A Survey of Two Open Problems of Privacy-Preserving Federated Learning: Vertically Partitioned Data and Verifiability

Horia-Claudiu Culea, Kaitai Liang, Rui Wang
Delft University of Technology

June 27, 2021

Abstract

Federated learning [1] is a machine learning technique proposed by Google AI in 2016, as a solution to the GDPR [2] regulations that made the classical Centralized Training, not only unfeasible, but also illegal, in some cases. In spite of its potential, FL has not gained much trust in the community, especially because of its susceptibility to data-privacy attacks. Incipient solutions to this problem were homomorphic encryption and differential privacy. These techniques, however, do not come with a solution to other open problems in the field, such as the difficulty to perform FL on vertically partitioned data, ensuring aggregation verifiability, resilience to user dropout etc. Fortunately, new strategies have been developed in the last years. This paper provides a comprehensive study of the state of the art privacy preserving techniques aimed to address two of these problems: vertical federated learning environments and aggregation verifiability. To this end, we study FedV, SecureBoost, MP-FEDXGB, VerifyNet, VFL, together with a verifiability approach exploiting bilinear aggregate signatures, analysing their security model, complexity and communication overhead, the accuracy impact, benefits and downsides in a comparative manner.

Keywords: privacy-preserving federated learning, vertical federated learning, hybrid privacy-preserving techniques, data-privacy attacks, homomorphic encryption, differential privacy, aggregation verifiability

1 Introduction

In 1997, the defeat of grandmaster Garry Kasparov by IBM's chess supercomputer, Deep Blue, with $3\frac{1}{2}$-$2\frac{1}{2}$, in a six game match, marked the dawn of an artificial intelligence dominated era. The following years have recorded numerous breakthroughs with regards to ML and AI state of the art, some of the most notable being MNIST (1998) [3], ImageNet (2009) [4], AlexNet (2011) [5], The Cat Experiment (2012) [6], and DeepMind (2016) [7]. Nowadays, the vast majority of AI-powered systems are trained with large amounts of decentralised data. In 2016, the inherent problems of Centralized Training[1], related to data security and privacy, became apparent, along with the introduction of the GDPR [2] regulations. In response, Google AI proposed federated learning [1], designed to achieve decoupling between the need to store data locally and the ability to do machine learning [8].

[1] machine learning technique involving the computation of a global model based on data aggregated from decentralised clients

Federated learning (FL) is a machine learning technique involving the decentralised training of a model without explicitly sharing data samples. During this process, the clients each train a local model with local samples of data and only share the parameters of their model with a central authority. After aggregating all clients' models and computing a global, improved model, the new corresponding parameters will be communicated back to the clients, which will update their local models accordingly. This process is repeated for multiple rounds, until convergence is achieved.

FL is prone to a wide array of attacks [9], such as poisoning attacks, backdoor attacks and inference attacks [10]. The main vulnerabilities that allow these attacks to happen are insecure communication protocols, the vicious manipulation of client data by adversaries, compromised aggregators and weak, insecure aggregation algorithms. [9]

Accounting for data security and privacy in FL involves both preventing inference over the messages exchanged during training, and ensuring that the resulting model has a sufficiently high prediction rate [11]. Classical techniques that are used in this endeavour are differential privacy (DP) [12] and homomorphic encryption (HE) [13]. However, both these schemes exhibit drawbacks [9], either in terms of accuracy or efficiency. Moreover, these techniques do not protect against a malicious aggregator, tampering with users' updates.

State of the art privacy-preserving techniques involve combining homomorphic encryption and differential privacy as to maximize efficiency and accuracy. However, newest privacy preserving schemes, such as VerifyNet [14], Adversial Training [15] and FedV [16] bring completely new approaches of security to the table. These techniques lay the ground work for future development of federated learning, and for overcoming its current challenges, such as trusted traceability [17] (backward tracking to identify which of the clients induced a certain change to the global model, without compromising privacy through logging), optimizing the trade-off between privacy protection enhancement and cost, and simplifying migrating and productionising FL environments [18].

There is, however, no metric universally used as to compare the different privacy-preserving strategies. In particular, training models with datasets with a decentralised feature set (vertical federated learning) is still an open problem, that few strategies tackle [19]. Moreover, hardly any methods implement a verification method for gradient aggregation [14]. Hence, formally, the research questions this paper aims to answer are:

- What are the privacy preserving schemes available for vertical federated learning, and how do they compare?
- What are the privacy preserving schemes providing aggregation verifiability, and how do they compare?

This paper is set out as follows: Section 2 will introduce concepts related to the research questions, and to the current state of FL. Section 3 will formalize the methodology employed in surveying the current state of the art of privacy-preserving techniques. Section 4 will analyse the identified techniques for vertically partitioned data individually. Section 5 will do the same for the verifiability techniques. Both sections 4 and 5 will end with a comparative analysis of the techniques approached in their beginning. Section 6 will touch upon the ethical aspects of the research process. Finally, Section 7 will conclude the paper, as well as discuss the future direction of research in this field.

2 Preliminaries

This section introduces background knowledge as to familiarize the reader with the topic of this paper. To this end, we discuss different categories of federated learning, the classical attacks on FL environments, classical privacy-preserving techniques and adversarial models.

2.1 Types of Federated Learning Environments

Based on the distribution characteristics of the data held by the clients in the federated learning environment, two main categories of FL can be differentiated [8]:

- horizontal federated learning: data of clients in the environment spans the same (or overlapping) feature space, but a different sample space
- vertical federated learning: data of clients in the environment spans the same (or overlapping) sample space, but different feature space

While Section 4 will address privacy-preserving federated learning techniques for vertical environments, in Section 5, we also encounter horizontal federated learning, as the approached verifiability strategies work exclusively on horizontally partitioned data.

Hence, understanding the practicality of both horizontal and vertical federated learning is essential to formalizing the approached problems. Let us consider two examples:

- a two-party example of horizontal FL is an ML model trained with data from two banks from different areas. Although these store the same data about their customers (overlapping feature space), they will most likely have no common clients (disjoint sample space).
- a two-party example of vertical FL is an ML model trained with data from a bank and a retailer within the same area; the retailer may be interested in its customers' purchasing power, while the bank may be interested in the customers' buying habits. The two entities hold different information about customers (different feature space), and, due to their locality, they will have common customers (overlapping sample space).

2.2 Types of attacks on Federated Learning Environments [10], [20], [21]

Based on the goal of the adversary, we distinguish two categories of attacks in FL: poisoning attacks, aiming to negatively influence the performance of the model, and data privacy attacks, aiming to infer information about data used within the federated learning environment. While only the latter are addressed by privacy-preserving techniques, interested readers may refer to [22] for more information on poisoning attacks. The most common types of data-privacy attacks are:

- model inversion attack [10]: inferring a specific data sample, given the parameters of the model and the output generated by the sample
- inference attack [20]: inferring information about a certain data sample or the federated model (membership inference, property inference, training inputs and labels inference etc.)
- GAN reconstruction attack [21]: training a generative adversarial network to generate prototypical samples of targeted private training data

Based on the adversarial model of the participants in the FL environment, for each technique, we will evaluate how these attacks are addressed, together with the colluding problem, described in 2.4.

2.3 Classical Privacy-Preserving Techniques

The majority of the hybrid privacy-preserving techniques employ either one or both of the classical techniques in their workflow. Hereby, we formally introduce the two techniques.

Homomorphic Encryption [13]

Homomorphic Encryption is an encryption mechanism that allows performing operations on the encrypted data. Upon decryption, the result is identical to that produced had the operations been applied to the raw data. Homomorphic encryption, usually, is based upon additive homomorphisms:

$$E(m_1) + E(m_2) = E(m_1 + m_2)$$

or multiplicative homomorphisms:

$$E(m_1) * E(m_2) = E(m_1 * m_2)$$

where E is the encryption function.

Worth mentioning also is the Paillier cryptosystem [23], an often used scheme:

$$E(m_1, r_1) * E(m_2, r_2) = E(m_1 + m_2, r_1 r_2)$$

Based on the number of mathematical operations that can be performed on the encrypted message, there are three types of HE: partially homomorphic encryption (allowing only one operation), somewhat homomorphic encryption (allowing both operations, a limited number of times) and fully homomorphic encryption (allowing multiple operations, an unlimited number of times).

Adding HE to a federated learning environment negatively influences the efficiency of the model, because of the computation overhead, generated by encryption/decryption, as well as the communication overhead, generated by the sharing of the keys required by the chosen cryptosystem. [9].

Differential Privacy [12]

Differential privacy is a method of openly disclosing information about a dataset by revealing patterns of groups within it, without releasing information about specific data samples. Formally, a process $P : X^n \longrightarrow Y$ is (ϵ, δ)-differentially private if for all neighbouring datasets $D_1, D_2 \in \mathrm{X}^n$ and all $T \in Y$:

$$Pr[P(D_1) \in T] \leq exp(\epsilon) Pr[P(D_2) \in T] + \delta$$

Therefore, differential privacy ensures that, if an item is added to or removed from the training set, this shall not affect the outcome of the model's analysis on unseen input. Most often, differential privacy is practically achieved by adding statistical noise to personal, sensitive attributes [9]. Because of this, adding differential privacy to a FL environment negatively influences the accuracy of the underlying model.

2.4 Adversarial models [24]

Based on the behaviour of the parties in a FL environment, we distinguish two adversarial models:

- semi-honest (honest-but-curious): a legitimate participant, not deviating from the defined protocol, but that attempts to learn all possible information from legitimately received messages
- malicious: an illegitimate participant, that may deviate from the defined protocol arbitrarily

The majority of privacy-preserving frameworks assume the honest-but-curious adversarial model. The frameworks providing a verifiability solution, however, assume a malicious aggregator.

An interesting security model that will be recurring in the analysis is that of honest-but-curious, colluding clients. Collusion represents the ability of multiple participants to coordinate an attack together [24]. Different frameworks provide an array of guarantees against the collusion problem, ranging from no resiliency whatsoever to a guarantee against $n - 1$ colluding users, where n is the total number of users within the environment.

3 Methodology

In order to answer the two questions posed by this research paper, different approaches to preserving privacy in vertical federated learning environments and that ensure aggregation verifiability are subjected to study. Every privacy-preserving technique is analyzed through the study of the research paper introducing that specific technique. These research papers not only introduce the privacy-preserving strategy employed within the technique, but also, usually, serve as a security white-paper. Additionally, efficiency and accuracy are often analysed, in the context of an experiment.

Therefore, the methodology employed in studying each privacy-preserving framework is based upon extracting or deriving the following key-points from the respective research materials: *1. workflow of the privacy-preserving protocol, 2. computational complexity, 3. communication overhead, 4. accuracy impact, 5. security model and prevented attacks*.

To boot, for each of the two research questions, a comparative analysis is performed. Each analysis argues aspects related not only to the theoretical study of the individual frameworks, but also to the experiments performed within the surveyed research papers, in order to compare real-life aspects related to the performance and implementation of the approached techniques.

4 Federated Learning on Vertically Partitioned Data

Most of the existing federated learning frameworks only work in the scenario of horizontally partitioned data [19]. Unlike horizontal FL, vertical FL requires a more complex mechanism to decompose the loss function at each party [25]. Moreover, many of productionised environments fit better within the vertical FL paradigm, where the class labels are only available to one party, and the entire feature set is only obtained upon the aggregation of multiple data sources [16]. Additionally, privacy-preserving vertical FL comes with the necessity of performing an extra step, before training: entity alignment/resolution, i.e. determining the intersection of different data sets in the sample space. Ensuring that entity resolution does not allow the inference of private data is crucial [16].

In this section, we study and compare three approaches of privacy-preserving vertical federated learning: FedV [16], SecureBoost [25], and MP-FEDXGB [26].

4.1 FedV [16]

FedV [16] is a generic, efficient privacy-preserving vertical FL framework, supporting stochastic gradient-decent based algorithms to train both linear and non-linear ML models.

The Entity Resolution Process (PER) within FedV employs an anonymous linking code technique called cryptographic longterm key, followed by the Dice coefficient matching method [27]. Interested readers may refer to [28] for the full description of the technique.

The entities participating within FedV are:

- multiple passive parties: clients of the FL environment, owning local data samples and features
- one active party: party which, aside from its local data, also has ownership of the class labels
- one aggregator: entity responsible for aggregating clients' data and training the global model
- one TPA (trusted third-party authority): honest entity, responsible for initially setting up the cryptosystem and for issuing decryption keys for the aggregator

Naively computing gradient descent in a vertical FL setting will lead to a significant risk of privacy leakage, since it would imply either peer-to-peer collaboration between parties, to exchange

their partial models, or clients exposing their data to the central server, which would be trusted to compute the gradient update. To mitigate this issues, FedV makes use of a functional encryption for inner-product (FEIP) scheme, in two settings:

1. feature-dimension secure aggregation: each client encrypts its partial model using a multi-input functional encryption (MIFE) scheme [29], with a public key from the TPA, and sends it to the aggregator. Upon receiving all the encrypted partial models, a fusion vector, containing the weights that each of the partial models will be attributed, is generated by the aggregator. This vector is sent to the TPA, which returns a secret function key. Upon computing the inner-product between the encrypted vector and the function key, the aggregator obtains the aggregated sum of the elements in the feature dimension.

2. sample-dimension secure aggregation: each party encrypts its batch samples using a single-input function encryption (SIFE) scheme [30], with a public key from the TPA. Using the result of the feature-dimension SA and a function secret key, issued by TPA, the aggregator is able to acquire the batch gradient.

At the start of the training process, the parties use a random seed as to generate a one-time password chain. Every training epoch, the chain and the epoch number are used as to randomly select samples to be included in the epoch's training batch. Therefore, the aggregator is unaware of the identity of the samples in any training round, and inference attacks are therefore prevented.

FedV works under the security model of honest-but-curious aggregator and malicious, colluding users. All channels of communication are expected to be secure, as well as the used cryptosystems. In order to prevent inference attacks, FedV makes use of an Inference Prevention Module within the TPA. The task of IPM is ensuring that the aggregator does not submit fusion vectors with weights designed as to isolate a specific sample of data or target a user. Trying to isolate a specific sample between two rounds of training is, once again, not possible, because of the randomised batch selection. Lastly, FedV can prevent against up to $t-1$ colluding users (t is a parameter of the IPM).

In terms of accuracy, FedV is a lossless framework both in training and prediction, since it does not require Taylor series approximations when tackling non-linear models. In an experimental setting, using the OptDigits dataset (64 features, 5620 samples, 3808 used in training) in Logistic Regression, FedV achieves an accuracy of 100%.

In terms of communication costs, FedV exhibits $O(n)$ (per epoch) overhead both during the Secure Stochastic Gradient Descent and the Secure Loss Computation, where n is the number of clients in the vertical FL environment [16]. While the framework only improves over [28] by a linear factor in the former, the latter improves by a polynomial factor, as Hardy's approach is $O(n^2)$. In e rounds of training, this becomes $O(en)$, while the clients maintain an $O(e)$ overhead.

Let us assume e is the number of epochs of training, s and f, the number of samples and features per user, respectively, and all data entries are comparable in size, such that we can consider the encryption/decryption process to take constant time. Each epoch, a client spends $O(ks+f)$ for encryption (k being a constant factor, corresponding to the fraction of records chosen by the randomization algorithm), due to computing cyphertexts in the sample and feature spaces separately. Similarly, the server spends $O(n(ks+f))$ for decryption. In e epochs of training, the runtime at the server will become $O(en(ks+f))$, while that at the client will be $O(e(ks+f))$.

4.2 SecureBoost [25]

SecureBoost is a privacy-preserving, lossless, tree-boosting federated algorithm and framework for vertically partitioned data, extending upon the XGBoost technique.

As opposed to traditional FL, where clients have ownership over local data, and the server only facilitates the aggregation of local gradients, SecureBoost defines two types of entities: the active party, which holds both a data matrix and the class labels, and the passive party, which exclusively holds the former. Therefore, the role of the aggregator is played by one of the clients.

The workflow of SecureBoost consists of two steps:

1. Privacy-Preserving Entity Alignment: based on the Inter-Database Intersection Protocol described in [31], the parties within the FL Environment find the common data samples across their databases, without revealing the non-shared parts.

2. Classification/Regression Model Training: described below.

Similarly to XGBoost, SecureBoost predicts the output by using K regression trees. During training, each iteration greedily adds a new tree f_t to minimize the loss function:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y_i}^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y, \hat{y}^{(t-1)})$.

In the end, the optimal weight of a leaf can be calculated as: $w_i^* = -\frac{\sum_{i \in I_i} g_i}{\sum_{i \in I_i} h_i + \lambda}$

Therefore, the evaluation of split candidates and that of optimal weights of leaves exclusively depend on g_i and h_i. The active party will broadcast these parameters, encoded using the Paillier cryptosystem. Each passive party will compute the same parameters for the local splits, and the active party will aggregate each party's gradients. In the end, the passive parties will index the local splits in a lookup table, containing the feature and the threshold. Each node, at the active party, will be represented by a tuple containing the ID of the party responsible for that certain split, and the ID of the split within the lookup table of the passive party.

The security model employed by SecureBoost is honest-but-curious, the same security assumption considered by [31]. However, there are unaddressed security concerns. As a client, the active party is in an advantageous position, since:

- it learns each split's instance space, as well as the party responsible for each decision point
- it learns all the possible g and h values during training (this is especially important, since the class label can be inferred with these two values [25])

Therefore, SecureBoost is susceptible to inference attacks.

In section 6, [25] proves the lossless property of SecureBoost, showing that a SecureBoost model has the same accuracy as an XGBoost model by making use of the properties of the Pailler cryptosystem. Therefore, SecureBoost has no impact on the accuracy of the underlying model.

According to [31], the entity alignment of two parties is linearly dependent on the number of samples in the databases. Therefore, the entity alignment of n passive parties each holding N records with the active party takes $O(nN)$, at the active party, and $O(N)$ at the passive parties.

Let us assume a tree of depth d, which will have an order of $O(2^d)$ nodes. For each node, the active party will perform $O(n)$ modular exponentiation procedures for encryption/decryption, scaled with the number of possible splits, assumed to be constant. According to [32], modular exponentiation costs $\lfloor logM \rfloor$ multiplications, which are presumed to be performed in constant time (M is the value of the gradient). For the purpose of the analysis, we shall assume that all gradients are equal in magnitude, and hence encryption and decryption can be considered to take constant time. Therefore, for T trees, the complexity overhead at the active party is $O(nN + 2^d Tn)$

For each node, each passive party will compute a constant number of modular exponentiation procedures. Hence, the complexity overhead at the clients will be $O(N + 2^d T)$.

During the entity alignment process, [31] proves that the communication overhead is $O(N)$. Once again, for each split, the passive parties will transmit a constant number of messages, while, at the server, this number scales linearly with the number of passive clients. Therefore, the communication overhead is $O(N + 2^d T)$ at the passive parties and $O(nN + n2^d T)$ at the active party.

4.3 MP-FEDXGB [26]

Multi-Party Federated XGBoost (MP-FEDXGB) [26] is the first multi-party federated XGB learning framework, operating on vertically partitioned data, based on the secret sharing (SS) technique introduced by Blakey [33] and Shamir [34]. This scheme also introduces a novel security mechanism, First-Layer-Mask, as to address the potential leakage in the instance space.

The idea behind SS relies on sharing of private data, by the participants within a distributed system, to the other participants, in such a way that none of them alone can recover the data [26]. The SS technique contains addition, subtraction and multiplication primitives. XGBoost, however, requires additional operations, such as *argmax* (e.g. in selecting the split candidate that maximizes the loss reduction function) and division (e.g. in the leaf weight computation). Therefore, [26] also proposes a computation reshaping method for effectively implementing the required operations.

It is to be noted that MP-FEDXGB does not provide an entity alignment strategy, but rather assumes that the data holders are have already agreed upon the dataset to be used in training. MP-FEDXGB assumes vertically partitioned data among the clients in the environment, possibly with overlapping feature sets between clients. The entities within MP-FEDXGB are the active party (holding a data matrix and a label vector), auxiliary participants (holding a data matrix), and a coordinator, as a trusted third party.

The training of an MP-FEDXGB model relies on three algorithms, whose pseudocode is presented in Fig 1, 2, 3. Algorithm 1, SecureTrain, will execute a T-round for loop, in order to generate T trees. Every round of the loop makes a call to Algorithm 2, SecureFit, which prepares the data and, in the case of the active party, also calculates the first and second order gradients of the loss function. Algorithm 3, SecureBuild, is used in order to recursively build a partial tree model. Split candidates with the biggest loss reduction are selected, traversing every value of every feature. At this point, SecureBuild is called recursively on the left and right branches. All the participants which have the feature that is being split within their feature space will retain the split information at the node in their local model, while all the other participants will mark it as 'Dummy'. During prediction, each party will perform a tree search with its partial model. Nodes marked as 'Dummy' will be searched in both branches they span. Each selected leaf is marked with an '1', and else, a '0'. By performing the inner product of all the indicator vectors generated by the outcome of the tree search performed by each participant, only one leaf is determined.

MP-FEDXGB is based on the honest-but-curious adversarial setting. The active participant is required to be honest. Under this assumption, all the auxiliary participants could cooperate as to try to recover private information. However, the active party, under the honest assumption, does not leak its share of the secret, which makes the inference of raw data and intermediate values impossible. Another inherent problem of the scheme is that instances within one leaf node will most likely have the same label, due to sharing the same instance space. The proposed First-Layer-Mask mechanism comes to address this, by ensuring that the root node will always be split at the active party. The following splits will once again be performed on the entire sample set, regardless of the participant that performs them, and therefore, the path between the root and the leaf is always ensured to be cut at depth 1, at the cost of an additional split.

MP-FEDXGB is proven to be, in theory, a lossless framework, due to not using non-linear operations, improving therefore over [28], which makes use of Taylor expansion. Experimental results

are presented in [26], with two datasets (with an 80/20 training-test ratio, first one containing 37 500 samples, while the second one contains 32 561 samples), four participants, three trees, varying depths and a 10%, 20%, 30%, 40% distribution of features. The results not only confirm the lossless nature of MP-FEDXGB, but also demonstrate that the First-Layer-Mask (denoted with '*') does not affect the performance of the model either. The results can be seen in the Fig 4.

In terms of computational cost, a first observation is that the training time linearly depends on the number of trees, T (Algorithm 2). Moreover, the same linear dependence can be observed when it comes to the number of features (f), number of possible values for a feature (s), and the number of nodes in the tree (2^d, where d is the depth), according to Algorithm 3. Although [26] does not analyse the dependence of the runtime on the number of users in the vertical FL environment, for the sake of this analysis, we shall consider that sending a message costs $O(1)$ time. Parties transmit shares for a constant number of messages (Algorithm 2 - first and second order gradient statistics of the loss function; Algorithm 3 - local indicator vectors), hence the dependence on the number of users n is linear. The computational overhead is $O((n + fs)T2^d)$. This analysis is consistent with the experimental setting, as shown in Fig 5, 6.

The communication overhead relies on the number of transmitted shares. For each tree and each split, a user will send a constant number of messages, to all other users, hence $O(nT2^d)$.

4.4 Comparison of Vertical FL Frameworks and discussion

All observations below are based on Table 1, containing the main takeaways of the individual analysis of each vertical FL framework, with the literals defined as before, in the individual subsections.

It is worth to be noted that SecureBoost, as opposed to FedV and MP-FEDXGB, has also been analysed in terms of the computational complexity and communication overhead required by the entity resolution process. For the sake of this comparison, we shall ignore the first term in the respective expressions, corresponding to the resources spent on the PER. Additionally, we mention that FedV operates on different models than SecureBoost and MP-FEDXGB, and hence the below comparison will be a purely theoretical one.

In terms of computational complexity, at the aggregator (i.e. the active party, in the cases of SecureBoost and MP-FEDXGB), the frameworks depend linearly on the number of epochs of training (i.e. the number of trees, in the case of SecureBoost and MP-FEDXGB) and the number of users.

At the clients, the linear dependence on the number of samples is observed in both MP-FEDXGB and FedV. At MP-FEDXGB, the overhead scales linearly with the number of users too, due to the secret sharing algorithm employed by the tree building protocol (each user will communicate with all the other users in the environment, during this stage).

From a practical perspective, the costs involved by the Paillier cryptosystem, that SecureBoost makes use of, are very high [26], both in terms of computation time and required amount of memory. Although better, MP-FEDXGB is computationally expensive too, as its complexity increases with the number of features and the data becoming more heterogeneous [26].

In terms of communication cost, FedV is superior to the other two schemes: the communication cost scales linearly with the number of users at the aggregator and is constant per user (in one epoch of training), while SecureBoost and MP-FEDXGB are also dependent on the depth of the tree exponentially. Once again, we see a linear dependence on the number of users both at the client and the aggregator in MP-FEDXGB (as opposed to the other two schemes, which only exhibit such a dependence at the aggregator), because of the SS protocol. Hence, the communication overhead is better in the case of SecureBoost, between the two XGBoost variants.

These observations are consistent with those empirically made by [16]. In an experimental setting, FedV has been shown to reduce the costs of communication between 80% to 90% over

similar schemes (along with the reduction of training time, from 10% to 70% over [28]).

Moreover, it is worth to be mentioned that FedV, as opposed to SecureBoost and MP-FEDXGB, does not depend on an underlying ML model. While both SecureBoost and MP-FEDXGB operate on protocols derived from XGBoost (and therefore only train Classification and Regression Trees), FedV can complement any model that supports updates through the Stochastic Gradient Descent method (such as linear models, logistic regression, support vector machines etc.).

From a security standpoint, it is interesting to compare MP-FEDXGB against SecureBoost, since the two represent different approaches to implementing the same algorithm in a federated setting: secret sharing based FEDXGB and homomorphic encryption based FEDXGB, respectively. The former proves to be superior from a security standpoint. HE-based FEDXGB only promises no raw data sharing. Intermediate information leakage remains a problem, due to the ability of the active party to find second order derivatives of data instances. Similarly, the density distribution can easily be leaked to the active party [26]. Both these problems are addressed by SS-based FEDXGB.

FedV also offers strong security guarantees, against the malicious user adversarial model: no inference, due to batch randomization and its IPM, as well as protection against user collusion up to t users. In contrast, MP-FEDXGB prevents user collusion up to $n-1$ users, where n is the number of users, as long as the active party is honest. This, however, is a hard requirement to fulfill, and, overall, a downside of this framework. SecureBoost offers no guarantees against collusion. We hence conclude that the strongest security guarantees are offered by FedV, while the least secure is SecureBoost. At the cost of higher computational complexity and a less secure adversarial setting, MP-FEDXGB offers the best guarantees against coordinated attacks of users.

Framework	FedV	SecureBoost	MP-FEDXGB
Computational Complexity at Aggregator	$O(en(ks+f))$	$O(nN+2^dTn)$	$O((n+fs)T2^d)$
Computational Complexity at Client	$O(e(ks+f))$	$O(N+2^dT)$	$O((n+fs)T2^d)$
Communication Overhead of Aggregator	$O(en)$	$O(nN+n2^dT)$	$O(nT2^d)$
Communication Overhead of Client	$O(e)$	$O(N+2^dT)$	$O(nT2^d)$
Accuracy Impact	lossless	lossless	losless
Security Model	honest-but-curious aggregator, malicious and colluding users	honest-but-curious aggregator and clients	honest-but-curious colluding auxiliary parties, honest active party
Machine Learning Model	linear and non-linear models supporting updates through SGD	Classification and Regression Trees	Classification and Regression Trees
Security Mechanisms	Inference Prevention Module, Batch Randomization, FEIP encryption (MIFE+SIFE)	Paillier Encryption	Secret-Sharing, First-Layer-Mask

Table 1: Comparison between Vertical Federated Learning Privacy Preserving Techniques

5 Verifiability in Federated Learning

The majority of solutions addressing data-privacy in FL assume the security model of a honest-but-curious aggregator. While this reduces the complexity of implementing these techniques, this assumption inherently involves no resiliency whatsoever to the aggregator tampering with the received gradients, and returning manipulated updates. Moreover, there is a correlation between users' privacy being compromised and the ability of the adversary to manipulate the data returned to users [14]. Attacks such as those described by [21] and [35] illustrate how, by analysing statistical characteristics of users' data, adversaries are able to return manipulated results, effecting users to submit sensitive data. Therefore, the verification of the aggregated result has to be properly handled.

In this section, we present three privacy-preserving techniques, providing verifiability for the server aggregation: VFL [36] (verification through Lagrange Interpolation), VerifyNet [14] (verification through homomorphic hash functions, integrated with pseudorandom technology) and the verification approach proposed by [37] (verification through bilinear aggregate signature).

5.1 VFL: Verifiable Federated Learning [36]

VFL is *a verifiable federated learning with privacy-preserving to achieve efficient and secure model training for industrial intelligent* [36]. This framework allows, aside from data privacy, the client verification of correct, unbiased aggregation by the server of submitted gradient updates.

The workflow of VFL consists of four main steps:

1. The initialization stage: The participants agree upon the structure of a neural network. A public key generator (PKG) initializes a model. The PKG sends to all participants m pairwise coprime numbers $g_1 ... g_m$ ($m \geq 3$ - security parameter). Two disjoint sets of numbers $\{a_i | i = 1, 2..m\}$ and $\{b_i | i = 1, 2..m\}$ to be used in the interpolation process are generated. PKG also sends the participants two seed sequences and the same pseudo-random generator (PRG).

2. The model training phase: The local models of all the participants are trained simultaneously. Each round of training, every participant executes backpropagation and SGD to calculate the local gradient. Local gradients are encrypted through a custom encryption scheme and submitted for aggregation to the server. Pseudocode of the gradient encryption algorithm can be seen in Fig. 7. In brief, the proposed encryption scheme relies on blinding the gradient vector using parameter sequences produced by the PRG, afterwards splitting the gradient and performing Lagrange Interpolation in order to find the corresponding polynomial function. The function output on sequence b_i together with g_i generate a system of linear congruences. Its solution, obtained via Chinese Remainder Theorem, represents the encrypted gradient.

3. The aggregation phase: The server aggregates the received gradients, by computing their sum. Since CRT satisfies additive homomorphism, the obtained result represents a system of linear congruences of m equations, where each element is a $m - 1$ degree polynomial.

4. The update/verification phase: The clients receive the sum of the all encrypted gradients, as computed by the aggregator previously. Each participant unpacks the cyphertext via a modulus operation. Lagrange Interpolation is afterwards conducted with b_i and the obtained result. In the obtained polynomial function F, the participants verify $F(a_m) = A$, where A is a parameter received from the PKG. If this holds, the local model is updated, otherwise the process is aborted. The pseudocode of this algorithm is presented in Fig. 8.

The security model employed by VFL is honest-but-curious colluding clients, while the server is presumed to be malicious. As proved in [36], the private gradient and the model parameters of

each participant will not be leaked to the aggregation server. This is because both sequences a_i and b_i, which would be needed in order to recover the values of the function, generated by the Lagrange Interpolation polynomial on the b_i sequence, are not disclosed to the aggregator. Therefore, inference attacks and the collusion problem are properly addressed. Moreover, [36] offers guarantees with regards to the privacy of gradients of non-colluding participants, in the case of collusion of maximum $n-2$ parties.

The experimental setting in which VFL has been tested, as reported by [36], makes use of a multi-layer perceptron, trained with 60 000 images of the MNIST dataset and tested on 10 000 samples of the same set. In terms of accuracy, after 400 rounds of training, VFL achieved an accuracy of 94%, in contrast with the original federated learning scheme, proposed by [38], which achieves 95%.

In terms of communication, the server will only transmit a gradient of size g to all n users, during stage 4. Therefore, the overhead is $O(egn)$, where e is the number of epochs of training. Similarly, the users only submit a vector of length g per epoch, therefore $O(eg)$.

Accounting for the computational complexity in VFL means assessing the overhead introduced by the encryption, decryption and verification processes. The only task that the server performs is adding n vectors of length g. Considering an addition of two numbers to be $O(1)$, in e rounds of training, this will yield a $O(egn)$ overhead. During the encryption process, gradient blinding and parameter selection is assumed to take constant time. According to [39], Lagrange Interpolation takes $O(m^2)$. Considering the magnitude of the parameters of the system of linear congruences equal, solving it via CRT is expected to take quadratic time too [40], dependent on the number of equations. Similarly, decryption will take, once again, $O(m^2)$. According to [36], both encryption and decryption depend linearly on the number of gradients too. Therefore, $O(em^2g)$.

5.2 VerifyNet [14]

VerifyNet is the first privacy- preserving approach supporting verification in the process of training neural networks. It provides a verifiable approach based on the homomorphic hash function and pseudorandom technologies to support the verifiability for each user. VerifyNet also provides a variant of secret sharing technology [34] along with key agreement protocol to protect the privacy of users' local gradients, and to address the users dropout problem during the training process.

The workflow of VerifyNet consists of five steps:

1. Initialization: A trusted authority generates, for each user, the keys that are going to be used for the homomorphic hash functions (δ, ρ), pseudorandom functions (K_1, K_2) and keys to be exploited during gradient encryption. The users submit their public keys to the server. The server needs to receive messages from at least t users, where t is a parameter of the secret sharing protocol. The server returns a statistic label to the users.

2. Key Sharing: Each client i generates a random number β_i, to be used as seed to the PRG in generating an offset for the double masking protocol, and sends shares of it to all other online clients, through Shamir's t-out-of-N secret sharing protocol.

3. Masked Input: Every two users n, m agree upon a random number $s_{n,m}$, to be used as seed to the PRG. Each user encrypts local gradient x_n as follows:

$$\hat{x_n} = x_n + PRG(\beta_n) + \sum_{m \in U: n<m} PRG(s_{n,m}) - \sum_{m \in U: n>m} PRG(s_{n,m})$$

   A group of five numerical values, based on the parameters distributed by the TPA in the Initialization stage and homomorphic hash function properties, are also computed and submitted, in order to verify the correctness of the results returned by the server in the Verification step.

4. Unmasking: The server receives the double-masked gradients. The server receives all shares β_i from the online users, and shares of N_i^{SK} for all users that dropped out, such that it is able to compute and undo the respective offset. Proof is calculated as the element-wise product of the 5-tuples received at the previous step. The aggregated result and proof are broadcasted.

5. Verification: users receive the aggregated result and the Proof of aggregation. Making use of the properties of hash functions, each user checks a set of four equalities. If these hold, the local parameters are accordingly updated. If any does not hold, the update is aborted.

The security model employed by VerifyNet is honest-but-curious, at the clients and the aggregator. TA is considered honest and to not collude with any entity. However, despite the approached adversarial model, the central server exhibits malicious aptness, such as the abilities to forge proof and modify the calculated results for deceiving users. As the authors of [14] argue, VerifyNet is able to protect both against joint attacks of multiple colluding users, and joint attacks of server and multiple users, as long as there is a maximum of $t - 1$ clients in collusion. Hence, data privacy is ensured as long as there are not enough colluding users within the system to centralize shares received through the secret sharing protocol as to revert the double mask of the individual gradients.

The experimental setting described in [14] has been conducted using a CNN network [41], consisting of two convolutional layers and two fully connected layers of 128 neurons each. The experiments have been conducted on the MNIST dataset (60 000 samples in training and 10 000 samples in testing). In theory, VerifyNet is a lossless framework. However, there has been no experiment, testing it against similar frameworks. The authors of [14] observe how accuracy increases, with an increase either in the number of users (shown in Fig. 11) or gradients, but eventually converges.

Let n be the total number of users in the environment, and let us assume that each user submits m gradients each epoch of training. During each round of training, every user will submit at least t (assumed to be $O(n)$) shares for the random offset β and its secret key. They will also submit an order of $O(m)$ gradients to the server, and an order of $O(n)$ shares for users dropping out. We assume the gradients and shares are equal in magnitude. The communication overhead, over e epochs of training, will therefore be $O(e(n+m))$. The server will communicate $O(m)$ gradients to each user. Hence the overhead will be $O(emn)$.

In terms of computational complexity, each user will take $O(n)$ time to calculate the agreed upon secret key with every other user, as well as generating and sending shares. We assume a call to the PRG takes $O(1)$. For each gradient, $O(n)$ calls to the PRG will be made. The rest of the processes (including encryption and decryption of a single gradient) are assumed to take constant time. Therefore, at the client, the computational overhead is $O(emn)$. The aggregation at the server (summing the gradients, multiplying the verification vectors) will be $O(mn)$. For each dropout user (who are assumed to be an order of $O(n)$), $O(n)$ computations are needed to eliminate the offset in the double-masked gradient. Therefore, the overhead is $O(emn^2)$. In the case there is no dropout, this overhead become $O(emn)$. (Fig. 9, 10)

5.3 Secure Verifiability [37]

Zhang et al.(2020) propose 'A Privacy-Preserving and Verifiable Federated Learning Scheme', a privacy-preserving technique that innovates by the lossless gradient processing method proposed, as well as a novel verification mechanism based on the Chinese Remainder Theorem and Bilinear Aggregate Signature (BAS) [42].

BAS [42] is a technique enabling the verification of signatures on different messages of different entities. It consists of five algorithms: KeyGen, Sign, Verify, AggregateSign and AggregateVerify.

Within the Secure Verifiability technique, the first four algorithms are employed: the KeyGen is performed by the TPA, the AggregateSign, by the aggregator, and the Sign and Verify, by the clients.

The workflow of the scheme proposed by [37] involves five stages:

1. Initialization: The participants agree upon a neural network architecture. A trusted public key generator issues the parameters to be used in the training process (the learning rate η, k pair-wise co-prime positive integers and a multiplicative cyclic group generator g). Since the system makes use of the Paillier cryptosystem, corresponding keys are also generated, as well as a bilinear key signature x.

2. Model Training: Each participant uses Stochastic Gradient Descent to train its local model. The output of this stage is a gradient matrix, each column corresponding to a layer of the locally trained neural network.

3. Data Processing: Each participant i splits each layer j of its local gradient in $\lceil \frac{W_i^{(j)}}{k} \rceil$ parts. These parts, together with the k co-prime numbers as remainders, define a system of linear congruences that, via the Chinese Remainder Theorem, yields a unique solution, $\tilde{W}_i$ ($\frac{1}{k}$ the size of the original gradient). $\tilde{W}_i$ is encrypted and submitted to the aggregator, together with bilinear signature σ_i of $\tilde{W}_i$.

4. Aggregation: The server aggregates the submitted gradients and signatures. The latter is produced by making use of the bilinear map e: $e(g, \sigma) = e(g, \prod_{i=1}^{n} \sigma_i)$

5. ResDecVerify: The clients download σ and encrypted $\tilde{W}$ from the server. Upon decrypting, they check $e(g, \sigma) = e(g^x, h(\tilde{W}))$. If this holds, the local model parameters are updated, otherwise, the round of training is aborted.

In terms of security model, this approach assumes honest-but-curious clients and a malicious server. The server cannot obtain the plaintext gradients of the participants, due to the security guarantees of the Paillier cryptosystem (in particular, the secret key is not disclosed to the aggregator). In addition, due to the ireversibility of hash functions and the confidentiality of the bilinear key signature, the server cannot calculate the gradients by making use of clients' signatures. Moreover, the Proof of Aggregation is not falsifiable, because of the interdependency between the aggregated gradient and the signatures. Forging signatures in this endeavour is, once again, impossible, as the server has no information of the signing function or of the bilinear key. Hence, this technique prevents inference, model inversion and gradient forging attacks altogether.

A simulation experiment [37] of a multi-layer perceptron conducted using 70 000 images of MNIST dataset (with a training batch of 60 000), yields an accuracy of 97% in the proposed scheme, as opposed to Original-FL's 98% [38].

Let us consider D, the number of model parameters, n, the amount of clients in the FL environment, and e, the number of epochs of training. Each client will spend a time proportional to D for updating parameters and a time proportional to $\frac{D}{k}$ for encryption and decryption. Additionally, CRT costs k^2 [40]. Therefore, the complexity at the client is $O(c_1 D + c_2 \frac{D}{k} + k^2)$ per epoch. However, k is set to be constant (for example, the authors of [37] set it to 7, in their experiments). Therefore, for all practical purposes, the clients' overhead is $O(eD)$.

The server will spend, for each client, time as to aggregate $\frac{D}{k}$ gradients (we assume aggregating a gradient takes constant time), as well as time to aggregate one signature. Therefore, the computation complexity at the server is $O(e\frac{nD}{k})$, or $O(enD)$, by fixing k.

In terms of communication, each client will submit an order of $\frac{D}{k}$ encrypted gradients, while the aggregator will submit n times this number. The communication overhead will hence be $O(e\frac{nD}{k})$ at the server, while the client's is $O(e\frac{D}{k})$. Fixing k, these become $O(enD)$ and $O(eD)$, respectively.

5.4 Comparison of Verifiable Frameworks and discussion

Framework	VFL	VerifyNet	Secure Verifiability
Computational Complexity at Aggregator	$O(egn)$	$O(emn^2)$	$O(enD)$
Computational Complexity at Client	$O(em^2g)$	$O(emn)$	$O(eD)$
Communication Overhead at Aggregator	$O(egn)$	$O(enm)$	$O(enD)$
Communication Overhead at Client	$O(eg)$	$O(e(n+m))$	$O(eD)$
Accuracy Impact	94% on MNIST (95% for [38])	not assessed	97% on MNIST (98% for [38])
Security Model	malicious server, honest-but-curious clients, colluding up to $n-2$	honest-but-curious clients (colluding up to $t-1$) and server (malicious aptness)	honest-but-curious clients and a malicious server
Machine Learning Model	neural networks	neural networks	neural networks
Security Mechanisms	Lagrange Interpolation, Pseudo-random technology, Chinese Remainder Theorem	Diffie-Hellman, Homomorphic Hash Functions, Double-Masking, Secret-Sharing, Pseudo-random technology	Paillier encryption, Chinese Remainder Theorem, Bilinear aggregate signature

Table 2: Comparison between Federated Learning Privacy Preserving Techniques providing Aggregation Verifiability

All observations below are based on Table 2, containing the main takeaways from the individual analysis of each verifiable FL framework, with the literals defined as in the individual subsections.

In terms of computational complexity, at the aggregator, the three frameworks exhibit a linear dependence on the number of epochs of training and the length of the gradient vector. The dependency on the number of users within the environment is, once again, linear at VFL and Secure Verifiability, while VerifyNet shows a quadratic dependency. This is because of the resiliency to user dropout. For every user dropping out, which is assumed to be a constant fraction of the total number of users, the aggregator will need to compute $O(n)$ partial offsets. In case no users dropout during training, the dependence of the runtime on the user count becomes linear too.

On the client side, aside from the linear dependence on the rounds of training, we can see that every framework introduces a different overhead. In the case of VFL, the dependence is quadratic on the security parameter m, and linear on g. However, in practical settings, m is set to a constant value. Authors of [36] use values such as 3, 4 or 6 in their experiments. VerifyNet's client runtime will once again, aside from the number of gradients, also depend linearly on the number of users, because of users having to agree upon mutual PRG seeds with all their peers. Finally, the method proposed by [37] will only depend on the training rounds and length of the gradient vector.

Hence, VerifyNet introduces the largest computational overhead, within the three studied schemas. In theory, by setting the security parameter m to be constant at VFL, both VFL and Secure Verifi-

ability introduce a comparable overhead, both at client and at the aggregating server. However, practically, the overhead of Secure Verifiability will be much larger, because of the costly Paillier encryption (more computationally expensive than VFL's custom encryption scheme).

The communication cost, at all the three frameworks, depends linearly on the number of epochs of training, the number of users, and the number of communicated gradients. At the client, similarly, the cost is linearly dependent on the number of epochs of training and the number of communicated gradients. However, VerifyNet's communication will also depend on the number of users, since every online user sends the server $O(n)$ shares, addressing the offset introduced by the dropout clients. If no clients dropout, the communication overhead is similar at all three frameworks.

In practice, however, [37]'s method is expected to have the least communication overhead, because of the k-order dimensionality reduction through CRT. VerifyNet exhibits the largest communication costs, due to the multiple rounds of communication with the TPA, as the protocol specifies.

From an accuracy perspective, further study is necessary in order to assess how the three frameworks compare. Although they have all been tested with the same dataset, both the model and hardware used for testing, as well as the setup of the federated environment differed.

From a security standpoint, all three frameworks offer similar guarantees. They all assume similar adversarial model and aim to address gradient forging attacks. However, while VFL offers data-privacy guarantee for up to $n-2$ colluding users, VerifyNet only offers the same guarantee for up to $t-1$ colluding users (t - security parameter used in the secret sharing protocol), and the method proposed by [37] does not address coordinated attacks of users. In terms of the verifiability techniques, further study is necessary as to determine their efficiency and accuracy, for each framework. We have only chosen to present techniques where verification can be performed in constant time (i.e. independent from the number of users in the environment). Assessing how these methods could be spoofed is necessary in order to evaluate how the given security guarantees perform in practice. (e.g. [14] argues that falsifying verification is equivalent to solving an NP-HARD problem.)

6 Responsible Research

The six approached privacy-preserving techniques (and the respective research papers) have been selected based on the following criteria:

- variety: the selected techniques exhibit diversity in their approaches, as to be able to present a full picture of the state-of-the-art of vertical federated learning and verifiability
- popularity: the papers approaching the selected techniques have a large number of citations. This metric is used to assess the community's support and recognition of the technique.
- accessibility: the papers that were surveyed are open access and can be consulted, if needed.

Additionally, all the other papers that were referenced throughout this study comply with the popularity and accessibility criteria described above.

All framework analysis and comparisons have been performed in a purely theoretical manner. The quoted experiments belong to the surveyed papers, and have been properly referenced. Where possible, the keypoints of the approached frameworks (computational overhead, communication costs, security model etc.) have been extracted from the respective research papers, and properly quoted. However, the surveyed papers do not always analyse the proposed frameworks according to our criteria. In these situations, we performed the analysis ourselves and documented our thinking process accordingly, in order to ensure the reproducibility of our methods. For example, methods used in assessing the computational complexity are the step-by-step analysis of the algorithm, and

inferring correlations between the variables and the runtime based on available experimental results. If additional papers were used in this process, these were accordingly referenced.

Privacy-preserving federated learning, as a subject of study, poses multiple ethical questions, related to data privacy, such as how is data stored, how is sensitive information protected, whether the right to be forgotten is ensured etc. This literature study, however, exclusively surveys the already existing federated learning solutions and does not attempt to address such matters.

Additionally, one should note that there has been no financing for the presented study, and no conflicts of interest that could have biased the researchers.

We consider this study to bring a positive impact to ethical computer science, as it aims to provide a short, but comprehensive guide to help those interested in vertical FL/verifiable frameworks to choose a solution well-suited for their use case, with performance and security implications in mind.

7 Conclusions and Future Work

In this paper, we have analysed six remarkable privacy-preserving techniques (three for vertical FL, and three providing aggregation verifiability), both from a security and a performance perspective. SecureBoost and MP-FEDXGB show the tradeoff between homomorphic encryption based and secret sharing based tree-boosting, in vertical FL environments. FedV, on the other hand, improves Hardy's [28] technique, being the first lossless vertical FL framework for models supporting SGD. VerifyNet is the first solution to provide both verifiability and user dropout resilience, with SS and homomorphic hash functions. We have also seen how VFL exploits Lagrange Interpolation and innovates through its custom encryption scheme, while [37]'s robust approach provides verifiability through Bilinear Aggregate Signature, at the cost of using an expensive encryption scheme.

We have also performed a comparative analysis of the two sets of privacy-preserving techniques. To this end, we compared their security models, the computational/communication complexity, their accuracy impact, the employed security mechanisms etc. Undoubtedly, there is no universal best privacy-preserving technique. We have highlighted the similarities and differences in the chosen range of techniques from a theoretical standpoint. However, our analysis could further be enhanced by a standardized, experimental analysis of the proposed frameworks. To this end, an evaluation with a unified dataset, on similar hardware would provide insight as to how the presented techniques compare, practically.

The future direction of research, as spanned by this paper, should be, first of all, pointed towards the development of a verifiability technique within a vertical federated learning environment. To our knowledge, no solution tackling vertically partitioned data can currently protect against a malicious aggregator. This is inherent to the fact that, as seen in the analysed algorithms, the aggregator within vertical FL is itself a client in the network, in most of the cases.

Secondly, another direction of study should be pointed towards enhancing the security guarantees of the studied frameworks. Surprisingly, none of the approached techniques offer direct support for implementing differential privacy. So, for example, a study initiative is including DP within these frameworks, and assessing what the trade-off between accuracy and data privacy is. This should more readily be a priority in the case of frameworks where vulnerabilities were identified, for example in HE based FEDXGB (SecureBoost).

To boot, another interesting initiative, as a followup of this study, is a survey into the privacy and security guarantees offered by each of the presented frameworks in a practical setting, through an adversarial lens, i.e. by simulating attacks on the respective federated environments. This would, once again, come as a step towards the process of standardizing a framework within which FL schemes can be impartially compared.

References

[1] *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. Apr. 2017. URL: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[2] Dragan Savic and Mladen Veinovic. "Challenges of General Data Protection Regulation (GDPR)". In: Jan. 2018, pp. 23–30. DOI: 10.15308/Sinteza-2018-23-30.

[3] Alejandro Baldominos, Yago Saez, and Pedro Isasi. "A Survey of Handwritten Character Recognition with MNIST and EMNIST". In: *Applied Sciences* 2019 (Aug. 2019), p. 3169. DOI: 10.3390/app9153169.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (2017), 84â90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[5] Md. Zahangir Alom et al. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches". In: *CoRR* abs/1803.01164 (2018). arXiv: 1803.01164. URL: http://arxiv.org/abs/1803.01164.

[6] Quoc V. Le et al. "Building high-level features using large scale unsupervised learning". In: *CoRR* abs/1112.6209 (2011). arXiv: 1112.6209. URL: http://arxiv.org/abs/1112.6209.

[7] *The Google DeepMind Challenge Match, March 2016*. URL: https://deepmind.com/alphago-korea.

[8] Qiang Yang et al. "Federated Machine Learning: Concept and Applications". In: *CoRR* abs/1902.04885 (2019). arXiv: 1902.04885. URL: http://arxiv.org/abs/1902.04885.

[9] Viraaji Mothukuri et al. "A survey on security and privacy of federated learning". In: *Future Generation Computer Systems* 115 (2021), pp. 619–640. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0167739X20329848.

[10] Malhar S. Jere, Tyler Farnan, and Farinaz Koushanfar. "A Taxonomy of Attacks on Federated Learning". In: *IEEE Security Privacy* 19.2 (2021), pp. 20–28. DOI: 10.1109/MSEC.2020.3039941.

[11] Stacey Truex et al. "A Hybrid Approach to Privacy-Preserving Federated Learning". In: *CoRR* abs/1812.03224 (2018). arXiv: 1812.03224. URL: http://arxiv.org/abs/1812.03224.

[12] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79228-4.

[13] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: *CoRR* abs/1704.03578 (2017). arXiv: 1704.03578. URL: http://arxiv.org/abs/1704.03578.

[14] Guowen Xu et al. "VerifyNet: Secure and Verifiable Federated Learning". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 911–926. DOI: 10.1109/TIFS.2019.2929409.

[15] Florian TramÃšr et al. *Ensemble Adversarial Training: Attacks and Defenses*. 2020. arXiv: 1705.07204 [stat.ML].

[16] Runhua Xu et al. "FedV: Privacy-Preserving Federated Learning over Vertically Partitioned Data". In: *CoRR* abs/2103.03918 (2021). arXiv: 2103.03918. URL: https://arxiv.org/abs/2103.03918.

[17] Abbas Yazdinejad et al. “P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking”. In: *Computers Security* 88 (2020), p. 101629. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.101629. URL: https://www.sciencedirect.com/science/article/pii/S0167404819301762.

[18] Keith Bonawitz et al. *Towards Federated Learning at Scale: System Design*. 2019. arXiv: 1902.01046 [cs.LG].

[19] Runhua Xu et al. “HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning”. In: *CoRR* abs/1912.05897 (2019). arXiv: 1912.05897. URL: http://arxiv.org/abs/1912.05897.

[20] Lingjuan Lyu et al. “Privacy and Robustness in Federated Learning: Attacks and Defenses”. In: *CoRR* abs/2012.06337 (2020). arXiv: 2012.06337. URL: https://arxiv.org/abs/2012.06337.

[21] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. “Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning”. In: *CoRR* abs/1702.07464 (2017). arXiv: 1702.07464. URL: http://arxiv.org/abs/1702.07464.

[22] Vale Tolpegin et al. “Data Poisoning Attacks Against Federated Learning Systems”. In: *CoRR* abs/2007.08432 (2020). arXiv: 2007.08432. URL: https://arxiv.org/abs/2007.08432.

[23] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology — EUROCRYPT ’99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238. ISBN: 978-3-540-48910-8.

[24] Peter Kairouz et al. “Advances and Open Problems in Federated Learning”. In: *CoRR* abs/1912.04977 (2019). arXiv: 1912.04977. URL: http://arxiv.org/abs/1912.04977.

[25] Kewei Cheng et al. “SecureBoost: A Lossless Federated Learning Framework”. In: *CoRR* abs/1901.08755 (2019). arXiv: 1901.08755. URL: http://arxiv.org/abs/1901.08755.

[26] Lunchen Xie et al. “An Efficient Learning Framework For Federated XGBoost Using Secret Sharing And Distributed Optimization”. In: *CoRR* abs/2105.05717 (2021). arXiv: 2105.05717. URL: https://arxiv.org/abs/2105.05717.

[27] Rainer Schnell, Tobias Bachteler, and Jorg Reiher. “A Novel Error-Tolerant Anonymous Linking Code”. In: *SSRN Electronic Journal* (Jan. 2011). DOI: 10.2139/ssrn.3549247.

[28] Stephen Hardy et al. “Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption”. In: *CoRR* abs/1711.10677 (2017). arXiv: 1711.10677. URL: http://arxiv.org/abs/1711.10677.

[29] Michel Abdalla et al. “Multi-Input Functional Encryption for Inner Products: Function-Hiding Realizations and Constructions Without Pairings”. In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Cham: Springer International Publishing, 2018, pp. 597–627. ISBN: 978-3-319-96884-1.

[30] Michel Abdalla et al. “Simple Functional Encryption Schemes for Inner Products”. In: *Public-Key Cryptography – PKC 2015*. Ed. by Jonathan Katz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 733–751. ISBN: 978-3-662-46447-2.

[31] Gang Liang and Sudarshan S. Chawathe. “Privacy-Preserving Inter-database Operations”. In: *Intelligence and Security Informatics*. Ed. by Hsinchun Chen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 66–82.

[32] Donald Knuth. *The Art Of Computer Programming, Volume 2: Seminumerical Algorithms, 3/E*. vol. 2. Pearson Education, 1998. ISBN: 9788177583359. URL: https://books.google.ro/books?id=OtLNKNVh1XoC.

[33] G. R. BLAKLEY. “Safeguarding cryptographic keys”. In: *1979 International Workshop on Managing Requirements Knowledge (MARK)*. 1979, pp. 313–318. DOI: 10.1109/MARK.1979.8817296.

[34] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (Nov. 1979), 612â613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[35] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. “Membership Inference Attacks against Machine Learning Models”. In: *CoRR* abs/1610.05820 (2016). arXiv: 1610.05820. URL: http://arxiv.org/abs/1610.05820.

[36] Anmin Fu et al. “VFL: A Verifiable Federated Learning with Privacy-Preserving for Big Data in Industrial IoT”. In: *CoRR* abs/2007.13585 (2020). arXiv: 2007.13585. URL: https://arxiv.org/abs/2007.13585.

[37] Xianglong Zhang et al. “A Privacy-Preserving and Verifiable Federated Learning Scheme”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148628.

[38] H. Brendan McMahan et al. “Federated Learning of Deep Networks using Model Averaging”. In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[39] Michael Hecht et al. “A Quadratic-Time Algorithm for General Multivariate Polynomial Interpolation”. In: (Oct. 2017).

[40] Rebecca N. Wright. “Cryptography”. In: *Encyclopedia of Physical Science and Technology (Third Edition)*. Ed. by Robert A. Meyers. Third Edition. New York: Academic Press, 2003, pp. 61–77. ISBN: 978-0-12-227410-7. DOI: https://doi.org/10.1016/B0-12-227410-5/00843-7. URL: https://www.sciencedirect.com/science/article/pii/B0122274105008437.

[41] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. “SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud”. In: *CoRR* abs/1706.10268 (2017). arXiv: 1706.10268. URL: http://arxiv.org/abs/1706.10268.

[42] Dan Boneh et al. “Aggregate and Verifiably Encrypted Signatures from Bilinear Maps”. In: *Advances in Cryptology — EUROCRYPT 2003*. Ed. by Eli Biham. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 416–432.

A Figures

Algorithm 1 Secure Training Framework

function SecureTrain(X,y,$\hat{y}$,T)
// Inputs: instance set X_m for each P_m; label y and prediction $\hat{y}$ from P_1; tree number T.
$Trees$ = [] // Initialize empty tree list.
$\hat{y} \leftarrow 0_{N\times 1}$
for $t = 1, 2, ..., T$ **do**
  $tree_t \leftarrow$ **SecureFit**(X, y, $\hat{y}$) // Generate T trees.
  $\hat{y}_t \leftarrow tree_t$.**SecurePredict**($X$)
  if in P_1 **then**
    $\hat{y} \leftarrow \hat{y} + \hat{y}_t$ // Only P_1 have valid predictions, thus it updates final predictions.
  end if
  $Trees.append(tree_t)$
end for
return $Trees$

Figure 1: MP-FEDXGB. Algorithm 1 - SecureTrain. Source: [26]

Algorithm 2 Fit the data

function SecureFit(X,y,$\hat{y}$)
// Inputs: instance set X_m for each P_m; label y and prediction $\hat{y}$ from P_1.
$s \leftarrow 1_{N\times 1}$ // Generate the initial indicator vector.
if in P_1 **then**
  $g, h \leftarrow \frac{\partial l(y,\hat{y})}{\partial \hat{y}}, \frac{\partial^2 l(y,\hat{y})}{\partial \hat{y}^2}$
  $\langle g \rangle, \langle h \rangle, \langle s \rangle \leftarrow$ **SHR**(g), **SHR**(h), **SHR**(s)
end if
Tree $\leftarrow$ **SecureBuild**($\langle g \rangle, \langle h \rangle, \langle s \rangle$)
return Tree

Figure 2: MP-FEDXGB. Algorithm 2 - SecureFit. Source: [26]

Algorithm 3 Secure Tree Building

function SecureBuild($\langle \boldsymbol{g} \rangle$,$\langle \boldsymbol{h} \rangle$,$\langle \boldsymbol{s} \rangle$,$\boldsymbol{X}$)
// Inputs: gradient vector $\langle \boldsymbol{g} \rangle^m$, hessian vector $\langle \boldsymbol{h} \rangle^m$, indicator vector $\langle \boldsymbol{s} \rangle^m$, and instance set $\boldsymbol{X}_m$ for each P_m;
$\langle g^\Sigma \rangle, \langle h^\Sigma \rangle \leftarrow \sum_n^N \langle \boldsymbol{g} \rangle_n, \sum_n^N \langle \boldsymbol{h} \rangle_n$ // Sum up elements within the vector.
if max depth reached **then**
  $\langle w \rangle \leftarrow$ **SecureLeafWeight**($\langle g^\Sigma \rangle, \langle h^\Sigma \rangle$)
  return Tree($\langle w \rangle$)
end if
$\langle loss_n \rangle, \langle loss_d \rangle \leftarrow \langle g^\Sigma \rangle \otimes \langle g^\Sigma \rangle, \langle h^\Sigma \rangle + \langle \lambda \rangle$
$\langle G \rangle, \langle H \rangle \leftarrow$ **SecureAggBucket**($\langle \boldsymbol{g} \rangle, \langle \boldsymbol{h} \rangle, \boldsymbol{X}$) // Aggregated statistics share $\langle G \rangle_{J \times K}$, $\langle H \rangle_{J \times K}$
for $j = 1, 2, ..., J$ **do**
  $\langle g_L^\Sigma \rangle, \langle h_L^\Sigma \rangle \leftarrow 0, 0$
  for $k = 1, 2, ..., K$ **do**
    $\langle g_L^\Sigma \rangle, \langle h_L^\Sigma \rangle \leftarrow \langle g_L^\Sigma \rangle + \langle G \rangle_{j,k}, \langle h_L^\Sigma \rangle + \langle H \rangle_{j,k}$
    $\langle g_R^\Sigma \rangle, \langle h_R^\Sigma \rangle \leftarrow \langle g^\Sigma \rangle - \langle g_L^\Sigma \rangle, \langle h^\Sigma \rangle - \langle h_L^\Sigma \rangle$
    $\langle \boldsymbol{\mathcal{G}}_L \rangle_{j,k}, \langle \boldsymbol{\mathcal{G}}_R \rangle_{j,k} \leftarrow \langle g_L^\Sigma \rangle \otimes \langle g_L^\Sigma \rangle, \langle g_R^\Sigma \rangle \otimes \langle g_R^\Sigma \rangle$
    $\langle \boldsymbol{\mathcal{H}}_L \rangle_{j,k}, \langle \boldsymbol{\mathcal{H}}_R \rangle_{j,k} \leftarrow \langle h_L^\Sigma \rangle + \langle \lambda \rangle, \langle h_R^\Sigma \rangle + \langle \lambda \rangle$
  end for
end for
$j^*, k^*, sign \leftarrow$ **SecureArgmax**($\langle \boldsymbol{\mathcal{G}}_L \rangle, \langle \boldsymbol{\mathcal{G}}_R \rangle, \langle \boldsymbol{\mathcal{H}}_L \rangle, \langle \boldsymbol{\mathcal{H}}_R \rangle, \langle loss_n \rangle, \langle loss_d \rangle$)
if *sign* is positive **then**
  if $P_{m'}$ have feature j^* **then**
    $val \leftarrow \boldsymbol{Q}_{j^*,k^*}^{m'}$//Quantile $\boldsymbol{Q}_{j^*,k^*}^{m'}$ records the k^*-th split bucket value for j^*.
    $\boldsymbol{s}_L \leftarrow 1_{N \times 1}$
    for $n = 1, ..., N$ **do**
      if $X_{n,j^*} > val$ **then**
        $\boldsymbol{s}_{Ln} \leftarrow 0$ // Set the n-th element to 0.
      end if
    end for
    $\boldsymbol{s}_R \leftarrow 1_{N \times 1} - \boldsymbol{s}_L$
    $\langle \boldsymbol{s}^L \rangle, \langle \boldsymbol{s}^R \rangle \leftarrow$ **SHR**($\boldsymbol{s}^L$), **SHR**($\boldsymbol{s}^R$) // Share the local indicator vectors to others.
  end if
  $\langle \boldsymbol{s}_L \rangle, \langle \boldsymbol{s}_R \rangle \leftarrow \langle \boldsymbol{s}_L \rangle \otimes \langle \boldsymbol{s} \rangle, \langle \boldsymbol{s}_R \rangle \otimes \langle \boldsymbol{s} \rangle$
  $\langle \boldsymbol{g}_L \rangle, \langle \boldsymbol{g}_R \rangle \leftarrow \langle \boldsymbol{g} \rangle \otimes \langle \boldsymbol{s}_L \rangle, \langle \boldsymbol{g} \rangle \otimes \langle \boldsymbol{s}_R \rangle$
  $\langle \boldsymbol{h}_L \rangle, \langle \boldsymbol{h}_R \rangle \leftarrow \langle \boldsymbol{h} \rangle \otimes \langle \boldsymbol{s}_L \rangle, \langle \boldsymbol{h} \rangle \otimes \langle \boldsymbol{s}_R \rangle$
  $branch_L \leftarrow$ **SecureBuild**($\langle \boldsymbol{g}_L \rangle, \langle \boldsymbol{h}_L \rangle, \langle \boldsymbol{s}_L \rangle, \boldsymbol{X}$)
  $branch_R \leftarrow$ **SecureBuild**($\langle \boldsymbol{g}_R \rangle, \langle \boldsymbol{h}_R \rangle, \langle \boldsymbol{s}_R \rangle, \boldsymbol{X}$)
  if $P_{m'}$ have feature j^* **then**
    return Tree($val, j^*, branch_L, branch_R$)
  else
    return Tree($branch_L, branch_R, Dummy$)
  end if
else
  $\langle w \rangle \leftarrow$ **SecureLeafWeight**($\langle g^\Sigma \rangle, \langle h^\Sigma \rangle$)
  return Tree($\langle w \rangle$)
end if

Figure 3: MP-FEDXGB. Algorithm 3 - SecureBuild. Source: [26]

Dataset	Depth	XGBoost			MP-FedXGB			MP-FedXGB*		
		ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC
Dataset1	3	0.9319	0.1354	0.8020	0.9324	0.1914	0.8410	0.9331	0.1824	0.8255
	4	0.9316	0.1818	0.8302	0.9321	0.2253	0.8377	0.9317	0.2099	0.8342
	5	0.9327	0.2337	0.8313	0.9329	0.2481	0.8379	0.9323	0.1962	0.8397
Dataset2	3	0.8455	0.6192	0.8639	0.8358	0.5965	0.8850	0.8332	0.5954	0.8844
	4	0.8500	0.6415	0.8844	0.8423	0.6403	0.8940	0.8404	0.6156	0.8936
	5	0.8561	0.6447	0.9014	0.8449	0.6481	0.8966	0.8424	0.6420	0.8945

Figure 4: MP-FEDXGB. Comparison with respect to Different Models under Different Datasets and Depths. Source: [26]

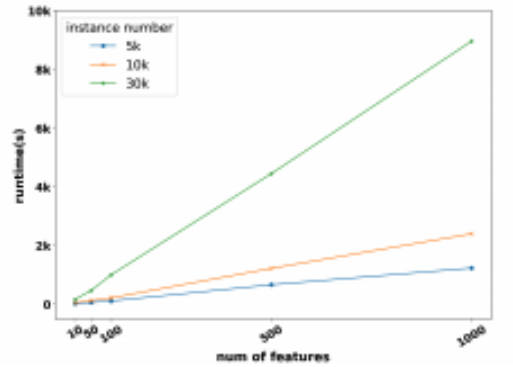
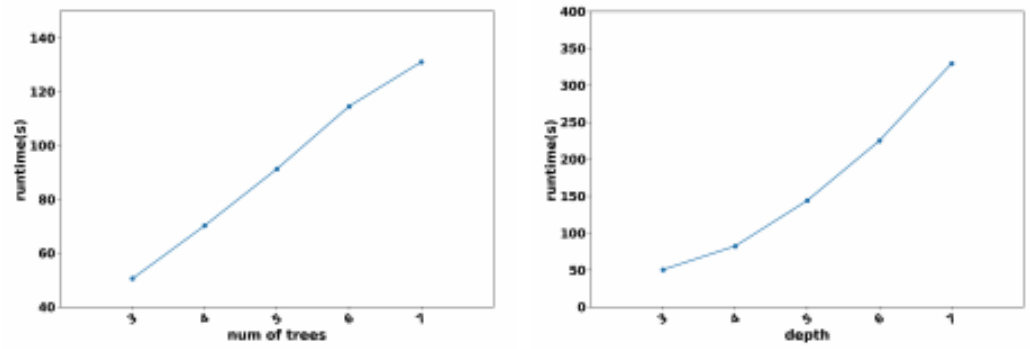


Figure 5: MP-FEDXGB. Relationships with respect to different instance sizes, feature sizes and runtime. Source: [26]



(a) Number of Trees v.s. runtime (b) Max depth v.s. runtime

Figure 6: MP-FEDXGB. Runtime in Different Tree Structures, Feature Numbers. Source: [26]

Algorithm 1 Gradient encryption algorithm.

Input:

Private gradient w_i, Pseudo-random generator $PRG(\cdot)$;

Two seed sequences $\left\{s_i^j\right\}$and $\left\{s_j^i\right\}$, $(j \in N, j \neq i)$;

Two constant sequences $\{a_j\}$ and $\{b_j\}$, $(j = 1, 2, ..., m)$.

Output:

Gradient ciphertext $[\![w_i]\!]$;

1: Blind the gradient w_i :
$w_i^{(j),t} \leftarrow PRG(s_i^{(j)})^t$; $w_j^{(i),t} \leftarrow PRG(s_j^{(i)})^t$;
$\widetilde{w_i} \leftarrow w_i - \sum_{j=1,j\neq i}^{n} w_i^{(j),t} + \sum_{j=1,j\neq i}^{n} w_j^{(i),t}$;

2: Randomly select $m-1$ parameters, which satisfy:
$\widetilde{w_i} = \sum_{j=1}^{m-1} v_{i,j}$;

3: Lagrange interpolation is performed on the data set $\{(a_1, v_{i,1}), (a_2, v_{i,2}), ..., (a_{m-1}, v_{i,m-1}), (a_m, A_i)\}$ to obtain the function $F_i(x)$:
$F_i(x) \leftarrow \sum_{j=1}^{m-1} \left[v_{i,j} \prod_{k=1,k\neq j}^{m} \frac{(x-a_k)}{(a_j-a_k)}\right] + \left[A_i \prod_{k=1}^{m-1} \frac{(x-a_k)}{(a_m-a_k)}\right]$;

4: Input $b_i (j = 1, 2, ..., m)$ into the function $F_i(x)$ and package the results to calculate the ciphertext $[\![w_i]\!]$:
$[\![w_i]\!] \leftarrow CRT\left[F_i(b_1), F_i(b_2), ..., F_i(b_m)\right]$;

5: Return $[\![w_i]\!]$.

Figure 7: VFL. Gradient encryption algorithm. Source: [36]

Algorithm 1 Gradient encryption algorithm.

Input:

Private gradient w_i, Pseudo-random generator $PRG(\cdot)$;
Two seed sequences $\left\{s_i^j\right\}$and $\left\{s_j^i\right\}$, $(j \in N, j \neq i)$;
Two constant sequences $\{a_j\}$ and $\{b_j\}$, $(j = 1, 2, ..., m)$.

Output:

Gradient ciphertext $[\![w_i]\!]$;

1: Blind the gradient w_i :
$w_i^{(j),t} \leftarrow PRG(s_i^{(j)})^t$; $w_j^{(i),t} \leftarrow PRG(s_j^{(i)})^t$;
$\widetilde{w_i} \leftarrow w_i - \sum_{j=1,j\neq i}^{n} w_i^{(j),t} + \sum_{j=1,j\neq i}^{n} w_j^{(i),t}$;
2: Randomly select $m-1$ parameters, which satisfy:
$\widetilde{w_i} = \sum_{j=1}^{m-1} v_{i,j}$;
3: Lagrange interpolation is performed on the data set $\{(a_1, v_{i,1}), (a_2, v_{i,2}), ..., (a_{m-1}, v_{i,m-1}), (a_m, A_i)\}$ to obtain the function $F_i(x)$:
$F_i(x) \leftarrow \sum_{j=1}^{m-1} \left[v_{i,j} \prod_{k=1,k\neq j}^{m} \frac{(x-a_k)}{(a_j-a_k)}\right] + \left[A_i \prod_{k=1}^{m-1} \frac{(x-a_k)}{(a_m-a_k)}\right]$;
4: Input $b_i (j = 1, 2, ..., m)$ into the function $F_i(x)$ and package the results to calculate the ciphertext $[\![w_i]\!]$:
$[\![w_i]\!] \leftarrow CRT\left[F_i(b_1), F_i(b_2), ..., F_i(b_m)\right]$;
5: Return $[\![w_i]\!]$.

Figure 8: VFL. Verification and decryption algorithm. Source: [36]

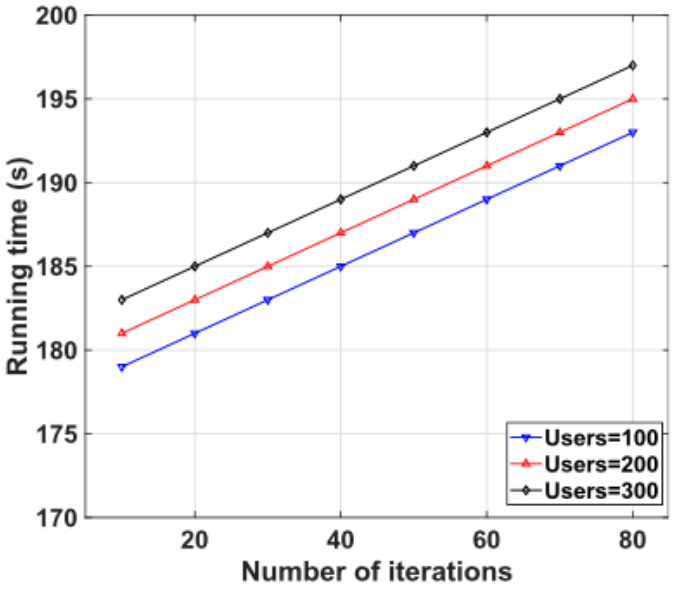


Figure 9: VerifyNet. Runtime dependence on number of iterations and number of users. Source: [14]

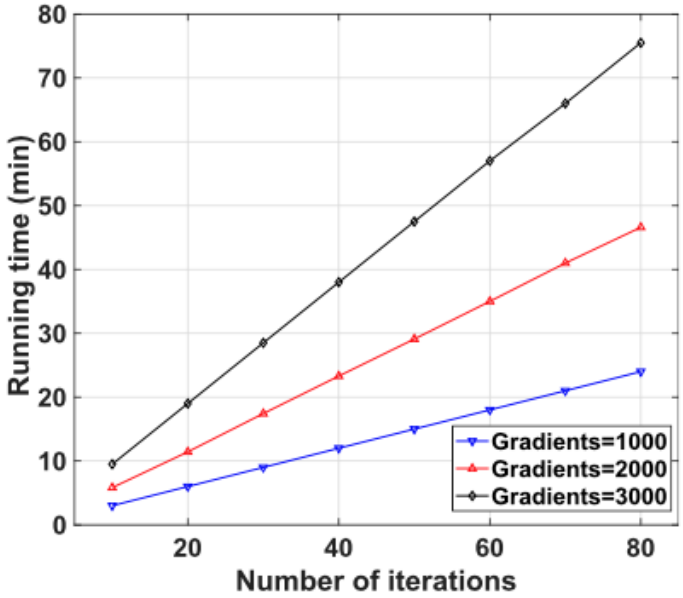


Figure 10: VerifyNet. Runtime dependence on number of iterations and number of gradients per user. Source: [14]

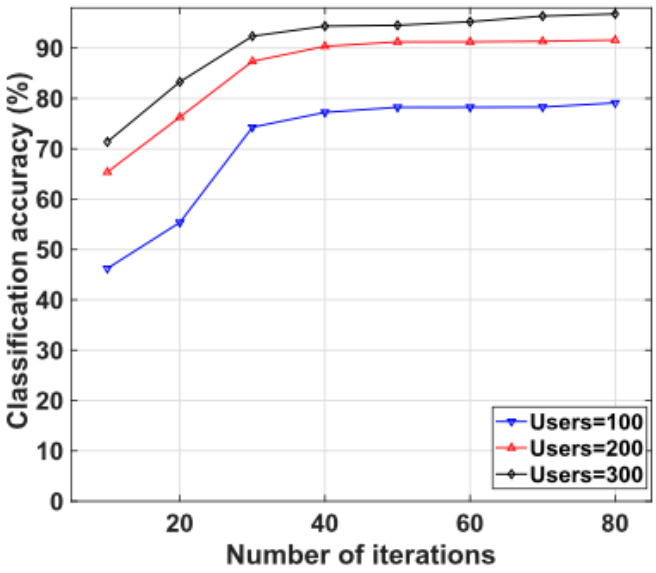


Figure 11: VerifyNet. Accuracy dependence on number of iterations and number of users. Source: [14]