Privacy-Preserving Outlier Detection with High Efficiency over Distributed Datasets

Guanghong Lu, Chunhui Duan*, Guohao Zhou, Xuan Ding, Yunhao Liu
School of Software and BNRist, Tsinghua University, China
*Corresponding author
Email: {lugh60, duanch09, zhough19, dingx04, yunhaoliu}@gmail.com

***Abstract*—The ability to detect outliers is crucial in data mining, with widespread usage in many fields, including fraud detection, malicious behavior monitoring, health diagnosis, *etc*. With the tremendous volume of data becoming more distributed than ever, global outlier detection for a group of distributed datasets is particularly desirable. In this work, we propose PIF (Privacy-preserving Isolation Forest), which can detect outliers for multiple distributed data providers with high efficiency and accuracy while giving certain security guarantees. To achieve the goal, PIF makes an innovative improvement to the traditional iForest algorithm, enabling it in distributed environments. With a series of carefully-designed algorithms, each participating party collaborates to build an ensemble of isolation trees efficiently without disclosing sensitive information of data. Besides, to deal with complicated real-world scenarios where different kinds of partitioned data are involved, we propose a comprehensive schema that can work for both horizontally and vertically partitioned data models. We have implemented our method and evaluated it with extensive experiments. It is demonstrated that PIF can achieve comparable AUC to existing iForest on average and maintains a linear time complexity without privacy violation.**

***Index Terms*—Outlier detection, Privacy-preserving, Distributed data, PIF**

I. Introduction

The volume of data is practically exploding by the day as more and more information is collected and stored. But due to a variety of reasons, including but not limited to equipment anomaly, noise interference, malicious activity, human error, the presence of outliers is usually inevitable in the acquired data. Generally speaking, an outlier is described as ‘an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data’ [1], [2]. Outliers are considered important because they often indicate rare but significant events and thus can promote critical actions to be taken in a wide range of applications. For example, anomalous behavior in credit card transactions could imply fraudulent activities such as stolen cards; on a factory production line, outliers in specific features of products may pinpoint manufacturing faults; and anomalies in an MRI image may indicate the existence of a malignant tumor. Consequently, outlier detection (also termed as anomaly detection) has always been a very crucial task and forms an essential research branch in data mining, with widespread use in domains such as activity monitoring [3], fraud detection, health diagnosis [4], *etc*.

There are numerous types of outlier detection approaches put forward in recent years, which can be categorized into supervised methods and unsupervised ones, based on whether we have a good understanding, or sufficient labeled training data, for both the normal and abnormal data points. As in most practical applications, there may be no training data available, here in this work, we focus on the unsupervised algorithms without assuming prior knowledge about the acquired data. State-of-the-art techniques mainly include: 1) distance-based algorithms, which view a data point as an outlier if it has a large deviation from its nearby neighbors [5], [6]; 2) density-based algorithms, which assumes the density around an outlier is considerably different to that around its neighbors [7], [8], [9], [10], [11]; 3) clustering-based algorithms, whose core principle is that outliers are not within or nearby any large or dense clusters [12], [13]. Nevertheless, these methods are typically designed to deal with centralized small-scale datasets because of their high computation complexity, or in other words, the data which needs to be analyzed (or mined) should be centrally stored as a single repository of small size.

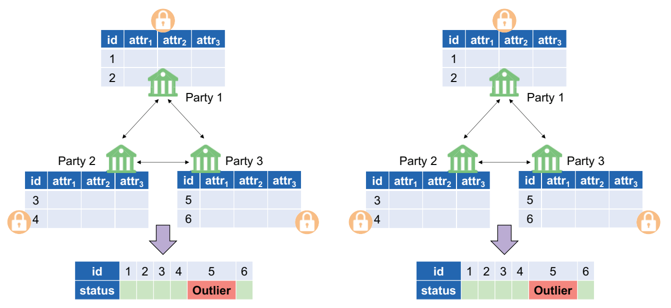
As a matter of fact, the growing amount of data itself is inherently becoming more distributed among multiple sources instead of centralized in a single source nowadays, which in turn requires the outlier detection task to be performed in a distributed way. For instance, in the medical field, patients’ medical data is often not collected and stored in a single institution, but scattered across different sites, such as various hospitals, health care centers, and pharmacies. Hence, the information in one institution is usually incomplete, and to detect outliers (*i.e.*, representing patients in abnormal physical conditions), it is necessary to comprehensively analyze the distributed information from several datasets. One straightforward way to achieve this is to integrate all the databases under one roof, but this strategy is usually unfeasible in practice, because the integration procedure of the massive multi-sourced data may not only cause prohibitively high cost and inefficiency, but also raise privacy concerns and result in the leak of sensitive information. For a variety of practical outlier detection applications, *privacy preservation* is a fundamental need where anomalies should be successfully detected while giving formal guarantees on the amount of private information disclosed. Recalling the aforementioned example in the medical domain, it is desired to efficiently identify the global outliers across multiple medical institutions without revealing any private information (patients' personal profile) to the untrusted.

In this work, we propose PIF (abbreviation for Privacy-preserving Isolation Forest), a *privacy-preserving* and *efficient* outlier detection approach with *high accuracy* for *distributed* data sources. To accomplish the goal, we design a novel structure based on traditional Isolation Forest (iForest) technique [14], which works by first constructing an ensemble of trees (called as iTree or Isolation Tree) with the given data and then detecting abnormal instances as those averagely isolated closer to the roots of the trees. Although the sub-sampling ability of iForest makes it applicable to extremely large data size with extraordinary detection performance, it can only handle centralized databases. Here in our work, we try to improve the traditional iForest for distributed scenarios taking privacy into consideration. By tailoring a series of secure algorithms regarding sub-tree construction and merging, PIF can work over multiple distributed datasets without privacy breach while maintaining high efficiency and accuracy. Besides, in actual applications, data can be distributed or partitioned among multiple parties either in a horizontal or vertical manner. We elaborately custom our method for both scenarios. The basic idea of PIF is sketched below. Each distributed party first generates a Privacy-preserving Isolation Sub-Tree (PIST) using their own data while considering the distribution of instances and attributes among all the parties. Then all parties collaborate to merge their PISTs into one Privacy-preserving Isolation Tree (PIT). Finally, with an ensemble of PITs, PIF can predict global outliers as those instances which have short average path lengths on these PITs.

Contributions. In summary, this paper makes the following key contributions:

- We propose an innovative approach to detect global outliers for distributed data sources. With multiple carefully-designed solutions, our method can insure data privacy without disclosing sensitive information.
- We extend the conventional iForest structure to enable it in distributed scenarios. A series of algorithms are devised to achieve high efficiency and accuracy with affordable computation and communication overhead.
- As a comprehensive outlier detection schema, PIF applies to both horizontally partitioned and vertically partitioned data models. The accompanying challenges are well analyzed and addressed.
- We have implemented and evaluated PIF with extensive experiments over various datasets. The final AUC is comparable to traditional iForest, and the actual CPU time goes linearly with the number of parties, which demonstrates the practicality and effectiveness of our design.

Roadmap. The remainder of the paper is organized as follows. We introduce background knowledge about our design in Section II. The technical details of PIF concerning horizontally partitioned and vertically partitioned data models are elaborated in Section III and Section IV respectively. We



(a) Horizontal partitioning (b) Vertical partitioning

Fig. 1: Illustration of distributed datasets

present the implementation and evaluation of our method in Section V. We review related work in Section VI, and finally conclude this paper in Section VII.

II. Preliminary

This section begins with background knowledge about data partitioning, followed by a brief introduction to the Isolation Forest and an explanation of the security model we adopt.

A. Data Partitioning

The data in our scene is distributed or partitioned across several parties. There are mainly two ways to partition the data: horizontal partitioning and vertical partitioning [15], as depicted in Fig. 1. In both of these cases, we assume that there are total k different parties $P_1, P_2, \ldots, P_k$ that hold data, and m attributes and n instances in the data.

Horizontally partitioned data. Here multiple parties hold the same set of attributes about different instances. Each party collects information about all m attributes $A_1, A_2, \ldots, A_m$, and there is no intersection of instances between any two parties. Formally, party P_i has n_i instances, such that $\sum_{i=1}^{k} n_i = n$. A typical example of horizontal partitioning is that banks store similar credit card transaction information but for different customers.

Vertically partitioned data. Here each party holds a different subset of the attributes for instances in the data store. Party P_i collects information about m_i ($m_i < m$) attributes, $A_{i,1}, A_{i,2}, \ldots, A_{i,m_i}$, for all the instances, and there is no intersection of attributes between any two parties. The total number of attributes, $\sum_{i=1}^{k} m_i = m$. For example, in a sensor network monitoring a certain area, there could be multiple types of sensors reporting entirely different information about environmental parameters (such as temperature, humidity, illumination). In case that one attribute exists in more than one party, we only use one copy of them.

B. Isolation Forest

Isolation Tree. The term *isolation* means 'separating an instance from the rest of the instances'. Since outliers have two potential characteristics: 'few and different', they are more susceptible to isolation than normal points. The work [14] uses a binary tree structure to realize the isolation, which

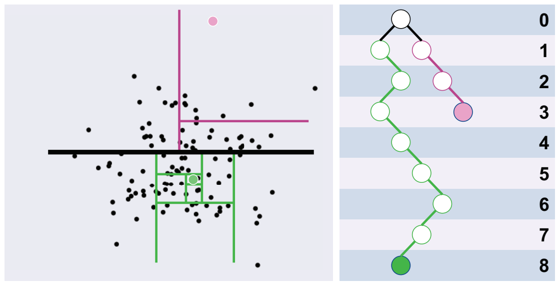


Fig. 2: Illustration of iTree

is called iTree (or Isolation Tree). It first samples a subset of data $X = \{x_1, x_2, \ldots, x_\psi\}$ ($\psi < n$) randomly from the original dataset. Then an iTree is generated from top to bottom, where the root contains all the data samples X. Each node is recursively divided by randomly selecting an attribute and a split value between the minimum and maximum values of this attribute, until either: a) the tree reaches a height limit, b) $|X| = 1$, or c) all instances left in the node have equal values. Every instance would be finally isolated to a leaf node when an iTree is fully grown. Fig. 2 gives an illustrative example of the random partitioning process. Normal points are denoted in green and black while the anomaly is in pink. It is clear from the figure that the path length (which represents the number of partitions) of anomalies is far shorter than normal instances.

Anomaly score. Each iTree can be seen as a weak classifier, which can further compose the iForest (also called as Isolation Forest) based on ensemble learning to achieve better results. Define $h(x)$ of an instance x as the path length on an iTree, which is measured by the number of edges x traverses from the root node until terminated at a leaf node. Formally, to reflect the degree of anomaly for an instance x, an **anomaly score** s is computed as expressed below [14]:

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}}, \tag{1}$$

where $E(h(x))$ denotes the average of $h(x)$ from a collection of iTrees (*i.e.*, an iForest). $c(\phi)$ is used to normalize $h(x)$ and defines the average of $h(x)$ given ψ. Namely,

$$c(\psi) = 2H(\psi - 1) - 2(\psi - 1)/\psi, \tag{2}$$

where $H(i)$ is the harmonic number which can be estimated by $\ln(i) + 0.5772156649$ (Euler's constant). From Eqn. 1 we can observe that: when $E(h(x)) \to c(\psi), s \to 0.5$; when $E(h(x)) \to 0, s \to 1$; and when $E(h(x)) \to \psi - 1, s \to 0$. This means that if the anomaly score of an instance approaches 1, it is likely to be an outlier.

C. Security Model

We are studying how to detect outliers in a distributed environment involving multiple parties without privacy leakage, so security issues should be considered. In this part, we will introduce our security model and the necessary privacy guarantees that we can provide.

Model assumptions. There are essentially two types of adversaries in Secure Multi-Party Computing (MPC): malicious adversaries and semi-honest adversaries. Malicious adversaries can deviate from the protocol instructions and follow an arbitrary strategy. For example, they may provide simulated or malicious data, disclose information to others, and collude with each other. A semi-honest adversary follows the protocol instructions, but it is curious and may record all intermediate results in the cooperation and try to get extra information based on these intermediate results. The semi-honest adversary model usually comes at an affordable cost and is more commonly used in MPC. In this paper, we assume that all k parties($k \geq 3$) are semi-honest and non-colluded, which means they will not share any information that they are not explicitly instructed to share with other parties. These are standard assumptions in related literature and apply to many real-life situations.

Privacy consideration. Here, we consider not only the privacy of users (original data), but also the privacy of participating parties (data characteristics).

a) The privacy of original data. This means any attribute value of an arbitrary instance (*e.g.*, a patient's 'age' value in the medical information database) cannot be disclosed, and even the valid value range is nearly impossible to be inferred. With such consideration, we can completely protect the privacy of users.

b) The privacy of data characteristics. This means that other parties cannot learn the properties of data (*e.g.*, the age distribution of patients in a hospital) from the related intermediate results, and even the valid value range of these properties can hardly be inferred. With such consideration, we can completely protect the privacy of parties.

III. Solution for Horizontally Partitioned Data

In this section, we will introduce our privacy-preserving outlier detection solution for horizontally distributed data.

A. Limitations of Prior Work

The key rationale of iForest is to construct a collection of iTrees each time with a small size of sampled data. And subsampling is conducted by the *random* selection of instances without replacement. Prior art [16] leverages the concept of iForest to enable secure outlier detection for distributed data. However, it is flawed in technical details, since it allows parties to generate their own trees independently without considering the data distribution among parties. Thus the premise of random sampling is broken, which violates the principle of the iForest algorithm and may result in detection errors. Besides, it can only deal with horizontally partitioned data and needs the participation of another two parties to complete the security calculation. Moreover, the encrypted numbers are mostly identical (*e.g.*, most of them are ciphertexts of 0s), making the ciphertexts easy to guess, which results in a low level of security.

In contrast, our work offers privacy-preserving solutions to detect outliers for both horizontally and vertically partitioned datasets with high accuracy and efficiency, without requiring the participation of third parties.

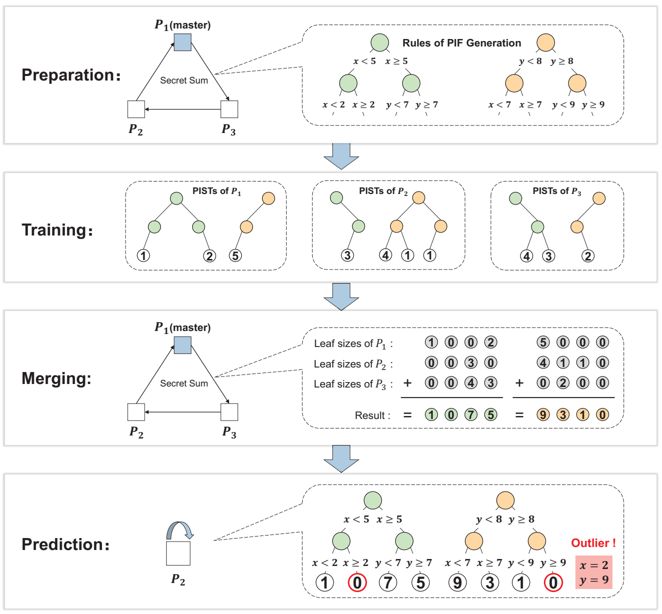


Fig. 3: Solution for horizontally partitioned data

B. Design Overview

PIF involves four stages. a) The *Preparation stage* requires all parties to cooperate to determine necessary parameters, including sub-samples of the global data, attributes and split values of partitions, *etc*. b) In the *training stage*, each party independently builds its Privacy-preserving Isolation Sub-Trees (PISTs) with the sampled training data. c) In the *merging stage*, all parties work together to merge their PISTs into a collection of Privacy-preserving Isolation Trees (PITs), *i.e.*, a Privacy-preserving Isolation Forest, or PIF for short. d) Finally, in the *prediction stage*, we can compute an anomaly score for each test instance by passing it through the PIF.

More technical details about the above steps will be given in the following parts.

C. PIF Workflow

Preparation. We decompose the building of a PIT into several sub-trees each generated by a distributed party. A collection of PITs form a PIF. There are some key parameters during PIF construction. Let t denote the *number of trees*, and ψ be the global *sub-sampling size* for each tree. The training samples should come from all the parties. Therefore, we raise our first question: *how to implement secure random sampling among all the parties without disclosing information on the data size of each party?* We utilize the Secure Sum algorithm [17] to achieve this. Suppose there are k parties $P_1, P_2, \ldots, P_k (k \geq 3)$ with data sizes of $n_1, n_2, \ldots, n_k$ $(n_1 + n_2 + \cdots + n_k = n)$ respectively. Let P_1 be a **master** among all the parties (the master can be elected through common consensus algorithms like Raft [18]) to lead the sampling task. The master first generates a random number r such that $r + n_1$ drops within a reasonable interval, and passes $r + n_1$ to party P_2. Then similarly P_2 generates n_2 and passes $r + n_1 + n_2$ to P_3. This process repeats until all the parties are involved. Afterward, the master can obtain the value of $r + \sum_{i=1}^{k} n_i$, which directly translates to n. Once the master knows n, it further broadcasts to all the other parties. Finally, for each sub-tree, P_i randomly samples a certain portion (*i.e.*, $\frac{n_i}{n}\psi$) of its own data. Note that when $r + n_1$ is random within a reasonable interval and the parties do not collude, they can never guess the output of other parties.

Once all parties finish their sub-sampling processes, the rationale of training sub-trees is to recursively partition the given training sets until instances are isolated or a specific *tree height limit* l is reached. Therefore, parties also need to jointly determine a series of data partitioning regulations including attributes and split values. Fig. 3 illustrates an example, where green and yellow represent different trees built with different training sets. l is automatically set as: $l = \lceil \log_2 \psi \rceil$ [14].

The attribute for each partitioning is randomly selected by the master. After the attribute is determined, each party P_i randomly chooses a *candidate* split value v_i between the minimum and maximum values of the attribute with its own data, and encrypts v_i with the master's public key to get $\texttt{Enc}(v_i)$. Then we leverage the **reservoir sampling** algorithm [19] to transmit the split values for security purpose. The master first passes $\texttt{Enc}(v_1)$ to P_2. The probability that P_2 chooses to pass $\texttt{Enc}(v_1)$ or $\texttt{Enc}(v_2)$ to P_3 is both $\frac{1}{2}$. More generally, suppose P_i receives $\texttt{Enc}(v)$ from P_{i-1}, then it has a probability of $\frac{i-1}{i}$ to transmit $\texttt{Enc}(v)$ to P_{i+1} and probability of $\frac{1}{i}$ to transmit $\texttt{Enc}(v_i)$. At last, the master decrypts the message it receives to get the *real* split values. It is easy to prove that all parties' split values have the same possibility to be selected. A party can learn nothing about other parties' candidate values, and the master also does not know which party the selected value comes from. The chosen split values are also broadcast from the master to any other parties to ensure the consistency of sub-tree structures.

Training. With the system parameters and partitioning regulations specified in the preparation stage, each party can independently train its own PISTs, as illustrated in Fig. 3. Our PIST is similar to the iTree in the iForest algorithm, but differs mainly in the following two aspects:

- *Sub-sampling size.* In an iTree, the number of samples equals ψ. But in our design, the sampling size ψ_i of each PIST equals $\frac{n_i}{n}\psi$ while the global sub-sampling size of all parties $\sum_{i=1}^{k} \psi_i$ sums to ψ.
- *Node termination condition.* In traditional iTree, a node would terminate partitioning if: a) the tree reaches a height limit, b) the current node contains only one sample, or c) all instances left in the node have equal values. However, since multiple parties train their own PISTs independently in our case, if we adopt the above strategy, PISTs would grow into quite diverse structures (*e.g.*, a leaf node in one PIST may correspond to an internal node in another PIST). To facilitate the later merging of several PISTs, *we require every node to keep partitioning until the tree height limit is reached, even if its instances are already isolated.* In this way, every leaf node would maintain the same height l.

Merging. Now that we get all parties' sub-trees, the next question is: *how to merge these sub-trees into a single PIT, which further composes a PIF?* Denote the p^{th} ($1 \leq p \leq t$) sub-tree built by party P_i as $T_{i,p}$. Note that the number of leaf nodes in each sub-tree could be variant with a maximum value of 2^l, so here for the sake of computation, we assume that there are total 2^l *virtual* leaf nodes at the last (a.k.a. l^{th}) level of each sub-tree. Let $w(i,p,q)$ be the size of the q^{th} ($1 \leq q \leq 2^l$) node, *i.e.*, number of instances contained. If a virtual node does not have any instance, we set its size as 0 ($w(i,p,q) = 0$). Then to obtain the p^{th} PIT, we try to merge all k parties' sub-trees by adding corresponding nodes at the last level. Let $u(p,q)$ be the merged value. We have,

$$u(p,q) = \sum_{i=1}^{k} w(i,p,q). \tag{3}$$

It is worth mentioning that parties are forbidden to directly share their own PISTs to each other for privacy purposes, as certain characteristics of data could be learned with the leaked PISTs. Under this premise, the problem we need to solve now is how to sum up all $w(i,p,q)$ in Eqn. 3 securely and efficiently?

We describe our method in matrix form. For any party P_i, it keeps a matrix $\mathbf{W_i}$, where each row representing a specific PIST and each column in one row denotes a leaf node. The element $\mathbf{W_i}[p,q]$ at the p^{th} row and q^{th} column is set to $w(i,p,q)$. First of all, the master (*i.e.*, P_1) generates a two-dimensional disturbance matrix $\mathbf{R}$ such that each element in $\mathbf{R} + \mathbf{W_1}$ is random within a reasonable interval (*e.g.*, $[0, 255]$ for $\psi = 256$). Then, it sends $\mathbf{R} + \mathbf{W_1}$ to party P_2. Thus P_2 can guess neither the real value of $\mathbf{W_1}$ nor its data characteristics. Generally, each party P_i repeats to send $(\mathbf{R} + \sum_{j=1}^{i} \mathbf{W_j}) \bmod \psi$ to the next party $P_{(i+1)\%k}$ until the data is sent back to the master. After that, the master can perform $(\mathbf{R} + \sum_{j=1}^{k} \mathbf{W_j} - \mathbf{R}) \bmod \psi$ to acquire $\sum_{j=1}^{i} \mathbf{W_j}$ (also denoted by $\mathbf{U}$), which directly corresponds to the final PIF. The master would also broadcast $\mathbf{U}$ to all other parties. In the end, every party would obtain a copy of $\mathbf{U}$ which is crucial for prediction in the next stage. Note that after broadcasting the PIF, each party would learn some specific characteristics of the global sampled data (*e.g.*, there are 20 instances whose ages are over 30). However, it can not relate the data characteristics to any single party.

Space complexity. The memory requirement of our method remains at a low level for each party. The intermediate results transmitted among parties are matrices with sizes of $t \times 2^l$ whose elements are natural numbers. According to the evaluation of iForest [20], Area Under Curve (AUC) is near optimal when sub-sampling size $\psi = 256$, and path lengths usually converge well before $t = 100$ [14]. Therefore, we can use an 8-bit binary to store each element in a matrix. Then the size of intermediate data that a party needs to transmit in our algorithm is about $25\,\mathrm{KiB}$, which is quite acceptable.

Prediction. Given a test instance x, to predict how likely it is an outlier, we pass it through each PIT in a PIF, to derive a

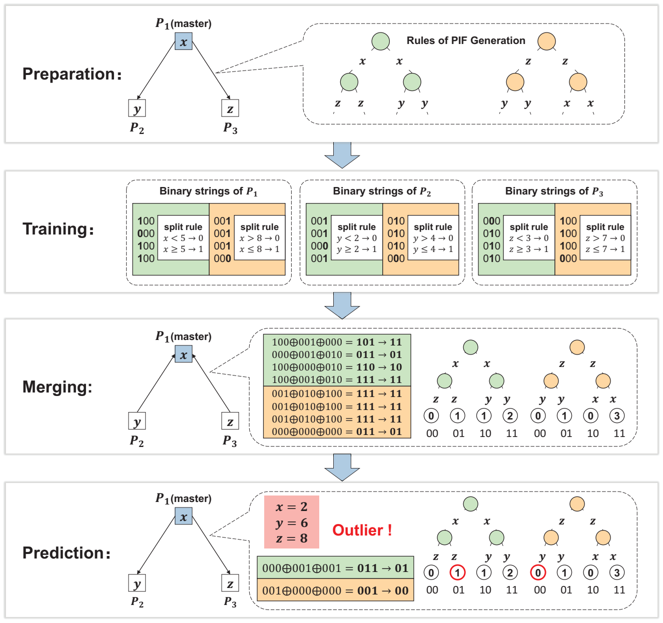


Fig. 4: Solution for vertically partitioned data

path length $h(x)$. Recall that in the prior work iForest, $h(x)$ is computed as the number of edges e an instance traverses from the root to a leaf node plus an adjustment $c(\mathsf{size})$, *i.e.*, $h(x) = e + c(\mathsf{size})$, where size denotes the size of the leaf node, and $c(\cdot)$ is defined in Eqn. 2. The adjustment accounts for an unbuilt sub-tree beyond the tree height limit. However in our case, any instance would reach the tree height limit l due to our tree building strategy. So we define $h(x)$ as $h(x) = l + c(\mathsf{size})$, and if the size of a leaf node $\mathsf{size} \leq 1$, we set $c(\mathsf{size}) = 0$. With the derived $h(t)$ for each PIT, an **anomaly score** $s(x, \psi)$ for the given instance can then be produced using Eqn. 1.

Note that the text instance may come from an arbitrary party. If a specific party wants to assess its own instances' anomaly, it can directly accomplish this with its PIF. Otherwise, if a party intends to make predictions on instances from a third party, the prediction results should be transmitted between the two parties in a secure way. There are many solutions in MPC, which will not be discussed in this work due to the page limit.

IV. Solution for Vertically Partitioned Data

When the data is vertically partitioned, each party collects information about m_i ($m_i < m$) attributes for all n instances, and there is no intersection of attributes between any two parties. Then parties collaborate to accomplish the goal of outlier detection without disclosing privacy.

Considering that the attributes of data owned by each party are not complete, hence parties are incapable of constructing their sub-trees independently. We try to deal with this problem from another point of view. Suppose that the attributes of partitions in the tree have been determined. Then if a party P_i happens to have certain attributes, it can partition its samples offline. An integrated tree can be built when all parties merge their partitioning results together. Similarly, we will introduce our solution in four stages: preparation, training, merging, and prediction, as shown in Fig. 4.

Preparation. Parties still need to elect a master at first. Since the instances owned by each party are the same, the master would conduct the sub-sampling task locally on behalf of other parties, and then sends the selected indexes of samples to others. It is reasonable to assume that the master knows the complete attributes (or the number of attributes at least) of all parties. Even if not, the master can request the attribute information from other parties respectively. Thereafter, the master randomly chooses the attributes for partitions in the tree, which is assumed to be a *complete binary tree* with height l. It is worth noting that the master does not specify the split values here.

Training. Since each party's attributes are incomplete and non-overlapping, instead of building a sub-tree, it only needs to maintain the partitioning that is related to its attributes. For ease of description, we first number the nodes in the final complete tree from the top level to bottom, and for each level, the number increases from left to right. Namely, denote the root node as a_1. Then the left child of a non-leaf node a_i is a_{2i}, and the right child is a_{2i-1}. The total number of non-leaf nodes equals $2^l - 1$.

According to the partitioning attributes determined in the preparation stage, each party independently chooses its own split values and split rules. During the training, for each instance, parties utilize a *binary string* str with a length of $2^l - 1$ to record their training results. str is initialized to be all '0's. If a party happens to hold an attribute b_i which corresponds to the node a_i, it rewrites the i^{th} element in the string: $\mathsf{str}[i]$ ='1' if the instance is split to the right child, or $\mathsf{str}[i]$='0' when split to the left child. As the example in Fig. 4 depicts, for the green tree, party P_2 holds attribute y and generates a split rule (split to the left child if $y < 2$ or to the right child if $y \geq 2$). Since y corresponds to the third node in the tree, then the binary string of an instance would be set to '00**1**' if the attribute satisfies $y \geq 2$ (*i.e.*, the instance is split to the right child). In this way, every instance is related to a binary string, which indicates how it is partitioned in the final tree.

Here note that to prevent the data characteristics from being disclosed, each party needs to randomly select an instance and then adopt its attribute value as the split value, instead of directly choosing a split value in the interval of $[\min, \max]$. Because, if a party chooses split values of its attributes from the value range $[\min, \max]$, the sizes of leaf nodes in PITs would easily be affected by the distribution of data. Thus other parties may infer certain data characteristics. For example, parties may deduce that the data is normally distributed if the sizes of leaf nodes are unbalanced, and even estimate the variance of data. Therefore, we require each party to randomly select an instance and use its attribute value for partitioning. In this way, the partitioning results would have no relation with the original data characteristics, such that the data privacy would not be exposed.

After training, each party gets a matrix $\mathbf{M_i}$, where $\mathbf{M_i}[p, q]$ denotes the binary string of the p^{th} sampled instance trained for the q^{th} tree. $\mathbf{M_i}$ is then transmitted to the master for the later merging stage.

Merging. The master XORs the received matrices from multiple parties, to obtain a collection of merged binary strings. Concretely, it computes $\mathbf{M_1}[p, q] \oplus \mathbf{M_2}[p, q] \oplus \cdots \oplus \mathbf{M_k}[p, q]$ (also called as *merged string*) for every possible p and q, like Fig. 4 shows. Remember that it is the number of instances contained in each leaf node that matters. For every instance in a tree, we can convert its merged string to a path traversed through the tree, which would finally terminate at a leaf node. By further counting the number of instances owned by each leaf node, we can construct a PIT (without split values) at last. An ensemble of PITs constitutes a PIF.

Security analysis. Each party would obtain a collection of PITs after merging. But from one PIT, a party can only learn the size of each leaf node without knowing the partitioning attributes and values for the internal nodes of other parties. To put it simply, a party only knows that the sampled instances are classified into several sets with known sizes, but has no idea of how these instances are classified by other parties.

Prediction. The PIF each party gets lacks the necessary split values in trees, so parties need to work together to perform the prediction task. Given a test instance, if a party wants to detect whether it is abnormal, it asks for help from all the other parties, which would generate the instance' binary strings respectively and send the results to the master. Then the master merges all the strings into a single one and converts it to a leaf node using the mechanism in the merging stage. An anomaly score can further be computed similarly as demonstrated in Section III-C. If a third party wants to know whether its instances are outliers, it can send the instances to any of our participating parties to finish the prediction.

V. Experimental Evaluation

In this section, we conduct performance evaluation of our approach and present the detailed experimental results.

A. Evaluation Methodology

We use 10 natural datasets widely used for evaluating outlier detection algorithms, including two biggest data subsets

TABLE I: Datasets properties

Datasets	n	m	Abnormal proportion
Http (KDDCUP99)	567497	3	0.4%
ForestCover	286048	10	0.9%
Smtp (KDDCUP99)	95156	3	0.03%
Shuttle	49097	9	7%
Mammography	11183	6	2%
Satellite	6435	36	32%
Pima	768	8	35%
Breastw	683	9	35%
Arrhythmia	452	274	15%
Ionosphere	351	32	36%

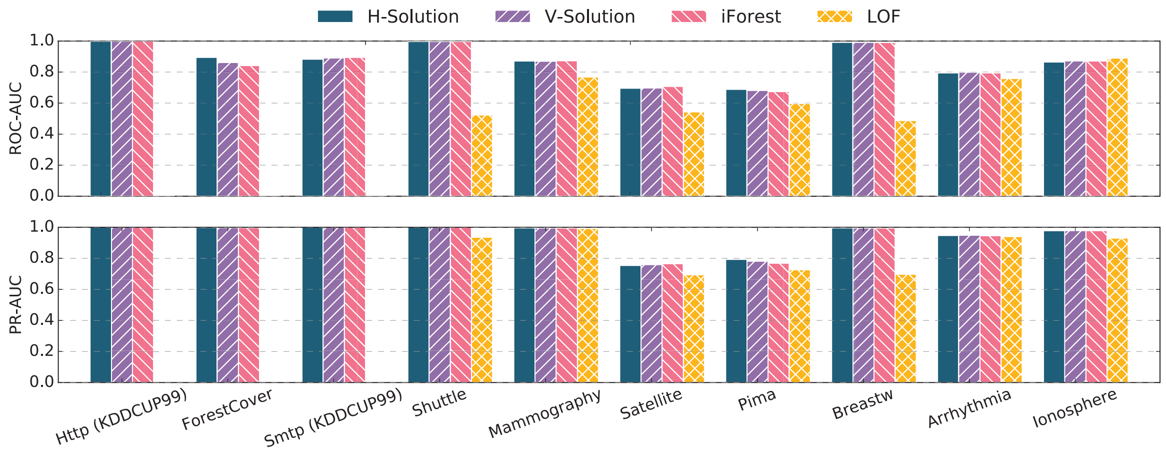


Fig. 5: Comparison of ROC-AUC and PR-AUC per dataset and algorithm

(Http and Smtp) of KDDCUP99 network intrusion data [20], Arrhythmia, Breastw, ForestCover, Ionosphere, Pima, Satellite, Shuttle [21] and Mammography [22]. Table I presents the properties of all the above datasets, where n is the number of instances, m is the number of attributes (dimensions), and abnormal proportion is the ratio of abnormal instances. Similar to [14], we find that setting ψ to 256 generally provides enough details to detect outliers over a wide range of data, and path lengths usually converge well before $t = 100$. So we set $\psi = 256, t = 100$ as default in our experiments. To intuitively show the results, we simply divide the whole workflow of PIF into two phases: train (to obtain the PIF) and test (to detect outliers). Every party is simulated on a workstation with a 3.20GHz Intel Core i7-8700 CPU and 16GB RAM.

ROC-AUC and PR-AUC. Receiver Operating Characteristic (abbreviated as ROC) curves and Precision-Recall (abbreviated as PR) curves are two evaluation tools that help in the interpretation of probabilistic forecasts for binary (two-class) classification predictive modeling problems. ROC curves are appropriate when the observations are balanced between each class, whereas PR curves are appropriate for imbalanced datasets [23]. The Area Under Curve (AUC) is used to clearly describe the correctness of a model. The closer AUC approaches 1, the more accurate the model is, and the closer AUC approaches 0.5, the more inclined the model is to random classification. We will combine the ROC-AUC and PR-AUC metrics to evaluate our solutions.

B. *Correctness and Effectiveness of PIF*

We first verify the correctness and effectiveness of PIF. In order to control variables, we do not consider the communication and encryption/decryption overhead here. We compare our solutions for horizontally partitioned data (abbreviated as H-Solution) and vertically partitioned data (abbreviated as V-Solution), with iForest [14], and LOF [7] (a well-known density-based algorithm) in terms of ROC-AUC, PR-AUC, and actual CPU time (runtime).

The detailed experimental results are shown in Fig 5. We observe that the ROC-AUC and PR-AUC of our solutions are almost the same as that of iForest, which also demonstrates the correctness of our model. Our solutions and iForest have better results than LOF on almost all datasets. In addition, our solutions are not affected no matter whether the data is balanced or not.

The detailed experimental results on effectiveness are shown in Table II. The runtime of our solutions and iForest is much better than LOF. We use 'NA' to denote time-consuming records. In the train phase, the time complexity of iForest and H-Solution is $O(t\psi \log \psi)$, and the time complexity of V-Solution is $O(t\psi^2)$. Since we set $\psi = 256$, $t = 100$, the runtime of all the datasets are basically the same for each method. H-Solution needs to merge PISTs into PIF, and each party in V-Solution needs to judge redundant nodes. So the runtime of our solutions is slightly higher than that of iForest. In the test phase, the time complexity of iForest and H-Solution is $O(nt \log \psi)$, and the time complexity of V-Solution is $O(nt\psi)$. So we can find that the runtime of these methods is linearly related to the dataset size, and V-Solution's runtime is larger than the other two. Note that the runtime of the Arrhythmia dataset in the train phase is longer because it contains more attributes, which requires more time to process during preparation.

In general, the performance of our PIF almost has no difference from iForest in terms of ROC-AUC and PR-AUC, and both of them are superior to LOF. In terms of runtime, no matter how the dataset size expands, the ROC-AUC and PR-AUC always converge when $\psi = 256$ and $t = 100$. So our method and iForest both have a linear time complexity. To summarize, PIF is very efficient for outlier detection, especially on large datasets.

C. *Performance in Distributed Environments*

Now we have learned the AUC and runtime of PIF in centralized environments. Then, to test the performance in distributed environments, we design the following experiment. We simulate 3 parties in the LAN to analyze the runtime of our H-Solution and V-Solution. The data transmissions between parties follow the HTTP protocol. We first evaluate the runtime

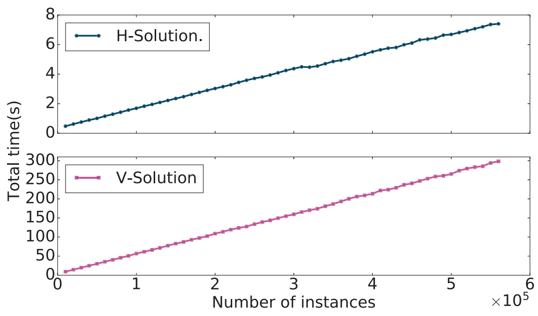
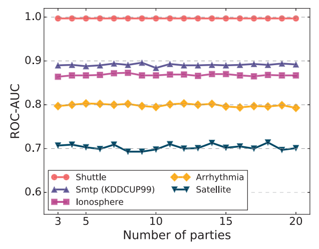
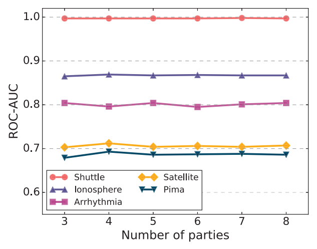


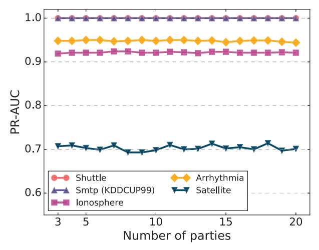
Fig. 6: Time vs. number of instances



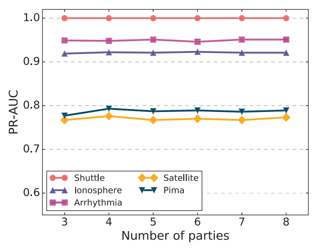
(a) ROC-AUC of H-Solution



(b) ROC-AUC of V-Solution



(c) PR-AUC of H-Solution



(d) PR-AUC of V-Solution

Fig. 7: AUC vs. number of parties

on each dataset, and then test the total time with regard to various quantities of instances.

The comparison of runtime is shown in Table III. From the table, we observe that:

- In the train phase, for each solution, the time spent on different datasets is almost the same, because t and ψ are fixed. The runtime in distributed environments is higher than that in centralized scenarios, since more communication overhead among parties is introduced. H-Solution needs to transmit the leaf values of PISTs, V-Solution needs to send all data on relevant nodes, which results in a little more time consumption than the above centralized scenario.
- In the test phase, for horizontally partitioned data, parties can independently perform outlier detection offline, so the runtime is the same as the above centralized scenario. But for vertically partitioned data, a party still needs help from all other parties to complete the detection, so the amount of communication is larger, which means more time is needed.

In order to figure out the impact of the number of instances on the total running time, we extract different numbers of instances from the Http (KDDCUP99) dataset for testing. The results are shown in Fig. 6. We find that although the total time of the two solutions are different, it changes linearly when the number of instances grows, which is beneficial for outlier detection in the large-scale distributed dataset.

D. *Influence from Quantity of Parties*

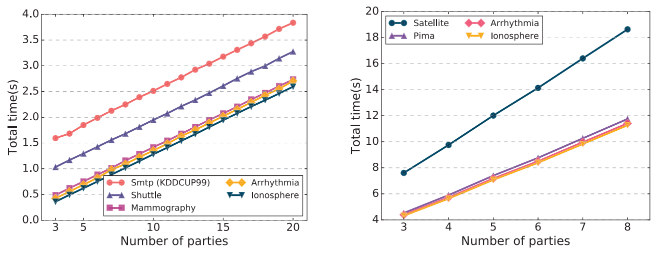
When detecting outliers in a distributed environment, an increase in the number of parties often cause higher time consumption and even reduce the detection accuracy. To study the impact of the number of parties, we design the following experiments.

For horizontally partitioned data, we change the number of parties from 3 to 20, while for vertically partitioned data, we set the maximum number of parties to 8 due to the limitation of attributes. Corresponding AUCs are reported. In order to clearly show the results, we select several datasets with different AUCs. As shown in Fig. 7, we find that both ROC-AUC and PR-AUC of each dataset are nearly the same across different scenarios, even if the number of parties increases.

Then we compute the communication times and traffic for our solutions, as presented in Table IV, where k is the number of parties and n is the number of instances. The communication times here refer to the actual data transmission times. We find that communication times are linearly related to the number of parties. The traffic here refers to the size of the valid data transmitted. We find that the traffic of the three solutions during the train phase is only related to the number of parties, and the traffic of V-Solution is the biggest. There is no communication in the test phase of H-Solution. V-solution's traffic at the test phase grows linearly with both

TABLE II: Comparison of runtime per dataset and algorithm

	H-Solution			V-Solution			iForest			LOF
	Train	Test	Total	Train	Test	Total	Train	Test	Total	
Http (KDDCUP99)	0.014	7.189	7.203	0.034	69.486	69.520	0.008	7.495	7.503	NA
ForestCover	0.014	3.761	3.774	0.034	34.474	34.508	0.009	3.774	3.782	NA
Smtp (KDDCUP99)	0.012	1.266	1.278	0.034	12.995	13.029	0.007	1.320	1.327	NA
Shuttle	0.014	0.695	0.709	0.036	7.133	7.169	0.007	0.709	0.715	1261.943
Mammography	0.013	0.145	0.158	0.034	1.487	1.521	0.006	0.150	0.156	64.938
Satellite	0.021	0.091	0.112	0.032	0.909	0.941	0.007	0.090	0.097	23.231
Pima	0.013	0.012	0.025	0.030	0.102	0.133	0.006	0.011	0.017	0.231
Breastw	0.014	0.009	0.023	0.029	0.096	0.125	0.006	0.009	0.015	0.240
Arrhythmia	0.078	0.006	0.084	0.030	0.062	0.092	0.006	0.006	0.012	0.475
Ionosphere	0.018	0.005	0.024	0.035	0.049	0.084	0.007	0.005	0.012	0.07



(a) H-Solution (b) V-Solution

Fig. 8: Runtime vs. number of parties

the number of parties and the number of instances.

Finally, we test the total time consumed for each dataset under different numbers of parties. As plotted in Fig. 8, we observe that the total time of H-Solution and V-Solution keep a linear relationship with the number of parties. Besides, when the dataset size becomes larger, the growing speed of V-Solution also increases. This is because most of the time consumed in V-Solution comes from the test phase, which is also directly related to the dataset size.

Overall, our proposed method has almost the same AUC with iForest on each dataset in a centralized environment and performs well for distributed scenarios, where iForest fails to work. Both the ROC-AUC and PR-AUC of PIF are nearly unaffected by the quantity of participating parties, no matter whether the data is balanced or not. And the total runtime is linearly related to both the data size and the number of parties. Generally speaking, all the above evaluations demonstrate that PIF can detect outliers effectively with high accuracy in distributed environments.

VI. Related Work

Outlier detection has been well studied in recent years. According to the scenarios targeted, we classify state-of-the-art approaches into the following two main categories.

Detection for centralized data. Existing methods mainly include the following three kinds. 1) Distance-based algorithms. Eskin *et al.* compute the anomaly score of an instance by calculating the sum of its distances of its k nearest neighbors [24]. The work in [5] computes the anomaly score of an instance by counting the number of neighbors with a distance no more than d. This algorithm can also be viewed as a density-based algorithm. 2) Density-based algorithms. LOF [7] defines the anomaly score as the ratio of the average local density of the k nearest neighbors of an instance and the local density of this instance. To improve the efficiency of LOF, Jin *et al.* present a variant in finding anomaly score only for the top n anomalies rather than all instances [25]. 3) Cluster-based algorithms. such algorithms declare any instance that does not belong to any cluster as an outlier. Well-known algorithms include DBSCAN [26], ROCK [27], and SNN [28]. The algorithms above are typically designed for a centralized system. With the expansion of the amount of data, big data distributed storage is becoming more and more common, and data privacy is a very sensitive topic.

TABLE III: Comparison of runtime in 3-party scenario

	H-Solution			V-Solution		
	Train	Test	Total	Train	Test	Total
Http (KDDCUP99)	0.152	7.241	7.393	4.106	305.431	309.538
ForestCover	0.155	3.923	4.077	4.119	147.889	152.008
Smtp (KDDCUP99)	0.151	1.409	1.560	4.113	52.547	56.659
Shuttle	0.154	0.861	1.015	4.094	26.756	30.851
Mammography	0.151	0.338	0.488	4.094	5.990	10.084
Satellite	0.159	0.288	0.447	4.125	3.485	7.610
Pima	0.150	0.212	0.362	4.119	0.414	4.533
Breastw	0.151	0.209	0.359	4.122	0.371	4.492
Arrhythmia	0.219	0.213	0.432	4.093	0.255	4.348
Ionosphere	0.156	0.204	0.360	4.081	0.198	4.279

TABLE IV: Comparison of communication traffic

	Communication times		Traffic (KiB)	
	H-Solution	V-Solution	H-Solution	V-Solution
Train	$6k-4$	$4k-4$	$274.04k-249.03$	$871.69k-871.69$
Test	0	$2k-2$	0	$3.11(k-1)n$

Detection for distributed data. A lot of studies on privacy-preserving outlier detection over distributed datasets have emerged. The authors in [29] propose a privacy-preserving nearest neighbor search method on horizontal data, which can be applied to LOF, but it performs slightly worse than the original LOF. Some researchers have proposed modifications of standard distributed data mining framework, so that parties only exchange locally trained models, *i.e.*, association rules [30], clustering [31] [32], principal component analysis [33], neural networks [34], *etc*. However, these algorithms are not efficient, and it is difficult to apply them in practice.

Zhang *et al.* propose a distributed outlier detection algorithm based on ensemble isolation principle (Secure Isolation Forest) and homomorphic encryption for horizontally partitioned data [16]. However, the samples in this algorithm are not obtained by random sampling, which does not conform to the principle of Isolation Forest. The authors in [35] propose an outlier detection algorithm based on Local Distance-based Outlier Factor (LDOF) and use MPC to protect privacy. Although this method has high efficiency with low communication cost, it can only apply to two-party computation situations.

VII. Conclusion

This work presents a privacy-preserving outlier detection method, which is applicable to distributed datasets with high efficiency and accuracy. PIF innovatively extends traditional iForest algorithm in distributed environments. With a series of solutions devised for both horizontally and vertically partitioned data models, PIF is capable of securely detecting outliers among multiple parties with as little computation and communication overhead as possible. Experimental results demonstrate the practicality and effectiveness of our design.

Acknowledgment

This research is supported in part by the National Key R&D Program of China (Grant No. 2018YFB1308601), and the National Natural Science Foundation of China (Grant No. 61902212).

References

[1] V. J. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[2] H. Wang, M. J. Bah, and M. Hammad, “Progress in outlier detection techniques: A survey,” *IEEE Access*, vol. 7, pp. 107 964–108 000, 2019.

[3] K. Alrawashdeh and C. Purdy, “Toward an online anomaly intrusion detection system based on deep learning,” in *Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2016, pp. 195–200.

[4] G. B. Gebremeskel, C. Yi, Z. He, and D. Haile, “Combined data mining techniques based patient data outlier detection for healthcare safety,” *International Journal of Intelligent Computing and Cybernetics*, vol. 9, no. 1, pp. 42–68, 2016.

[5] E. M. Knorr and R. T. Ng, “Algorithms for mining distance-based outliers in large datasets,” in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. Citeseer, 1998, pp. 392–403.

[6] M. Sugiyama and K. Borgwardt, “Rapid distance-based outlier detection via sampling,” vol. 26, pp. 467–475, 2013.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 2000, pp. 93–104.

[8] L. Duan, L. Xu, F. Guo, J. Lee, and B. Yan, “A local-density based spatial clustering algorithm with noise,” *Information Systems*, vol. 32, no. 7, pp. 978–986, 2007.

[9] J. Gao, W. Hu, Z. M. Zhang, X. Zhang, and O. Wu, “Rkof: robust kernel-based local outlier detection,” in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2011, pp. 270–283.

[10] B. Tang and H. He, “A local density-based approach for outlier detection,” *Neurocomputing*, vol. 241, pp. 171–180, 2017.

[11] C. Zhang, A. Yin, Y. Deng, P. Tian, X. Wang, and L. Dong, “A novel anomaly detection algorithm based on trident tree,” in *Proceedings of the International Conference on Cloud Computing*. Springer, 2018, pp. 295–306.

[12] Z. Chen, A. W.-C. Fu, and J. Tang, “On complementarity of cluster and outlier detection schemes,” in *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2003, pp. 234–243.

[13] Z. He, X. Xu, and S. Deng, “Discovering cluster-based local outliers,” *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1641–1650, 2003.

[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, pp. 413–422.

[15] J. Vaidya and C. Clifton, “Privacy-preserving outlier detection,” in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 2004, pp. 233–240.

[16] C. Zhang, H. Liu, Y. Li, A. Yin, Z. L. Jiang, Q. Liao, and X. Wang, “A novel privacy-preserving distributed anomaly detection method,” in *Proceedings of the International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*. IEEE, 2017, pp. 463–468.

[17] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, “Tools for privacy preserving distributed data mining,” *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 28–34, 2002.

[18] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *Proceedings of the USENIX Annual Technical Conference*, 2014, pp. 305–319.

[19] J. S. Vitter, “Random sampling with a reservoir,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[20] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, “On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms,” *Data Mining and Knowledge Discovery*, vol. 8, no. 3, pp. 275–300, 2004.

[21] A. Asuncion and D. Newman, “Uci machine learning repository,” 2007.

[22] K. S. Woods, C. C. Doss, K. W. Bowyer, J. L. Solka, C. E. Priebe, and W. P. Kegelmeyer Jr, “Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 6, pp. 1417–1436, 1993.

[23] J. Brownlee, *Probability for Machine Learning: Discover How To Harness Uncertainty With Python*. Machine Learning Mastery, 2019.

[24] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, “A geometric framework for unsupervised anomaly detection,” in *Applications of Data Mining in Computer Security*. Springer, 2002, pp. 77–101.

[25] W. Jin, A. K. Tung, and J. Han, “Mining top-n local outliers in large databases,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 293–298.

[26] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[27] S. Guha, R. Rastogi, and K. Shim, “Rock: A robust clustering algorithm for categorical attributes,” *Information Systems*, vol. 25, no. 5, pp. 345–366, 2000.

[28] L. Ertöz, M. Steinbach, and V. Kumar, “Finding topics in collections of documents: A shared nearest neighbor approach,” in *Clustering and Information Retrieval*. Springer, 2004, pp. 83–103.

[29] M. Shaneck, Y. Kim, and V. Kumar, “Privacy preserving nearest neighbor search,” in *Machine Learning in Cyber Trust*. Springer, 2009, pp. 247–276.

[30] G. Deshmeh and M. Rahmati, “Distributed anomaly detection, using cooperative learners and association rule analysis,” *Intelligent Data Analysis*, vol. 12, no. 4, pp. 339–357, 2008.

[31] S. Rajasegarar, C. Leckie, M. Palaniswami, and J. C. Bezdek, “Distributed anomaly detection in wireless sensor networks,” in *Proceedings of the IEEE Singapore International Conference on Communication Systems*. IEEE, 2006, pp. 1–5.

[32] Y.-F. Zhang, Z.-Y. Xiong, and X.-Q. Wang, “Distributed intrusion detection based on clustering,” in *Proceedings of the International Conference on Machine Learning and Cybernetics*, vol. 4. IEEE, 2005, pp. 2379–2383.

[33] V. Chatzigiannakis, S. Papavassiliou, M. Grammatikou, and B. Maglaris, “Hierarchical anomaly detection in distributed large-scale sensor networks,” in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2006, pp. 761–767.

[34] N. Srinivasan and V. Vaidehi, “Anomaly detection in a distributed environment using neural networks on a cluster,” in *Proceedings of the IASTED International Conference on Communication, Network, and Information Security (CNIS)*, 2005.

[35] Z. Wei, Q. Pei, X. Liu, and L. Ma, “Efficient privacy preserving cross-datasets collaborative outlier detection,” in *Proceedings of the International Symposium on Cyberspace Safety and Security*. Springer, 2019, pp. 343–356.