(2)}) \tag{8}$$

which denotes the inference result of node v_i.

After producing all labeled nodes' inference results through forward propagation, the average cross-entropy loss can be calculated by using all labeled nodes' labels and inference results:

$$\mathcal{L} = -\frac{1}{|\mathcal{T}|}\sum_{v_i\in\mathcal{T}}\sum_{j\in[1,C]} \mathbf{T}_{v_i,j} \ln \mathbf{Z}_{v_i,j}, \tag{9}$$

where $\mathcal{T}$ is the set of labeled nodes and $\mathbf{T}_{v_i,j}$ is the classification label of node v_i, class j. Finally, each weight $\mathbf{M}_{i,j} \in \mathbf{M}^{(1)} \cup \mathbf{M}^{(2)}$ can be updated by its gradient:

$$\mathbf{M}_{i,j} = \mathbf{M}_{i,j} - \rho \frac{\partial \mathcal{L}}{\partial \mathbf{M}_{i,j}},$$

where ρ is the learning rate. After the GCN model is trained, the classification label of each unlabeled node can be inferred through the forward propagation process.

B. Additive Secret Sharing

The 2-out-of-2 additive secret sharing of a secret value x is denoted as $[\![x]\!]$, which can have the following two types [22]:

- *Arithmetic sharing*: $[\![x]\!]^A = \langle x \rangle_1 + \langle x \rangle_2$ where $x, \langle x \rangle_1, \langle x \rangle_2 \in \mathbb{Z}_{2^k}$, and $\langle x \rangle_1, \langle x \rangle_2$ held by two parties, respectively.
- *Binary sharing*: $[\![b]\!]^B = \langle b \rangle_1 \oplus \langle b \rangle_2$ where $b, \langle b \rangle_1, \langle b \rangle_2 \in \mathbb{Z}_2$, and $\langle b \rangle_1, \langle b \rangle_2$ held by two parties, respectively.

The basic operations in the secret sharing domain under a two-party setting are as follows. (1) *Linear operations.* Linear operations on secret-shared values only require local computation. In arithmetic sharing, if α, β, γ are public constants and $[\![x]\!]^A$, $[\![y]\!]^A$ are secret-shared values, then

$$[\![\alpha x + \beta y + \gamma]\!]^A = (\alpha\langle x \rangle_1 + \beta\langle y \rangle_1 + \gamma, \alpha\langle x \rangle_2 + \beta\langle y \rangle_2).$$

Each party can compute their respective shares locally based on the secrets they hold. (2) *Multiplication.* Multiplication on secret-shared values requires one round of online communication. To multiply two secret-shared values: $[\![z]\!]^A = [\![x]\!]^A \times [\![y]\!]^A$, the two parties should first share a Beaver triple $[\![w]\!]^A = [\![u]\!]^A \times [\![v]\!]^A$ in the offline phase. After that, the party $P_{i\in\{0,1\}}$ locally computes $\langle e \rangle_i = \langle x \rangle_i - \langle u \rangle_i$ and $\langle f \rangle_i = \langle y \rangle_i - \langle v \rangle_i$, and then opens e, f to each other. Finally, P_i holds $\langle z \rangle_i = i \times e \times f + f \times \langle u \rangle_i + e \times \langle v \rangle_i + \langle w \rangle_i$.

In binary sharing, the operations are similar to arithmetic sharing. In particular, the addition operation is replaced by the XOR ($\oplus$) operation and multiplication is replaced by the AND ($\otimes$) operation.

IV. Problem Statement

A. System Architecture

There are two kinds of entities in SecGNN: the data owner and the cloud. The data owner (e.g., an online shopping enterprise or a social media service provider) wants to leverage the power of cloud computing to train a GNN model over his proprietary graph data as well as provide on-demand inference services once the model is trained. Due to privacy concerns and that the graph data is proprietary, it is demanded that security must be embedded in the outsourced service, safeguarding the graph data, the trained model, as well as the inference results along the whole service flow. The cloud providing the secure GNN training and inference is split into three cloud servers $P_{\{1,2,3\}}$ which can be operated by independent cloud service providers (e.g., AWS, Google, and Microsoft) in practice. Such multi-server model has also gained increasing traction in prior works on building efficient secure systems for other application domains [26], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59]. In addition to the adoption in academia, such multi-server model has also been deployed in industry. For example, Mozilla provides a service of lightweight private collection of telemetry data about Firefox under the non-colluding multi-server model [28]; Apple and Google cooperatively provide automated alerts about potential COVID-19 exposure to users, while providing strong privacy protections [29]. SecGNN also follows such trend and contributes a new design for enabling privacy-preserving training and inference of GNNs in the cloud.

From a high-level point of view, the data owner in SecGNN will encrypt the graph by *adequately* splitting the graph-structured data into secret shares under 2-out-of-2 additive secret sharing, as per our design. The secret shares are sent to P_1 and P_2, respectively. Upon receiving the encrypted graph-structured data, $P_{\{1,2,3\}}$ perform our SecGNN to train the encrypted GNN model in the secret sharing domain. Once the encrypted GNN model is trained, the data owner can query the cloud service to obtain encrypted classification labels for unlabeled nodes for decryption. It is noted that the major computation in SecGNN is undertaken by the cloud servers P_1 and P_2 while P_3 provides necessary assistance, so as to simplify the interactions (and so the system implementation and deployment) as much as possible.

B. Threat Model

Similar to prior security designs in the three-server setting [24], [26], [27], we consider a semi-honest adversary setting where each of the three cloud servers honestly follow our protocol, but may *individually* attempt to learn the private information of the data owner. The rationality of the non-collusion assumption is that the cloud service providers hosting the three cloud servers are normally business-driven and well-established parties, who are thus unwilling to risk their valuable commercial reputation by colluding with each other to intentionally breach data privacy [60], [61], [62]. We consider that the data owner wishes to keep the following information private: (i) the features $\mathbf{F}$ and labels $\mathbf{T}$ of nodes, (ii) the adjacency matrix $\mathbf{A}$ encoding the structural information regarding the neighboring nodes of each node, the number of neighbors of each node, and the edge weight between each pair of connected nodes, (iii) the model weights $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(2)}$, and (iv) the inference results for (unlabeled) nodes.

V. Secure GNN Training and Inference

A. SecGNN Overview

Without loss of generality, we will use the two-layer GCN in (1) to illustrate the design of secure training and inference in SecGNN. Fig. 2 provides an overview of the core components in SecGNN. We will start with designing a *secure input preparation* method, which allows the data owner to adequately encrypt its graph-structured data so that they can support secure training and inference at the cloud. Subsequently, we design the following essential components to support the secure training and inference procedure at the cloud: (i) *secure initialization* where the

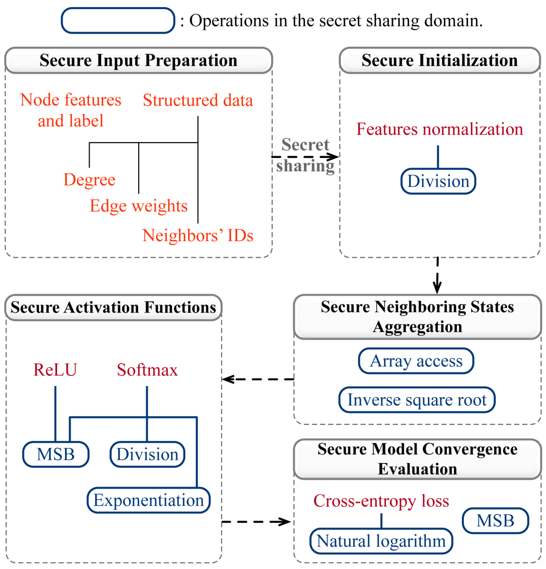


Fig. 2. Overview of the core components in SecGNN.

cloud normalizes the encrypted features for each node, (ii) *secure neighboring states aggregation* where the cloud computes the encrypted aggregate state (as shown in (5) and (7)) for each node, (iii) *secure activation functions* where the cloud activates the encrypted aggregate state for each node, and (iv) *secure model convergence evaluation* where the cloud performs a secure and fine-grained protocol to evaluate the convergence of the training process. Finally, we will elaborate on how to bridge the designed secure components to give the complete protocol for secure GCN training and inference.

B. Secure Input Preparation

Encrypting Node Features and Labels: Given each node v_i's initial feature vector with length L: $\mathbf{F}_{v_i} \in \mathbb{Z}_{2^k}^L$, the data owner generates a random vector $\mathbf{r} \in \mathbb{Z}_{2^k}^L$. Then the arithmetic ciphertext of $\mathbf{F}_{v_i}$ is the secret shares $\langle \mathbf{F}_{v_i} \rangle_1 = \{(\mathbf{F}_{v_i,s} - \mathbf{r}_s) \bmod \mathbb{Z}_{2^k}\}_{s=1}^L$ and $\langle \mathbf{F}_{v_i} \rangle_2 = \{\mathbf{r}_s\}_{s=1}^L$ where $\langle \mathbf{F}_{v_i} \rangle_j$ is sent to $P_j, j \in \{1,2\}$. Similarly, the data owner splits each labeled node's label vector $\mathbf{T}_{v_i}$ into secret shares.

Encrypting Structural Information: The structural information includes 1) each node's degree d_i; 2) the neighbors' IDs $ne_{i,j}$ of all nodes; 3) edge weights $w_{i,j}$ between all connected nodes. To protect the structural information, a simple method is to split the adjacency matrix $\mathbf{A}$ into secret shares. However, this method is inefficient and unnecessary since the adjacency matrix $\mathbf{A}$ is usually sparse.

Instead, our insight is to devise a set of data structures to properly store and represent the necessary structural information so that they can be encrypted efficiently as well as be used for GCN training and inference. In particular, we represent the structural information with an array-like data structure where each array element refers to a node's neighbor ID list and an edge weight list, and the array index is the node's ID.

It is noted that as the degrees of nodes are different, the length of nodes' neighbor ID lists varies. To protect each node's degree, the data owner pads several dummy neighbors' IDs to each node's neighbor ID list so that all nodes have the same number of neighbors. Namely, the secure neighbor ID list of v_i is

$$\mathbf{Ne}_{v_i} = \{ne_{i,1}, \ldots, ne_{i,d_i}\} \cup \{ne'_{i,1}, \ldots, ne'_{i,d_{max}-d_i}\},$$

where ne' are dummy neighbors' IDs, d_i is v_i's degree and d_{max} is the maximum degree in the graph. However, if these dummy neighbors' IDs point to nodes that do not exist in the graph, the cloud servers will distinguish them from $\mathbf{Ne}_{v_i}$ when accessing these dummy neighbors, while if the dummy neighbors' IDs point to real nodes in the graph, the accuracy of the trained model will be degraded dramatically since dummy neighbors' states will change node v_i's aggregate state.

Our solution is based on the observation that in GCN or GNN, a node's aggregate state is the weighted sum of its neighboring states, where the weights are relevant with v_i's edge weights $w_{i,j}$. Therefore, we can set the edge weights between v_i and its dummy neighbors to 0. Namely, v_i's secure edge weight list is $\mathbf{W}_{v_i} = \{w_{i,1}, \ldots, w_{i,d_i}\} \cup \{0_j\}_{j=1}^{d_{max}-d_i}$. By this way, the effect of the dummy neighbors will be eliminated, which will be understood clearly in Section V-D. After padding dummy neighbors, the data owner splits each node's secure neighbor ID list $\mathbf{Ne}_{v_i}$ and secure edge weight list $\mathbf{W}_{v_i}$ into secret shares:

$$[\![\mathbf{Ne}_{v_i,j}]\!]^A = \langle \mathbf{Ne}_{v_i,j} \rangle_1 + \langle \mathbf{Ne}_{v_i,j} \rangle_2, j \in [1, d_{max}],$$

$$[\![\mathbf{W}_{v_i,j}]\!]^A = \langle \mathbf{W}_{v_i,j} \rangle_1 + \langle \mathbf{W}_{v_i,j} \rangle_2, j \in [1, d_{max}],$$

where $i \in [1, N], j \in [1, d_{max}]$. Finally, the data owner sends all secret shares $[\![\mathbf{F}]\!]^A$, $[\![\mathbf{T}]\!]^A$, $[\![\mathbf{Ne}]\!]^A$ and $[\![\mathbf{W}]\!]^A$ to P_1 and P_2, respectively. Assuming that the nodes in the graph are indexed from 1 to N, i.e., $v_1 = 1, \ldots, v_N = N$. The encrypted graph-structured data can be regarded as an array-like data structure where each array element is a node's encrypted data and the index is the node's ID v_i.

C. Secure Initialization

Each node's initial features need to be normalized before model training [63]. Without loss of generality, we will work with a common feature normalization method:

$$\overline{x}_i = \frac{x_i}{\sum_{j \in [1,L]} x_j}, i \in [1, L], \tag{10}$$

where L is the number of features. Obviously, the *sum* operation is directly supported in the secret sharing domain, but the *division* operation is hard to be directly supported and calls for a tailored protocol for secure division in the secret sharing domain.

Our solution is to approximate the division operation using basic operations (i.e., $+$, $\times$) supported in the secret sharing domain. We observe that the main challenge in computing division is to compute the reciprocal $[\![\frac{1}{x}]\!]^A$. Inspired by the recent work [64], we approximate the reciprocal by the iterative Newton-Raphson algorithm [65]:

$$y_{n+1} = y_n(2 - xy_n), \tag{11}$$

which will converge to $y_n \approx \frac{1}{x}$. Obviously, both subtraction and multiplication are naturally supported in the secret sharing domain. In addition, a faster convergence can be achieved by

Subroutine 1: Secure Feature Normalization.

```
    Input: Node $v_i$'s encrypted features $\{[\![\mathbf{F}_{v_i,j}]\!]^A\}_{j\in[1,L]}$.
    Output: Encrypted normalized features $\{[\![\overline{\mathbf{F}}_{v_i,j}]\!]^A\}_{j\in[1,L]}$.
1: $P_{\{1,2\}}$ locally calculate $[\![S]\!]^A = \sum_{l\in[1,L]}[\![\mathbf{F}_{v_i,l}]\!]^A$.
    //$P_{\{1,2\}}$ calculate the approximate $[\![\frac{1}{S}]\!]^A$:
2: $[\![y_0]\!]^A = 3\times[\![e^{0.5-S}]\!]^A + 0.003$;
3: for $n=0$ to $\mathcal{N}$ do
4:     $[\![y_{n+1}]\!]^A = [\![y_n]\!]^A \times (2 - [\![S]\!]^A \times [\![y_n]\!]^A)$.
5: end for
    //$P_{\{1,2\}}$ calculate the normalized features:
6: for $l=1$ to $L$ do
7:     $[\![\overline{\mathbf{F}}_{v_i,l}]\!]^A = [\![\mathbf{F}_{v_i,l}]\!]^A \times [\![\frac{1}{S}]\!]^A$.
8: end for
```

initializing y_0 as:

$$y_0 = 3e^{0.5-x} + 0.003. \tag{12}$$

How to compute e^x in the secret sharing domain will be introduced in Section V-E2. Subroutine 1 describes our protocol for secure feature normalization.

D. Secure Neighboring States Aggregation

During the state propagation process, the k_{th}-layer aggregate state $\overline{\mathbf{x}}^{(k)}_{v_i}$ of node v_i is computed by node v_i's k_{th}-layer state $\mathbf{x}^{(k)}_{v_i}$ and its neighbors' k_{th}-layer states $\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}$, where node v_i's 0_{th}-layer state is its normalized feature vector $\overline{\mathbf{F}}_{v_i}$. As shown in (5) and (7), the aggregate state is the weighted sum of these states. However, since only the encrypted neighbors' IDs are uploaded to the cloud rather than the whole adjacency matrix $\mathbf{A}$, it raises a challenge on how to compute $\hat{\mathbf{A}}$ and perform all subsequent operations.

Our insight is to first transform the aggregation method in (5) and (7) to the other form that can be calculated in the secret sharing domain. The k_{th}-layer aggregate state of node v_i can be denoted as $(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(k)})_i$ where $()_i$ denotes the i_{th} row of the matrix, and its equivalent form is $\sum_{j=1}^{N}\frac{\tilde{\mathbf{A}}_{i,j}}{\sqrt{\tilde{\mathbf{D}}_{i,i}\tilde{\mathbf{D}}_{j,j}}}\mathbf{X}^{(k)}_j$, where $\sum_{k=1}^{N}\tilde{\mathbf{D}}^{-\frac{1}{2}}_{i,k} = \tilde{\mathbf{D}}^{-\frac{1}{2}}_{i,i}$ because $\tilde{\mathbf{D}}$ is a diagonal matrix. Since $\tilde{\mathbf{A}}_{i,j}=0$ if node v_j is not v_i's neighbors, $w_{i,i}=1$ and $sw_{v_i} = \tilde{\mathbf{D}}_{v_i,v_i}$ (i.e., (2)), a more simper form of v_i's k_{th}-layer aggregate state is:

$$\overline{\mathbf{x}}^{(k)}_{v_i} = \frac{1}{sw_{v_i}}\mathbf{x}^{(k)}_{v_i} + \sum_{j\in[1,d_{max}]}\frac{\mathbf{W}_{v_i,j}}{\sqrt{sw_{v_i}}\cdot\sqrt{sw_{\mathbf{Ne}_{v_i,j}}}}\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}. \tag{13}$$

It is noted that since the edge weights between v_i and its dummy neighbors are 0, the effect of these dummy neighbors can be eliminated using (13).

When securely computing node v_i's k_{th}-layer aggregate state $\overline{\mathbf{x}}^{(k)}_{v_i}$ by (13), the cloud servers should first securely access the neighboring nodes' k_{th}-layer states $\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}, j\in[1,d_{max}]$, and then sum these states by securely multiplying its weight $\frac{\mathbf{W}_{v_i,j}}{\sqrt{sw_{v_i}}\cdot\sqrt{sw_{\mathbf{Ne}_{v_i,j}}}}$. However, it is challenging to access the neighboring nodes' states since the neighbors' IDs are encrypted. Meanwhile, the square root is not naturally supported in the secret sharing domain.

To overcome the two obstacles, we design a protocol for *secure neighboring states access* which allows the cloud servers to securely access the neighboring nodes' states, and a protocol for *secure neighboring states summation* allowing the cloud servers to securely perform the square root calculation and the summation of the accessed neighboring states.

1) Secure Neighboring States Access: Neighboring states access is challenging in the secret sharing domain, because we need to access each neighbor's state with both the neighbor's ID $\mathbf{Ne}_{v_i,j}$ and state $\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}$ being encrypted. Furthermore, the accessed result should still be encrypted. Our insight is to first transform it to the array access problem in the secret sharing domain, i.e., the state vector $\mathbf{x}^{(k)}_{v_i}$ of each node in the graph is treated as an array element and node IDs (1 to N) serve as array indexes. We then consider how to securely access the *encrypted element* at the *encrypted location* from the *encrypted array*.

From the literature, we identify the existence of the state-of-the-art secure array access protocol in the secret sharing domain by Blanton et al. [66], which works in a similar three-party setting and uses 2-out-of-2 secret sharing. This method requires communicating $\mathbf{4m+4}$ elements in *two* rounds, where m is the length of the encrypted array. In the protocol of [66], the cloud needs to send random values to each other during accessing the encrypted array element. These shared random values will be used to hide the shares of each array element, and will be offset in the sum of shares.

Through careful inspection on the protocol, we manage to design a more efficient protocol which only requires communicating $\mathbf{2m+2}$ elements in *one* round. In particular, instead of letting the cloud servers send random values to each other, our idea is to enable them to locally generate *correlated random values* (i.e., $c^1+c^2+c^3=0$) based on a technique from [67], which will be used to hide the shares of each array element, and will be offset in the sum of shares. More specifically, in the system initialization phase, the cloud server $P_i, i\in\{1,2,3\}$ samples a key k_i and send k_i to P_{i+1} where $P_{3+1=1}$. Then P_i's j_{th} correlated random value is

$$\mathbf{c}^i[j] = \mathbb{F}(k_i,j) - \mathbb{F}(k_{i-1},j),$$

where $k_{1-1=3}$ and $\mathbb{F}$ is a pseudorandom function (PRF). Meanwhile, an agreed random value r between each two cloud servers can also be generated by their shared key. The j_{th} agreed random value between P_i and P_{i+1} is $r^i_j = \mathbb{F}(k_i,j) \mod m$, where m is the length of the secret array.

Given a secret array $[\![\mathbf{a}]\!]^A = \langle\mathbf{a}\rangle_1 + \langle\mathbf{a}\rangle_2$ and a secret index $[\![I]\!]^A = \langle I\rangle_1 + \langle I\rangle_2$ held by P_1 and P_2, respectively, our protocol, as shown in Subroutine 2, for securely accessing the element $[\![\mathbf{a}[I]]\!]^A$ is as follows:

1) P_1 first rotates its shares r^1 locations:

$$\langle\mathbf{a}[1]\rangle_1,\ldots,\langle\mathbf{a}[m]\rangle_1 \;\circlearrowright\blacktriangledown$$

Subroutine 2: Secure Array Access.

```
Input: $P_{\{1,2\}}$ hold the secret shares $\langle \mathbf{a} \rangle_{\{1,2\}}$ and $\langle I \rangle_{\{1,2\}}$,
  respectively; $P_{\{1,2,3\}}$ hold random value array $\mathbf{c}^{\{1,2,3\}}$,
  respectively; $P_{\{1,2\}}$ and $P_{\{1,3\}}$ hold the agreed random
  values $r^1$ and $r^3$, respectively.
Output: $P_{\{2,3\}}$ hold the accessed element $[\![\mathbf{a}[I]]\!]^A$.
// $P_1$ locally performs:
1: $\langle \mathbf{a} \rangle_1 = \langle \mathbf{a} \rangle_1 \circlearrowright r^1$. $\langle \mathbf{a}' \rangle_1 = \langle \mathbf{a} \rangle_1 + \mathbf{c}^1$.
   $\langle \mathbf{a}'' \rangle_2 = \langle \mathbf{a}' \rangle_1 \circlearrowright r^3$.
2: $\langle h \rangle_1 = (\langle I \rangle_1 + r^1 + r^3) \mod m$.
3: Sending $\langle h \rangle_1$ and $\langle \mathbf{a}'' \rangle_2$ to $P_2$.
   // $P_2$ locally performs:
4: $h = (\langle h \rangle_1 + \langle I \rangle_2) \mod m$.
5: $P_2$ sets the secret share of $\mathbf{a}[I]$ as $\underline{\langle \mathbf{a}''[h] \rangle_2}$.
6: $\langle \mathbf{a} \rangle_2 = \langle \mathbf{a} \rangle_2 \circlearrowright r^1$. $\langle \mathbf{a}' \rangle_2 = \langle \mathbf{a} \rangle_2 + \mathbf{c}^2$.
7: Sending $h$ and $\langle \mathbf{a}' \rangle_2$ to $P_3$.
   // $P_3$ locally performs:
8: $\langle \mathbf{a}'' \rangle_3 = \langle \mathbf{a}' \rangle_2 + \mathbf{c}^3$. $\langle \mathbf{a}'' \rangle_3 = \langle \mathbf{a}'' \rangle_3 \circlearrowright r^3$.
9: $P_3$ sets the secret share of $\mathbf{a}[I]$ as $\underline{\langle \mathbf{a}''[h] \rangle_3}$.
```

$$\langle \mathbf{a}[m-r^1] \rangle_1, \ldots, \langle \mathbf{a}[m] \rangle_1, \langle \mathbf{a}[1] \rangle_1, \ldots, \langle \mathbf{a}[m-r^1+1] \rangle_1.$$

Then, P_1 sets the new array as $\langle \mathbf{a}'[j] \rangle_1 = \langle \mathbf{a}[j] \rangle_1 + \mathbf{c}^1[j], j \in [1, m]$, and rotates it r^3 locations:

$$\langle \mathbf{a}'[1] \rangle_1, \ldots, \langle \mathbf{a}'[m] \rangle_1 \quad \circlearrowright \blacktriangledown$$

$$\langle \mathbf{a}'[m-r^3] \rangle_1, \ldots, \langle \mathbf{a}'[m] \rangle_1, \langle \mathbf{a}'[1] \rangle_1, \ldots, \langle \mathbf{a}'[m-r^3+1] \rangle_1.$$

The new array is denoted as $\langle \mathbf{a}'' \rangle_2$. Finally, P_1 sets $\langle h \rangle_1 = (\langle I \rangle_1 + r^1 + r^3) \mod m$, then sends $\langle h \rangle_1$ and $\langle \mathbf{a}'' \rangle_2$ to P_2.

2) P_2 sets $h = (\langle h \rangle_1 + \langle I \rangle_2) \mod m$, then P_2's share of the accessed element $\mathbf{a}[I]$ is $\underline{\langle \mathbf{a}''[h] \rangle_2}$.

3) P_2 first rotates its shares of the raw array r^1 locations:

$$\langle \mathbf{a}[1] \rangle_2, \ldots, \langle \mathbf{a}[m] \rangle_2 \quad \circlearrowright \blacktriangledown$$

$$\langle \mathbf{a}[m-r^1] \rangle_2, \ldots, \langle \mathbf{a}[m] \rangle_2, \langle \mathbf{a}[1] \rangle_2, \ldots, \langle \mathbf{a}[m-r^1+1] \rangle_2.$$

Then, P_2 sets the new array as $\langle \mathbf{a}'[j] \rangle_2 = \langle \mathbf{a}[j] \rangle_2 + \mathbf{c}^2[j], j \in [1, m]$, and sends $\langle \mathbf{a}' \rangle_2$ and h to P_3.

4) P_3 first sets $\langle \mathbf{a}''[j] \rangle_3 = \langle \mathbf{a}'[j] \rangle_2 + \mathbf{c}^3[j], j \in [1, m]$, then rotates them r^3 locations:

$$\begin{aligned} &\langle \mathbf{a}''[1] \rangle_3, \ldots, \langle \mathbf{a}''[m] \rangle_3 \quad \circlearrowright \blacktriangledown \\ &\langle \mathbf{a}''[m-r^3] \rangle_3, \ldots, \langle \mathbf{a}''[m] \rangle_3, \\ &\langle \mathbf{a}''[1] \rangle_3, \ldots, \langle \mathbf{a}''[m-r^3+1] \rangle_3. \end{aligned} \tag{14}$$

Finally, P_3's share of $\mathbf{a}[I]$ is $\underline{\langle \mathbf{a}''[h] \rangle_3}$.

It is noted that, the correlated random values $\mathbf{c}^{\{1,2,3\}}$ and agreed random values $r^{\{1,3\}}$ all do not require online communication because they are generated by the PRF and shared keys. Therefore, our protocol only requires communicating $2m + 2$ elements in one round, i.e., in steps 1), 3).

Correctness Analysis: P_2's shares $\langle \mathbf{a}'' \rangle_2$ are generated by $\langle \mathbf{a}'' \rangle_2 = (\langle \mathbf{a} \rangle_1 \circlearrowright r^1 + \mathbf{c}^1) \circlearrowright r^3$, where "$\circlearrowright$" denotes "rotate". P_3's shares $\langle \mathbf{a}'' \rangle_3$ are generated by $\langle \mathbf{a}'' \rangle_3 = (\langle \mathbf{a} \rangle_2 \circlearrowright r^1 + \mathbf{c}^2 + \mathbf{c}^3) \circlearrowright r^3$. Based on $\mathbf{c}^1[j] + \mathbf{c}^2[j] + \mathbf{c}^3[j] = 0$, we can obtain $\langle \mathbf{a}'' \rangle_2 + \langle \mathbf{a}'' \rangle_3 = \mathbf{a} \circlearrowright r^1 \circlearrowright r^3$, namely, for $j \in [1, m]$, $\langle \mathbf{a}''[(j + r^1 + r^3) \mod m] \rangle_2 + \langle \mathbf{a}''[(j + r^1 + r^3) \mod m] \rangle_3 = \mathbf{a}[j]$. Since $h = I + r^1 + r^3$, the accessed element is exactly $\langle \mathbf{a}''[h] \rangle_2 + \langle \mathbf{a}''[h] \rangle_3 = \mathbf{a}[I]$.

It is noted that since $P_{\{1,2\}}$ perform main computations in our protocol but P_3 holds the secret share of the accessed element, P_3 should re-share its secret $\langle \mathbf{a}''[h] \rangle_3$ to P_1 and P_2. More specifically, P_3 generates a random value s, and then sends $s, \langle \mathbf{a}''[h] \rangle_3 - s$ to P_1 and P_2, respectively. Finally, the shares held by P_1 and P_2 are s and $\langle \mathbf{a}''[h] \rangle_2 + \langle \mathbf{a}''[h] \rangle_3 - s$, respectively.

2) Secure Neighboring States Summation: After performing the above secure neighboring states access protocol, $P_{\{1,2\}}$ hold all encrypted neighboring states $\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}$ of node v_i. In addition, as shown in (13), the sum of v_i's each neighbor's own edge weights $sw_{\mathbf{Ne}_{v_i,j}}$ (i.e., (2)) are used in calculating the aggregate state $\overline{\mathbf{x}}^{(k)}_{v_i}$. Since each neighbor's own edge weights are attached with its ID like its state, similar to accessing neighboring states, the cloud servers should access each neighbor's edge weights using the above secure array access protocol. After that, the cloud servers can obtain node v_i's each neighbor's encrypted state $\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}$ and the encrypted sum of edge weights $sw_{\mathbf{Ne}_{v_i,j}}$ using each encrypted neighbor's ID. Then the cloud uses (13) to calculate node v_i's aggregate state $\overline{\mathbf{x}}^{(k)}_{v_i}$. However, the *square root* is not naturally supported in secret sharing.

Inspired by the very recent work [64], we resort to the approach of approximating the inverse square root by iterative Newton-Raphson algorithm [65]:

$$y_{n+1} = \frac{1}{2} y_n (3 - x y_n^2), \tag{15}$$

which will converge to $y_n \approx \frac{1}{\sqrt{x}}$. Obviously, both subtraction and multiplication are naturally supported in the secret sharing domain. The initialization y_0 can be set as $y_0 = 3e^{0.5-x} + 0.003$.

After securely accessing each neighboring node's state $[\![\mathbf{x}^{(k)}_{\mathbf{Ne}_{v_i,j}}]\!]^A$ for node v_i, the cloud servers utilize the above secure inverse square root protocol to perform the secure neighboring states summation, as shown in Subroutine 3.

E. Secure Activation Functions

After a node v_i's k_{th}-layer aggregate state $\overline{\mathbf{x}}^{(k)}_{v_i}$ is calculated and multiplied with the trainable weight matrix $\mathbf{M}^{(k+1)}$, i.e., $\overline{\mathbf{x}}^{(k)}_{v_i} \mathbf{M}^{(k+1)}$, an activation functions needs to be applied over $\hat{\mathbf{x}}^{(k)}_{v_i} = \overline{\mathbf{x}}^{(k)}_{v_i} \mathbf{M}^{(k+1)}$ to calculate v_i's $(k+1)_{th}$-layer state $\mathbf{x}^{(k+1)}_{v_i}$, according to (6) and (8) respectively. In this section, we will introduce how to securely compute the activation functions in the secret sharing domain.

Subroutine 3: Secure Neighboring States Summation.

```
Input: Node $v_i$'s $k_{th}$-layer state and edge weights: $[\![\mathbf{x}_{v_i}^{(k)}]\!]^A$ and $[\![\mathbf{W}_{v_i,j}]\!]^A, j \in [1, d_{max}]$, and $v_i$'s neighboring $k_{th}$-layer states and their edge weight sum: $[\![\mathbf{x}_{\mathbf{Ne}_{v_i,j}}^{(k)}]\!]^A$ and $[\![sw_{\mathbf{Ne}_{v_i,j}}]\!]^A, j \in [1, d_{max}]$.
Output: $v_i$'s $k_{th}$-layer encrypted aggregate state $[\![\overline{\mathbf{x}}_{v_i}^{(k)}]\!]^A$.
1: $P_{\{1,2\}}$ first calculate the approximate $[\![\frac{1}{sw_{v_i}}]\!]^A$ by (11).
   //$P_{\{1,2\}}$ calculate each approximate $[\![\frac{1}{\sqrt{sw_{id}}}]\!]^A$:
2: for each $id \in \{v_i\} \cup \{\mathbf{Ne}_{v_i,j}\}_{j\in[1,d_{max}]}$ do
3:     $[\![y_0]\!]^A = 3 \times [\![e^{0.5-sw_{id}}]\!]^A + 0.003$.
4:     for $n = 0$ to $\mathcal{N}$ do
5:         $[\![y_{n+1}]\!]^A = \frac{1}{2} \times [\![y_n]\!]^A \times (3 - [\![sw_{id}]\!]^A \times [\![y_n]\!]^A \times [\![y_n]\!]^A)$.
6:     end for
7: end for
   //$P_{\{1,2\}}$ calculate the aggregate state:
8: $[\![\overline{\mathbf{x}}_{v_i}^{(k)}]\!]^A = [\![\frac{1}{sw_{v_i}}]\!]^A \times [\![\mathbf{x}_{v_i}^{(k)}]\!]^A$.
9: for $j = 1$ to $d_{max}$ do
10:    $[\![\overline{\mathbf{x}}_{v_i}^{(k)}]\!]^A = [\![\overline{\mathbf{x}}_{v_i}^{(k)}]\!]^A + [\![\mathbf{W}_{v_i,j}]\!]^A \times [\![\frac{1}{\sqrt{sw_{v_i}}}]\!]^A \times [\![\frac{1}{\sqrt{sw_{\mathbf{Ne}_{v_i,j}}}}]\!]^A \times [\![\mathbf{x}_{\mathbf{Ne}_{v_i,j}}^{(k)}]\!]^A$.
11: end for
```

1) Secure ReLU Function: The function ReLU $:= max(x, 0)$ is a popular activation function in neural network, whose core is to test whether $x > 0$ or not. However, the comparison operation is not naturally supported in the secret sharing domain. We note that given the computation is in $\mathbb{Z}_{2^k}$, it suffices to tailor a protocol for testing whether the Most Significant Bit (MSB) of $[\![x]\!]^A$ is 0 or not [23], [68]. Mohassel et al. [23] propose to compute the MSB using secure bit decomposition (only directions briefly mentioned without a concrete construction though). It is noted that different from our system, their security design uses replicated secret sharing, which runs among three cloud servers and needs them to interact with each other throughout the process. Inspired by their work, we provide an alternative design to evaluate the MSB under additive secret sharing that suits our system, in which the computation is mainly conducted by P_1 and P_2 while P_3 just provides necessary triples in advance. The details of our design are as follows.

Given two fixed point numbers' complement A and B, which can represent the shares of a secret value, the MSB of $A + B$ can be computed by a tailored Parallel Prefix Adder (PPA) [69]. Fig. 3 illustrates an 8-bit tailored PPA. We can apply the tailored PPA to the secret shares. In particular, given the k-bit secret sharing $[\![x]\!]^A = \langle x \rangle_1 + \langle x \rangle_2$ held by P_1 and P_2, they first locally decompose the complement of $\langle x \rangle_i$ into bits: $\langle x \rangle_i = x_i[1], \ldots, x_i[k], i \in \{1, 2\}$. After that, they input the bits into a k-bit tailored PPA to perform secure AND and XOR calculations. Given a k-bit number, the tailored PPA can calculate its MSB in $\log k$ rounds. In addition, as shown in Section III-B, in additive secret sharing, a AND gate requires

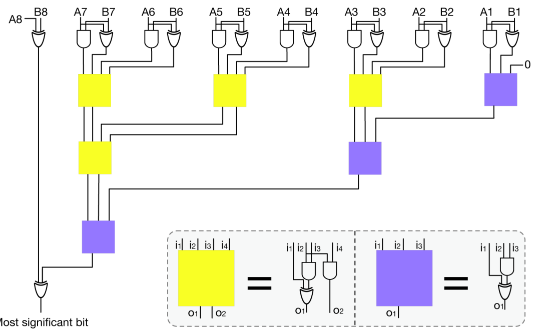


Fig. 3. An 8-bit tailored PPA.

online communication 4 bits in one round, while an XOR gate does not require communication. Therefore, to calculate the MSB of a k-bit number in additive secret sharing, our alternative design requires the two cloud servers to online communicate $12k - 12 - 4\log k$ bits in $\log k$ rounds. It is noted that, using the above method, $msb(x) = 1$ if $x < 0$, and $msb(x) = 0$ if $x >= 0$. To be compatible with the subsequent operation, one of P_1 and P_2 flips its share $\langle msb(x) \rangle_1$ or $\langle msb(x) \rangle_2$ so that $msb(x)' = 0$ if $x < 0$, and $msb(x)' = 1$ if $x >= 0$.

However, using the above method, the cloud servers only obtain $[\![msb(x)']\!]^B$, not ReLU(x), and the cloud servers also need to calculate $[\![msb(x)']\!]^B \times [\![x]\!]^A$ when P_1 and P_2 hold $\langle msb(x)' \rangle_1, \langle x \rangle_1$ and $\langle msb(x)' \rangle_2, \langle x \rangle_2$, respectively. Inspired by [23], we design a tailored protocol for securely evaluating $[\![\text{ReLU}(x)]\!]^A$ in additive secret sharing:

1) P_1 randomly generates $r \in \mathbb{Z}_{2^k}$ and defines $m_{b\in\{0,1\}} := (b \oplus \langle msb(x)' \rangle_1) \times \langle x \rangle_1 - r$, and sends them to P_2.
2) P_2 chooses m_b based on $\langle msb(x)' \rangle_2$, namely, P_2 chooses m_0 if $\langle msb(x)' \rangle_2 = 0$, and otherwise P_2 chooses m_1. Therefore, the secret share held by P_2 is $m_{\langle msb(x)' \rangle_2} = msb(x)' \times \langle x \rangle_1 - r$, and the secret share held by P_1 is r.
3) For the other secret share $\langle x \rangle_2$, P_2 acts as the sender and P_1 acts as the receiver to perform step 1) and 2) again.

Finally, $P_{\{1,2\}}$ hold the secret shares $\langle msb(x)' \times x \rangle_{\{1,2\}}$. It is noted that, in [23], $P_{\{1,2\}}$ should re-share their shares to P_3 since they work on replicated secret sharing. Subroutine 4 describes our protocol for secure ReLU function.

2) Secure Softmax Function: GCN usually considers a multi-classification task, which requires the Softmax function (i.e., (4)) to normalize the probabilities of inference results. Therefore, we need a protocol to securely compute the Softmax function.

First, to avoid error from calculating the exponential function on very large or very small values, a frequently-used method is to calculate the Softmax function on $\mathbf{x} - max(\mathbf{x})$. When calculating $max(\mathbf{x})$ in the secret sharing domain, to reduce the overhead, we can use the binary-tree form, e.g., $max(max([\![x_1]\!]^A, [\![x_2]\!]^A), max([\![x_3]\!]^A, [\![x_4]\!]^A))$, which requires $\log C$ rounds comparison and C is the number of classifications. We can directly use the secure ReLU function introduced above to perform $max()$:

$$max([\![x_1]\!]^A, [\![x_2]\!]^A) = \text{ReLU}([\![x_1]\!]^A - [\![x_2]\!]^A) + [\![x_2]\!]^A.$$

Subroutine 4: Secure ReLU Function.

```
Input: $P_{\{1,2\}}$ hold node $v_i$'s $0_{th}$-layer encrypted state :
  $\{[\![\hat{\mathbf{x}}^{(0)}_{v_i,1}]\!]^A,\ldots,[\![\hat{\mathbf{x}}^{(0)}_{v_i,H}]\!]^A\}$, where $\hat{\mathbf{x}}^{(0)}_{v_i}=\overline{\mathbf{x}}^{(0)}_{v_i}\mathbf{M}^{(1)}$.
Output: $P_{\{1,2\}}$ hold node $v_i$'s $1_{st}$-layer encrypted state:
  $\{[\![\mathbf{x}^{(1)}_{v_i,1}]\!]^A,\ldots,[\![\mathbf{x}^{(1)}_{v_i,H}]\!]^A\}$.
1: for $j=1$ to $H$ do
2:     Securely calculating $[\![msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})]\!]^B$ by tailored PPA.
3:     One of $P_{\{1,2\}}$ flips its share, and then $P_{\{1,2\}}$'s shares
       are $\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_1$ and $\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_2$, respectively.
4:     $P_1$ randomly generates $r\in\mathbb{Z}_{2^k}$, and sends $m_b:=$
       $(b\oplus\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_1)\times\langle\hat{\mathbf{x}}^{(0)}_{v_i,j}\rangle_1-r, b\in\{0,1\}$ to
       $P_2$.
5:     $P_2$ chooses $m_b$ based on $\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_2$.
6:     $P_2$ randomly generates $r'\in\mathbb{Z}_{2^k}$, and sends $m'_b:=$
       $(b\oplus\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_2)\times\langle\hat{\mathbf{x}}^{(0)}_{v_i,j}\rangle_2-r', b\in\{0,1\}$ to $P_1$.
7:     $P_1$ chooses $m'_b$ based on $\langle msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\rangle_1$.
8:     Finally, $P_1$ holds $msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\times\langle\hat{\mathbf{x}}^{(0)}_{v_i,j}\rangle_2-r'+r$
       and $P_2$ holds $msb(\hat{\mathbf{x}}^{(0)}_{v_i,j})'\times\langle\hat{\mathbf{x}}^{(0)}_{v_i,j}\rangle_1-r+r'$.
9: end for
```

After that, the cloud servers should compute $[\![e^x]\!]^A$. Since $[\![e^x]\!]^A$ is not naturally supported in the secret sharing domain, we first approximate e^x using its limit characterization [64]:

$$e^x \approx (1+\frac{x}{2^n})^{2^n}. \tag{16}$$

However, the approximation is inefficient if the cloud servers serially calculate the multiplication, which will require to calculate 2^n multiplications in 2^n rounds communication. Our solution is to calculate the approximation by the binary-tree form. More specifically, the core of (16) is to calculate $([\![x]\!]^A)^{2^n}$, thus $P_{\{1,2\}}$ first calculate $([\![x]\!]^A)^2$ in one round, and then set $[\![y]\!]^A=([\![x]\!]^A)^2$ followed by calculating $([\![y]\!]^A)^2$ in one round. Therefore, $P_{\{1,2\}}$ can calculate $([\![x]\!]^A)^{2^n}$ in $\log 2^n=n$ rounds. Subroutine 5 describes our protocol for secure Softmax function.

F. Secure Model Convergence Evaluation

So far we have presented our solution for securely realizing the forward propagation process as given in (1) in the secret sharing domain. We now show how to securely evaluate the convergence of the model training process.

We note that prior works (e.g., [23], [25], [26]) on secure CNN training generally terminate the training process at a specified number of epochs. However, the convergence of the training process is unpredictable, which can depend on various factors such as the training data set, the learning parameter setting, and random factors in the nature of model training. A fixed number of epochs without considering the property of models may easily lead to overfitting or underfitting [70]. Therefore, instead of specifying a certain number of epochs, it is much more desirable to directly evaluate the model convergence in a secure manner.

Our solution is to calculate the encrypted cross-entropy loss and then calculate the difference in the encrypted cross-entropy

Subroutine 5: Secure Softmax Function.

```
Input: $P_{\{1,2\}}$ hold node $v_i$'s $1_{th}$-layer encrypted state :
  $\{[\![\hat{\mathbf{x}}^{(1)}_{v_i,1}]\!]^A,\ldots,[\![\hat{\mathbf{x}}^{(1)}_{v_i,C}]\!]^A\}$, where $\hat{\mathbf{x}}^{(1)}_{v_i}=\overline{\mathbf{x}}^{(1)}_{v_i}\mathbf{M}^{(2)}$.
Output: $P_{\{1,2\}}$ hold node $v_i$'s $2_{nd}$-layer encrypted state:
  $\{[\![\mathbf{x}^{(2)}_{v_i,1}]\!]^A,\ldots,[\![\mathbf{x}^{(2)}_{v_i,C}]\!]^A\}$.
1: Calculate $[\![Q]\!]^A=max\{[\![\hat{\mathbf{x}}^{(1)}_{v_i,j}]\!]^A\}_{j\in[1,C]}$ by ReLU().
2: Locally calculate
     $[\![\hat{\mathbf{x}}'^{(1)}_{v_i,j}]\!]^A=[\![\hat{\mathbf{x}}^{(1)}_{v_i,j}]\!]^A-[\![Q]\!]^A, j\in[1,C]$.
     //$P_{\{1,2\}}$ calculate the approximate $[\![e^{\hat{\mathbf{x}}'^{(1)}_{v_i,j}}]\!]^A$:
3: for $j=1$ to $C$ do
4:     $[\![y_0]\!]^A=1+\frac{[\![\hat{\mathbf{x}}'^{(1)}_{v_i,j}]\!]^A}{2^{\mathcal{N}}}$.
5:     for $n=0$ to $\mathcal{N}$ do
6:         $[\![y_{n+1}]\!]^A=[\![y_n]\!]^A\times[\![y_n]\!]^A$.
7:     end for
8: end for
9: $P_{\{1,2\}}$ first locally calculate $[\![S]\!]^A=\sum_{j\in[1,C]}[\![e^{\hat{\mathbf{x}}'^{(1)}_{v_i,j}}]\!]^A$,
   and then calculate the approximate $[\![\frac{1}{S}]\!]^A$ by (11).
     //$P_{\{1,2\}}$ calculate the $2_{nd}$-layer encrypted state:
10: for $j=1$ to $C$ do
11:     $[\![\mathbf{x}^{(2)}_{v_i,j}]\!]^A=[\![e^{\overline{\mathbf{x}}'^{(1)}_{v_i,j}}]\!]^A\times[\![\frac{1}{S}]\!]^A$.
12: end for
```

loss between two adjacency epochs. If the difference is smaller than a public threshold α and lasts for a window size, the cloud servers $P_{\{1,2\}}$ will conclude that the model is convergent and will terminate the training. From the computation, $P_{\{1,2\}}$ know nothing except the *necessary* fact about whether the difference in the cross-entropy loss between two adjacency epochs is less than α.

A new challenge arises, namely, how to calculate the cross-entropy loss in the secret sharing domain. In (9), the *natural logarithm* is not naturally supported in the secret sharing domain, and requires a tailored protocol. Inspired by [64], we approximate $\ln x$ by:

$$y_{n+1}=y_n-\sum_{k\in[1,\mathcal{K}]}\frac{1}{k}(1-xe^{-y_n})^k, \tag{17}$$

which will converge to $y_n\approx\ln x$. The initial value can be set as $y_0=\frac{x}{120}-20e^{-2x-1}+3$ [64]. Obviously, both subtraction and multiplication are naturally supported in secret sharing domain, and $[\![e^{-2x-1}]\!]^A$ can be calculated by (16).

After obtaining the encrypted loss $[\![\mathcal{L}^j]\!]^A$ and $[\![\mathcal{L}^{j+1}]\!]^A$ of two adjacent epochs, $P_{\{1,2\}}$ first calculate the absolute value of their difference:

$$\begin{aligned}|[\![\mathcal{L}^{j+1}]\!]^A-[\![\mathcal{L}^{j}]\!]^A| &= \mathrm{ReLU}([\![\mathcal{L}^{j+1}]\!]^A-[\![\mathcal{L}^{j}]\!]^A)\\ &\quad+\mathrm{ReLU}([\![\mathcal{L}^{j}]\!]^A-[\![\mathcal{L}^{j+1}]\!]^A).\end{aligned}$$

Then, the model convergence flag is calculated by $[\![msb(\alpha-|[\![\mathcal{L}^j]\!]^A-[\![\mathcal{L}^{j+1}]\!]^A|)]\!]^B$. $P_{\{1,2\}}$ open the flag to each other, and then decide whether to terminate the training. Subroutine 6 describes our protocol for secure model convergence evaluation.

Subroutine 6: Secure Model Convergence Evaluation.

```
Input: $P_{\{1,2\}}$ hold all labeled nodes' encrypted inference
  results $\{[\![\mathbf{Z}_{v_i}]\!]^A\}_{v_i \in \mathcal{T}}$ and encrypted labels $\{[\![\mathbf{T}_{v_i}]\!]^A\}_{v_i \in \mathcal{T}}$,
  the public threshold $\alpha$, the public window size $\beta$ and the
  public maximum number of epochs $\gamma$.
Output: Nothing.
1: for $e = 1$ to $\gamma$ do
2:     $[\![\mathcal{L}^e]\!]^A = 0$.
       // $P_{\{1,2\}}$ calculate the approximate $[\![\ln \mathbf{Z}_{v_i,j}]\!]^A$:
3:     for each $v_i \in \mathcal{T}, j \in [1, C]$ do
4:         $[\![y_0]\!]^A = \frac{[\![\mathbf{Z}_{v_i,j}]\!]^A}{120} - 20 \times [\![e^{-2 \times \mathbf{Z}_{v_i,j} - 1}]\!]^A + 3$.
5:         for $n = 0$ to $\mathcal{N}$ do
6:             $[\![y_{n+1}]\!]^A =$
           $[\![y_n]\!]^A - \sum_{k=1}^{\mathcal{K}} \frac{1}{k} \times (1 - [\![\mathbf{Z}_{v_i,j}]\!]^A \times [\![e^{-y_n}]\!]^A)^k$.
7:         end for
8:     end for
       // $P_{\{1,2\}}$ calculate the cross-entropy loss $[\![\mathcal{L}^e]\!]^A$:
9:     for each $v_i \in \mathcal{T}, j \in [1, C]$ do
10:        $[\![\mathcal{L}^e]\!]^A - = [\![\mathbf{T}_{v_i,j}]\!]^A \times [\![\ln \mathbf{Z}_{v_i,j}]\!]^A$.
11:    end for
       // $P_{\{1,2\}}$ determine whether to stop the training:
12:    $|[\![\mathcal{L}^e]\!]^A - [\![\mathcal{L}^{e-1}]\!]^A| = \mathrm{ReLU}([\![\mathcal{L}^e]\!]^A - [\![\mathcal{L}^{e-1}]\!]^A) +$
     $\mathrm{ReLU}([\![\mathcal{L}^{e-1}]\!]^A - [\![\mathcal{L}^e]\!]^A)$.
13:    flag$=[\![msb(\alpha - |[\![\mathcal{L}^j]\!]^A - [\![\mathcal{L}^{j+1}]\!]^A|)]\!]^B$.
14:    $stop = (flag == 1) ? 0 : (stop + 1)$.
15:    if $(stop \geqslant \beta)$ then terminating the training process.
16: end for
```

G. Putting Things Together

Secure Training: When training the GCN model, the cloud servers first securely normalize all nodes' initial features through *secure feature normalization*. After that, the cloud servers securely perform the forward propagation (1) through *secure neighboring states aggregation*, and *secure activation functions* for each labeled node $v_i \in \mathcal{T}$ to obtain the inference results $\mathbf{Z}_{v_i,j}, j \in [1, C]$. Subsequently, the cloud servers securely calculate the average cross-entropy loss $\mathcal{L}$ between each labeled node's inference result $\mathbf{Z}_{v_i}$ and its true label $\mathbf{T}_{v_i}$ and then securely evaluate the model convergence.

If convergence is not yet achieved, the cloud servers perform backward propagation to calculate each trainable weight's gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{M}_{i,j}}$ followed by updating each weight using its gradients. Based on the chain rule [71], if the cloud servers can calculate the derivatives of all non-linear functions, they can calculate the complete derivative of (1). In (1), the first non-linear function is the cross-entropy loss function, and its derivative is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{v_i,j}} = -\frac{\mathbf{T}_{v_i,j}}{\mathbf{Z}_{v_i,j}}, v_i \in \mathcal{T}, j \in [1, C],$$

where the division can be securely calculated by using the design in Section V-C. The second non-linear function is the Softmax function, and its derivative is:

$$\frac{\partial z_j}{\partial x_i} = \begin{cases} z_j(1 - z_j) & \text{if } i = j, \\ -z_j z_i & \text{if } i \neq j, \end{cases}$$

where $z_j = Softmax(x_j)$, which can be securely calculated by using Subroutine 5 in Section V-E. The third non-linear function is the ReLU function, and its derivative is:

$$\frac{\partial \mathrm{ReLU}(x)}{\partial x} = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0, \end{cases}$$

which can be securely calculated by using the tailored PPA in Section V-E1. So this is the whole process of secure training in our system.

Secure Inference: Secure inference for an unlabeled node corresponds to a forward propagation through the trained GCN model in the secret sharing domain. In particular, the data owner provides the cloud servers with the ID of the unlabeled node. Upon receiving the ID, the cloud servers securely conduct the forward propagation process (i.e., (1)) in the secret sharing domain, and output the encrypted inference result about its label, which is then sent to the data owner for reconstruction.

VI. Security Analysis

We follow the standard ideal/real world paradigm to analyze the security of SecGNN. In the ideal/real world paradigm, a protocol is secure if the view of the corrupted party during the real execution of a protocol can be generated by a simulator given only the party's input and legitimate output, which can be defined as follows:

Definition 1: Let $P_{\{1,2,3\}}$ engage in a protocol π which computes function $f : (\{0,1\}^*)^3 \rightarrow (\{0,1\}^*)^3$. P_i's view during the execution of protocol π on inputs $\mathbf{x}$, denoted as $\mathrm{View}_i^\pi(\mathbf{x})$, consists of its input $\mathbf{in}_i$, its internal random values $\mathbf{r}_i$ and the messages $\mathbf{m}_i$ received during the execution. We say that π computes f with security in the semi-honest and non-colluding setting, if there exists a probabilistic polynomial time simulator Sim such that for each P_i: $\mathrm{Sim}(\mathbf{in}_i, f_i(\mathbf{x})) \approx \mathrm{View}_i^\pi(\mathbf{x})$.

Recall that SecGNN consists of several secure sub-protocols: 1) secure division `secDIV`; 2) secure array access `secACCESS`; 3) secure square root `secROOT`; 4) secure ReLU function `secRELU`; 5) secure Softmax function `secSoftmax`; 6) secure natural logarithm `secLOG`. We use $Sim_{\mathtt{X}}^{P_i}$ to denote the simulator which can generate P_i's view in sub-protocol `X` on corresponding input and output.

Theorem 1: Our SecGNN is secure according to Definition 1.

Proof: It is noted that the inputs and outputs of each sub-protocol are secret shares, with each sub-protocol being invoked in order as per the processing pipeline. If the simulator for each sub-protocol exists, then our complete protocol is secure [72]. It is easy to see that the simulators $Sim_{\mathtt{X}}^{P_{i \in \{1,2,3\}}}$ ($\mathtt{X} \in \{\mathtt{secDIV}, \mathtt{secROOT}, \mathtt{secSoftmax}, \mathtt{secLOG}$) must exist, because they are all calculated through approximations which are realized via basic operations (i.e., addition and multiplication) in the secret sharing domain. Therefore, SecGNN is secure if the simulators for the remaining sub-protocols exist, i.e., `secACCESS` in Section V-D1 and `secRELU` in Section V-E1. The existence of these simulators is given in *Theorems* 2 and 3.

Theorem 2: The protocol `secACCESS` for secure neighboring states access is secure according to Definition 1.

Proof: We consider the simulator of P_1, P_2 and P_3 in turn.

- $Sim^{P_1}_{\texttt{secACCESS}}$: The simulator is simple since P_1 receives nothing in the real execution. Therefore, it is clear that the simulated view is identical to the real view.
- $Sim^{P_2}_{\texttt{secACCESS}}$: To analyze P_2's view, we see that P_2 has k_1, k_2, and shares $\langle I\rangle_2$, $\langle \mathbf{a}\rangle_2$ at the beginning, and later receives new shares $\langle \mathbf{a}''\rangle_2$ and $\langle h\rangle_1$ in step 1). In the simulated view, P_2 receives random values in step 1). Therefore, we need to prove that $\langle \mathbf{a}''\rangle_2$ and $\langle h\rangle_1$ are uniformly random in the view of P_2.
  - $\langle \mathbf{a}''\rangle_2$ are uniformly random in P_2's view: First, $\langle \mathbf{a}''\rangle_2 = (\langle \mathbf{a}\rangle_1 \circlearrowleft r^1 + \mathbf{c}^1) \circlearrowleft r^3$ and $\mathbf{c}^1[j] = \mathbb{F}(k_1, j) - \mathbb{F}(k_3, j), j \in [1, m]$. Though P_2 has k_1, it does not have k_3, thus $\mathbb{F}(k_3, j)$ is uniformly random in P_2's view. It implies that $\mathbf{c}^1[j]$ is also uniformly random in P_2's view since $\mathbb{F}(k_3, j)$ is independent of $\mathbb{F}(k_1, j)$ used in the generation of $\mathbf{c}^1[j]$ [67]. Similarly, the array $\langle \mathbf{a}''\rangle_2$ is uniformly random in P_2's view since $\mathbf{c}^1$ is independent of $\langle \mathbf{a}\rangle_1$ used in the generation of $\langle \mathbf{a}''\rangle_2$. Therefore, the distribution over the real $\langle \mathbf{a}''\rangle_2$ received by P_2 in the protocol execution and over the simulated $\langle \mathbf{a}''\rangle_2$ generated by the simulator is identically distributed.
  - $\langle h\rangle_1$ is uniformly random in P_2's view: In a similar way, $\langle h\rangle_1 = \langle I\rangle_1 + r^1 + r^3$, where $r^1 = \mathbb{F}(k_1, j)$ and $r^3 = \mathbb{F}(k_3, j)$. Though P_2 has k_1, it does not have k_3, thus r^3 is uniformly random in P_2's view, furthermore, $\langle h\rangle_1$ is uniformly random in P_2's view. Therefore, the distribution over $\langle h\rangle_1$ received by P_2 in the protocol execution and over the $\langle h\rangle_1$ generated by the simulator is identically distributed.
- $Sim^{P_3}_{\texttt{secACCESS}}$: To analyze P_3's view, we see that P_3 has k_2, k_3 at the beginning, and later receives share $\langle \mathbf{a}'\rangle_2$ and h in step 3). It is noted that the proof of $Sim^{P_3}_{\texttt{secACCESS}}$ is similar to the proof of $Sim^{P_2}_{\texttt{secACCESS}}$ since P_3 and P_2 receive similar messages during the protocol execution, thus we omit the proof of $Sim^{P_3}_{\texttt{secACCESS}}$.

Theorem 3: The protocol `secRELU` for the ReLU function is secure according to Definition 1.

Proof: Obviously, the $[\![msb([\![x]\!]^A)]\!]^B$ function is secure since the tailored PPA consists of basic AND and XOR gates, so we only prove that $[\![msb(x)]\!]^B \times [\![x]\!]^A$ function is secure. In the case of P_1 acting as the sender and P_2 acting as the receiver, we consider the simulator of P_1, P_2 and P_3 in turn.

- $Sim^{P_1}_{\texttt{secRELU}}$: The simulator is simple since P_1 receives nothing in the real execution. Therefore, it is clear that the simulated view is identical to the real view.
- $Sim^{P_2}_{\texttt{secRELU}}$: To analyze P_2's view, we see that P_2 has $\langle msb(x)'\rangle_2$ and $\langle x\rangle_2$ at the beginning, and later receives messages $m_b := (b \oplus \langle msb(x)'\rangle_1) \times \langle x\rangle_1 - r, b \in \{0,1\}$. In the simulated view, P_2 receives two random values. Therefore, we need to prove that $m_{\{1,2\}}$ are uniformly random in the view of P_2. Obviously, the above claim is valid, because r is uniformly random in P_2's view, which implies that $m_{\{1,2\}}$ are also uniformly random in P_2's view since r is independent of other values used in the generation of $m_{\{1,2\}}$. Therefore, the distribution over the real $m_{\{1,2\}}$ received by P_2 in the protocol execution and over the simulated $m_{\{1,2\}}$ generated by the simulator is identically distributed.
- $Sim^{P_3}_{\texttt{secRELU}}$: The simulator is simple since P_3 does not participate in the protocol and receives nothing in the real execution. Therefore, it is clear that the simulated view is identical to the real view.

Similarly, in the case of P_2 acting as the sender and P_1 acting as the receiver, the protocol is also secure.

Discussion: As the first research endeavor towards privacy-preserving training and inference of GNNs outsourced to the cloud, the current design of SecGNN only considers the commonly assumed non-colluding and semi-honest threat model, where the three cloud servers $P_{\{1,2,3\}}$ will not collaboratively launch inference attacks, e.g., model inversion attack [73]. On another hand, we are aware that there exist effective mechanisms for bounding information leakage even if $P_{\{1,2,3\}}$ collude with each other, which can also be smoothly integrated into SecGNN for security enhancement. Specifically, we observe that local differential privacy (LDP) [74] and dummy edges padding are promising techniques, of which the blueprint is as follows. It is noted that the private information in the graph-structured data that needs to be protected is the node features and labels and the edges between nodes. First, before encrypting the graph-structured data, the data owner perturbs the node features and labels by the LDP-based obfuscation mechanism [42], which is specifically designed for GNNs. Second, the data owner adds dummy edges with random weights between some pairs of unconnected nodes in the graph-structured data to obfuscate the existence and weights of edges. Finally, the data owner encrypts the graph-structured data after obfuscation by the encryption method introduced in Section V-B. Since the node features and labels and the edges between nodes in the graph-structured data are obfuscated, $P_{\{1,2,3\}}$ cannot learn the accurate original graph-structured data even if they collude with each other. So the above is the blueprint for prevent $P_{\{1,2,3\}}$ from colluding with each other to launch inference attacks in SecGNN, for which it is important to explore how to make the decreased accuracy of the trained GNN model (a natural trade-off) as small as possible upon concrete realizations.

VII. Experiments

A. Setup

The implementation is written in C++ using the standard library. All experiments are performed on a workstation with Intel Core i7-10700K and 64GB RAM running Ubuntu 20.04.2 LTS. Consistent with prior art [16], [26], we consider a Local Area Network (LAN) environment with a network bandwidth of 625MB/s and an average latency of 0.22 ms. For all experiments, we split our computation and communication into data-dependent online phase and data-independent offline phase, and report the end-to-end protocol execution time and the total communication traffic. Our implementation is available at https://github.com/songleiW/SecGNN.

Graph Datasets: We use three graph datasets commonly used in GCN: Citeseer[1], Cora[2] and Pubmed[3] in our experiments. Their statistics are summarized in Table I.

1.https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz
2.https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz
3.https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz

TABLE I
DATASET STATISTICS

Dataset	Nodes	Edges	d_{max}	Classes	Features
Citeseer	3,327	4,732	100	6	3,703
Cora	2,708	5,429	169	7	1,433
Pubmed	19,717	44,338	171	3	500

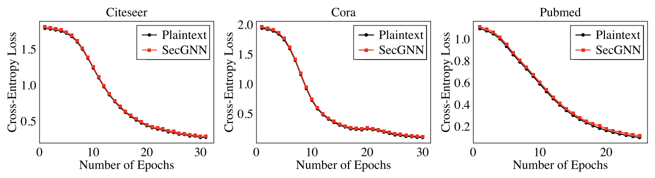


Fig. 4. Evolution of the cross-entropy loss in SecGNN and plaintext, with varying number of epochs over different datasets.

Model Hyperparameters: Similar to [31], we use the two-layer GCN described in (1). For training, we use 40 labeled samples per class but use feature vectors of all nodes. We perform batch gradient descent using the full training set for each epoch. The learning rate is 0.2 and the size of the hidden layer is 16. Early stopping with a window size 5 and public threshold 0.02. We randomly initialize model parameters by the uniform distribution $\mathbf{M}^{(0)} \sim (\frac{-1}{\sqrt{E}}, \frac{1}{\sqrt{E}})$, where E is the number of neurons. We use the same hyperparameters in plaintext and SecGNN.

Protocol Instantiation: We instantiate the sub-protocols in Section V using the following parameter settings. Machine learning algorithms usually perform on real numbers, while the additive secret sharing is restricted to computations over integers. Following previous works [23], [26], we use a fixed-point encoding of real numbers in our secure protocols. Specifically, for a real number x, we consider a fixed-point encoding with t bits of precision: $\lfloor x \cdot 2^t \rceil$. Note that when multiplying two fixed-point encoding numbers, since both of them are multiplied by 2^t, the two parties additionally need to rescale the product scaled by 2^{2t}, where we use the truncation technique from [22]. In our experiments, we consider the ring $\mathbb{Z}_{2^{64}}$ with $t = 15$ bits of precision. The number of iterations of (11) is set to 13, (16) is set to 8, (15) is set to 18, (17) is set to 3 and k is set to 8.

B. Evaluation on Secure GNN Training

Cross-Entropy Loss: We first compare the cross-entropy loss between SecGNN and plaintext training. The results are summarized in Fig. 4. It is observed that the cross-entropy loss of SecGNN is slightly higher than that of plaintext, but they exhibit consistent behavior. Meanwhile, it is revealed that the training processes of SecGNN and plaintext terminate at the same number of epochs, which demonstrates that SecGNN, with security assurance, does not adversely affect the convergence of the training process. This, in turn, also validates the effectiveness of our secure model convergence evaluation protocol in Section V-F.

Validation Set Accuracy: In addition to comparing the evolution of the cross-entropy loss, we first evaluate and compare

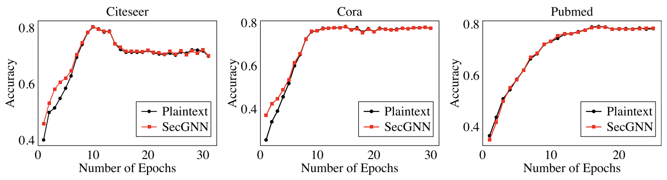


Fig. 5. Evolution of the validation set accuracy in SecGNN and plaintext, with varying number of epochs over different datasets.

the validation set (500 samples excluding the training samples) accuracy between SecGNN and plaintext. The results are summarized in Fig. 5. It can be seen that although the difference in the validation set accuracy between SecGNN and plaintext is obvious at the very beginning, the difference rapidly decreases as the number of epochs grows and eventually vanishes.

Computation and Communication Performance: We now report SecGNN's computation and communication performance in secure training. The results are given in Table II, where the number of training epochs on the three datasets is as follows: Citeseer: 30, Cora: 30, and Pubmed: 25 (as shown in Fig. 4). Over the three tested datasets, the online communication traffic in SecGNN ranges from 3.6 GB to 9.1 GB, and the online end-to-end training time varies from 31.2 minutes to 94 minutes. It is noted that the secure training procedure in SecGNN is full conducted on the cloud and the cost is one-off.

C. Evaluation on Secure GNN Inference

Inference Accuracy: We evaluate the Top-1 inference accuracy in SecGNN which performs inference with models trained in the ciphertext domain via our protocols, and compare it against with plaintext inference which is based on models trained over plaintext graphs. In addition, we compare the average relative error in inference results between SecGNN and plaintext. Table III summarizes the results, from which we can observe that the Top-1 accuracy of SecGNN *exactly* matches that of plaintext.

Computation and Communication Performance: We examine the computation and communication performance of secure inference in SecGNN. Table II shows the cost of inference for a single unlabeled node. Over the three tested datasets, the online end-to-end runtime of the sophisticated secure GNN inference for a single unlabeled node in SecGNN varies from 25.3 seconds to 69.6 seconds, with the online communication traffic ranging from 0.4 GB to 1 GB.

It is worth noting that the average cost of inferring a node's label decreases as the number of test nodes increases. That is because in secure inference, to calculate the encrypted 1_{st}-layer aggregate state (i.e., (5)) for a single unlabeled node, the cloud servers must calculate the encrypted 1_{st}-layer state for all nodes since the cloud servers do not hold the IDs of the unlabeled node's neighboring nodes in plaintext. Therefore, if the cloud servers infer labels for a number of unlabeled nodes in a single batch, the cost can be amortized, so the average cost of individual node inference will go down. Figs. 6 and 7 show the average time and communication cost with varying number of test nodes for inference.

TABLE II
SecGNN's Computation and Communication Performance for Secure GNN Training and Inference

Dataset	Training				Inference (a single unlabeled node)			
	Time (seconds)		Comm. (GB)		Time (seconds)		Comm. (GB)	
	Online	Offline	Online	Offline	Online	Offline	Online	Offline
Citeseer	5,640	168	9.1	13.5	49.7	13.1	0.7	1
Cora	2,664	54	3.6	5.3	25.3	6.7	0.4	0.6
Pubmed	1,872	72	5.1	7.3	69.6	18.6	1	1.5

TABLE III
Inference Accuracy Performance

Dataset		Accuracy	Average relative error
Citeseer	SecGNN	68.3%	0.12%
	Plaintext	68.3%	
Cora	SecGNN	78%	0.11%
	Plaintext	78%	
Pubmed	SecGNN	78.6%	0.12%
	Plaintext	78.6%	

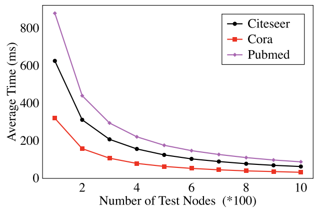


Fig. 6. Amortized runtime cost of secure inference as we vary the number of test nodes.

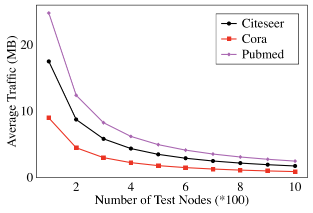


Fig. 7. Amortized traffic of secure inference, as we vary the number of test nodes.

D. Performance Benchmarks on Sub-Protocols

In this section, we will first demonstrate the performance advantage of our proposed secure array access protocol over the state-of-the-art [66] (referred to as the BYK20 protocol hereafter). After that, we evaluate the performance of secure MSB extraction which is used in secure activation functions.

Secure Array Access: To demonstrate the performance advantage of our secure array access protocol over the BYK20 protocol, we evaluate the cost of securely accessing a node's feature vector from an encrypted array, where each array element is a graph node's feature vector. The size of the encrypted array is $N \times L$, where N is the number of graph nodes and L is the length of each node's feature vector. The runtime costs are provided in Table IV. In addition, we further compare the runtime costs of our protocol and the BYK20 protocol, with varying array sizes. The results are given plotted in Fig. 8. It is observed that the efficiency gain of our protocol over the BYK20 protocol increases as the array size grows.

TABLE IV
Performance Comparison of Secure Array Access

		Time (seconds)	Comm. (GB)
Citeseer	BYK20 [66]	22.4	0.37
	Ours	**17.9**	**0.19**
Cora	BYK20 [66]	9.8	0.12
	Ours	**8.1**	**0.06**
Pubmed	BYK20 [66]	27	0.29
	Ours	**23**	**0.15**

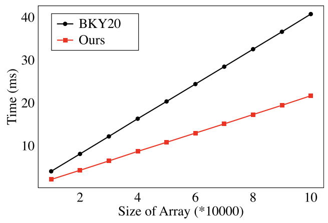


Fig. 8. Runtime comparison of secure array access.

TABLE V
Theoretical Communication Performance of Secure MSB Extraction ($K = 64$)

Scheme	Rounds	Online (bit)	Offline (bit)
ABY^3 [23]	7	**677**	**0**
Ours	**6**	732	1026

TABLE VI
Runtime Comparison of Secure MSB Extraction (in ms)

$h \times k$	16×64	128×128	20×576
ABY^3 [23]	6.08	**74.33**	**52.72**
Ours	**5.89**	74.55	52.81

Secure MSB Extraction: We evaluate and compare the performance of secure MSB extraction with ABY^3 [23]. Table V gives a comparison on the theoretical communication complexity of secure MSB extraction between SecGNN and ABY^3. The protocol in SecGNN consumes one less round, at the cost of more communication bits. Furthermore, we conduct experiments to compare the practical efficiency under different $h \times k$ settings: number of values $\times$ bit-length. The results are given in Table VI. It is observed that our protocol has comparable performance to ABY^3 [23], and is a bit more efficient with a small size setting (16×64). However, it is noted that different from our system, ABY^3's security design uses replicated secret sharing, which runs among three cloud servers and needs them to interact with each other throughout the process. In contrast, we provide an alternative design to evaluate the MSB under additive secret sharing, which requires only two cloud servers $P_{\{1,2\}}$ to interact online, while the third cloud server P_3 just provides necessary triples in offline phase.

VIII. Conclusion

In this article, we design, implement, and evaluate SecGNN, the first system supporting privacy-preserving GNN training and inference as a cloud service. Building on lightweight cryptographic techniques and a multi-server decentralized-trust setting, SecGNN can effectively allow the cloud servers to train a GNN model without seeing the graph data as well as provide secure inference service once the encrypted GNN model is trained. Extensive experiments on real-world datasets demonstrate that SecGNN achieves comparable plaintext training as well as inference accuracy, with practically affordable performance on the cloud. For future work, it would be interesting to explore how to extend our initial research effort to support secure GNN training and inference under a stronger active adversary model, as well as the possibility of leveraging the recent advances in trusted hardware for performance speedup.

References

[1] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surveys*, vol. 52, no. 1, pp. 5:1–5:38, 2019.

[2] J. Kim and M. Hastak, "Social network analysis: Characteristics of online social networks after a disaster," *Int. J. Inf. Manage.*, vol. 38, no. 1, pp. 86–96, 2018.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[4] M. Wu, S. Pan, L. Du, and X. Zhu, "Learning graph neural networks with positive and unlabeled nodes," *ACM Trans. Knowl. Discov. Data*, vol. 15, 2021, Art. no. 101.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.

[7] J. Kim, T. Kim, S. Kim, and C. D. Yoo, "Edge-labeling graph neural network for few-shot learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11–20.

[8] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5171–5181.

[9] Amazon Web Services, "What is Amazon machine learning?," 2021. Accessed: Jul. 15, 2021. [Online]. Available: https://docs.aws.amazon.com/machine-learning/?id=docs_gateway

[10] Microsoft Azure, "Azure AI: Make artificial intelligence real for your business today," 2021. Accessed: Jul. 15, 2021. [Online]. Available: https://azure.microsoft.com/en-us/overview/ai-platform/

[11] Z. Qin, J. Weng, Y. Cui, and K. Ren, "Privacy-preserving image processing in the cloud," *IEEE Cloud Comput.*, vol. 5, no. 2, pp. 48–57, Mar./Apr. 2018.

[12] P. Jiang et al., "Building in-the-cloud network functions: Security and privacy challenges," *Proc. IEEE*, vol. 109, no. 12, pp. 1888–1919, Dec. 2021.

[13] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[14] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 619–631.

[15] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. USENIX Secur. Symp.*, 2018, pp. 1651–1668.

[16] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. USENIX Secur. Symp.*, 2019, pp. 1501–1518.

[17] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High throughput 3PC over rings with application to secure prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 81–92.

[18] A. Patra and A. Suresh, "BLAZE: Blazing fast privacy-preserving machine learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[19] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. USENIX Secur. Symp.*, 2020, pp. 27–30.

[20] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 336–353.

[21] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 325–342.

[22] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.

[23] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[24] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[25] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proc. Privacy Enhancing Technol.*, vol. 2021, no. 1, pp. 188–208, 2021.

[26] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1021–1038.

[27] P. Mohassel, P. Rindal, and M. Rosulek, "Fast database joins and PSI for secret shared data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1271–1287.

[28] Mozilla Security Blog, "Next steps in privacy-preserving Telemetry with Prio," 2019. Accessed: Nov. 01, 2022. [Online]. Available: https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/

[29] Apple and Google, "Exposure notification privacy-preserving analytics (ENPA) White Paper," 2021. Accessed: Nov. 01, 2022. [Online]. Available: https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf

[30] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–20.

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[32] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–12.

[33] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug Discov. Today*, vol. 23, no. 6, pp. 1241–1250, 2018.

[34] H. Wang et al., "Exploring high-order user preference on the knowledge graph for recommender systems," *ACM Trans. Inf. Syst.*, vol. 37, no. 3, pp. 32:1–32:26, 2019.

[35] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2021, pp. 1202–1211.

[36] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "FedGNN: Federated graph neural network for privacy-preserving recommendation," in *Proc. Int. Workshop Federated Learn. User Privacy Data Confidentiality*, 2021, pp. 1–7.

[37] F. Chen, P. Li, T. Miyazaki, and C. Wu, "FedGraph: Federated graph learning with intelligent sampling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1775–1786, Aug. 2022.

[38] C. Chen et al., "Vertically federated graph neural network for privacy-preserving node classification," in *Proc. Int. Joint Conf. Artif. Intell.*, 2022, pp. 1959–1965.

[39] M. Jiang, T. Jung, R. Karl, and T. Zhao, "Federated dynamic graph neural networks with secure aggregation for video-based distributed surveillance," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 4, pp. 1–23, 2022.

[40] Y. Pei et al., "Decentralized federated graph neural networks," in *Proc. Int. Workshop Federated Transfer Learn. Data Sparsity Confidentiality Conjunction IJCAI*, 2021, pp. 1–7.

[41] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[42] S. Sajadmanesh and D. Gatica-Perez, "Locally private graph neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2130–2145.

[43] X. Miao et al., "P ^{2} CG: A privacy preserving collaborative graph neural network training framework," *VLDB J.*, 2022, doi: 10.1007/s00778-022-00768-8.

[44] C. Dwork, "Differential privacy," in *Proc. Int. Colloq. Automata Lang. Program.*, 2006, pp. 1–12.

[45] B. Wang, J. Guo, A. Li, Y. Chen, and H. Li, "Privacy-preserving representation learning on graphs: A mutual information perspective," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2021, pp. 1667–1676.

[46] P. Rodríguez, M. Á. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image Vis. Comput.*, vol. 75, pp. 21–31, 2018.

[47] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

[48] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 507–516.

[49] Y. Zheng, H. Duan, and C. Wang, "Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 10, pp. 2475–2489, Oct. 2018.

[50] W. Chen and R. A. Popa, "Metal: A metadata-hiding file-sharing system," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–15.

[51] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica, "DORY: An encrypted search system with distributed trust," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2020, Art. no. 62.

[52] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, "Secure graph analysis at scale," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 610–629.

[53] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 762–776.

[54] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, "Waldo: A private time-series database from function secret sharing," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 2450–2468.

[55] S. Wang, Y. Zheng, X. Jia, and X. Yi, "Privacy-preserving analytics on decentralized social graphs: The case of eigendecomposition," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2022.3185079.

[56] J. Bell et al., "Distributed, private, sparse histograms in the two-server model," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 307–321.

[57] S. Wang, Y. Zheng, X. Jia, H. Huang, and C. Wang, "OblivGM: Oblivious attributed subgraph matching as a cloud service," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 3582–3596, Sep. 2022.

[58] Y. Zheng, W. Wang, S. Wang, X. Jia, H. Huang, and C. Wang, "SecSkyline: Fast privacy-preserving skyline queries over encrypted cloud databases," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2022.3220595.

[59] S. Wang, Y. Zheng, X. Jia, and X. Yi, "PeGraph: A system for privacy-preserving and efficient search over encrypted social graphs," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 3179–3194, Aug. 2022.

[60] Q. Wang, J. Wang, S. Hu, Q. Zou, and K. Ren, "SecHOG: Privacy-preserving outsourcing computation of histogram of oriented gradients in the cloud," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 257–268.

[61] H. Chun, Y. Elmehdwi, F. Li, P. Bhattacharya, and W. Jiang, "Outsourceable two-party privacy-preserving biometric authentication," in *Proc. 9th ACM Asia Conf. Comput. Commun. Secur.*, 2014, pp. 401–412.

[62] Z. Qin, J. Yan, K. Ren, C. W. Chen, and C. Wang, "Towards efficient privacy-preserving image feature extraction in cloud computing," in *Proc. 22nd ACM Asia Conf. Comput. Commun. Secur.*, 2014, pp. 497–506.

[63] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[64] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 4961–4973.

[65] S. Akram and Q. U. Ann, "Newton Raphson method," *Int. J. Sci. Eng. Res.*, vol. 6, no. 7, pp. 1748–1752, 2015.

[66] M. Blanton, A. Kang, and C. Yuan, "Improved building blocks for secure multi-party computation based on secret sharing with honest majority," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2020, pp. 377–397.

[67] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 805–817.

[68] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "MediSC: Towards secure and lightweight deep learning as a medical diagnostic service," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2021, pp. 519–541.

[69] D. Harris, "A taxonomy of parallel prefix networks," in *Proc. 37th Asilomar Conf. Signals Syst. Comput.*, 2003, pp. 2213–2217.

[70] W. M. P. van der Aalst, V. A. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: A two-step approach to balance between underfitting and overfitting," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 87–111, 2010.

[71] J.-H. He, S. Elagan, and Z. Li, "Geometrical explanation of the fractional complex transform and derivative chain rule for fractional calculus," *Phys. Lett. A*, vol. 376, no. 4, pp. 257–259, 2012.

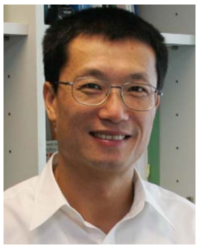
[72] M. Curran, X. Liang, H. Gupta, O. Pandey, and S. R. Das, "ProCSA: Protecting privacy in crowdsourced spectrum allocation," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2019, pp. 556–576.

[73] Z. Zhang, Q. Liu, Z. Huang, H. Wang, C.-K. Lee, and E. Chen, "Model inversion attacks against graph neural networks," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2022.3207915.

[74] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. D. Smith, "What can we learn privately?," *SIAM J. Comput.*, vol. 40, no. 3, pp. 793–826, 2011.

Songlei Wang received the BE degree in Internet of Things from the China University of Petroleum (East China), Qingdao, China, in 2018, and the ME degree in computer technology from the Harbin Institute of Technology, Shenzhen, China, in 2021. He is currently working toward the PhD degree with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing security and secure machine learning.

Yifeng Zheng received the PhD degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He is an assistant professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He worked as a postdoc with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and City University of Hong Kong, respectively. His work has appeared in prestigious conferences (such as ESORICS, DSN, ACM AsiaCCS, IEEE PERCOM, IEEE INFOCOM, and IEEE ICDCS), as well as journals (such as the *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, and *IEEE Transactions on Knowledge and Data Engineering*). He received the Best Paper Award in ESORICS 2021. His research interests lie at the intersection of security, privacy, cloud computing, IoT, machine learning, and data mining.



Xiaohua Jia (Fellow, IEEE) received the BSc and MEng degrees from the University of Science and Technology of China, in 1984 and 1987, and the DSc degree in information science from the University of Tokyo, in 1991. He is currently chair professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile computing. He is an editor of the *IEEE Internet of Things*, *IEEE Transactions on Parallel and Distributed Systems* (2006-2009), *Wireless Networks*, *Journal of World Wide Web*, *Journal of Combinatorial Optimization*, etc. He is the general chair of ACM MobiHoc 2008, TPC co chair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symposium, area-chair of IEEE INFOCOM 2010 and 2015.