Horizontal Federated Learning Frameworks: A Literature Study

MÁRTON SOOS[1] , **KAITAI LIANG**[1] , **RUI WANG**[1]
[1]TU Delft

Abstract

Federated Learning (FL)[1] is a type of distributed machine learning that allows the owners of the training data to preserve their privacy while still being able to collectively train a model.

FL is a new area in research and several challenges regarding privacy and communication cost still need to be overcome. Gradient leakage[1], for example is the possibility of partially reconstructing the private data of a participant based on the weight gradients they send over the network during FL, which poses a great risk for privacy. Mitigations against this problem lead to an increase in the computational complexity of the scheme or affect the performance of the resulting fully trained model. Other issues regarding trusting the central server or the clients, or accounting for clients that loose connection or drop out during training also exist.

This paper is a literature survey about frameworks for Horizontal Federated Learning (HFL), which is a subset of Federated Learning, in which all clients have the same type of data (same set of features). The survey presents a summary of how 7 different Horizontal Federated Learning frameworks work, and compares them in terms of their performance and the security guarantees they provide. Moreover, a summary of how each of the studied frameworks resolves the trade-offs among data privacy, framework performance and resulting model performance is also provided.

Based on the studied frameworks it is concluded that the privacy and performance issues of HFL still need to be researched. Suggestions for future research topics are also provided.

1 Introduction

Federated Learning is a distributed machine learning technique which allows training machine learning models without the need to centralize large amounts of training data and to compromise the privacy of the involved parties.

In the present, more and more software systems rely on machine learning models to meet the needs of humanity. However training these models requires large amounts of data, which is often collected at the expense of the privacy of the users.

Federated Learning is a highly researched topic nowadays, as it offers a way to train models on large volumes of data without compromising the privacy of any individual. However, the originally proposed algorithm for federated learning [1] is vulnerable to several attacks such as reconstruction through inference [2], which entails extracting private information from the weight gradients shared throughout the training process, or model or data poisoning [2] attacks, in which malicious participants send weight gradients constructed to prevent the convergence of the training process or to insert a backdoor into the resulting model.

Horizontal Federated Learning has been used in multiple fields such as: natural language processing [3], spoken language understanding [4], vision and language grounding problems [5], healthcare [6] [7], and IoT [8]. These applications all employ different frameworks for performing training in a HFL setup. These frameworks all make use of various privacy preserving techiques such as encrypting data with Homomorphic Encryption, adding noise to the transmitted weight gradients to ensure differential privacy, or removing the need for transmitting weight gradients altogether.

This paper aims to answer the following questions:

- How is Horizontal Federated Learning implemented?
- What privacy and performance trade-offs are made when designing HFL frameworks?
- How do HFL frameworks compare in terms of computational complexity, communication cost and privacy guarantees?

In order to offer a clear picture of how existing horizontal federated learning frameworks compare, this paper summarizes the algorithms behind 7 different HFL frameworks and compares them in terms of performance and security guarantees.

The paper will start with a section which provides background on the main concepts referenced throughout the document, followed by a section presenting an analysis of each of the studied implementations of Horizontal Federated Learning. Next, a section summarizing implementations of HFL

will follow. Finally, the reproduced results of a subset of the studied frameworks will be presented along with the conclusions of the study.

2 Background

This section presents the definitions of Federated Learning, Horizontal Federated Learning and other related concepts or algorithms that are mentioned in this paper.

2.1 Federated Learning

Federated Learning is a type of distributed machine learning which helps ensure data privacy. In federated learning the data on which the model is trained is stored on a number of data owners. The model can be trained on the entire dataset while each data owner keeps its data private.

Federated Learning is usually implemented with the client-server architecture. In this context, the data owners act as clients and a central server coordinates the training process. The server sends out the current version of the model (initially, a model with random parameters) to all clients (data owners). The clients train the model on their private data and return the trained model parameters (or the parameter adjustments) to the server. The server aggregates these by averaging the parameter values it receives from each client. This constitutes a round of federated learning. After multiple rounds, the model typically converges and performs similarly to a model that is trained on the aggregated private data of all data owners.

This algorithm is the simplest form of Horizontal Federated Learning, as described in [1]. In the report this algorithm will be referred to as Classic HFL.

2.2 Horizontal Federated Learning

Horizontal Federated Learning is a type of Federated Learning in which all data owners have similar types of data. Ideally, in an HFL setup, the samples on each client are different and have the same set of features. In practice, a scenario is considered to be a Horizontal Federated Learning scenario when the overlap between the feature sets of the data owners is larger than the overlap between the set of samples.

2.3 Homomorphic Encryption

Homomorphic Encryption is a type of encryption which allows mathematical operations to be performed on the ciphertexts. [9]

The benefit of homomorphic encryption is that "it can separate the ownership and processing rights of data". [10] In the context of federated learning it is used as a measure to protect against attackers gaining insight into the private data of the data owners by examining the gradients or parameters they return to the server. Using homomorphic encryption, these parameters can be encrypted and aggregated by the server without the server being able to obtain the parameter values or the aggregated result.

A homomorphic encryption scheme that is often used in FL, is the Paillier scheme. The homomorphic property of this scheme is that the product of two ciphertexts is decrypted to the sum of the two corresponding plain text values. [11] In other words:

$$E_k(m_1) * E_k(m_2) = E_k(m_1 + m_2) \tag{1}$$

where E_k is the encryption function using the public key k.

2.4 Differential Privacy

Differential privacy is a security guarantee that ensures that no risk is incurred by joining a statistical database. More precisely, ϵ-differential privacy ensures that it is only possible to determine whether or not an element is part of a dataset with probability of at most ϵ based on an exposed summary of the dataset.

According to Dwork [12], differential privacy has the following formal definition:

A randomized function K gives ϵ differential privacy if for all datasets D_1 and D_2 differing on at most one element, and all $S \subset Range(K)$:

$$P[K_1(D_1) \in S] \leq exp(\epsilon) \times P[K(D_2) \in S] \tag{2}$$

In most scenarios, differential privacy is achieved by adding noise (usually Gaussian or Laplacian noise) to the exposed summary. The amount of added noise depends on the desired level of privacy (ϵ) and the sensitivity of the summarizing function.

In the case of federated learning, the summarizing function maps the private data of each user to the set of gradients returned by it to the main server. By adding noise to these gradients it becomes increasingly difficult to reach conclusions about the underlying private samples of each participant.

3 Methodology

In order to achieve the goals defined in the introduction and to answer the research questions posed a number of research papers presenting HFL frameworks were read. Each of these frameworks has been analyzed theoretically in terms of time complexity and in terms of communication cost. These have been determined based on the description of the algorithms or the provided pseudocode and were expressed in terms of key parameters of the HFL setup (such as the number of involved clients, the number of samples per client or the number of training rounds). Moreover, the privacy guarantees of each framework and the attacks that they are vulnerable to have been determined and presented in the report.

For a selected subset of the studied frameworks a reproduction of their experimental results was attempted. The process of running these experiments and their results are presented in Section 6.

4 Frameworks

This section will present a summary of each of the studied frameworks, followed by an analysis of their computational complexity and privacy guarantees.

For each framework, the time complexity, as well as the number and size of exchanged messages will be presented for the participants/clients and, if applicable for the server(s). These will be expressed in terms of the variables defined in Table 1.

Variable	Definition
R	Number of rounds of Federated Learning
C	Number of clients / participants
N	Total number of samples of all participants
n	Number of samples per participant
M	Number of features
T / T_L / T_G	Training time of (local/global) model
S / S_L / S_G	Size of (local/global) model
E	Number of epochs of training (locally)

Table 1: Variables used when discussing computational complexity of frameworks

4.1 FederBoost [13]

FederBoost is a Horizontal and Vertical Federated Learning framework for training Gradient Boosting Decision Trees (GBDTs).

In FederBoost participants collaboratively train a model. At the beginning of the algorithm one of the participants is selected to act as the leader and be responsible of coordinating the training. In this section we refer to the selected leader as the server and to the other participants as the clients.

The main distinguishing characteristic of FederBoost is that during this protocol clients do not share any gradient or weight values thus only lightweight secure aggregation is required to ensure the privacy of the participants. This is due to the fact that the training process of GBDTs only relies on the order of the samples (and how these are split into buckets) and does not need any actual values from the training data sets.

As this literature study is about HFL, only the horizontal variant of FederBoost will be analyzed and described below.

Gradient Boosting Decision Trees

Gradient Boosting Decision Trees consist of a number of decision trees. As with other decision trees each internal node of the tree contains a condition which splits the samples between its children nodes, based on the value of a feature. Each leaf node contains a label.

When using a GBDT for classifying a sample, first the sample is passed through each decision tree and the values of the leaves to which the sample corresponds are summed. Based on this sum the sample is then classified into one of the predefined categories.

Training a GBDT

To train a GBDT one has to build a set of decision trees which together accurately classify the samples in the training data. Constructing a decision tree can be reduced to determining a set of conditions which split the training samples into leaves that are pure or as pure as possible. When training on a centralized dataset these conditions would be chosen such that they maximize the following expression in terms of first and second order gradients of the loss function:

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] \quad (3)$$

In practice the samples are sorted by each feature, split into buckets of consecutive samples and only splits between the buckets are considered. This reduces the amount of computations required to find a reasonably good split.

In FederBoost, the bucket construction is performed with the newly proposed Distributed Bucked Construction algorithm.

During Distributed Bucket Construction, the central server performs a binary search trying to find the cut-off point between the first and the second bucket (the largest value in the first bucket). At every iteration, it sends a proposed cut-off value to all the clients and they respond with the number of samples they have that would be part of the first bucket (are smaller than the proposed cut-off point). These values aggregated using a lightweight aggregation protocol similar to double masking. If the aggregated number of samples below the proposed cut-off point is too large or too small, the binary search continues in the expected manner and the process is applied in a similar fashion to find the subsequent cut-off points.

Note that for discreet features, a bucket is constructed for each of the possible values of the feature, and the aforementioned algorithm is not needed.

After determining the buckets, the sum of the first and second order gradients (G and H) of the samples in each bucket are computed and aggregated with lightweight secure aggregation. Then, each of the trees in the GBDT are constructed by the clients and the server collectively by finding the best splits for each node. The first tree is trained to predict as well as possible the target labels of the samples, subsequent trees all aim to predict the difference between the previous prediction and the actual labels.

Performance

After performing a theoretical analysis of the training process of FederBoost, the computational complexity and communication cost of the framework were determined in terms of the variables defined in Table 1, with the additional variables: q, Z, $|t|$ denoting the number of buckets, number of decision trees and the size of the decision trees used, respectively.

The time complexity for each client is $\mathcal{O}(C + M * (q * logN * (logn + C)) + Z * (M * n + |t| * M * q * C + n))$.

For the server, it is $\mathcal{O}(C + M * (q * logN * (logn + C)) + Z * (M * n + |t| * (M * q * C + C) + n))$.

The number of messages exchanged during the protocol is:

- $(C-1)^2$ messages sent for setting up secure aggregation
- $2 * M * q * logN * C$ messages sent for distributed bucket construction
- $Z * |t| * q * C$ messages sent for secure aggregation of G and H values
- $Z * |t| * (C - 1)$ messages sent for transmitting the best split to each client

Each message has a constant size ($\mathcal{O}(1)$).

Security Guarantees

FederBoost assumes honest-but-curious participants and server. The secure aggregation protocol used ensures the privacy of the local data. The privacy of any given participant

is only compromised if all other participants are malicious and colluding or the participant chosen to play the role of the server is malicious.

The convergence of the resulting model is only guaranteed if all participants follow the protocol, therefore the framework is vulnerable to data and model injection attacks.

In terms of robustness, the framework assumes that all participants are available at all times during the training process.

4.2 GRAFFL [14]

GRAFFL (from Gradient-free Federated Learning) is a Horizontal Federated Learning framework for training generative machine learning models without exchanging gradients or weights during training.

GRAFFL assumes that each participant has horizontally distributed data with possibly different distributions and intends to train a generative model that can reflect the sample diversity found in each client's dataset.

Training within GRAFFL is done through the ABC rejection sampling algorithm.

ABC rejection sampling

In summary, ABC rejection sampling is a process in which the central server proposes various parameters for the generative model it is training and evaluates these parameters by sampling the distribution defined by them and comparing these samples to summaries of the training samples of each client.

Before performing ABC rejection sampling, each client summarizes the samples in their dataset using an autoencoder, as described in the following section. Next, the central server chooses a set of parameters for the generative model and generates a number of samples for each of the parameters. The central server splits these samples up among the clients and sends them to them. Each client compares the received samples with their own summarized samples and reports back the value of a discrepancy metric between these. This process is repeated a number of times. Finally, the server selects the parameters with the lowest discrepancy values as the actual model parameters.

SuffiAE

In GRAFFL, clients make use of an autoencoder network called SuffiAE to summarize their training samples.

An autoencoder is a type of neural network that consists of two parts: the encoder and the decoder. The encoder receives samples of D dimensions (D features) and outputs summaries of d ¡ D dimensions. The decoder network reconstructs the original samples based on these summaries.

A sufficient autoencoder such as SuffiAE, has the additional properties that the summaries it outputs contain enough information to fully reproduce the original samples and these summaries cannot be converted back to the original samples unless the weights of the decoder are known.

By summarizing samples in this way, GRAFFL ensures that no private data is leaked, as the original samples are never directly compared to the generated samples, only their summaries are.

Performance

After performing a theoretical analysis on GRAFFL, the computational complexity and communication cost of the framework were determined in terms of the variables defined in Table 1 and the additional variables d and $T_{SuffiAE}$ denoting the number of dimensions of the summarized samples and the training time of SuffiAE, respectively.

The time complexity for each client is: $\mathcal{O}(C+T_{SuffiAE}+R*n*d)$. For the server, it is $\mathcal{O}(C+R*N*d+R*N*log(R*N))$.

The number of messages exchanged among the clients (to agree on d) is $C*(C-1)$. The size of these messages is of order $\mathcal{O}(1)$. The number of messages exchanged between the clients and the server is $2*R*C$. The size of these messages is of order $\mathcal{O}(d)$

Security Guarantees

Due to the fact that all samples are summarized with SuffiAE before training, the protocol protects the privacy of the participants' data against a malicious server and/or clients, even in case of collusion among them.

The protocol requires every client and the server to be available at all times during training.

4.3 SplitFed [15]

SplitFed is a machine learning framework which combines Federated Learning with Split Learning in order to obtain the benefits of both of these approaches.

By using Split Learning, it ensures an almost fully parallelizable training process and, by making use of Federated Learning (and differential privacy), it aims to increase the data privacy of the participants.

Split Learning [15] is a distributed machine learning technique which consists of splitting a neural network into two networks, one consisting of the first K layers and the other half containing the rest, and training the first half of the network in parallel on the clients that have the training data and training the latter half on a central server. The forward- and backpropagation are also done partially by the clients and partially by the server.

SplitFed's architecture consists of two servers and a number of clients. The clients own the data and train the client side network. One of the servers, referred to as the central server, plays the role of the server in split learning, having the responsibility of training the server side half of the network. The other server, referred to as the federated server, plays the role of the server in a federated learning setup and has the responsibility of aggregating the client side networks in a privacy preserving manner.

One training round consists of a number of forward- and backpropagation (across the clients and central server), followed by the aggregation of the local models on the federated server and updating the local models to the results of the aggregation.

Performance

After performing a theoretical analysis on SplitFed, the computational complexity and communication cost of the framework were determined in terms of the variables defined in

Table 1 and the additional variables S_C, L_S and L_C, denoting the size of the cut layer, the number of layers in the server side of the network and the number of layers in the client side of the network respectively.

The time complexity for each client is $\mathcal{O}(R * (L_C + S_C))$. For the central server, it is $\mathcal{O}(R * C * (L_S + S_C))$. For the federated server, it is $\mathcal{O}(R * C * L_C)$.

The number of messages exchanged between the clients and the central server is $2 * R * C$. The size of these messages is of order $\mathcal{O}(S_C)$.

The number of messages exchanged between the clients and the federated server is $2 * R * C$. The size of these messages is of order $\mathcal{O}(L_C)$

Security Guarantees

The framework assumes honest-but-curious participants and servers. It protects the privacy of the data by adding noise to the shared smash data and local weight gradients. It does not defend against data or model poisoning attacks.

In case a client drops out during the protocol, training can continue without it.

4.4 Fusion Learning [16]

Fusion Learning is a one-shot, machine learning technique. One-shot, in this context, refers to the method requiring only a single communication round between the server and each client to fully train a model.

Fusion Learning aims to overcome the high communication overhead of the classic HFL algorithm, which requires an exchange of model parameters or gradients between the server and each client in every round.

When applying Fusion Learning, each client is required to send a model trained on their local dataset, along with the distributions of each of the features present in this dataset, to the server. The server then uses each client's distribution to generate samples that are characteristic to them, and labels these using the model sent by the same client. Then, these generated samples are used to train the final model on the server, without requiring any further communication.

Therefore, Fusion Learning can be seen as a variation of federated learning in which the clients send one, fully trained model to the server instead of training a model over multiple rounds and transmitting gradients repeatedly.

Kasturi, Ellore, and Hota [16] claim that Fusion Learning is able to achieve accuracies that are comparable to those achieved by the classic HFL algorithm. The paper presents several experiments and shows a list of accuracies achieved by models trained using Fusion Learning on various benchmark datasets. These experiments have been partially reproduced and are presented in more detail in Section 6

Performance

After performing a theoretical analysis on Fusion Learning, the time complexity and and communication cost have been determined in terms of the variables from Table 1, with the additional variable ng denoting the number of generated samples per client.

The time complexity for each client is $\mathcal{O}(M + T_L + S_L)$. For the server, it is $\mathcal{O}(C * (S_L + ng) + T_G)$.

During the protocol C messages are sent to the server and C messages are sent from the server to the clients.

The size of each message sent to the server is $\mathcal{O}(M + S_L)$ and the messages sent to the clients have the size $\mathcal{O}(S_G)$.

Security Analysis

This approach does not offer complete data privacy to its participants as the distribution of each of the features in the private dataset is shared by each client with the server.

The privacy issue inherent to this approach is that the client must provide enough information to the server to allow it to generate and label samples similar to the ones found in the client's private dataset. Although the private samples themselves are kept secret, a potentially large amount of private information can be obtained about the client by generating and analyzing samples that are similar to their private data.

4.5 Two-Phase Multi-Party Computation Enabled Privacy-Preserving Federated Learning [17]

Two-Phase Multi-Party Computation Enabled Privacy-Preserving Federated Learning is a federated learning algorithm which makes use of Two-Phase MPC to ensure the data privacy of every participant involved in training.

Two-Phase MPC is a novel approach to MPC that improves upon the previously used Peer-to-Peer MPC by decreasing the communication overhead of the protocol at the cost of its resilience to colluding malicious participants.

Peer-to-Peer MPC is a technique which allows secure computation over the private data of multiple parties, without requiring the parties to share this data.

In the context of federated learning it solves the problem of securely summing (or averaging) the gradient tensors of each participant.

This can be achieved using the Additive Secret-Sharing MPC Protocol, which works in the following way. In case of a network of N clients, every client generates a set of N random tensors which sum to their secret gradient tensor. These are the secret shares of the client, and they are given each to a separate client. Then, every client sums the secret shares it receives and broadcasts the sum to all other clients. Finally, every client sums the broadcasted tensors to obtain the result of the computation, which is equal to the sum of the secret tensors of each client.

A more secure alternative to Additive Secret-Sharing is Shamir's Secret Sharing Protocol [18], which works similarly but makes use of the multiplication operation defined over polynomial rings instead of using addition over real numbers. [17]

Two-Phase MPC

Two-Phase MPC consists of two phases. In the first phase, a committee of M clients is elected using Peer-to-Peer MPC (e.g. through random voting). In the second phase, each client only creates M secret shares and sends each of them to one committee member. The committee then aggregates the shares of all the clients using Peer-to-Peer MPC.

The number of messages exchanged in this protocol is lower than in the Peer-to-Peer case, as the first phase (which requires $C * (C - 1)$ messages to be exchanged) is only executed once. In every round of training fewer messages are

exchanged, as the secure MPC is performed within the committee of $K << C$ members.

Performance
After performing a theoretical analysis on Two-Phase Multi-Party Computation Enabled Privacy-Preserving Federated Learning, the computational complexity and communication cost of this framework were determined in terms of the variables from Table 1, with the additional variable K, denoting the number of committee members.

The computational cost for each client is $\mathcal{O}(C + R * (T + K * S))$. For committee members, it is $\mathcal{O}(C + R * (T + K * S + C * S + C/K * S))$.

During the protocol $2 * C * (C - 1)$ small messages and $R * C * K + R * K * (K - 1 + C/K)$ large messages are exchanged. The size of each small message is of order $\mathcal{O}(1)$ and the size of each large message is of order $\mathcal{O}(S)$.

Security Analysis
The Two-Phase MPC protocol provides data-privacy for each participant, assuming that not all committee members are malicious and colluding. The correctness of the protocol is only guaranteed if the participants are honest. The protocol can continue if participants outside the committee drop out, however the availability of committee members is required at all times.

4.6 FLOP: Federated Learning on Medical Datasets using Partial Networks [19]

FLOP is a FL framework which defends against inference attacks by only requiring clients to share a subset of the weights of their locally trained models.

The training process of FLOP is similar to the classic HFL algorithm's, with the only difference being the fact that the models trained on each client are split into two disjoint models: the private and the shared model. The private model is only trained on the private data of the client while the shared model is trained with federated averaging, on the data of all clients. The example presented in the [19] is an image classifier that is split up into a shared feature extractor and a private classifier.

Accuracy
Similarly to Fusion Learning, FLOP also alters how the resulting model is trained (beyond just optimizing the process of training it), therefore it is interesting to discuss the performance achieved by models trained with this framework.

The paper presenting FLOP includes experiments on both real life and benchmark datasets. Experiments on the benchmark datasets have been reproduced for this literature study and are presented in Section 6.2.

Performance
Based on a theoretical analysis of FLOP, its computational complexity and communication cost have been determined in terms of the variables from Table 1, with the additional variable S_S denoting size of the shared model.

The computational cost for each client is $\mathcal{O}(R * (T + S_S))$. For the server, it is $\mathcal{O}(R * C * S_S)$.

The number of messages exchanged during the protocol is $2 * R * C$. The size of each message is of order $\mathcal{O}(S_S)$.

Security Analysis
As FLOP is essentially a variation of the classic HFL algorithm in which only a subset of the model is shared, it will have at least the same privacy guarantees as this algorithm.

Yang et al. [19] claim that FLOP reduces privacy and security risks by only sharing a subset of the model, however the extent to which this is the case is not discussed or presented. More research is required to verify this claim and to quantify the impact that sharing a partial model has on privacy.

4.7 Paillier Federated Multi-Layer Perceptron (PFMLP) [20]

PFMLP is a federated learning framework that ensures the privacy of the participants by encrypting the weight gradients with a homomorphic encryption algorithm, specifically with an optimized version of the Paillier scheme.

Using homomorphic encryption in FL, has the great disadvantage of adding a large computational overhead to training. To address this, PFMLP proposes the use of the improved version of the Paillier scheme described in [21], which has the computational complexity of $\mathcal{O}(|n|^2\alpha)$, instead of the complexity of the naive Paillier scheme, which is $\mathcal{O}(|n|^3)$ (where n denotes the size of the encryption key and $\alpha < n$ is a parameter of the algorithm.

Another disadvantage of using a homomorphic encryption scheme is that it requires the presence of an additional entity, the key server, which is responsible with generating and distributing the encryption keys to the participants and must be trusted.

Training Process
Before the training starts, each client requests a key pair from the key server. Then, each client initializes their local model and training begins.

In round i of federated learning, each client trains their local model, encrypts the weight gradients with the public key of client i and sends them to the central server. The server aggregates the encrypted gradients and sends back the aggregated gradient to each client. The clients then use the private key of client i to decrypt the aggregated gradient and adjust their weights accordingly.

Performance
Based on a theoretical analysis of FLOP, its time complexity and communication cost have been determined in terms of the variables from Table 1, with the additional variables k and α, denoting the size of encryption key and a parameter of the encryption scheme respectively.

The time complexity for each client is $\mathcal{O}(R*(T+k^2*\alpha*S)$. For the server, it is $\mathcal{O}(R * C * S)$.

During the protocol $2*R*C$ large messages and $C+R*C$ small messages are exchanged.

The size of each large message is of order $\mathcal{O}(S)$.

The small messages are used to request the encryption keys from the key server and have a constant size.

Security Analysis
The PFMLP protocol provides privacy against an honest-but-curious server, keyserver and honest-but-curious clients, if

there is no collusion between the keyserver and any other involved party.

The availability of the server, keyserver and clients is required for the entire duration of the training process.

5 Framework comparisons

The studied frameworks make various trade offs among model accuracy, data privacy and performance (training time and communication cost). This section will briefly summarize the advantages and disadvantages of each framework by comparing them with each other.

FederBoost has low time complexity as it leverages properties of GBDTs to simplify the training process. However, it has a relatively high communication cost due to the secure aggregation algorithm it uses. It is also limited to only training GBDTs. To address its high communication cost, using homomorphic encryption instead of the lightweight secure aggregation could be considered. As the messages exchanged are relatively small in size the encryption will likely not add a significant overhead to the algorithm, but will reduce the communication cost.

GRAFFL has the strongest privacy guarantees out of all the algorithms. This privacy is given by autoencoder network it uses. However, it has a relatively high communication cost and it is limited to only training parametric generative models.

SplitFed is one of the more efficient frameworks presented, however it does not offer privacy protection against inference attacks, therefore it is not suitable in most situations, as most use cases of FL require data privacy.

Fusion Learning is an efficient framework in terms of communication cost, however it exposes the distribution of the private client data, which compromises the privacy of the participants and is unacceptable in most situations. It also has the disadvantage of training the final global model on a single machine, which is inefficient compared other frameworks.

Two-phase MPC is more efficient than the classic Peer-to-Peer MPC, however it is still slower than most of the other studied frameworks due to its high communication cost. It has the advantage of not requiring a central server and doing all computations in a distributed fashion within a peer-to-peer network and does not sacrifice the privacy of its participants.

FLOP is promising in terms of the apparent amount of privacy it provides at little to no cost in model accuracy or computational performance [19], however it needs a more in depth security analysis to determine the precise security and privacy guarantees it offers.

PFMLP offers protection against gradient leakage without sacrificing the accuracy of the final model. However, even though it uses an improved version of Paillier encryption, it is still comparatively slow to other studied frameworks (FederBoost, FLOP). Perhaps applying this encryption algorithm to frameworks which exchange little data would result in a faster HFL framework with high privacy guarantees.

Tables 2, 3, 4 and 5 summarize the results of the analysis performed for this literature study. The variables used in the tables are defined in the previous sections and are listed in Appendix A.

Framework	Time Complexity
Classic HFL	$O(R*(T+C*S))$
FederBoost	$O(M*logN*(logn+C))+M*n+M*C)$
GRAFFL	$O(C+T_{SuffiAE}+R*n*d+R*N*log(R*N))$
SplitFed	$O(R*(S_C+L_S+C*L_C))$
Fusion Learning	$O(M+T_L+C*(S_L+ng)+T_G)$
Peer-to-Peer MPC	$O(R*(T+C*S))$
Two-Phase MPC	$O(C+R*(T+K*S+C*S+C/K*S))$
FLOP	$O(R*(T+C+S_S))$
PFMLP	$O(R*(T+k^2*\alpha*S+C*S))$

Table 2: Comparison of time complexities of the studied frameworks

Table 2 contains the total time complexities of the algorithms, combining the client and server side time complexities and accounting for operations that happen in parallel. Some of the complexities have been expressed in terms of fewer variables to simplify comparisons, and algorithms such as the classic HFL algorithm and the Peer-to-Peer MPC based HFL algorithm have been added for reference.

Framework	Communication cost	Messages sent
Classic HFL	$O(R*C*S)$	$2*R*C$
FederBoost	$O(C^2+2*M*logN*C)$	$(C-1)^2+Z*\|t\|*(C-1)+$ $2*M*q*logN*C+Z*\|t\|*q*C$
GRAFFL	$O(C^2+R*C*d)$	$C*(C-1)+2*R*C$
SplitFed	$O(R*C*(S_C+L_C))$	$4*R*C$
Fusion Learning	$O(C*(S_L+M)+C*S_G)$	$2*C$
Peer-to-Peer MPC	$O(C^2*S)$	$R*C*2*(C-1)$
Two-Phase MPC	$O(C^2+(R*C*K+$ $R*K^2*S)$	$2*C*(C-1)+R*(C*K+$ $K*(K-1+C/K))$
FLOP	$O(R*C*S_S)$	$2*R*C$
PFMLP	$O(R*C*S)$	$4*R*C+C$

Table 3: Comparison of communication cost of studied frameworks

The communication cost in Table 3 is the order of the number of bytes exchanged in the protocols.

Framework	Privacy-preserving measure	Ensures privacy against
Classic HFL	None	None
FederBoost	Simplified Masking Protocol	Honest-but-curious clients
GRAFFL	Using summaries of private data Not sharing gradients/weights	Fully-dishonest, curious (and colluding) server and/or clients
SplitFed	Differential Privacy	Fully-dishonest, curious (and colluding) server and/or clients
Fusion Learning	Not sharing gradients/weights	None
Peer-to-Peer MPC	Secure MPC	Honest-but-curious participants without collusion
Two-Phase MPC	Secure MPC	Honest-but-curious participants without collusion
FLOP	Only sharing part of model	Unknown
PFMLP	Homomorphic Encryption	Honest-but-curious server, keyserver and clients without collusion

Table 4: Comparison of privacy measures and guarantees of the studied frameworks

Framework	Supported models
Classic HFL	Neural Networks
FederBoost	GBDTs
GRAFFL	Parametric Generative Models
SplitFed	Neural Networks
Fusion Learning	Any Model
Peer-to-Peer MPC	Neural Networks
Two-Phase MPC	Neural Networks
FLOP	Neural Networks
PFMLP	Neural Networks

Table 5: Comparison of the rage of models supported by the studied frameworks

In conclusion the main trade-offs made when choosing a Federated Learning Framework are among the final model's accuracy, the performance (training time and communication cost) and the security guarantees of the protocol.

6 Reproduced Results

For two of the aforementioned frameworks, experiments have been reproduced. This section presents the setup of these experiments and the obtained results.

The two studied frameworks are Fusion Learning and FLOP. In the case of both frameworks, experiments which assess the accuracy and loss of the trained models have been reproduced. This was achieved by simulating the behavior of clients and servers within one Python script.

The advantage of this approach is that the results can easily be reproduced, given the fact that the experiments can be run locally, on a single machine, without requiring a network of computers. Moreover the resulting trained models will be equivalent to the models trained by distributed implementations of these frameworks, therefore allowing the analysis of model performance obtained with these frameworks.

The main disadvantage of this approach is that no conclusions can be drawn regarding the runtime of the training or the communication cost of these frameworks. For these aspects of the framework, a theoretical analysis has been conducted and presented in the previous section of the report.

6.1 Fusion Learning experiments

In [16] four experiments are presented, which apply Fusion Learning to train a model on four different benchmark datasets: the Credit Card, Breast Cancer, Gender Voice and Audit datasets from [22]. Unfortunately the Gender Voice dataset was not present in the aforementioned repository and thus the experiment on it could not be reproduced.

To reproduce these experiments a simulation of this framework has been implemented in Python. The code for this simulation can be found on GitHub [23].

The simulation first randomly splits the training set into 10 subdatasets, which represent the private datasets of the 10 clients. Then, for each subdataset individually, the best fitting distribution is determined for each feature, according to the Kolmogorov–Smirnov test. Next, 10 models are trained, one on each of the subdatasets. These models represent the local models of the 10 clients participating in the protocol.

Following this, the central server's behavior is simulated. First, for each of the 10 clients, 1000 samples are generated by sampling the best fit distributions found previously. Each of these samples is labeled as classified by the model trained on its corresponding client's dataset. Finally, a model is trained on the 10000 generated samples and the accuracy of the model is measured on the test dataset.

In order to achieve the accuracies presented in [16], a slight deviation from the original experiment was required, namely, the use of upsampling to combat the imbalanced datasets.

The reproduces accuracies are presented in Table 6.

Dataset	Original Accuracy	Reproduced Accuracy
Credit Card	81.09	78
Breast Cancer	95.62	95.91
Audit Data	97.42	97.4359

Table 6: Fusion Learning: Reproduced results compared to original results

Interestingly, the results of the experiment on the credit card dataset could not be reproduced. The models trained with the aforementioned simulator achieved around 78% accuracy, which is an unacceptable performance considering that only 22% of the samples have the positive label (and the rest have the negative label).

Based on these results we can conclude that Fusion Learning can indeed be used to train machine learning models that have a comparable performance to the original Horizontal Federated Learning framework. However, some of the results presented in [16] cannot be reproduced, which suggests that some implementation details of the experiment might have been omitted from the paper.

6.2 FLOP experiment

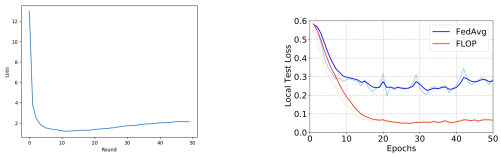
Experiments on the FLOP framework have also been conducted. Specifically, the experiment performed on the Fashion MNIST dataset that is presented in [19] has been reproduced. The FLOP framework has been simulated locally, in one Python script, similarly to how the Fusion Learning experiments have been conducted. The experiment's code can be found on GitHub [24].

To compare the results of the reproduced experiment with the results of the original one, in every FL round the test set loss of the clients is averaged and plotted in Figure 1.

The plot clearly shows that the model converges, confirming that a model can be trained with HFL while only sharing a subset of the weights. When comparing this plot with the plot in the original paper, shown in Figure 1. it can be seen that the range of values is different, indicating a possible difference in the way it is computed.

7 Responsible Research

The main aspect of responsible research that relevant to literature studies containing experiments is reproducibility. To ensure the reproducibility of the presented experiments,



(a) Loss in reproduced experiment

(b) Loss in original experiment

Figure 1: Loss over epochs in FLOP experiment

the code written for each experiment has been published to GitHub and a detailed description of how the experiment can be reproduced has been provided in the repositories, along with a list of versioned dependencies.

This literature study will have a positive impact on ethical computer science, as it is intended to help developers or researchers compare HFL frameworks and choose the most suitable one for their use-case, thus encouraging the use of FL. Using FL is more ethical than collecting user data in most scenarios, as it honours the value of individual privacy by not collecting or forcing the users to expose their private data to participate in training a model.

8 Conclusions and Future Work

In conclusion, there are multiple approaches to achieving secure federated learning. Some improve upon previously developed frameworks (such as Two-Phase MPC Enabled Federated Learning), others exploit particularities of certain classes of models to increase privacy (e.g. FederBoost).

These frameworks each have their own advantages and disadvantages. The trade-offs made by them can be viewed as trade-offs among the following three factors: privacy of individual users, performance/accuracy of the resulting model and performance of framework (runtime and communication overhead). The various techniques employed by the studied frameworks can be used to shift the focus of FL from one factor, to another. For example, by adding various degrees of noise to the transmitted gradients, the trade-off between model accuracy and data privacy can be explored, while having little to no impact on the training time or communication cost. Employing Homomorphic Encryption, ensures that no information can be leaked through the weight gradients, while not affecting the quality of the model, so this shifts the focus of the framework to privacy and model accuracy, while sacrificing the performance of the overall framework.

The framework comparisons section presents how the studied frameworks fit in this model and which of the three aspects they emphasize or sacrifice. Undoubtedly, there is no single best framework for all use cases of FL, as the importance of each of the aforementioned factors varies based on the situation. The summaries and analysis presented in this paper are intended to help with determining which FL framework is most suitable for any given use case.

In the future it would be beneficial to develop and study more approaches to Federated Learning, especially approaches similar to FederBoost, which manage to achieve all of the three aforementioned desired qualities of a FL framework, by limiting the range of models that can be trained with it.

Moreover it would be valuable to further study the security and privacy guarantees offered by FLOP, as these cannot be trivially deduced from the description of the algorithm.

Appendix

A Variables used when discussing computational complexity of frameworks summarized

Variable	Definition	Specific to framework
R	Number of rounds of Federated Learning	
C	Number of clients / participants	
N	Total number of samples on all participants	
n	Number of samples owned by one participant	
M	Number of features	
T / T_L / T_G	Training time, training time of local/global model	
S / S_L / S_G	Size of model, size of local/global model	
E	Number of epochs of training (locally)	
q	Number of buckets / quantiles	FederBoost
Z	Number of decision trees in the GBDT	FederBoost
$\|t\|$	Size of a decision tree in the GBDT	FederBoost
d	Number of dimensions of summarized samples	GRAFFL
$T_{SuffiAE}$	Training time of SuffiAE	GRAFFL
S_C	Size of cut layer	SplitFed
L_S	Number of layers on the server side of the network	SplitFed
L_C	Number of layers on the client side of the network	SplitFed
ng	Number of generated samples	Fusion Learning
K	Number of committee members	Two-Phase MPC enabled FL
S_S	Size of shared model	FLOP
k	Size of encryption key	PFMLP
α	Parameter of encryption scheme	PFMLP

Table 7: Complete list of variables used when discussing computational complexity of frameworks and their definition

References

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

[2] Viraaji Mothukuri et al. "A survey on security and privacy of federated learning". In: *Future Generation Computer Systems* (Oct. 2020). DOI: 10.1016/j.future.2020.10.007.

[3] Andrew Hard et al. *Federated Learning for Mobile Keyboard Prediction*. 2019. arXiv: 1811.03604 [cs.CL].

[4] Zhiqi Huang, Fenglin Liu, and Yuexian Zou. "Federated Learning for Spoken Language Understanding". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 3467–3478. DOI: 10.18653/v1/2020.coling-main.310. URL: https://www.aclweb.org/anthology/2020.coling-main.310.

[5] Fenglin Liu et al. "Federated Learning for Vision-and-Language Grounding Problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (Apr. 2020), pp. 11572–11579. DOI: 10.1609/aaai.v34i07.6824.

[6] Xiaoxiao Li et al. "Multi-site fMRI Analysis Using Privacy-preserving Federated Learning and Domain Adaptation: ABIDE Results". In: *Medical Image Analysis* 65 (July 2020), p. 101765. DOI: 10.1016/j.media.2020.101765.

[7] Dashan Gao et al. *HHHFL: Hierarchical Heterogeneous Horizontal Federated Learning for Electroencephalography*. 2020. arXiv: 1909.05784 [eess.SP].

[8] Dinh C. Nguyen et al. "Federated Learning for Internet of Things: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* (2021), pp. 1–1. ISSN: 2373-745X. DOI: 10.1109/comst.2021.3075439. URL: http://dx.doi.org/10.1109/COMST.2021.3075439.

[9] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: *ACM Comput. Surv.* 51.4 (July 2018). ISSN: 0360-0300. DOI: 10.1145/3214303. URL: https://doi.org/10.1145/3214303.

[10] Xiaoxian Yang et al. "BFLP: An Adaptive Federated Learning Framework for Internet of Vehicles". In: *Mobile Information Systems* 2021 (2021), p. 6633332. DOI: 10.1155/2021/6633332. URL: https://doi.org/10.1155/2021/6633332.

[11] 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.

[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] Zhihua Tian et al. *FederBoost: Private Federated Learning for GBDT*. 2020. arXiv: 2011.02796 [cs.CR].

[14] Seok-Ju Hahn and Junghye Lee. *GRAFFL: Gradient-free Federated Learning of a Bayesian Generative Model*. 2020. arXiv: 2008.12925 [cs.LG].

[15] Chandra Thapa, M. A. P. Chamikara, and Seyit Camtepe. *SplitFed: When Federated Learning Meets Split Learning*. 2020. arXiv: 2004.12088 [cs.LG].

[16] Anirudh Kasturi, Anish Reddy Ellore, and Chittaranjan Hota. "Fusion Learning: A One Shot Federated Learning". In: *Computational Science – ICCS 2020*. Ed. by Valeria V. Krzhizhanovskaya et al. Cham: Springer International Publishing, 2020, pp. 424–436. ISBN: 978-3-030-50420-5.

[17] Renuga Kanagavelu et al. "Two-Phase Multi-Party Computation Enabled Privacy-Preserving Federated Learning". In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020, pp. 410–419. DOI: 10.1109/CCGrid49817.2020.00-52.

[18] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[19] Qian Yang et al. *FLOP: Federated Learning on Medical Datasets using Partial Networks*. 2021. arXiv: 2102.05218 [cs.LG].

[20] Haokun Fang and Quan Qian. “Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning”. In: *Future Internet* 13.4 (2021). ISSN: 1999-5903. DOI: 10 . 3390 / fi13040094. URL: https://www.mdpi.com/1999-5903/13/4/94.

[21] Taiwo Blessing Ogunseyi and Tang Bo. “Fast Decryption Algorithm for Paillier Homomorphic Cryptosystem”. In: *2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*. 2020, pp. 803–806. DOI: 10.1109/ICPICS50287.2020.9202325.

[22] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[23] *Fusion Learning Simulation*. URL: http://github.com/Marton6/fusion-learning/.

[24] *FLOP Simulation*. URL: http://github.com/Marton6/flop/.