An overview of hybrid approaches in Horizontal Federated Learning

EDUARD FILIP , **KAITAI LIANG** , **RUI WANG**
TU Delft

Abstract

Federated Learning starts to give a new perspective regarding the applicability of machine learning in real-life scenarios. Its main goal is to train the model while keeping the participants' data in their devices, thus guaranteeing the privacy of their data. One of the main architectures is the Horizontal Federated Learning, which is the most common one implemented. However, the challenges of the model (security attacks, data leakage), led to using some privacy-enhancing elements. Even those have their own trade-offs that make Federated Learning a challenge to implement in real-life scenarios (communication cost, training time).

One question may rise up based on the aforementioned challenges: What if there is a hybrid way of implementing the Federated Learning so that we can overcome the challenges of the present implementations and expand its potential? This paper aims to answer this by diving into five hybrid models that use a variety of components for preserving privacy (Differential Privacy combined with Secure Multiparty Computation, blockchain) and will be compared with each other. Based on that, a reader can get an overview of how the hybrid approach affects the evolution of Horizontal Federated Learning.

1 Introduction

Artificial Intelligence and Machine Learning reached a highly developed state in 2016. Yet, the models used until that moment were based on centralized training data. When it comes to real-life scenarios, the current models seem to face some new challenges. The new legal concerns about the users' private data such as GDPR [1] increase the difficulty of acquiring the necessary data to train the models. Another new challenge is the distribution of data. Organizations may want to use their data together to train models, however, the structure is not the same, and not centralized [2]. Google introduced in 2016 the Federated Learning (FL) model, which overcomes these challenges in the industry [3].

This model can be categorized based on the characteristics of the data: Horizontal Federated Learning (HFL), Vertical Federated Learning (VFL) and Federated Transfer Learning (FTL) [4]. The horizontal model is used in the cases where the datasets of the participants share the same feature space, but the space in samples differs. Such architecture is present in the existing FL algorithms directed to applications in smart devices or networks containing similar medical information between different hospitals [2].

While this new approach does expand the usability of AI in more complex structures, FL presents a new set of vulnerabilities that can be exploited. The main concerning aspects are security, privacy, and performance. Although the main aim of the FL model is to preserve privacy, researchers found out that the gradient descent can be exploited to unveil the private data provided by a certain participant to train the model [5]. Also, the FL is vulnerable to security attacks such as model update poisoning, data poisoning, and inference evasion. This is caused by the fact that the server can observe the individual updates and interfere with the training process. Another cause is that the participants can see the global parameters and thus control their parameter uploads maliciously.[5]

In order to tackle the security and privacy risks of the HFL, privacy-enhancement tools were investigated and led to two main ones. One is Homomorphic Encryption (HE), a specific type of Secure Multiparty Computation (SMC) [6], which allows to perform computations on encrypted data and leads to a result that, when decrypted, is the same as if the computations were made on the plain text [7]. This approach is used, for instance, for training data on the cloud. Its challenge is to find the right trade-off between accuracy and privacy caused by evaluating non-linear functions [4]. The other one, Differential Privacy (DP), consists of adding noise to the data, such that the third party cannot easily recover the data transmitted, therefore ensuring privacy. The challenge here is to find the right trade-off between accuracy and privacy caused by the complexity of the process in which the other necessary parameters of the DP method are transmitted to obtain the desired results [8].

The two methods of privacy-preserving in FL, along with their advantages and disadvantages, led the researchers and developers to look up for hybrid approaches that can achieve the privacy-preserving of the FL, as well as exploit the potential of the two methods. Some researchers ([9, 10] looked into whether combining a (SMC) algorithm with DP can significantly boost the performance of the framework while keeping a high level of privacy. Some others tried a protocol that could bring a high performance while preserving a good level of privacy ([11]). Other approaches were considered as well, such as having the privacy level guaranteed by the properties of the blockchain, thus boosting the performance of the FL algorithm used to its maximum ([12, 13]).

This paper aims to take a look at the current implementations of hybrid models for HFL and explore how they perform over the state-of-the-art models. The analysis will be made from three main criteria: privacy level, performance (e.g. accuracy, model training time complexity), communication costs and security risks. Also, this paper compares the studied implementations between each other to identify what direction is more suitable in the future, as well as what missing points should be further investigated to make the HFL more practical in real scenarios.

This paper is organized as follows. Besides the introduction, we present an overview of the two methods of privacy-preserving for FL in Section 2, as well as the current security challenges the HFL is put up against. Section 3 presents the methodology of the analysis done in this paper, highlighting what and how the key features of the studied frameworks are examined. In Section 4, we will analyze five FL models that use different main components for preserving privacy. The next section, Section 5, a comparison between these frameworks is made, obtaining a better perspective over the features that they have and how they work out in different scenarios. Here we will also highlight which framework is better from different standpoints. Next comes Section 6 which tackles the ethical aspect of the analysis done throughout this paper. Section 7 discusses what are the remaining open problems for HFL, as well as future directions for further research. In the end, Section 8 concludes the paper.

2 Background

In this section, a quick overview of the key concepts that need a better understanding for our analysis will be presented. Firstly, an overview will be made about the current challenges the HFL has regarding security and privacy Then the main privacy enhancements will be presented, along with their advantages, disadvantages, and their performances in the state of the art model. This information will be used in the following sections of the paper, since the hybrid models will also be compared to these frameworks to see whether the hybrid approach is the next step in the development of the HFL.

2.1 Security threats and privacy risks of HFL

FL, as a model of machine learning, promotes the privacy of the data of the participants by not needing to take the data out of the device for training the model. However, recent work shows that just the model does not provide sufficient privacy guarantees, as the gradient results received by the central server can still reveal sensitive information about the data used by the user in the model training process [5]. Also, FL is prone to security attacks that enable the attackers to interfere and significantly change the performance of the global model. There are three main types of attacks. The first one is model update poisoning, in which the attacker can directly manipulate the model updates sent to the server. This can be achieved either by directly changing the updates of a participant or by a man-in-the-middle attack. The second type is data poisoning, in which the attacker can only interfere with the participant's data, thus leading to erroneous model updates. The last one is inference evasion, in which the attacker can manipulate the samples that are fed into the model. This is most visible when the test inputs are perturbed, thus the model assuming that its model is not in a good version. For the poisoning attacks, these can be performed for both targeted attacks (alter the model's behavior for a minority of samples) and untargeted attacks (reduce global model's accuracy, leading to breaking the model) [14].

To understand better what vulnerabilities each framework has, a threat model is shaped when creating an HFL model. The attacks can be carried out by either insiders or outsiders. Insiders are the ones that are part of the training process of the model, while outsiders are the ones who want to interfere with the communication between participants and the central server. As for the type of setting of the FL training, we can have trusted, honest-but-curious or malicious parties (participants and central server) [5].

2.2 SMC and Homomorphic Encryption

The first privacy-enhancing method used for the state-of-the-art HFL is Secure Multiparty Computation (SMC). This has the goal of providing methods for participants to be able to perform computation on encrypted data. In this way, the inputs (the gradient results of the training) remain private. A specific method of SMC is Homomorphic Encryption (HE), which can compute over encrypted data without accessing the secret key. The result, which is still encrypted, can then be used by the central server, which has the private key, to improve the model. The challenge here is the threshold which tells that no subset of participants smaller than it is able to decrypt the shared data given to the central server. [9] The higher the number, the more complex the calculations get, thus increasing the training time of the model.

2.3 Differential Privacy

The second privacy-enhancing method is Differential Privacy (DP), which is a mathematical framework that adds noise to the data such that third parties cannot distinguish the data. This guarantees the privacy of the data sent to the central server. Therefore, it stops an attacker's ability to infer the membership of a participant. [9]. However, this comes at a cost: the performance of the model. The accuracy decreases the more noise is added (i.e the higher the privacy level, the lower the accuracy). Also, it requires sending additional data necessary for processing the result by the central server to ensure a good accuracy level, thus increasing the communication costs of the model training [8].

When it comes to the algorithms used to implement DP, [15] presents the Global Differential Privacy and its performance in an FL framework. Here is highlighted a theorem which shows that there is a need for additional noise terms (notated n_D) to be added by the server in order to satisfy the desired (ε, δ)-DP requirement. This is affected by the number of aggregation times and the number of clients. The 'Noising before Aggregation' FL algorithm, proposed by [15], accomplishes the DP requirement by adding noise at both the client and the server. Its performance was studied on the MNIST dataset and the results show that the performance of the model is affected by the following three elements: the protection level - the lower, the better the performance is; the number of clients - the larger, the better; the number of maximum aggregation times - the performance can be increased by the larger number of aggregations, yet the higher variance of the noise will a negative impact on it. This highlight the challenges that the HFL models that use DP have, and they will be taken into account when comparing this model to the hybrid ones.

3 Methodology

This section presents an overview of the main aspects used to analyze the hybrid models presented in this paper, along with the meaning and importance of the criteria considered. This can be considered a literature study, since there was little time to implement an FL model that can evaluate the performance of the five analyzed implementations.

Firstly, for each paper, we get a better understanding of the main components used for preserving privacy and how the training process works. In this way, the reader can easily see the main features of the framework and observe the diversity of the hybrid approaches presented. The next element analysed is its performance. This can be shaped by a couple of aspects. Model training represents the speed at which the machine learning model evolves. One way of analyzing this is the training loss after a certain number of aggregations. Another way of measuring performance is the computation complexity. Here the algorithm of the framework is analyzed, and based on the privacy-preserving methods applied to the clients and servers, time complexities can be calculated. Here the best-effort approach was taken, since not all the papers had very concrete implementations for both the server and the client, thus making some estimations for comparing all the frameworks in all criteria. In this part, we will also consider the communication cost since it influences the scalability of the framework. as well as the network overhead. Lastly, security and privacy will be examined. Here the threat model will be considered, which shows the level of privacy guaranteed by the framework, as well as the security attacks that they can be immune to (e.g collusion, malicious client, malicious aggregator, whether a trusted third party is needed or not). We will also identify what are the security risks that are still open in these implementations.

4 New approaches for FL

When it comes to real applications of the Federated Learning framework, there are scenarios in which the state-of-the-art model does not provide enough to make the participants satisfied with the results of the trained model. Therefore scientists looked for methods that can use a combination of components so that they can bring the advantages of them in one place, and even expand the potential of the HFL. In the following part of this section we will present 4 hybrid models, highlighting the components used to implement them, as well as their key features.

4.1 A hybrid approach with DP and SMC [9]

This first approach presented in this paper tackles this concept: use SMC and DP to balance the trade-offs that each enhancement brings up [9]. The three security aspects that are addressed with this model at the following: inference during the learning process, inference over the output (in case the model would be deployed as a service after it has been trained well enough), and trust. The scenario presented in the paper considers a set of N parties, each one having its disjoint dataset, and an aggregator. Along with this, the additional parameters taken into account for the algorithm are: f_m (the training algorithm), t (minimum number of honest, non-colluding parties), and ε(the level of differential privacy that is satisfied). The aggregator runs a learning algorithm that cosists of k or fewer queries, each one requesting information from the N datasets. Each participant will calculate a response for the query received, encrypt the results using homomorphic encryption, use a differential privacy dependent on the algorithm to add the necessary amount of noise to their data, and then send it to the aggregator. Homomorphic encryption is used to reduce the noise that the DP algorithm needs to add, thus increasing the performance and accuracy. In the meantime, the aggregator queries at least *N-t+1* participants to decrypt the aggregate value, then updates the model. Once the new version of the model has been made, it is exposed to all the participants. The algorithm is presented for more analysis in Figure 1.

Based on the algorithm, we can calculate the time complexity of this model. In our case we consider the encryption step to be $\mathcal{O}(1)$, since this can help us get a better perspective of the architecture of the framework, rather than how it is affected by the concrete implementations of the privacy-preserving enhancements. Looking at the algorithm, we can see that for each step of the entire training (labeled by Q_s), we have $\mathcal{O}(N)$ for obtaining the data from all participants, then $\mathcal{O}(N-t+1)$ for querying t participants to partially decrypt the aggregate value. The rest of the parts of the algorithm are $\mathcal{O}(1)$. Therefore, for one step we have the total complexity of $\mathcal{O}(2N-t+1)$, which shows that it is linear. For an entire training process, the time complexity is $\mathcal{O}(Nk)$, based on the number of queries needed for each f_m. Another thing worth mentioning here is that the communication complexity is $\mathcal{O}(2N-t+1)$, which shows that we need two cycles of data transfers: one for the results of the query, and one for the partially decrypted aggregate value.

This model has been experimented with three learning models: decision trees (DT), convolutional neural networks (CNN) and linear Support Vector Machines (SVM). For DT, the hybrid approach achieves an F1 score of 0.8 and it is kept at this value when increasing the number of participants, on the other hand, the local DP significantly decreases, getting to 0.4 when there are more than 25 participants. For CNN, MNIST data was used and the performance of the hybrid approach has a F1-score of 0.9, which is higher than local DP by 0.2. Also, it is close to the performance of a Central Data holder model, with an F1 score of 0.95. Lastly, For SVM, the performance of the hybrid model has an F1-score of 0.87, which is almost equal to the central DP, while having a higher score than local DP by 0.1.

```
Input: ML algorithm $f_M$; set of data parties $\mathcal{P}$ of size $N$, with each $P_i \in \mathcal{P}$ holding
a private dataset $D_i$ and a portion of the secret key $sk_i$; minimum number of honest,
non-colluding parties $t$; privacy guarantee $\epsilon$
$\tilde{t} = n - t + 1$
for each $Q_s \in f_M$ do
  for each $P_i \in \mathcal{P}$ do
    $\mathcal{A}$ asynchronously queries $P_i$ with $Q_s$
    $P_i$ sends $r_{i,s} = Enc_{pk}(Q_s(D_i) + noise(\epsilon, t))$
  end for
  $\mathcal{A}$ aggregates $Enc_{pk}(r_s) \leftarrow r_{1,s} \circ r_{2,s} \circ \ldots \circ r_{N,s}$
  $\mathcal{A}$ selects $\mathcal{P}_{dec} \subseteq \mathcal{P}$ such that $|\mathcal{P}_{dec}| = \tilde{t}$
  for each $P_i \in \mathcal{P}_{dec}$ do
    $\mathcal{A}$ asynchronously queries $P_i$ with $Enc_{pk}(r_s)$
    $\mathcal{A}$ receives partial decryption of $r_s$ from $P_i$ using $sk_i$
  end for
  $\mathcal{A}$ computes $r_s$ from partial decryptions
  $\mathcal{A}$ updates $M$ with $r_s$
end for
return $M$
```

Figure 1: Algorithm used for implementing FL with DP and SMC

4.2 HybridAplpha [10]

HybridAplha is a model which focuses on improving the training time of the model while keeping the privacy level high. This is achieved by using functional encryption as the main secure multiparty computation (SMC) protocol, along with DP. One thing that it can do compared to other models is that it allows dynamic participants, meaning that some can drop out while others can join without affecting the overall performance of the model. Also, the training is made using only one round, excluding the key distribution communication, which is more efficient than other models.

The framework contains three main components: the participants, the aggregator, and a Trusted Third Party (TPA). The TPA is present due to the use of functional encryption as the main privacy-preserving enhancement. This component sets up a master public key and a master private key. These will be used to derive multiple public keys that will be sent to the parties that want to participate in the training process. With the key received, the participants will encrypt the results of the training with their own dataset.

The HypridAlpha algorithm is as follows. Firstly, there is the setup stage, where the TPA generates the master public key and master private key. Then, it derives a large number of public keys and distributes them to the participants. The new aspect here, compared to other frameworks, is that it allows new participants to join the training even after it started. When a new participant wants to join, the TPA will just give them one of the available public keys that were generated in the setup stage. The next stage is the learning stage. Here the aggregator asynchronously queries each participant with a query to train the given learning algorithm. The participants, during the training process, add differential privacy noise to the model parameters, then encrypts the resulting model and sends it back to the aggregator. Once all the results have been received, the aggregator requests a key from the TPA based on the number of participants that joined. In case a participant drops out, then the aggregator can request a key based on the number of responses received. Then it will decrypt the set of results received and update the global model. The detailed algorithm for further analysis is provided in Figure 2.

When it comes to evaluating the framework, HybridAlpha

```
Input: $\mathcal{L}_{FL}$ := Machine learning algorithms to be trained;
$\epsilon$ := privacy guarantee; $\mathcal{S_P}$ := set of participants, where
$\mathcal{P}_i \in \mathcal{S_P}$ holds its own dataset $\mathcal{D}_i$; $N$ := maximum number of
expected participants; $t$ := minimum number of aggregated
replies
Output: Trained global model $\mathcal{M}$
function TPA-initialization($1^\lambda, N, \mathcal{S_P}$)
    mpk, msk ← MIFE.Setup($1^\lambda, \mathcal{F}_N^1$) s.t. $N \gg |\mathcal{S_P}|$;
    foreach $\mathcal{P}_i \in \mathcal{S_P}$ do
        pk$_i$ ← MIFE.PKDistribute(mpk, msk, $\mathcal{P}_i$);
    end
function aggregate($\mathcal{L}_{FL}, \mathcal{S_P}, t$)
    foreach $\mathcal{P}_i \in \mathcal{S_P}$ do
        asynchronously query $\mathcal{P}_i$ with msg$_{q,i}$ = ($\mathcal{L}_{FL}, |\mathcal{S_P}|$) ;
    end
    do
        $\mathcal{S}_{\text{msg}_{\text{recv}}}$ ← collect participant response msg$_{r,i}$ ;
    while $|\mathcal{S}_{msg_{recv}}| \geq t$ and still in max waiting time;
    if $|\mathcal{S}_{msg_{recv}}| \geq t$ then
        specify $w_\mathcal{P}$ vector; request the sk$_{f,w_\mathcal{P}}$ from TPA;
        $\mathcal{M}$ ← MIFE.Decrypt(sk$_{f,w_\mathcal{P}}$, $w_\mathcal{P}$, $\mathcal{S}_{\text{msg}_{\text{recv}}}$);
    end
    return $\mathcal{M}$
function participant-train($\epsilon, t, msg_{q,i}, \mathcal{D}_i, \mathbf{pk}_i$)
    $\mathcal{M}_i \leftarrow \mathcal{L}_{FL}(\mathcal{D}_i)$;
    $\mathcal{M}_i^{DP}$ ← DP($\epsilon, \mathcal{M}_i, t$);
    msg$_{r,i}$ ← MIFE.Encrypt($\mathcal{M}_i^{DP}$, pk$_i$);
    sends msg$_{r,i}$ to aggregator;
```

Figure 2: Algorithm used for implementing HybridAlpha

was used to train a CNN on the MNIST dataset of handwritten digits [16]. It was compared to the performance of two SMC models. When it comes to both the communication cost and the training time, the theoretical analysis made in the paper shows that HybridAlpha, compared to SMC models, has the smallest one, which is $\mathcal{O}(mn + m + n)$, m representing the number of aggregators, and N the number of participants. This shows that HybridAlpha has the simplest aggregation process, which has a great impact on the scalability of the framework. As for the experiments made, it can be seen that the F1-score of HybridAlpha is close to the 0.9, which a similar score to other SMC models that also use DP. When it comes to the training time, HybridAlpha is similar to an FL without any privacy, which means that the extra privacy added to the model does not affect significantly the training time needed. Another element analyzed here is the network transmission efficiency, which was measured by the volume of the encrypted parameters sent throughout the network. The results show that HybridAlpha uses a size 15mb, compared to 100mb and 140mb that the other models need. This has a great impact on the efficiency of data transmission, which helps participants with an unstable connection to be part of the training process with a smaller chance of dropout.

To improve the security level of the framework, HybridAlpha implements an inference prevention module, which solves the threat model of having an honest but curious aggregator. This module is implemented with TPA and it inspects the requests for the key from the aggregator. In this way, the framework makes sure that the aggregator cannot ask for certain keys that could lead to revealing the model updates of a certain participant.

4.3 Turbo-Aggregate [11]

Turbo-aggregate focuses on optimizing the aggregation overhead of the main server. It manages to reduce it from quadratic time complexity to $\mathcal{O}(NlogN)$. As HybridAlpha, it is tolerant to the users who drop out in case of poor connection, thus not affecting the overall performance of the training model. As the main privacy enhancement component, it uses an improved version of the secure aggregate protocol, which is capable of providing up to 40x speedup in running time.

Turbo-aggregate consists of three stages. In the first one, a multi-group circular aggregation structure is created to enable fast model aggregation. Here, a network with N users is partitioned into L groups. In the second stage, to ensure the privacy of the participants, the framework uses additive secret sharing by adding randomness to the resulting models so that they will be canceled out once all the trained models are aggregated. This is done to ensure the privacy of the user models against collusions between the interactive participants (between users or between users and server). In the last stage, aggregation redundancy is added by using Lagrange polynomial in the model updates that are passed from one group to the other. This redundancy can be used to reconstruct the aggregated model even when in the case some users dropped out during the process.

The framework performs L steps sequentially. In each step, the users in a specific group encode their inputs, which include their trained models and the partial results from the previous groups, and then send them to the next group. The following group recovers the missing data due to dropped users and then perform an aggregation on the received messages.

The analysis of the time complexity can be extracted based on the algorithm presented in Figure 3 and the formulas mentioned in the algorithm, which are further explained and demonstrated in [11]. For each user in a group, it takes and $\mathcal{O}(l)$ for computing the masked model and generate the encoded model (l represents the number of members in the following group), and $\mathcal{O}(k)$ for calculating the code aggregate value and updating the aggregate value (k represents the number of members in the previous group). The last step, sending the parameters to the users in the following group, takes $\mathcal{O}(l)$. This totals to $\mathcal{O}(l + k)$. Since both l and k represent a partition of N, they can be replaced by $logN$. Therefore, the total time complexity for all users becomes $\mathcal{O}(NlogN)$. The final aggregation is similar in the approach, with the exception that all parameters are sent to the main server, thus a similar time complexity like the one for each group is needed. In the end, the server computes the aggregation that will update the global model, which takes $\mathcal{O}(logN)$. This shows that the algorithm has a time complexity of $\mathcal{O}(NlogN)$.

Communication-wise, we can see that each user sends k messages, meaning that it sends their result to all the other members of the following group. Therefore, for a user, the total communication cost is $\mathcal{O}(k)$. Since k is a partition of N, it can be rewritten as $logN$. Therefore, for the total communication cost of the algorithm, it is still $\mathcal{O}(NlogN)$. This shows that the communication cost has been optimized, compared to the state-of-the-art protocol that requires quadratic cost (i.e. $\mathcal{O}(N^2)$).

The performance of the framework is compared to a benchmark protocol that represents the performance of the state-of-the-art secure-aggregation. Two versions of Turbo-Aggregate are evaluated: Turbo-Aggregate, which is the one just presented, and Turbo-Aggregate+, which parallelizes the execution stages. The experiments have been performed over up to 200 users. When comparing the total running time, it can be seen that Turbo-Aggregate has a 5.8x speedup compared to the benchmark, while Turbo-Aggregate+ has a 40x speedup. Also, having an aggregation overhead of $\mathcal{O}(NlogN)$ compared to the quadratic complexity significantly improves the scalability of the framework, which is a key element for real applications. As for the security aspect, the secure-aggregate protocol ensures a good level of data privacy. If higher levels are desired, then DP can be used in the framework.

This framework ensures two important aspects regarding its privacy and security. The first one is that it guarantees privacy against up to $N/2$ colluding users (thus treating the security threat of honest but curious users). This shows that the more participants join a training epoch, the higher the privacy level gets. This is useful in real scenarios where the number of participants can get to the order of thousands. The other aspect is that the privacy of the aggregate of any subset of user models is guaranteed as long as a collusion between the server and the participants cannot reveal the aggregation of the random masks used by the participants. This is achieved by the fact that the server knows only N/N_l equations, N_l representing the number of members in a group, while there are at least $N/2$ masks generated by the participants. Therefore, the server cannot remove the random masks, which solves the threat model of honest but curious server/aggregate.

```
input  Local models $\mathbf{x}_i^{(l)}$ of users $i \in [N_l]$ in group $l \in [L]$.
output Aggregated model $\sum_{l\in[L]} \sum_{i\in\mathcal{U}_l} \mathbf{x}_i^{(l)}$.
 1: for group $l = 1, \ldots, L$ do
 2:     for user $i = 1, \ldots, N_l$ do
 3:         Compute the masked model $\{\widetilde{\mathbf{x}}_{i,j}^{(l)}\}_{l\in[N_{l+1}]}$ from (5).
 4:         Generate the encoded model $\{\bar{\mathbf{x}}_{i,j}^{(l)}\}_{j\in[N_{l+1}]}$ from (10).
 5:         if $l = 1$ then
 6:             Initialize $\widetilde{\mathbf{s}}_i^{(1)} = \bar{\mathbf{s}}_i^{(1)} = \mathbf{0}$.
 7:         else
 8:             Reconstruct the missing values in $\{\widetilde{\mathbf{s}}_k^{(l-1)}\}_{k\in[N_{l-1}]}$ due to
                the dropped users in group $l - 1$.
 9:             Update the aggregate value $\widetilde{\mathbf{s}}_i^{(l)}$ from (6).
10:             Compute the coded aggregate value $\bar{\mathbf{s}}_i^{(l)}$ from (11).
11:         Send $\{\widetilde{\mathbf{x}}_{i,j}^{(l)}, \bar{\mathbf{x}}_{i,j}^{(l)}, \widetilde{\mathbf{s}}_i^{(l)}, \bar{\mathbf{s}}_i^{(l)}\}$ to user $j \in [N_{l+1}]$ in group $l + 1$
            ($j \in [N_{final}]$ if $l = L$).
12: for user $i = 1, \ldots, N_{final}$ do
13:     Reconstruct the missing values in $\{\widetilde{\mathbf{s}}_k^{(L)}\}_{k\in[N_L]}$ due to the
        dropped users in group $L$.
14:     Compute $\widetilde{\mathbf{s}}_i^{(final)}$ from (14) and $\bar{\mathbf{s}}_i^{(final)}$ from (15).
15:     Send $\{\widetilde{\mathbf{s}}_i^{(final)}, \bar{\mathbf{s}}_i^{(final)}\}$ to the server.
16: Server computes the final aggregated model from (16).
```

Figure 3: Algorithm used for implementing Turbo-Aggregate

4.4 BlockFLA [12]

This FL uses blockchain as the main element, however, even here the scientists go a step further. They use a hybrid blockchain, using both private and public blockchains to get the advantages of both in one place and to overcome each one's weaknesses. This framework is powerful against model poisoning attacks since it uses smart contracts to automatically detect, then punish the participants that turn out to be attackers by giving monetary penalties. Each participant has a wallet and when they want to update the central model, they need to pay for it. If it turns out that they were honest about their involvement, they are awarded and can get more involved in the process. In case they are attackers, the central server can identify them and no longer give them any 'money' (the model uses Ethereum as their currency of the transactions). This means that they can no longer contribute to the training of the model.

This approach contains a couple of key components. The overall architecture is presented in Figure 4. The participants in this scheme are the ones who train the model. They are called 'workers' and they directly communicate with both types of blockchain and the Secure Cloud. The private blockchain performs aggregation and then sends the updated global model back to the participants. Since aggregation essential part of the model training, having high transaction throughput performance and low latency is essential, which is guaranteed by the private blockchain. Also, it ensures privacy and confidentiality of the transactions made by workers, which is the key property of FL, so having that by the use of this component is great. The public blockchain is the one who hosts smart contracts, In this implementation, Ethereum is used as the currency, which has low transaction throughput and high latency on transaction confirmation. The last component is the Secure Cloud, which logs all the updates each worker sent to the private chain during an epoch. This is important to reinforce the trojan detection process. These are also useful whenever a worker suspects another participant of being malicious. They can get one-time access to the logs of the suspect and perform the trojan detection themselves.

The training process is as follows. In the first step, the workers send the local parameters resulted from model training to the private blockchain. This is also logged in the Secure Cloud. The next step consists of the workers sending the generated hashes of their local updates to the public blockchain for verification. In this way, the potential attackers can be identified and penalized properly. The public blockchain also rewards honest participants. After that, the private blockchain aggregates all the local updates received from the involved workers to generate a global update. In the last step, the new model is sent back to the workers for the next epoch.

The performance of BlockFLA can be extracted based on the process itself, as well as some results observed in the experiment made in the paper. The raw performance of the FL algorithm used in this framework is considered as the one without any privacy enhancements since there is no need to use any on top of it. This is the result of the architecture of the blockchain that achieves the necessary level of privacy the FL needs. For step 1, the time complexity is $\mathcal{O}(N)$, since all the workers send their parameters to the private chain and the hashes to the public chain. Then, the aggregation process depends on the aggregation algorithm used. In the paper, the ones discussed are SignSGD [17] and FedAvg [18]. Out of these two, SignSGD performs way better than FedAvg, with a difference in deploy and aggregation time of around 10.5 minutes for 5-10 epochs. However, we will consider the time complexity of FedAvg, since with SignSGD it was not possible to implement the attacker detection algorithm. Therefore, for the aggregation process, we consider the time complexity as $\mathcal{O}(1/sqrt(NT) + N/T)$, with T the number of stochastic updates performed by each worker. [19]. We assumed $E = 1$ since there is direct communication between the worker and the private chain, and the throughput is very efficient. Therefore, the total time is $\mathcal{O}(N + 1/sqrt(NT) + N/T)$. Communication wise, the total time is $\mathcal{O}(2N)$, since each worker has to send messages to both chains.

The performance of the attacker detection algorithm has also been evaluated. The results show that when identically and distributed data (IID) is used, the algorithm easily detects the attackers. However, in the non-IID scenario, the algorithm also included honest agents. This shows that the algorithm is useful for identifying the attackers, but other detection algorithms can be used along with this one for better detection.

When it comes to the security of the framework, this framework achieves a couple of things. In the honest but curious participants threat model, the framework has the attacker detection algorithm, which works at its best if at least 51% of the participants are honest. In the case an attack has been undetected, the private chain can perform a re-simulation to identify the malicious data introduced. Then the attacker is penalized properly. The honest but curious aggregator threat model does not apply to this framework, since it is represented by the private blockchain that is responsible for ensuring the participants are properly authenticated and their updates are kept safe. Also, attackers cannot change the aggregation process, thus the private chain can be considered trustworthy.

As for privacy, the data each worker sends to the private chain is kept safe because each worker has a separate channel to the main server. This enables strict access control on the shared data between the worker and the server. Also, only the participants who are an authorization can join the private chain, which means that the data is inaccessible to external users. This ensures that the data cannot be leaked to third parties. Therefore, the privacy guarantees are of a high level.

4.5 Blockchain empowered Asynchronous FL [13]

This implementation of FL is studied on the Internet of Vehicles (IoV) scenario, to solve the challenges regarding the bandwidth, security, and privacy of the data shared between the vehicles for collaborative analysis. The key element here is using an asynchronous FL along with a hybrid blockchain mechanism. The blockchain, named PermiDAG, consists of a main permissioned blockchain and the local Directed Acyclic Graph (DAG). These are responsible for the synchronous global aggregation and the asynchronous local training in the FL model. The advantage of the hybrid blockchain is that it allows a part of the participating nodes to run the blockchain effectively. Also, by letting the vehicles store the local DAG and letting the Road Side Units (RSUs) store the permis-

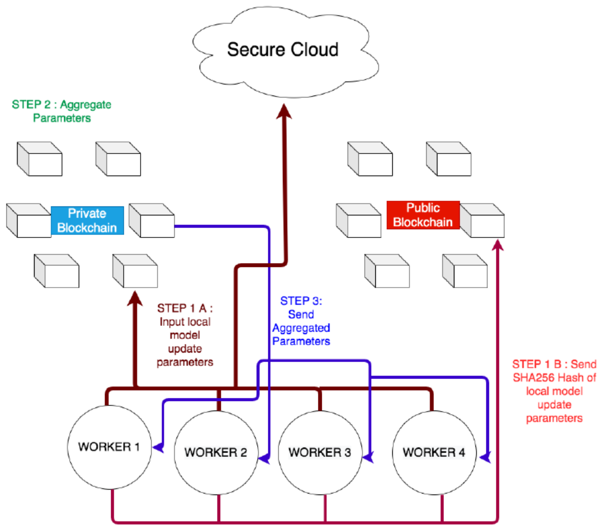
Secure Cloud
STEP 2 : Aggregate Parameters
Private Blockchain
Public Blockchain
STEP 1 A : Input local model update parameters
STEP 3: Send Aggregated Parameters
STEP 1 B : Send SHA256 Hash of local model update parameters
WORKER 1
WORKER 2
WORKER 3
WORKER 4

Figure 4: BlockFLA architecture diagram

sioned blockchain, the storage efficiency is significantly improved. The RSU is responsible for optimizing the network efficiency and enhance security with a malicious IoV detection algorithm [20]

This framework contains three phases: node selection, local training, and global aggregation. The node selection algorithm chooses the nodes with a higher amount of resources to participate in the federated learning process. This improves the running efficiency and training accuracy. Once the nodes are selected, they vote for the delegates (RSUs) that maintain the permissioned blockchain. In the next one, local training, the participants, which are the vehicles in this case, train their local model and send it to the local DAG for verification and aggregation. The vehicles also retrieve the verified models of other participants from the local DAG and perform a local aggregation to improve their local model. In the last phase, the aggregation, the delegates (RSUs) retrieve the local models from the local DAG and perform a global aggregation. Once this is completed, the new model will be sent to all delegates.

The performance of the framework is analyzed based on the algorithm presented in Figure 5, as well as the formulas presented in the paper. For each vehicle that participates in the training process, the time complexity is $\mathcal{O}(1)$, since all the steps are constant (lines 4-10). The most time-consuming step is the computation of the local gradient descent. For an episode, the total time complexity is $\mathcal{O}(tP)$, where t is the number of local training rounds made by each vehicle, and P the number of selected nodes. The steps outside the loop (lines 12-14) take at most $\mathcal{O}(P)$. For having a final global model, the total time complexity is $\mathcal{O}(etP)$, where e is the number of episodes (in the other papers they are called epochs). In the case of selecting all the participant, the result becomes $\mathcal{O}(etN)$

The evaluation of the framework is made on the MNIST dataset. The results are compared to the ones achieved by local and centralized CNN models. The accuracy of the framework is similar to a centralized CNN, having very close values as the number of iterations increases. As for the time cost, the asynchronous FL performs better, reducing the cost by six times compared to the centralized CNN. This shows that the approach is efficient. The aspect to note is that the cost grows quickly as the network becomes more complex (the number of RSUs and vehicles rises). Therefore, there may be challenges regarding its scalability.

Privacy and security are achieved by the implementation of the hybrid blockchain mechanism. By storing and verifying the model parameters sent by the vehicles, the framework makes sure that the data is not modified maliciously and also that they cannot be leaked to the other members in a way that can lead to the leak of the training data used. However, there is no discussion regarding whether the aggregator can be curious, or if some participants collude with the training data. Some of these concerns may be solved by the use of blockchain, which by its nature reduces the risk of data leakage and gives the participants more control over the access of the shared data, but further investigation should be made in that matter.

```
Input: The registering vehicles as participating nodes
  $V_I = \{v_1, v_2, \ldots, v_N\}$, the dataset of vehicle $i$, $d_i \in D$.
Input: Initialize the permissioned blockchain $B$ and DAG.
  Initialize the initial global model $M_0$
Input: Select the participating vehicles $V_P \subset V_I$ by
  running node selection algorithm. Vote the delegates
  $r_1, r_2, \ldots, r_n$
 1: for each episode $e$ do
 2:   Select a leader $r_0$ from delegates
 3:   for each time slot $t$ do
 4:     for each vehicle $v_i \in V_P$ do
 5:       $v_i$ retrieves global model $M_{t-1}$ from
          permissioned blockchain $B$
 6:       $v_i$ executes the local training on its local data $d_i$
          based on Eq. (6)
 7:       $v_i$ retrieves local model updates from DAG
 8:       $v_i$ executes local aggregation and obtain updated
          local model $m_i(t)$
 9:       $v_i$ add the parameters of model $m_i(t)$ as a
          transaction to the DAG
10:     end for
11:   end for
12:   The leader $r_0$ retrieves the current verified updated
      models from DAG, and aggregates the models into
      $M(e)$ based on Eq. (7)
13:   $r_0$ broadcasts $M(e)$ to other delegates for
      verification, and collects all transactions into a new
      block
14:   $r_0$ appends the block including the global model
      $M(e)$ to the permissioned blockchain
15: end for
16: return The parameters of the final global model $M$.
```

Figure 5: The Hybrid Blockchain Empowered FL algorithm

Framework	Privacy-preserving enhancement	Security threat model	Time complexity	Communication cost
FL with DP and SMC	DP and SMC	honest but curious	$O(Nk)$	$O(2N-t+1)$
HybridAlpha	DP and SMC	honest but curious	$O(mN + m + N)$	$O(mN + m + N)$
Turbo-Aggregate	Secure Aggregate	honest but curious	$O(N\, logN)$	$O(N\, logN)$
BlockFLA	public + private blockchain	honest but curious	$O(N + \frac{1}{\sqrt{NT}} + \frac{N}{T})$	$O(2N)$
PermiDAG	hybrid blockchain	honest	$O(etN)$	$O(etN)$

Table 1: Comparison between the presented frameworks

5 Comparison

Five FL architectures have been presented in the previous section. To understand better their potential and how they shape the evolution of FL, a thorough comparison between them will be made in this section. The main elements that will be taken into account are training time complexity, communication cost, and security threat model. In addition, we will also discuss the set of features that they have that stand out. A summary of the comparison is presented in Table 1.

The interesting part, to begin with, is that two frameworks use a combination of DP and SMC, and two make use of blockchain. Comparing the first two ones, it can be seen that their communication cost and training times are quite similar: they are both linear in regards to the number of participants. One thing to note is that HybridAlpha treats the dishonest threat model, which increases the level of security guaranteed. Another thing that can be compared is their F1 scores since they were both tested on the MNIST data. The FL with DP and SMC has a score of 0.9, while HybridAlpha with DP has a score of about 0.9. This shows that they are similar regarding precision and sensitivity. Another thing worth noting is that the communication of HybridAlpha is achieved through one set of messages, compared to two sets of messages in the other one, since it needs to ask the participants to decrypt the aggregate value, which can affect the communication overhead when thousands of participants contribute to the model training in an epoch.

BlockFLA and PermiDAG both looked for better privacy-preserving enhancements in the properties of the blockchain mechanism. Both consider that blockchain enables access control by the participant over the data shared with the others while keeping the attacks to the sent data to a minimum. Although they use different architectures regarding the blockchain, the complexities are quite similar in regards to the number of participants. BlockFLA guarantees a higher level of security and privacy since it has an honest but curious security threat model. Also, it increases the level of security by having an attack detection algorithm that works quite well in certain uniform distributed data scenarios. Communication-wise, BlockFLA performs slightly better, since there is not much overhead on the throughput. PermiDAG asks the participants to perform local aggregation with the shared parameters from the other participants. On the other hand, BlockFLA uses the private chain that is much more efficient on the throughput and can easily act as the main aggregator.

Turbo-Aggregate seems the one that stands out from multiple perspectives. Firstly, it uses a protocol that the others do not have. It supports DP for extra privacy, but it does not impact its performance signifficantly. Compared to the others, it can easily be implemented in scenarios in which no main aggregator is needed (e.g. peer-to-peer networks). Also, with Turbo-Aggregate+, it can easily parallelize the aggregation process, while the others perform the global aggregation sequentially. Another thing worth mentioning is that it supports user dropout of up to 50%, which is a feature that HybridAlpha also has. The difference is that HybridAlpha also supports users to join even after the training process started.

Looking at the security threat model, it can be said that most of the frameworks focus on the honest and curious threat model. The ones who do not use blockchain mechanisms even treat the scenario of colluding users, which requires that the data of the honest users is not revealed even when a certain amount of users try to work together to reveal that data. In the frameworks where blockchain was not used, they support collusion of up to $N/2$. BlockFLA is a better-suited framework, since it also has a detection algorithm that can penalize the attacker in case it sends malicious model updates, thus no longer being able to participate in the following training epochs. However, HybridAlpha is second on the list, since it also implements an inference prevention filter, which increases its defense against inference attacks.

As for the time complexity, most of them try to be linear in regards to the number of participants. The most efficient one is BlockFLA, since it gets to use the raw performance of the FL model. This is achieved since the privacy and security concerns are solved by the combination of private and public blockchains.

For communication cost, the most efficient is HybridAlpha, since it only needs one round of messages for an epoch, compared to the FL with DP and SMC, which needs three. Turbo-Aggregate needs one round of messages, but the number of messages sent is way higher, since each member of a group has to send messages to all the members of the following group, which can have an impact when thousands of participants contribute to the model training.

6 Responsible Research

The analysis made throughout this paper has been achieved by applying a set of key principles. Firstly, the papers considered had a high number of citations, meaning that the work done in the papers is recognized. Once the first set of papers has been selected, we filtered based on the approach taken in the implementation of the FL, as well as the level of details provided regarding the proposed architecture. This ensures that the variety of approaches analyzed in the paper is large (we have DP combined with SMC, secure aggregation protocol, blockchain mechanisms), while also being able to compare similar implementations.

All the calculations of the performance of the presented models have been made based on analyzing the algorithms, the results of the experiments, and the formulas provided in the analyzed papers. Therefore, a good part of them can easily be replicated. If the paper does not provide such in-depth data to easily make these calculations, then a rough estimate is made based on the components used in the framework, the design of the FL model, the training process, as well as interpreting the results of the experiments. For instance, for BlockFLA no concrete algorithm is used, therefore the process is the main aspect to study for the complexities, along with the aggregation algorithm used. For the complexity of the algorithm, an analysis of papers that dived into the algorithm is made.

As for the papers picked for the first sections (introduction and background), papers with a good amount of citations were used, which can show that the work presented in them is believable and of good quality. For definitions and main concepts, booksm and articles were used to easily present the concepts to a new reader.

Overall, the literature study presented in this paper helps the other researchers get an overview of the hybrid approaches that are implemented, as well as their performance, trade-offs, security, and privacy levels. Also, this study can help in identifying new directions of the evolution of HFL. Therefore, it will have a positive impact on ethical computer science.

7 The future of the hybrid approaches in HFL

The future for the HFL looks promising. Looking back to the state-of-the-art implementations with either DP or HE, it can be seen that their performance was not suitable for real scenarios. However, the models presented in this paper are suitable for a variety of cases, such as the IoV, peer-to-peer network, asynchronous training. The performance of these frameworks has a significant boost, which is helpful in scalability, network efficiency, and aggregation overhead.

The challenges that still remain tackle the security threats and attacks. Most of the frameworks presented are able to resist targeted model update attacks. BlockFLA is also able to prevent data poisoning, while HybridAlpha can defend against inference-time evasion attacks by having an inference prevention filter. Yet, these solutions are not the best. The BlockFLA's attacker detection algorithm works better if combined with other detection algorithms. HybridAlpha's threat model does not take into account the scenario in which a malicious aggregator will interrupt the network or replace the model update sent by an honest participant. [11] suggests that further investigation regarding the Byzantine attacks should be made if the security of the Turbo-Aggregate framework.

The attacks that were not quite handled are the following: inference-time evasion, untargeted and Byzantine model update poisoning, and Byzantine-robust aggregation. By not treating these, the frameworks will significantly lose their accuracy, which can further lead to breaking the global model (for untargeted attacks) and alter the behavior on a minority of samples (for targeted attacks). Therefore, a nice future path is to see how could the privacy advantages of the provided mechanisms (blockchain, DP, Secure-Aggregate) can be used to challenge the open security risks that the FL faces. Also, it will be nice to see if the threat model can be further developed, as in try out the frameworks in riskier scenarios. By doing this, possible defense mechanisms can be identified.

One other thing that can be noted is that in the case of FL frameworks that use blockchain, the FL algorithm does not use any privacy-preserving enhancements. It would be interesting to dive into hybrid FL models that use blockchain and also a combination of SMC and DP. For instance, [21] presents an architecture consisting of a permissioned blockchain and an FL module that uses DP. Similar approaches should be further investigated in the future since it can lead to making the FL immune to more security threats.

8 Conclusion

This paper presents how hybrid approaches in HFL opened new horizons in the evolution of the HFL model. This is achieved by diving deeper into the main concepts of the HFL: the main purpose, security and privacy challenges, main privacy-preserving enhancements. Then, five hybrid models are examined, highlighting the main component of the framework, the training process, as well as the security threat model used. Two models (FL with DP and SMC and HybridAlpha) combine the advantages of DP and SMC, while minimizing their trade-offs. Turbo-Aggregate uses a new improved protocol based on secure aggregation, focusing on optimizing the aggregation overhead. The last two models presented, BlockFLA and PermiDAG, use blockchain mechanisms to ensure the privacy level needed by FL. Therefore, the performance of the FL algorithm is improved significantly.

The performance and the communication cost of the presented concepts are analyzed based on the algorithms implemented. Also, their performance in real scenarios is discussed by diving into the results of the experiments made in the papers. The security guarantees are also discussed for each implementation. After that, a comparison between the five frameworks is made to highlight the most suitable one regarding the security level achieved, the training time performance, and communication cost. Also, by comparing the studied frameworks, the reader can see how a similar approach (e.g. combining DP with SMC) can be implemented differently and identify the key elements that improve the FL model. The features that are quite unique for each framework are also highlighted here.

The implementations presented open new problems to be examined. It would be interesting to make a hybrid model that uses both blockchain and a privacy-preserving enhancement and see whether other security risks are solved. Also, by experimenting with the implementations against the threat models that have not been treated, new aspects can be identified regarding the direction of the development of the HFL.

References

[1] Paul Voigt and Axel Von dem Bussche. "The eu general data protection regulation (gdpr)". In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10 (2017), p. 3152676.

[2] Li Li et al. "A review of applications in federated learning". In: *Computers & Industrial Engineering* 149 (2020), p. 106854. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106854. URL: https://www.sciencedirect.com/science/article/pii/S0360835220305532.

[3] Beandan McMahan and Daniel Ramage. "Federated Learning: Collaborative Machine Learning without Centralized Training Data". In: *Google AI Blog* (2017). URL: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[4] Qiang Yang et al. "Federated Machine Learning: Concept and Applications". In: *ACM Trans. Intell. Syst. Technol.* 10.2 (Jan. 2019). ISSN: 2157-6904. DOI: 10.1145/3298981. URL: https://doi.org/10.1145/3298981.

[5] Lingjuan Lyu, Han Yu, and Qiang Yang. "Threats to federated learning: A survey". In: *arXiv preprint arXiv:2003.02133* (2020).

[6] Chuan Zhao et al. "Secure multi-party computation: Theory, practice and applications". In: *Information Sciences* 476 (2019), pp. 357–372.

[7] Xun Yi, Russell Paulet, and Elisa Bertino. "Homomorphic Encryption". In: *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, pp. 27–46. ISBN: 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8_2. URL: https://doi.org/10.1007/978-3-319-12229-8_2.

[8] Katrina Ligett et al. "Accuracy first: Selecting a differential privacy level for accuracy-constrained erm". In: *arXiv preprint arXiv:1705.10829* (2017).

[9] Stacey Truex et al. "A Hybrid Approach to Privacy-Preserving Federated Learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. AISec'19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1–11. ISBN: 9781450368339. DOI: 10.1145/3338501.3357370. URL: https://doi.org/10.1145/3338501.3357370.

[10] Runhua Xu et al. "Hybridalpha: An efficient approach for privacy-preserving federated learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 2019, pp. 13–23.

[11] Jinhyun So, Başak Güler, and A Salman Avestimehr. "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning". In: *IEEE Journal on Selected Areas in Information Theory* 2.1 (2021), pp. 479–489.

[12] Harsh Bimal Desai, Mustafa Safa Ozdayi, and Murat Kantarcioglu. "Blockfla: Accountable federated learning via hybrid blockchain architecture". In: *arXiv preprint arXiv:2010.07427* (2020).

[13] Yunlong Lu et al. "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles". In: *IEEE Transactions on Vehicular Technology* 69.4 (2020), pp. 4298–4311.

[14] Peter Kairouz et al. "Advances and open problems in federated learning". In: *arXiv preprint arXiv:1912.04977* (2019).

[15] Kang Wei et al. "Federated learning with differential privacy: Algorithms and performance analysis". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469.

[16] Yann LeCun, Corinna Cortes, and Christopher J Burges. "MNIST handwritten digit database. 2010". In: *URL http://yann. lecun. com/exdb/mnist* 7 (2010), p. 23.

[17] Jeremy Bernstein et al. "signSGD: Compressed optimisation for non-convex problems". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 560–569.

[18] Felix Sattler et al. "Robust and communication-efficient federated learning from non-iid data". In: *IEEE transactions on neural networks and learning systems* 31.9 (2019), pp. 3400–3413.

[19] Zhaonan Qu et al. "Federated Learning's Blessing: FedAvg has Linear Speedup". In: *arXiv preprint arXiv:2007.05690* (2020).

[20] Sudha Anbalagan et al. "Machine Learning-based Efficient and Secure RSU Placement Mechanism for Software Defined-IoV". In: *IEEE Internet of Things Journal* (2021), pp. 1–1. DOI: 10.1109/JIOT.2021.3069642.

[21] Yunlong Lu et al. "Blockchain and federated learning for privacy-preserved data sharing in industrial IoT". In: *IEEE Transactions on Industrial Informatics* 16.6 (2019), pp. 4177–4186.