RESEARCH-ARTICLE

Collaborative Fraud Detection on Large Scale Graph Using Secure Multi-Party Computation

XIN LIU, Tsinghua University, Beijing, China

XIAOYU FAN, Tsinghua University, Beijing, China

RONG MA, Tsinghua University, Beijing, China

KUN CHEN, Ant group, Hangzhou, Zhejiang, China

YI LI

GUOSAI WANG

View all

Published: 21 October 2024

CIKM '24: The 33rd ACM International Conference on Information and Knowledge Management
October 21 - 25, 2024
ID, Boise, USA

Conference Sponsors:
SIGIR

Collaborative Fraud Detection on Large Scale Graph Using Secure Multi-Party Computation

Xin Liu
Tsinghua University
Beijing, China
liuxin19@mails.tsinghua.edu.cn

Xiaoyu Fan
Tsinghua University
Beijing, China
fanxy23@mails.tsinghua.edu.cn

Rong Ma
Tsinghua University
Beijing, China
rong_ma13@126.com

Kun Chen
Ant Group
Beijing, China
ck413941@antgroup.com

Yi Li
Independent Researcher
Beijing, China
xiaolixiaoyi@gmail.com

Guosai Wang
Tsingjiao Information Technology Co. Ltd.
Beijing, China
guosai.wang@tsingj.com

Wei Xu
Tsinghua University
Beijing, China
weixu@tsinghua.edu.cn

Abstract

Enabling various parties to share data enhances online fraud detection capabilities considering fraudsters tend to reuse resources attacking multiple platforms. Multi-party computation (MPC) techniques, such as secret sharing, offer potential privacy-preserving solutions but face efficiency challenges when handling large-scale data. This paper presents a novel approach, SecureFD (Secure Fraud Detector), aimed at detecting fraud in multi-party graph data, ensuring privacy, accuracy, and scalability. We propose a graph neural network EPR-GNN, which is MPC-friendly, as the base detector. Then we design a framework that allows multiple parties to train EPR-GNN collaboratively on secure sparse graphs in a privacy-preserving manner. The oblivious node embedding sharing protocol in the collaborative training procedure achieves up to a 45× speed-up, supporting over four million users compared to the naive solution. Additionally, we further reduce secure computation by locally pruning a significant number of non-suspicious users and selecting only the most valuable resources for sharing. Experiments on real datasets demonstrate that by securely integrating data from different parties, SecureFD achieves superior detection performance compared to state-of-the-art local detectors. And the local pruning greatly improves the scalability without compromising detection accuracies.

CCS Concepts

• **Security and privacy → Domain-specific security and privacy architectures**.

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679863

Keywords

privacy-preserving, secure multi-party computation, fraud detection, graph neural network

ACM Reference Format:
Xin Liu, Xiaoyu Fan, Rong Ma, Kun Chen, Yi Li, Guosai Wang, and Wei Xu. 2024. Collaborative Fraud Detection on Large Scale Graph Using Secure Multi-Party Computation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679863

1 Introduction

Internet fraud results in significant losses for businesses and individuals alike [1, 34]. The motives behind these fraudulent activities vary widely, from boosting the popularity of a video by purchasing likes and follows [8, 46], to manipulating public perception with deceptive reviews [23], or registering fake accounts for illicit coupon collection and fraudulent loan applications.

Fraudsters often use a limited set of resources, such as specific IP addresses or consecutive phone numbers, to execute substantial fraudulent activities at lower costs and higher profits [11, 47]. Moreover, these fraudulent activities are not confined to a single company; the same resources may target multiple companies. For instance, a fraudster might sell fake likes on both Twitter and Facebook, or post spam reviews on both Taobao and JD.com like in Figure 1. However, each attacked company typically operates fraud detection independently, relying solely on their local datasets.

Intuitively, fraud detection mechanisms would be more effective if these companies collaborated. For example, they could discover that users B, C, and D share the same resources (IP addresses, target items, and rating stars) to post reviews as illustrated in Figure 1. Consequently, both platforms could identify review spam more accurately. Even though some sophisticated fraudsters reduce their resource reuse on individual companies (like user D) and evade

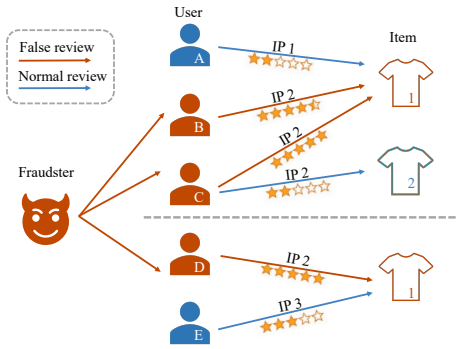

Figure 1: An example of fraudsters posting review spam on two websites reusing the same resources–IP addresses, target items, and rating stars.

localized detectors, they could still be identified through a collaborative approach that captures global resource reuse patterns.

Nevertheless, collaboration poses three challenges: privacy, accuracy and scalability. First, companies are reluctant to share their business data directly, making it essential to ensure user data privacy and provide provable security guarantees. One solution is to share independently computed fraud scores [3], but this shallow data sharing is insufficient to catch sophisticated fraudsters. Another possibility is to deploy existing detection methods within secure multi-party computation (MPC) frameworks [28, 32] which guarantees privacy. However, operations on MPC are often costly [18]. Training even a logistic regression requires careful design to handle large-scale data [13], let alone more complex detection algorithms. Thus, scalability presents another challenge. The third choice is federated learning. But as far as we are concerned, the current works [42, 48] could not model the resource reusing across multiple parties adequately with strong privacy guarantees.

To address these challenges, we propose a novel approach, SecureFD (Secure Fraud Detector), designed with MPC characteristics in mind. Our contributions are summarized as follows.

1) We formulate the multi-party collaborative fraud detection problem on graphs, following current single-party works [17, 29, 30]. Our secure sparse graph representation encrypts minimal data to allow fast operations while meeting privacy requirements.

2)We design an MPC-friendly and powerful graph neural network, EPR-GNN. It features no complex neighborhood aggregation functions and is tailored for MPC training.

3) We propose an oblivious node embedding sharing protocol that enables multiple parties to train EPR-GNN collaboratively on secure sparse graphs from different parties.

4) To further accelerate secure training with minimal impact on detection accuracy, we use local detection results to filter out a significant number of less suspicious user nodes and select only the most suspicious resource nodes for sharing on MPC.

Experiments on real datasets demonstrate the effectiveness of SecureFD, showing the benefits of integrating data from multiple parties. And our optimizations enable SecureFD to process millions of users in less than an hour.

2 Background and Related Work

The Economics of Fraud. Previous studies [39] report that fraudsters often purchase resources from black markets to commit fraud. For example, $250 can buy 15,000-30,000 IP addresses for one month, and $140-420 can buy 1,000 mobile numbers. With these resources, fraudsters can register fake accounts and post fake likes or reviews for profit. For instance, 1,000 follows on Twitter sell for $4-20 [39]. Obviously, fraudsters need to reuse these resources to maximize their profits.

Fraud Detection Methods. The reuse of resources leads to synchronized behavior among the accounts fraudsters manipulate and becomes a key to detection [6, 11]. Since fraud labels are expensive to obtain, many practical detection methods are unsupervised. Some model the data as a graph and identify the densest regions as fraudulent behavior [7, 23, 36, 37]. Other methods use domain-specific clustering algorithms to find users with high similarities [11, 40]. Meanwhile, supervised and semi-supervised methods are also emerging. For example, [30] uses graph neural networks to detect manipulated users in Alipay, while [16] addresses the camouflage problem in fraud detection. And [29] tackles the class imbalance issue, particularly in picking neighbors of nodes. [43] focuses on detecting fraud in temporal transaction graphs.

There also exist efforts to allow multiple parties to collaborate. [3] enables service providers to jointly detect spammers, but it only allows sharing independently computed fraud scores, which cannot catch sophisticated fraud that shares resources across parties.

Secret-Sharing (SS) Based Multi-Party Computation (MPC). MPC is a well-known cryptographic technique. Suppose C parties, each with its private dataset, want to jointly compute a function using these datasets as inputs. With MPC, they will not leak any private information other than the final outputs. Among various MPC schemes [2, 5, 9, 19, 20, 28, 31, 32], secret sharing (SS) has relatively low overhead. Thus, many MPC systems [2, 28, 32] and applications [24, 38, 41, 45] leverage the SS scheme. There are multiple computing servers in SS each of which receives a share of the private data. To reveal some private data like the final results, the shares from all computing servers are joined together. Under certain security assumptions, such as no collusion between computing servers, the privacy of the secret-shared data is preserved. A general-purpose SS system provides basic operations like addition and multiplication and allows us to construct more complex algorithms based on them.

However, the basic operations are not always efficient [18]. In most SS systems, such as [28, 32], operations that do not involve communication among computing servers are nearly as fast as plaintext computation, such as adding secure scalars or retrieving elements from a secure vector by plain indices. But other operations, including multiplication and comparisons, require rounds of communication and are much slower than plaintext operations. Consequently, matrix multiplication and non-linear function approximations built from basic operations are time-consuming, too.

PrivPy [28] is an MPC system employing a two-out-of-four secret sharing framework, achieving high-performance speeds for operations like matrix multiplications and secure comparisons. Besides, it provides easy-to-use Python programming front-end. So we choose it as the underlying MPC system in the experiments.

Federated Graph Neural Networks. Recently, research on federated GNNs [42, 44, 48, 49] has emerged. [49] considers the vertical setting, where each party has the same node set but different edges and node feature dimensions. Conversely, [44] focuses on the horizontal setting, where the graph nodes are different but the feature dimensions are the same. This is more applicable to our problem. As pointed out by [48], one of the key challenges in the horizontal setting is managing cross-party graph connections. [48] trains a missing neighbor generator to handle missing links across local graphs. [42] proposes a central node embedding server that allows training parties to expand local subgraphs with exchanged features or embeddings without revealing the local graph topology. However, it exposes the embeddings of the one-hop neighbors, while our secure collaborative graph forwarding does not.

3 Problem Formulation

We first formulate the fraud detection problem for a single party as a heterogeneous bipartite graph node classification task and then extend it to the multi-party scenario.

Consider a data party that owns a dataset, such as user reviews or access records on a website or application. A heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed from these records. The i-th node in $\mathcal{V}$ has a unique ID denoted by v_i. For example, v_1 might be a timestamp *2022-01-01* and v_2 a book titled *Deep Learning*. Besides, the nodes $\mathcal{V}$ are divided into two subsets: user nodes $\mathcal{V}_1$ and resource nodes $\mathcal{V}_2$. Each edge connects a user node to a resource node and is associated with a weight. Thus, the graph $\mathcal{G}$ is bipartite. Additionally, the edges have R types, $\mathcal{E} = \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_R$. The corresponding adjacency matrices are denoted as $\mathbf{A}_1, \cdots, \mathbf{A}_R$. And the feature matrix are denoted as $\mathbf{X}$ with D feature dimensions. Some users are labeled as fraud or normal, with y_i equal to 1 or 0, respectively. But many remain unlabeled. Our goal is to predict the probability that each unlabeled user is fraudulent. Obviously, this is a semi-supervised node classification task.

Now, consider C data parties, each owning a local graph $\mathcal{G}^{(i)} = (\mathcal{V}^{(i)}, \mathcal{E}^{(i)})$ for $i = 1, \ldots, C$ and planning to detect fraud collaboratively. We make three assumptions for the multi-party fraud detection task. 1) All local graphs $\mathcal{G}^{(i)}$ follow the same paradigm, having the same edge types and node feature dimensions. 2) We assume that each party has their own user ID system and could not be matched. Even if there exist identifiers for global matching, like identity card numbers, we still consider them resources that users connect to. Thus, the user node sets are non-overlapping, i.e., $\mathcal{V}_1^{(i)} \cap \mathcal{V}_1^{(j)} = \emptyset$ for all $i \neq j$. 3) Unlike user nodes, resource nodes share a global ID system. For example, IP address nodes in different parties can be matched exactly. Therefore, the resource node sets have some non-empty intersections, meaning at least one pair (i, j) satisfies $\mathcal{V}_2^{(i)} \cap \mathcal{V}_2^{(j)} \neq \emptyset$.

4 Methods

4.1 Graph Neural Network Detector

Given the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ formulated in Sec.3 with adjacency matrices $\mathbf{A}_r$ for multiple edge types, let the hidden embedding of all nodes at layer l for edge type r in the graph neural network be $\mathbf{H}_{l;r}$ and the initial embedding be $\mathbf{H}_0 = \mathbf{X}$.

The graph neural network used in SecureFD is adapted from the generalized PageRank graph neural network (GPR-GNN) [15], and we call it EPR-GNN (Enhanced PageRank-GNN). It consists of L graph layers followed by two MLP layers. There are two main differences between our model and GPR-GNN. First, we extend GPR-GNN to support multiple edge types by adopting the message passing and aggregation architecture from [22]. The forwarding process of EPR-GNN for the l-th graph layer is:

$$\mathbf{H}_{l;r} = \mathbf{A}_r \mathbf{H}_{l-1}, \tag{1}$$

$$\mathbf{H}_l = \mathbf{H}_{l;1} \oplus \cdots \oplus \mathbf{H}_{l;R} \oplus \mathbf{H}_{l-1}. \tag{2}$$

where Eq. (1) generates the hidden embedding $\mathbf{H}_{l;r}$ for edge type r, and Eq. (2) concatenates the hidden embeddings for each edge type with the previous layer's embedding $\mathbf{H}_{l-1}$. Second, instead of training the GPR weights, we adopt specialized values similar to [27], resulting in using only the last graph layer to output probabilities. The last hidden embedding $\mathbf{H}_L$ is fed into two MLP layers with trainable weights $\mathbf{W}_1$ and $\mathbf{W}_2$ to get the final probabilities

$$\mathbf{P} = \text{Sigmoid}(\text{ReLU}(\mathbf{H}_L \mathbf{W}_1)\mathbf{W}_2). \tag{3}$$

The two enhancements improve the model's capacity for better fraud node classification performance.

We want to emphasize that the GNN structure in SecureFD enables fast execution on MPC. The first L graph layers have no weights, so the hidden embeddings of these layers remain fixed once the graph structure and node features are set. During training, the time-consuming graph neural network layers before the MLP could be forwarded only in the first epoch. In subsequent epochs, we reuse the hidden embedding $\mathbf{H}_L$ for forwarding, back-propagation, and updating the weights $\mathbf{W}_1$ and $\mathbf{W}_2$ in the MLP layers.

4.2 Secure Sparse Graph Representation

Next, we introduce how we represent a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrices $\mathbf{A}_r$ for multiple edge types in a secret-sharing based MPC system. First, the node ID vector $\mathbf{v} = (v_1, v_2, \ldots)$ is encrypted in secret shares, ensuring that no one knows the actual meaning of the nodes. Second, as most real-world graphs are sparse, we only store the indices and weights of edges in the adjacency matrix $\mathbf{A}_r$ to save space. While this is straightforward in plaintext, retrieving weight values is expensive when indices are secret-shared [26]. To address this, we choose to expose the row and column indices plaintext and keep only the values secure. The privacy of the edges is maintained because the node IDs are encrypted. Thus, it is impossible to infer the meaning of an edge, such as a user reviewing a product. In Sec.5, we further protect the edges using *edge differential privacy*. Finally, the node embeddings $\mathbf{H}$ are also encrypted in secret shares. The space complexity of the secure sparse graph is then $O(|\mathcal{E}| + D_H|\mathcal{V}|)$, where D_H is the embedding dimension. The time complexity of multiplying the secure sparse adjacency matrix $\mathbf{A}_r$ by the embedding matrix involved in Eq.(1) is $O(D_H|\mathcal{E}_r| + D_H|\mathcal{V}|)$.

4.3 Collaborative Graph Layer Forwarding

Now, we consider how to enable C data parties to jointly forward embeddings described in Eq.(1)-(2) on the secure sparse graphs. The key challenge is handling cross-party graph connections [42,

Algorithm 1 Secure Collaborative Graph Layer Forwarding

Input: Secure adjacency matrices $\mathbf{A}_r^{(c)}(r = 1, \ldots, R)$, secure input embedding $\mathbf{H}_{l-1}^{(c)}$, secure node ID vectors $\mathbf{v}^{(c)}$, for $c = 1, \ldots, C$

Output: Secure output embedding $\mathbf{H}_l^{(c)}$ for $c = 1, \ldots, C$

1: // Forwarding on each party's secure graph
2: **for** each data party $c = 1, \ldots, C$ **do**
3: **for** each edge type $r = 1, \ldots, R$ **do**
4: Compute local embedding $\tilde{\mathbf{H}}_{l;r}^{(c)} = \mathbf{A}_r^{(c)}\mathbf{H}_{l-1}^{(c)}$
5: // Oblivious node embedding sharing
6: **for** each data party $c = 1, \ldots, C$ **do**
7: **for** each edge type $r = 1, \ldots, R$ **do**
8: Compute shared embedding $\mathbf{H}_{l;r}^{(c)}$ using $\{\tilde{\mathbf{H}}_{l;r}^{(k)}, \mathbf{v}^{(k)}\}_{k=1}^{C}$ in Algorithm 2, with party c as the first input party
9: // Output embedding
10: **for** each data party $c = 1, \ldots, C$ **do**
11: Compute $\mathbf{H}_l^{(c)} = \mathbf{H}_{l;1}^{(c)} \oplus \cdots \oplus \mathbf{H}_{l;R}^{(c)} \oplus \mathbf{H}_{l-1}^{(c)}$
12: **return** $\mathbf{H}_l^{(c)}$ for $c = 1, \ldots, C$

48]. For instance, a resource node in party 1 may share the same ID with another resource node in party 2 and thus have edges in both parties. Naively, we could merge the secure graphs from all parties into a global graph and perform graph forwarding on it. However, this is time-consuming. So we present Algorithm 1, which avoids the need for such global merging. The algorithm first performs forwarding on each party's secure graph and then calls the oblivious node embedding sharing protocol, described in the next section, to handle cross-party connections. Figure 2 illustrates Algorithm 1 with an example.

It is guaranteed that the resulting node embeddings from Algorithm 1 are identical to those obtained by forwarding on a globally merged graph. The graph forwarding in Eq.(1) means that a node's embedding is the weighted sum of all its neighbors, with weights given by the edges. For a user node $v_i^{(c)}$ in party c, embedding sharing is unnecessary as it only connects to resource nodes within the same party. For a resource node $v_i^{(c)}$ in party c, we will show that its embedding on the global graph equals the sum of its embeddings on each party's secure graph. According to our assumptions, it may share the same ID with resource nodes in other data parties. Suppose it shares the same ID as the i_k-th node in party k, which Sec.4.4 will show how to compute securely. Its potential neighbors include the user node set $\mathcal{V}_2^{(k)}$ of every party k. Since the user node sets are non-overlapping, the hidden embedding of node $v_i^{(c)}$ at layer l is given by:

$$\mathbf{H}_{l;r;i}^{(c)} = \sum_{k=1}^{C} \sum_{j_k=1}^{|\mathcal{V}_2^{(k)}|} \mathbf{A}_{r;i_k,j_k}^{(k)} \mathbf{H}_{l-1;j_k}^{(k)} \tag{4}$$

$$= \sum_{k=1}^{C} \tilde{\mathbf{H}}_{l;r;i_k}^{(k)}, \tag{5}$$

where $\tilde{\mathbf{H}}_{l;r;i_k}^{(k)}$ is the local embedding of resource node $v_{i_k}^{(k)}$ at layer l computed on the secure graph in party k.

The time complexity of Algorithm 1 has two terms. The first is the local graph forwarding $O(D_{l-1} \sum_c |\mathcal{E}^{(c)}| + RD_{l-1}|\mathcal{V}|)$ where $D_{l-1} = (R+1)^{l-1}D$ is the input embedding dimension and $|\mathcal{V}|$ is the total number of nodes from all parties. The second is the oblivious node embedding sharing $O(CRD_{l-1}|\mathcal{V}| \log |\mathcal{V}|)$ (See Sec.4.4.2).

4.4 Oblivious Node Embedding Sharing

Here we demonstrate how to share embeddings obliviously without exposing how the resource nodes are matched across different data parties. For simplicity, we omit the layer l and edge type r in this section. Additionally, although node embeddings are usually multi-dimensional, we use one-dimensional embeddings as an example for clarity. In this case, the embedding matrix $\mathbf{H}$ reduces to a vector denoted as $\mathbf{h}$.

The oblivious node embedding sharing problem is as follows. Let the secure node embedding vector computed on the local graph of party i be denoted as $\mathbf{h}^{(i)}$, and the secure node ID vector as $\mathbf{v}^{(i)}$, for $i = 1, \ldots, C$. Without loss of generality, we consider that the first party wants to retrieve the embeddings of all its nodes $\mathbf{v}^{(1)}$ from sharing parties $2, \ldots, C$ and add them to $\mathbf{h}^{(1)}$.

4.4.1 Naive Solution. Consider a node $v_i^{(1)}$ in the first party that needs to get embeddings shared from other parties. We compare its node ID $v_i^{(1)}$ to every node $v_s^{(k)} \in \mathbf{v}^{(k)}$ in another party k using a secure comparison protocol, obtaining a secure boolean $\mathbb{1}\{v_i^{(1)} = v_s^{(k)}\}$. The embedding $\mathbf{h}_i^{(1)}$ of node $v_i^{(1)}$ is updated using the following formula:

$$\mathbf{h}_i^{(1)} = \mathbf{h}_i^{(1)} + \sum_{k=2}^{C} \sum_{s=1}^{|\mathbf{v}^{(k)}|} \mathbb{1}\{v_i^{(1)} = v_s^{(k)}\} \mathbf{h}_s^{(k)}. \tag{6}$$

If the secure boolean $\mathbb{1}\{v_i^{(1)} = v_s^{(k)}\}$ is True, the compared nodes have the same ID, and the embedding $\mathbf{h}_s^{(k)}$ from party k is added to the node embedding in party 1. Otherwise, the adding is not performed. If the node ID $v_i^{(1)}$ does not exist in party k, all comparison booleans will be 0, and it does not get any sharing from party k.

This solution requires computing a secure comparison for every node pair across all parties, resulting in quadratic complexity. Such complexity makes it impractical for large datasets with millions of nodes. Hence, we propose the more efficient Algorithm 2.

4.4.2 Faster Solution. The idea of Algorithm 2 is inspired by [26], which provides oblivious vector read and write protocols. Algorithm 2 first combines all nodes (lines 2-3) and sorts them by their IDs (line 7) so that nodes with the same ID from different data parties are adjacent. The stable sort ensures that among nodes with the same ID, the first party's node is always the first, and so on. Next, we only need to compare a node's ID securely with its adjacent nodes. If the IDs match, the embedding is shared; otherwise, it is not (lines 10-12). Finally, all nodes are shuffled back to their original positions, and the first party's node embeddings after sharing are extracted (line 15). Figure 2 illustrates how nodes in party 1 obliviously receive sharing from party 2 in a two-party setting.

Algorithm 2 uses a secure functionality Sortperm that sorts the first input $\mathbf{v}$ and permutes the subsequent inputs by the same order. This is adapted from a provably secure quicksort protocol [21]

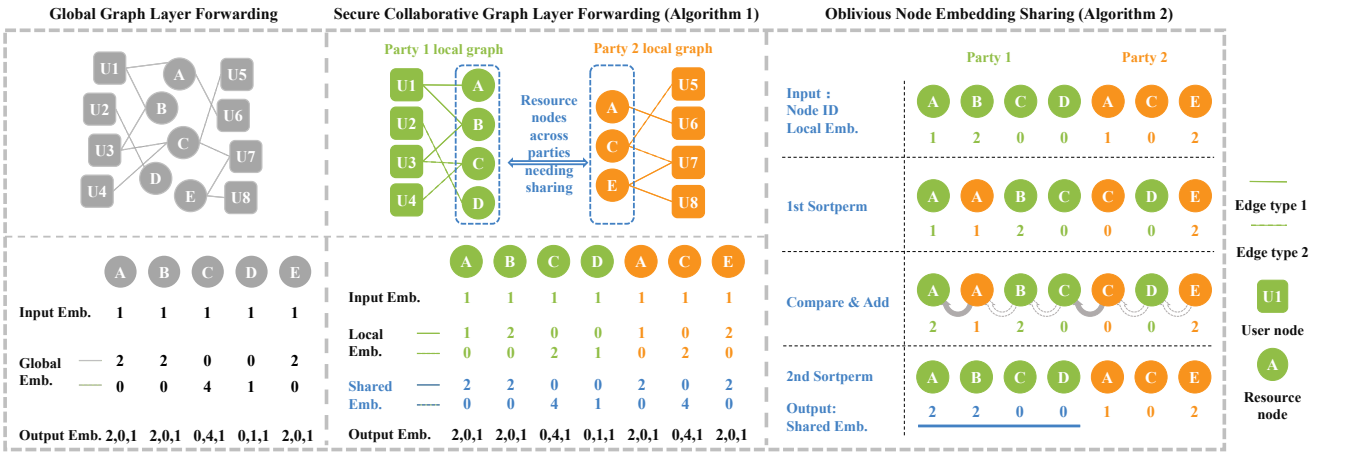

Figure 2: Illustration of the secure collaborative graph layer forwarding. All nodes have a one-dimensional input embedding of 1. After one graph layer forwarding, the output embedding reflects the node degree. Our collaborative graph layer forwarding (Algorithm 1) produces the same embeddings as those obtained by forwarding on the merged global graph. The figure also shows the oblivious node embedding sharing (Algorithm 2), which is called by Algorithm 1 to handle cross-party connections.

Compared to plaintext quicksort, the secure quicksort (1) first obliviously shuffles the input randomly and (2) uses secure comparison, revealing the outcome to decide whether to swap or not. The oblivious random shuffling at the beginning ensures that revealing the comparison outcome does not leak any information. Based on this quicksort protocol, we implement the secure Sortperm. We only need to shuffle and swap the next inputs simultaneously when the secure quicksort protocol shuffles and swaps the first input $\mathbf{v}$. It is straightforward to show that Sortperm is as secure as the secure quicksort in [21]. To make Sortperm a stable sort, we process the first input $\mathbf{v}$ to ensure all its elements are unique without changing the resulting sorting order. For example, we add an increasing vector on a small scale to $\mathbf{v}$ if it is a numerical vector. Additionally, the time complexity of Sortperm is $O(|\mathbf{v}| \log |\mathbf{v}|)$. The oblivious shuffling, like that provided by PrivPy [28], typically has no more than linear complexity. Therefore, the complexity of Sortperm is dominated by the recursive secure comparisons.

The overall oblivious node embedding sharing described in Algorithm 2 achieves an efficient sub-quadratic time complexity of $O(|\mathbf{v}| \log |\mathbf{v}|)$, where $|\mathbf{v}| = \sum |\mathbf{v}^{(i)}|$ is the total number of nodes. Given that the number of data parties C is typically much smaller than $|\mathbf{v}|$, the complexity $O(C|\mathbf{v}|)$ in lines 7-9 is absorbed.

Algorithm 2 Oblivious Node Embedding Sharing

Input: Secure node embedding vectors $\mathbf{h}^{(i)}$ and node IDs $\mathbf{v}^{(i)}$, $i = 1, \ldots, C$. Party 1 merges embeddings from parties $2, \cdots, C$.
Output: Party 1's shared node embedding vector $\hat{\mathbf{h}}$

1: // Combine all nodes and their embeddings
2: Concatenate $\mathbf{h}^{(i)}$, $i = 1, \ldots, C$ into a new vector $\mathbf{h}$
3: Concatenate $\mathbf{v}^{(i)}$, $i = 1, \ldots, C$ into a new vector $\mathbf{v}$
4: Let m be the length of $\mathbf{v}$
5: Let secure $\mathbf{index} = (1, 2, \ldots, m)$
6: // Sort nodes by their IDs
7: $\mathbf{h}', \mathbf{v}', \mathbf{index}' = \text{Sortperm}(\mathbf{v}, (\mathbf{h}, \mathbf{v}, \mathbf{index}))$
8: // Compare IDs of adjacent nodes and share embeddings
9: Initialize $\mathbf{o}$ as a zero vector of length m
10: **for** $i = 0, \ldots, C-1$ **do**
11: **for** $j = 1, \ldots, m-i$ **do**
12: $\mathbf{o}_j = \mathbf{o}_j + \mathbb{1}\{\mathbf{v}'_j = \mathbf{v}'_{j+i}\}\mathbf{h}'_{j+i}$
13: // Shuffle nodes back to original positions
14: $\hat{\mathbf{o}} = \text{Sortperm}(\mathbf{index}', (\mathbf{o}))$
15: Extract the head of $\hat{\mathbf{o}}$ with the same length as $\mathbf{h}^{(1)}$ to get $\hat{\mathbf{h}}$
16: **return** $\hat{\mathbf{h}}$

4.5 MLP Layers Training

The forwarding of MLP layers is described in Eq.(3). Here, the model weights, intermediate training results such as gradients, and training hyperparameters like the learning rate and regularization coefficient are all encrypted in secret shares. The ReLU activation function is implemented using a secure $\geq$ comparison protocol, and the Sigmoid activation function is approximated using the Newton-Raphson method.

Recall that the graph neural network layers before the MLP layers have no parameters. Therefore, during training, the inputs to the MLPs remain unchanged. Let the training set merged from all parties be $\mathcal{V}_{train} = \cup_i \mathcal{V}^{(i)}_{train}$. For a mini-batch $\mathcal{V}_{batch} \subset \mathcal{V}_{train}$ sampled from the training set, the cross-entropy loss to minimize is given by:

$$J = \sum_{v_i \in \mathcal{V}_{batch}} y_i \ln p_i + (1 - y_i) \ln(1 - p_i). \tag{7}$$

In fact, we do not need to compute the value of the loss function involving the non-linear function ln. As the gradient back-propagation formula is straightforward to derive, it is omitted here. We employ the Adam optimizer to update the weights iteratively.

Given T training epochs, the time complexity of two-layer MLP training is $O(TD_L D' |\mathcal{V}_{train}|)$, where $D_L = (R+1)^L D$ is the input

embedding dimension of MLP layers and D' is the hidden embedding dimension of the MLP layers. The complexity of predicting all user nodes is $O(D_L D'|\mathcal{V}|)$.

4.6 Further Speed-up Using Local Computation

Before a data party sends its private data to MPC for collaborative detection, it first runs a local detection algorithm with decent accuracy, such as EPR-GNN, to reduce the subsequent MPC computation as follows.

4.6.1 Local User Node Pruning. Denote a user's local fraud probability (or score) as $\tilde{p}_i^{(c)}$, for a node $v_i^{(c)} \in \mathcal{V}_1^{(c)}$. If a user is predicted to be normal with high confidence, this user will not be included in the secure graph for collaborative detection. Specifically, we sort the users by their fraud probabilities (or scores) $\tilde{p}_i^{(c)}$ and filter out a fraction β_1 of unlabeled users. This fraction β_1 is referred to as the *pruning* ratio. As normal users typically take the majority, we can filter out a large portion of normal users, e.g., $\beta_1 = 70\%$. Now, two types of users will be used for collaborative detection: 1) labeled fraud and normal users[1] for training; 2) the unlabeled users that are not filtered and need to be predicted more accurately. Consequently, resource nodes not connected to the remaining users are also pruned. As the pruning ratio β_1 increases, the graph size is reduced, making all layers of SecureFD faster on MPC.

4.6.2 Selecting the Most Suspicious Resource Nodes for Sharing. Next, we propagate the users' fraud probabilities (or scores) to the resource nodes they connect to and select some candidates to share node embeddings with. The suspicious score $\tilde{p}_i^{(c)}$ of a resource node $v_i^{(c)} \in \mathcal{V}_2^{(c)}$ in party c is computed by the following formula:

$$\tilde{p}_i^{(c)} = \sum_{r=1}^{R} \sum_{j=1}^{|\mathcal{V}_1^{(c)}|} A_{r;i,j}^{(c)} \tilde{p}_j^{(c)}. \tag{8}$$

We choose a small fraction β_2 of the most suspicious resource nodes as sharing candidates $\mathcal{V}_{share}^{(c)}$. This fraction β_2 is referred to as the *sharing* ratio. Previously in Algorithm 1, we assume all data parties input all resource node embeddings when calling Algorithm 2 for sharing. Now, they only input the sharing candidates $\mathcal{V}_{share}^{(c)}$, $c = 1, \ldots, C$. Unlike local user pruning, selecting the most suspicious resource nodes for sharing only influences the graph layer forwarding in SecureFD. MLP layers are not affected. We expect that a smaller sharing ratio β_2 will result in faster training.

4.6.3 Advancing MPC Computation to Plaintext. The first graph layer's local forwarding can be done in plaintext. Additionally, since the training set contains only user nodes, we only need the hidden embeddings of user nodes for the last graph layer and do not have to perform the oblivious resource node embedding sharing.

4.7 The Overall Procedure of SecureFD

Finally, we summarize the overall collaborative fraud detection procedure in SecureFD. All data parties train one EPR-GNN model with $L = 2$ graph layers, which are already powerful enough for node representation [25, 30]. In brief, the detection process is divided into local plaintext computation (steps 1-3) and secure multi-party computation (steps 4-7).

Step 1: Data preprocessing. Each data party constructs the local heterogeneous graph, prepares the node ID, features, labels, the training set and sparse adjacency matrices.

Step 2: Local plaintext fraud detection. Each data party runs a local fraud detector. Based on the detection results, prune the least suspicious unlabeled users and select the most suspicious resources for later sharing.

Step 3: Local plaintext graph forwarding. Perform the first layer's local graph forwarding in plaintext as described in Sec.4.6.3.

Step 4: MPC initialization. Initialize the secure multi-party computation service. Each data party sends the adjacency matrices, node features, node IDs, and precomputed results from steps 2-3 as secret shares to the MPC platform. The selected sharing resource and training node indices are sent in plaintext, as their privacy is protected by the secure node IDs.

Step 5: Completing the rest of the graph forwarding on MPC. Perform the oblivious resource node embedding sharing for the first graph layer using Algorithm 2 and compute the user nodes' second graph layer embedding using Eq. (1)-(2).

Step 6: Training MLP layers on MPC. Fix the user nodes' second layer graph embeddings as input to the MLP layers. Randomly initialize the weights. Then iteratively forward, backpropagate, and update the weights by minimizing the loss on the training nodes using the Adam optimizer.

Step 7: Predicting unknown fraud users on MPC. Compute the fraud probabilities of unlabeled user nodes and return them in plaintext to the original data parties.

5 Security Analysis

Security Model and Guarantees. We use the standard and easy-to-implement MPC assumptions: 1) All parties are *semi-honest*: they follow the designed MPC protocol but are curious to mine others' information if given the chance. As deviations from the protocol can be easily detected, this assumption is feasible to uphold in a consortium setting. 2) The communication channels between all parties are secure, which could be implemented using well-known encryption protocols like SSL. And we also assume that all implementation codes are accessible to all parties.

We analyze security under the Universal Composability (UC) framework [10] like common discussions of MPC protocols. The basic secure protocols in SecureFD are provided by the secret-sharing MPC system that builds SecureFD and are proven secure already. Therefore, the security of SecureFD can be proven using the composition theorem [10]. These protocols include addition, multiplication, equal comparison, greater-than comparison, the reciprocal, oblivious random shuffle, oblivious shuffle, and matrix dot product.

Enhancing Privacy of Secure Sparse Graphs. In the representation of secure sparse graphs, although the node IDs are encrypted, there are still potential risks in revealing the row and column indices, as the edges are exposed to some degree. To mitigate this, we let each data party randomly include additional edges locally

[1] We down-sample the labeled normal users for label balance.

with weights equal to zero when constructing the secure sparse graph. Specifically, for every non-linked user-resource node pair, an edge with zero weight is generated with probability ρ. Since the edge weights are secret-shared locally before collaborative detection, these zero-weight edges are indistinguishable from real edges during training. This ensures that adversaries cannot determine which user nodes connect to which resource nodes, thereby preventing further information leakage.

Obviously, these zero-weight edges do not affect the training results. Additionally, the computation overhead introduced to the graph neural network layer forwarding is linear in ρ. For a data party c with $|\mathcal{V}_1^{(c)}|$ user nodes and $|\mathcal{V}_2^{(c)}|$ resource nodes, the expected number of newly introduced edges is $\rho|\mathcal{V}_1^{(c)}||\mathcal{V}_2^{(c)}|$.

To formally analyze the privacy of this approach, we focus solely on the connections between nodes on MPC, disregarding the weights as they are encrypted. We employ the *edge differential privacy* [12, 35] notation, where two graphs $\mathcal{G}$ and $\mathcal{G}'$ are neighbors if they differ by exactly one edge. Denote the process of including generated zero-weight edges as $B(\cdot)$. For all sets S of possible outputs produced by $B(\cdot)$, we claim that

$$\Pr[B(\mathcal{G}') \subseteq S] \leq 1/\rho \Pr[B(\mathcal{G}) \subseteq S] + (1-\rho). \tag{9}$$

Therefore, the closer ρ is to 1, the harder it is to determine if the edges on MPC genuinely exist.

The proof is as follows. Suppose that $\mathcal{G}$ is missing an edge e from $\mathcal{G}'$. For a possible output graph set S produced by $B(\cdot)$,

$$\Pr[B(\mathcal{G}') \subseteq S] = \sum_{\mathcal{G}^* \in S} \Pr[B(\mathcal{G}') = \mathcal{G}^*].$$

If $\mathcal{G}^*$ does not contain the differing edge e, as the process $B(\cdot)$ will not delete the existing edge e from $\mathcal{G}'$, we have $\Pr[B(\mathcal{G}') = \mathcal{G}^*] = 0$. Otherwise, $\Pr[B(\mathcal{G}) = \mathcal{G}^*] = \rho \Pr[B(\mathcal{G}') = \mathcal{G}^*]$ by first inserting the missing edge e with probability ρ into $\mathcal{G}$ and then independently transforming $\mathcal{G}'$ into $\mathcal{G}^*$ during the process $B(\cdot)$. Therefore,

$$\Pr[B(\mathcal{G}') \subseteq S] \leq \sum_{\mathcal{G}^* \in S} 1/\rho \Pr[B(\mathcal{G}) = \mathcal{G}^*] = 1/\rho \Pr[B(\mathcal{G}) \subseteq S]. \tag{10}$$

Conversely, suppose that $\mathcal{G}$ has one more edge e than $\mathcal{G}'$. $\Pr[B(\mathcal{G}') \subseteq S]$ can be expressed as the sum of two terms

$$\sum_{(\mathcal{G}^* \in S)\&(e \in \mathcal{G}^*)} \Pr[B(\mathcal{G}') = \mathcal{G}^*] + \sum_{(\mathcal{G}^* \in S)\&(e \notin \mathcal{G}^*)} \Pr[B(\mathcal{G}') = \mathcal{G}^*].$$

If $\mathcal{G}^* \in S$ contains the differing edge e, $\Pr[B(\mathcal{G}') = \mathcal{G}^*] = \rho \Pr[B(\mathcal{G}) = \mathcal{G}^*]$. Hence the first term

$$\sum_{(\mathcal{G}^* \in S)\&(e \in \mathcal{G}^*)} \Pr[B(\mathcal{G}') = \mathcal{G}^*] = \sum_{(\mathcal{G}^* \in S)\&(e \in \mathcal{G}^*)} \rho \Pr[B(\mathcal{G}) = \mathcal{G}^*]$$
$$\leq \sum_{\mathcal{G}^* \in S} \rho \Pr[B(\mathcal{G}) = \mathcal{G}^*] = \rho \Pr[B(\mathcal{G}) \subseteq S].$$

And the second term

$$\sum_{(\mathcal{G}^* \in S)\&(e \notin \mathcal{G}^*)} \Pr[B(\mathcal{G}') = \mathcal{G}^*] \leq \sum_{e \notin \mathcal{G}^*} \Pr[B(\mathcal{G}') = \mathcal{G}^*] = 1 - \rho,$$

as the probability of not generating e into $\mathcal{G}'$ is $1-\rho$. So,

$$\Pr[B(\mathcal{G}') \subseteq S] \leq \rho \Pr[B(\mathcal{G}) \subseteq S] + (1-\rho). \tag{11}$$

Combing Eq.(10) and Eq.(11), we prove the claim in Eq.(9).

6 Experiments

6.1 Setup

Datasets and Multi-Party Simulation. As multi-party datasets are difficult to obtain, we follow common practice in MPC studies [14, 38] and partition the following four datasets to emulate the multi-party scenario. `S-FFSD` [43] is a financial fraud semi-supervised dataset with partially labeled records. The labels are provided by consumers and further confirmed by domain experts. We regard the records as users to detect. `AmazonMovie`, `AmazonBook`, and `AmazonAll` are based on real public datasets that collect users' reviews on Amazon in the corresponding categories [33]. Following the method in [4, 23], we inject fraud review records into the normal datasets. We add multiple fraud groups to the dataset, each comprising some existing normal users who give extremely high (5.0) or low (1.0) review scores to newly injected products within one day. These *hijacked camouflaged* fraudsters have both legitimate and fake reviews to boost or hurt target products, making them the most challenging to detect. Details are in Table 1.

To emulate multi-party behaviors, we partition the datasets into three to five parties, allowing each party to have a different number of users to reflect the imbalanced data distribution found in the real world. The total user ratios are 1:2:3 for three parties, 1:2:3:4 for four parties, and 1:2:3:4:5 for five parties.

Methods in Comparison. We compare SecureFD with popular fraud detection algorithms to demonstrate the benefits of collaborative detection. We run algorithms 1-9 independently on each party's dataset and then compute the *area under the receiver operating characteristic curve (AUC)* on the testing set as the quality metric for each party. 1) M-Zoom [36]: unsupervised fraud detection method identifying dense subgraphs as fraud. 2) M-Biz [36]: improved version of M-Zoom, guaranteeing locally optimal subgraph densities. 3) D-Cube [37]: scalable dense block detection algorithm for web-scale data. 4) AnoGraph [7]: online anomaly detection in dynamic graphs. 5) GraphSAGE [22]: inductive node classification algorithm aggregating node features from neighborhoods. 6) CARE-GNN [16]: graph neural network for fraud detection with enhanced aggregation modules against fraud camouflages. 7) PC-GNN [29]: graph neural network addressing label imbalance in fraud detection. 8) GTAN [43]: gated temporal attention network for credit-card and other fraud detection tasks. 9) EPR-GNN: the graph neural network used in SecureFD, also serving as a local detection baseline.

We also compare SecureFD with the *Oracle* solution, where all data parties combine their data in plaintext without privacy protection and use the baseline with the highest AUC. This serves as a reference to analyze the secure collaborative detection.

Implementation.[2] We build the SecureFD prototype using the 2-out-of-4 secret-sharing MPC system PrivPy [28]. Each secure multi-party computation job has four computation services, scattered across a Kubernetes cluster with machines featuring Intel Xeon Gold 5218R 2.10GHz CPUs and 10 Gbps network bandwidth. Each secure computation service runs in a Kubernetes pod with a limit of 8 CPU cores and 32 GB of memory. We have sufficient computational resources to execute model training and prediction jobs

[2] Our source code will be available at https://github.com/LiuXin11235/SecureFD.git.

Table 1: Data statistics.

Dataset	#User nodes	#Resource nodes	#Edges	Fraud ratio	Training/Validation/Testing
S-FFSD	77,881	123,227	623,048	6.75%	10%-10%-80%
AmazonMovie	123,960	84,549	5,092,599	1.61%	10%-10%-80%
AmazonBook	603,667	407,253	26,694,123	1.66%	5%-5%-90%
AmazonAll	4,285,799	2,334,858	227,449,275	1.87%	5%-5%-90%

Table 2: The average test AUC (%) across all data parties and the standard deviation. Note that the *Oracle* solution combines all data parties' data in plaintext without privacy protection and serves only as a reference for analyzing secure collaborative detection. The best local method is underlined. And the best result excluding Oracle is in bold.

Colla.	Method	S-FFSD			AmazonMovie			AmazonBook		
		3 parties	4 parties	5 parties	3 parties	4 parties	5 parties	3 parties	4 parties	5 parties
No	M-Biz	61.7 ± 0.1	63.3 ± 2.3	62.4 ± 2.1	53.8 ± 1.3	53.3 ± 2.4	54.2 ± 1.2	52.1 ± 0.9	52.6 ± 0.7	52.9 ± 1.2
	M-Zoom	61.7 ± 0.1	63.3 ± 2.3	62.4 ± 2.1	55.8 ± 0.3	54.3 ± 0.4	54.4 ± 1.2	52.2 ± 1.0	52.9 ± 1.1	52.9 ± 1.1
	D-Cube	64.4 ± 5.5	64.3 ± 2.1	63.5 ± 3.7	54.7 ± 6.2	51.9 ± 1.6	52.6 ± 3.0	52.2 ± 1.3	54.0 ± 0.5	51.7 ± 0.8
	AnoGraph	62.8 ± 2.2	63.0 ± 1.9	62.6 ± 3.9	80.2 ± 0.4	80.4 ± 1.0	80.3 ± 0.6	79.8 ± 0.2	79.7 ± 0.3	79.8 ± 0.2
	GraphSAGE	68.5 ± 0.8	67.6 ± 3.3	67.9 ± 2.0	60.2 ± 1.8	59.9 ± 1.3	60.2 ± 2.3	55.9 ± 8.7	56.4 ± 5.9	57.2 ± 4.7
	CARE-GNN	68.9 ± 1.1	68.3 ± 3.9	67.2 ± 4.9	92.1 ± 1.6	91.3 ± 0.8	89.2 ± 5.6	82.6 ± 5.4	81.5 ± 5.1	81.0 ± 3.9
	PC-GNN	67.3 ± 2.7	69.4 ± 2.9	69.9 ± 3.4	91.8 ± 1.7	91.6 ± 1.6	90.6 ± 1.0	78.1 ± 2.1	78.7 ± 2.5	79.1 ± 1.4
	GTAN	75.2 ± 0.7	73.2 ± 1.4	74.8 ± 3.3	72.3 ± 12.4	78.5 ± 18.4	72.4 ± 12.8	56.8 ± 11.4	65.5 ± 15.9	62.7 ± 11.8
	EPR-GNN	73.5 ± 6.3	71.0 ± 8.2	71.9 ± 7.3	86.4 ± 2.2	87.1 ± 3.4	86.1 ± 1.8	88.5 ± 2.3	88.8 ± 1.2	88.0 ± 2.3
Secure	SecureFD	**82.2 ± 2.5**	**81.0 ± 3.0**	**79.6 ± 3.4**	**93.2 ± 1.0**	**92.6 ± 1.1**	**92.8 ± 1.2**	**91.6 ± 0.6**	**90.1 ± 0.7**	**91.4 ± 1.1**
Plain	Oracle	80.7 ± 2.0	81.8 ± 2.4	81.0 ± 3.0	93.1 ± 0.6	93.3 ± 0.2	93.5 ± 0.9	94.3 ± 0.1	94.4 ± 0.3	94.3 ± 0.2

Table 3: SecureFD training time on MPC in minutes.

Dataset	# Users	3 parties				4 parties				5 parties			
		Layer 1	Layer 2	MLPs	Overall	Layer 1	Layer 2	MLPs	Overall	Layer 1	Layer 2	MLPs	Overall
AmazonMovie	124K	0.0	0.6	2.7	3.3	0.0	0.4	2.7	3.1	0.0	0.3	2.6	3.0
AmazonBook	604K	0.1	2.2	3.7	6.0	0.1	1.9	3.7	5.7	0.1	1.8	3.7	5.6
AmazonAll	4,286K	0.4	20.0	9.5	29.9	0.6	15.1	9.5	25.1	0.6	12.3	9.3	22.3

simultaneously for all data parties. The current implementation of SecureFD uses PrivPy, but we want to emphasize that SecureFD can operate with any secret-sharing based MPC system providing the basic operations listed in Sec.5.

For the hyper-parameters of SecureFD, the hidden embedding size of the MLP layers in EPR-GNN is set to 48. We use the Adam optimizer to update the weights, with 100 training epochs, L2 regularization of 5e-4, and a learning rate of 0.01. For the pruning hyper-parameters, the user pruning ratio β_1 is 0.7, and the resource node sharing ratio β_2 is 0.01.

The graphs constructed for S-FFSD have four resource node types: sources, targets, locations, and types [43]. Graphs for AmazonMovie, AmazonBook, and AmazonAll have three resource node types: products, timestamps, and the concatenation of timestamps and review scores [29]. Each resource node type yields an edge type. The node features contain two parts: 1) user statistical features; and 2) one-hot encoding indicating the node type. For other baselines, we transform the graphs appropriately to their input formats.

6.2 Detection Performance

Table 2 shows the average AUC and standard deviation across all data parties. Excluding the Oracle solution, which is impractical in reality, SecureFD consistently performs the best across all datasets, regardless of the number of data parties. Additionally, SecureFD achieves performance that is close to the Oracle solution. The latter has the highest AUC (except in the 3-party scenarios of S-FFSD and AmazonMovie) and the lowest standard deviation, indicating that plain collaboration is the most robust.

Second, the standard deviation of SecureFD is also reasonably low, indicating that all data parties benefit stably from privately sharing their data. In the experiment, we partition the original data unequally into three to five parties, with some parties owning more data and some less. Even if one party believes it already has enough data (for example, half of the global data for party 3 in the three-party scenario), it can still benefit from collaboration.

In terms of other metrics including precision and recall, SecureFD is also better than non-collaborated baselines on all datasets if we

Table 4: Speed-up of EPR-GNN compared to GCN in the same MPC environment. The table shows the overall MPC training time in minutes in the three-party scenario.

Dataset	# Users	GCN	EPR-GNN	Speed-up
AmazonMovie	124K	43.3	3.3	13.1×
AmazonBook	604K	148.5	6.0	24.8×
AmazonAll	4,286K	924.7	29.9	30.9×

predict the top suspicious users as fraud. Here we omit the detection results on the AmazonAll dataset in Table 2 since the user-user graph required by baselines PC-GNN, CARE-GNN and GTAN are too dense to fit into memory. Nevertheless, when there are 4 or 5 parties, SecureFD still outperforms the non-collaborative EPR-GNN by 0.5% in the average test AUC.

6.3 Scalability Analysis

Here we investigate the scalability of SecureFD in terms of secure computation time on MPC. Table 3 presents the scenarios with three to five parties on AmazonMovie, AmazonBook, and AmazonAll datasets as they are more consistent. Overall, the entire detection design enables reasonable scalability on current hardware. It takes less than an hour to process more than four million users and just a few minutes to handle fewer than a million users. Comparing the runtime in Table 3 with respect to the number of parties, more data parties means each party has less data, resulting in shorter runtime, especially on the largest AmazonAll dataset.

To demonstrate that EPR-GNN is an MPC-friendly graph neural network, we implement GCN [25] also on PrivPy. It adopts the same oblivious node embedding sharing algorithm described in Sec.4.4 and local acceleration optimizations in Sec.4.6. As shown in Table 4, EPR-GNN is significantly faster than GCN (13.1× on AmazonMovie, 24.8× on AmazonBook, and 30.9× on AmazonAll). As the number of users increases, the efficiency advantage becomes more obvious. Because other graph neural networks, such as PC-GNN and GTAN, are more complex than GCN, implementing their secure collaborative versions would result in even longer running time, making them less efficient to handle large scale graph data.

In Table 5, we analyze the oblivious node embedding sharing by comparing the faster solution (Algorithm 2) to the naive one. As the number of users grows, the speed-up becomes more significant, from 1.5× on 124K users to 45.7× on 4,286K users. Additionally, Table 5 shows the runtime with the resource sharing ratio $\beta_2 = 0.01$. A larger β_2 would contrast the speed-up more significantly.

6.4 Ablation Study

Finally, we study how local detection contributes to detection accuracy and scalability on the AmazonBook dataset. In Figure 3, as the pruning ratio increases, the AUC initially drops slightly and then increases because noise irrelevant to detecting fraud is removed. However, the ratio should not exceed 0.9 to ensure unknown fraud users are kept for collaborative detection. For the running time, it always decreases as the pruning ratio increases. Our choice of 0.7 turns out to be a good trade-off between accuracy and scalability.

Table 5: Speed-up of oblivious node embedding sharing in SecureFD. The table shows the MPC training time of layer 1 in minutes in the three-party scenario.

Dataset	# Users	Naive	Faster	Speed-up
AmazonMovie	124K	0.06	0.04	1.5×
AmazonBook	604K	0.51	0.09	5.7×
AmazonAll	4,286K	19.18	0.42	45.7×

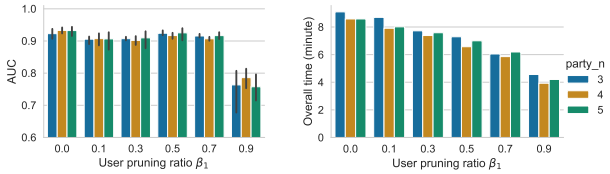

Figure 3: Influence of local user pruning on detection AUC and overall training time on the AmazonBook dataset.

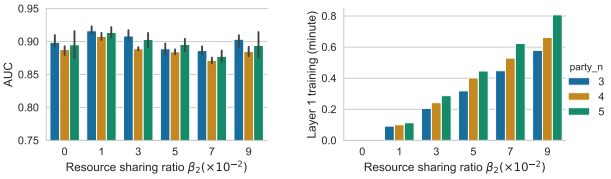

Figure 4: Influence of resource sharing ratio on detection AUC and layer 1 training time on the AmazonBook dataset.

As shown in Figure 4, a smaller resource sharing ratio means less time to share the layer 1 hidden embedding of resource nodes. Additionally, a reasonably small sharing ratio helps to select only the crucial resource nodes that fraudsters use. Consequently, the AUC achieves its highest value when the sharing ratio is 0.01.

7 Conclusion

Fraud detection requires industry-wide collaboration, as fraudsters often attack multiple companies using the same resources. In this work, we propose a method for multiple parties to collaboratively train a graph neural network detector, achieving better performance. By integrating MPC-efficient models, data structures, designed protocols and local detection pruning, our approach can scale to millions of users. SecureFD opens many opportunities for future work. Scalability can be further improved through multi-round collaboration and dynamic sharing levels. Besides, our experiments involve up to five data parties, and managing a larger number (e.g., fifty) with skewed data distribution poses additional challenges.

Acknowledgements

This work is supported by the National Key Research and Development Program of China under Grant 2023YFC3304800. Additionally, the authors would like to thank Dr. Ling Huang and Dr. Yitao Duan for their continuing support, and Hu Li and Lu Chen for their assistance in deploying the PrivPy environment.

References

[1] Panos Alexopoulos, Kostas Kafentzis, Xanthi Benetou, Tassos Tagaris, and Panos Georgolios. 2008. Towards a Generic Fraud Ontology in e-Government. In *International Conference on E-Business*. 269–276.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.

[3] Muhammad Ajmal Azad, Samiran Bag, Shazia Tabassum, and Feng Hao. 2020. Privy: Privacy Preserving Collaboration Across Multiple Service Providers to Combat Telecom Spams. *IEEE Transactions on Emerging Topics in Computing* 8, 2 (2020), 313–327. https://doi.org/10.1109/TETC.2017.2771251

[4] Yikun Ban, Xin Liu, Yitao Duan, Xue Liu, and Wei Xu. 2019. No Place to Hide: Catching Fraudulent Entities in Tensors. In *Proceedings of The Web Conference 2019*.

[5] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of Garbled Circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 784–796.

[6] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior In Social Networks. In *WWW*. 119–130.

[7] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, Neil Shah, Philip S. Yu, and Bryan Hooi. 2023. Sketch-Based Anomaly Detection in Streaming Graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 93–104.

[8] Richard J. Bolton and David J. Hand. 2002. Statistical Fraud Detection: A Review. *Statist. Sci.* 17, 3 (2002), 235–249.

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Breaking the Circuit Size Barrier for Secure Computation Under DDH. In *CRYPTO 2016*.

[10] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[11] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 477–488.

[12] Bo Chen, Calvin Hawkins, Kasra Yazdani, and Matthew Hale. 2021. Edge Differential Privacy for Algebraic Connectivity of Graphs. In *2021 60th IEEE Conference on Decision and Control (CDC)*. 2764–2769.

[13] Chaochao Chen, Jun Zhou, L. xilinx Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Xiaoxi Ji, Alex X. Liu, Hao Wang, and Cheng Hong. 2021. When Homomorphic Encryption Marries Secret Sharing: Secure Large-Scale Sparse Logistic Regression and Applications in Risk Control. In *27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

[14] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. SecureBoost: A Lossless Federated Learning Framework. *IEEE Intelligent Systems* 36, 6 (2021), 87–98.

[15] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*. https://openreview.net/forum?id=n6jl7fLxrP

[16] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020).

[17] Yingtong Dou, Guixiang Ma, Philip S. Yu, and Sihong Xie. 2020. Robust Spammer Detection by Nash Reinforcement Learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 924–933.

[18] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. 2022. NFGen: Automatic Non-linear Function Evaluation Code Generator for General-purpose MPC Platforms. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*.

[19] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC '09*.

[20] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. *IACR Cryptology ePrint Archive* 2014 (2014), 106.

[21] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. 2012. Practically Efficient Multi-Party Sorting Protocols from Comparison Sort Algorithms. In *15th International Conference on Information Security and Cryptology*. 202–216.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[23] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 895–904.

[24] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. In *IACR Cryptology ePrint Archive*.

[25] Thomas N. Kipf and Max Welling. 2007. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the International Conference on Learning Representations*.

[26] Peeter Laud. 2015. Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees. *Proceedings on Privacy Enhancing Technologies* 2015 (2015), 188 – 205.

[27] Pan Li, Eli Chien, and Olgica Milenkovic. 2019. Optimizing generalized PageRank methods for seed-expansion community detection. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 1050, 12 pages.

[28] Yi Li and Wei Xu. 2019. PrivPy: General and Scalable Privacy-Preserving Data Mining. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[29] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection. In *Proceedings of the Web Conference 2021*. 3168–3177.

[30] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. In *CIKM*. ACM, 2077–2085.

[31] Wenjie Lu, Shohei Kawasaki, and Jun Sakuma. 2016. Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data. *IACR Cryptology ePrint Archive* 2016 (2016), 1163.

[32] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[33] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *EMNLP*.

[34] U.S. Department of Justice Federal Bureau of Investigation. 2015. 2015 Internet Crime Report. https://pdf.ic3.gov/2015_IC3Report.pdf.

[35] Sofya Raskhodnikova and Adam Smith. 2016. *Differentially Private Analysis of Graphs*. Springer New York, New York, NY, 543–547. https://doi.org/10.1007/978-1-4939-2864-4_549

[36] Kijung Shin, Bryan Hooi, and Christo Faloutsos. 2018. Fast, Accurate, and Flexible Algorithms for Dense Subtensor Mining. *TKDD* 12, 3 (2018), 28.

[37] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. D-Cube: Dense-Block Detection in Terabyte-Scale Tensors. In *WSDM*. 681–689.

[38] Erez Shmueli and Tamir Tassa. 2017. Secure Multi-Party Protocols for Item-Based Collaborative Filtering. In *RecSys '17*.

[39] Kurt Thomas, Danny Huang, David Wang, Elie Bursztein, Chris Grier, Thomas J. Holt, Christopher Kruegel, Damon McCoy, Stefan Savage, and Giovanni Vigna. 2015. Framing Dependencies Introduced by Underground Commoditization. In *Workshop on the Economics of Information Security*.

[40] Tian Tian, Jun Zhu, Fen Xia, Xin Zhuang, and Tong Zhang. 2015. Crowd fraud detection in internet advertising. In *WWW*. 1100–1110.

[41] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proceedings on Privacy Enhancing Technologies* 2019 (2019), 26 – 49.

[42] Xueyu Wu, Zhuoran Ji, and Cho-Li Wang. 2023. Embedding Communication for Federated Graph Neural Networks with Privacy Guarantees. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. 305–315. https://doi.org/10.1109/ICDCS57875.2023.00029

[43] Sheng Xiang, Mingzhi Zhu, Dawei Cheng, Enxia Li, Ruihui Zhao, Yi Ouyang, Ling Chen, and Yefeng Zheng. 2023. Semi-Supervised Credit Card Fraud Detection via Attribute-Driven Graph Representation. In *The Annual AAAI Conference on Artificial Intelligence*.

[44] Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated Graph Classification over Non-IID Graphs. *Advances in Neural Information Processing Systems* 34 (2021), 18839–18852.

[45] Ibrahim Yakut and Huseyin Polat. 2012. Arbitrarily Distributed Data-Based Recommendations with Privacy. *Data & Knowledge Engineering* 72 (2012), 239–256.

[46] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. 2012. Analyzing Spammers' Social Networks for Fun and Profit: A Case Study of Cyber Criminal Ecosystem on Twitter. In *21st International Conference on World Wide Web*.

[47] Han Zhang, Wenhao Zheng, Charley Chen, Kevin Gao, Yao Hu, Ling Huang, and Wei Xu. 2020. Modeling Heterogeneous Statistical Patterns in High-dimensional Data by Adversarial Distributions: An Unsupervised Generative Framework. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. 1389–1399. https://doi.org/10.1145/3366423.3380213

[48] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph Federated Learning with Missing Neighbor Generation. *Advances in Neural Information Processing Systems* 34 (2021), 6671–6682.

[49] Jun Zhou, Chaochao Chen, Longfei Zheng, Huiwen Wu, Jia Wu, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2022. Vertically Federated Graph Neural Network for Privacy-Preserving Node Classification. In *Proceedings of the International Joint Conference on Artificial Intelligence 2022*.