Federated Learning: An Investigation of Weaknesses, Countermeasures, and Attacks

Eric Tillmann Bill
RWTH Aachen University
Aachen, Germany
eric.bill@rwth-aachen.de

***Abstract*—Federated Learning (FL) is a novel approach to distributed, scalable, and privacy-preserving machine learning. However, these very characteristics make it vulnerable to various attacks, including the class of model poisoning attacks.**

In this work, we will classify attacks on FL in general into coarse categories and highlight the associated vulnerabilities of FL against each category. In particular, we will explore the underlying mechanism of model poisoning attacks in detail by examining common countermeasures that have been proposed to mitigate these attacks but are still vulnerable to modern attacks. With these findings, we conclude the paper by saying that countermeasures are just a form of statistical outlier detection, that some countermeasures risk leaking sensitive private data, and that most countermeasures do not slow down the training process.

***Index Terms*—federated learning, poisoning attacks**

I. Introduction

In the past decades, machine learning has achieved remarkable success in terms of advantages in performance and scalability. Nowadays, the amount of data needed to train modern models is gigantic and constantly growing, which poses two challenges. Firstly, storing this amount of data and then training a model using only a single machine is infeasible due to the resulting long computation times. Therefore, distributed systems are commonly used for large models such as [1]. Secondly, but most importantly, collecting these volumes of data, especially when working with personal data, is difficult due to various hurdles along the way, e.g., each participant must give unrestricted consent to the sharing of his or her data, or because there have been numerous political acts in recent decades to protect people and their sensitive data. An approach that addresses both problems was proposed in the work of McMahan et al. [2] and has been termed Federated Learning (FL).

FL is an approach to machine learning that allows for decentralized training of models without sharing the training data. In its most straightforward configuration, FL performs multiple iterations to train a global model, where in each iteration, it starts by sharing the current global model with all participants. Then all participants use the model for training locally on their private data. After the local training is completed, each participant sends their updated version of the model back to the server, which aggregates them into a single new global model. Thereby, each participant never shares their sensitive data, only the updated model. Moreover, since the actual training and storing are delegated to the participants, FL works in a distributed fashion, allowing for better scalability than centralized machine learning. Hence, it is fair to say that FL solves both issues at once [2].

To date, there are numerous cases where FL is used as an alternative to centralized machine learning models. Of particular interest are the use cases where centralized machine learning cannot be implemented due to non-technical reasons, such as ethical reasons concerning the training data. For example, Jochems et al. [3] used FL in a healthcare environment where several hospitals from different nations participated in training a predictive model for assessing the condition of dyspnea (shortness of breath) in patients. Most remarkable is that with the data never leaving the hospital, they were able to avoid legal, ethical, and administrative barriers that would normally hinder the sharing of information for such use cases. Similarly, Dayan et al. [4] developed an application in another health environment, where several institutes used patients' data to train a predictive model to assess the future oxygen requirements of symptomatic patients with COVID-19 in an FL fashion. Finally, with the prospect of overcoming these non-technical problems and the scalability advantages already mentioned, FL is a promising technology.

Although FL has many benefits, no technology is perfect from the start. In the case of FL, there are still two significant drawbacks to consider.

1) Despite the fact that each participant never shares their private data with the server, the updates they send to the server were computed using their private data. Thus, the updates contain information about their sensitive training data, which can sometimes be used to reconstruct the original training data [5], [6], thereby proposing a risk to the client's privacy.
2) The risk of adversarial clients who try to manipulate the learning process such that the final model performs as the adversarial attacker desires, like having a backdoor [7] or performing miserably on its intended task [8].

The problems have been thoroughly studied in the literature. In this work, we will focus on the second one and aim to help the reader understand why FL architectures are vulnerable to these kinds of attacks, how they can defend against them, and why creating a foolproof solution is nearly impossible.

Therefore, the remainder of this paper is structured as follows, in Section II, we will introduce a formal notation and

definition of FL that will be used throughout the rest of the paper. In Section III, we introduce a classification of attacks on FL, with which we will transition into Section IV, where we talk about fundamental problems in FL. After that, we will explain current defense mechanisms in Section V and show how to develop attacks against these defense mechanisms in Section VI. Finally, in the last part, in Section VII, we will discuss the use case in light of these attacks and the necessary future research for FL.

II. Federated Learning

In this section, we will briefly introduce a formal notation and definition of FL that we will use throughout the paper to ensure a coherent understanding.

We refer to participants of an FL architecture as clients. The number of clients can vary and is generally referred to as K, such that all clients can be numbered from 1 to K. There will only be a single server. Further, the data is partitioned over all clients with a partition $\mathcal{P}$, such that $\mathcal{P}_i$ is the local dataset of client i. The overall model we will train is of the format $f_\theta : \mathcal{X} \to \mathcal{Y}$, where f_θ is a differentiable classifier that is parameterized by θ, e.g., a multi-layer perceptron. Further, we denote θ_t being the global parameters and $\theta_{t,i}$ the local parameters of client i at iteration t. Finally, we denote with l a well-defined differentiable and suitable loss function and with η the learning rate parameter.

Using this notation, we will formally introduce one of the simplest forms of FL. This architecture will run for multiple iterations, where in each iteration t, the following procedure is executed:

1) The server sends out the current global model f_{θ_t} to all clients.
2) Each client then trains on its local data and applies gradient descent to derive its updated parameters by computing:

$$\theta_{t,i} \leftarrow \theta_t - \eta \cdot \nabla_{\theta t}\, \mathbb{E}_{(x,y)\in\mathcal{P}_i}[l(f_{\theta_t}(x), y)]. \quad (1)$$

   Each client then sends its update $\theta_{t,i}$ to the server.
3) The server aggregates all updates by averaging them and thereby derives the new global parameters θ_{t+1}:

$$\theta_{t+1} \leftarrow \frac{1}{K} \cdot \sum_{i\in\{1,\dots K\}} \theta_{t,i} \quad (2)$$

See Figure 1 for an illustration of such an iteration.

A short note on this model: as stated before, it is one of the most straightforward FL architectures and is, in fact, a streamlined version of the initially proposed model [2]. By understanding this model, it is easy to generalize this concept to more complex training techniques like batch normalization [9] and more safe model aggregation methods like Krum [10], Buylan [11] or Trimmed mean [12]. In addition, FL is not limited to classification tasks and can easily be adapted to regression or other tasks. Finally, the number of clients is not fixed but can vary from iteration to iteration due to the design principle that clients can join or leave at any time [2].

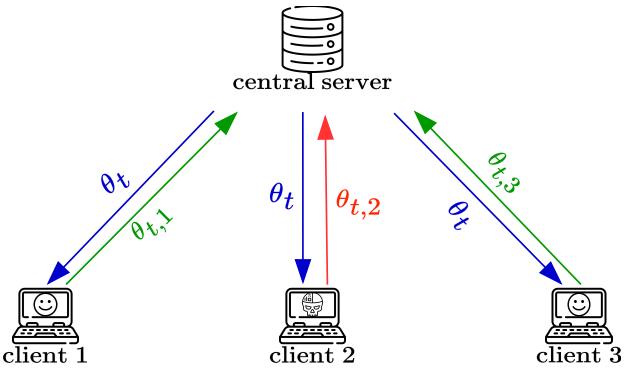

Fig. 1: An illustration of an exemplary FL environment with three clients. Here, the central server sends the current model θ_t to all its clients (blue arrows). Each client optimizes this model and sends an updated version back to the central server. The difference is that clients 1 and 3 are benign (indicated by the green-colored arrows), while client 2 is malicious (red-colored).[1]

However, for ease of understanding, we assume a constant number of clients throughout all iterations.

A very important detail that we have not yet addressed and that also made FL initially so popular, is that the authors of [2] have shown that the underlying data distribution $\mathcal{P}$ is not required to be independently and identically distributed (i.i.d.), but can, in fact, be very unbalanced and is therefore referred to as non-i.i.d. data. This characteristic of FL is unlike traditional machine learning methods, including distributed machine learning techniques such as [13]–[15], where the training data must usually be i.i.d. to ensure convergence. For a better understanding, recall the example from the introduction where FL was applied to a healthcare setting: Since the data distribution does not have to be i.i.d., it does not matter that, for example, the dataset of hospital A has proportionally more fatalities than the dataset of hospital B or that all patients in hospital A have certain diseases that none of the patients in the dataset of hospital B have.

III. Classification of Attacks

Before introducing the classification, we would like to emphasize the fact that studies on possible attacks are essential as they provide a basis for the robustness of defense techniques, as stated in [5]. Additionally, since FL is a technology that allows for better scalability by utilizing distributed computation, standard failures such as memory corruption, network loss, etc., are very common at such a scale among the clients and must therefore be also accounted for by defense mechanisms. For this reason, clients are usually regarded and referred to as Byzantine clients. This concept, introduced in [16], covers all

[1]The graphic was designed using the freely available and open-source platform `https://www.draw.io`. Additionally, the icons utilized in the graphic were sourced from various libraries. The central server icon and clients icon were obtained from Freepik's library, the skull icon was sourced from SmashIcons, and the happy face icon was obtained from th studio.

malicious effects a client could have on an FL architecture, whether they are deliberate or not.

Attacks on FL can be categorized by their objective into two categories: 1) inference attacks and 2) poisoning attacks [17]:

1) Interference, also known as privacy attack, are a collection of attacks that try to infer any information about the sensitive training data or training parameters [6], [17] by leveraging the information that is contained in the updates sent from the clients to the server. Attacks of this category are further categorized, by the information they try to comprise, into class representative attacks, properties of the underlying model attacks, and training data attacks.
2) On the other hand, poisoning attacks are a collection of attacks that attempt to manipulate the global model in terms of performance [7], [17]. For example, these attacks typically aim to reduce the global model's accuracy or introduce a backdoor into the global model. In the literature, [17], these attacks are finer classified into data poisoning and model poisoning attacks. The difference is that in the class of data poisoning attacks, an attacker only has access to the local training data and, therefore, can only modify certain properties of that data, e.g., changing labels. The other class assumes that an attacker has (almost) full control over a compromised client and can, thus, fully change its updates sent to the central server. As one can assume, the latter class is more powerful in an FL architecture. This is due to the fact that data poisoning attacks have been researched even before FL was formally introduced. Therefore, most data poisoning attacks focus on centralized architectures; however, their findings can be, to some extent, applied to FL as well.

As outlined in the introduction, this paper focuses on model poisoning attacks and explains why it is almost impossible to protect against them fully. The classification presented earlier aims to provide readers with an encompassing understanding of the various attack types. Furthermore, in Section VII, we will explore a subtle connection between both attack types.

IV. Issues in the FL architecture

The fundamental principles of FL [2], that the data is non-i.i.d. and is never shared with the server, leads to the fact that the server is unaware of the typical values of the updates it receives. If the data were i.i.d. across all clients, the server could expect all updates to be very similar to each other. However, because the data is non-i.i.d., the updates sent by clients are also non-i.i.d., making it challenging for the server to distinguish between malicious and benign updates.

We quickly demonstrate how an attacker that only controls a single client in an FL architecture can manipulate the global model arbitrarily. This attack is inspired by the work [10]. We assume the FL training process uses the equations provided in Section II. Further, we assume that an attacker can estimate the number of clients K and has full control over client j. The attack works as follows, instead of using equation 1, the client j performs a gradient ascent step with a learning rate of K', with $K' >> K$ being sufficiently larger than K:

$$\theta_{t,j} \leftarrow \theta_t + K' \cdot \nabla_{\theta t}\ \mathbb{E}_{(x,y)\in\mathcal{P}_j}[l(f_{\theta_t}(x), y)]. \tag{3}$$

With this attack, the attacker achieves that the impact of all other clients is negligible in the aggregation, and the global model parameters will be approximately the update sent by the compromised client j. This attack works because the updates are not bounded, and an attacker can enforce its updates by multiplying it with a sufficiently large factor, here K'.

$$\frac{K'}{K} \cdot \theta_{t,j} \ \approx\ \frac{1}{K} \cdot \sum_{i\in\{1,\dots K\}} \theta_{t,i} \tag{4}$$

This simple example beautifully demonstrates where countermeasures can occur, namely only in the aggregation rule. For example, in the work of [10], they even proved that an FL architecture can never be safe from one or more adversarial clients if the server uses a linear combination as an aggregation function. However, as we will learn in the next section, the server can detect malicious updates if the majority of clients are benign.

V. Countermeasures

In this section, we will present three safe aggregation methods called Krum, TrimmedMean, and Median, all of which are very effective in their own way. Note that these defensive mechanisms represent only a small fraction of the proposed approaches available. Further, it is worth mentioning that these methods may not be the most secure due to more advanced and intricate methods such as Buylan, which are difficult to explain and prove in a short space. Noteworthy are also the methods BRUTE [11] and GeoMed [10], [18].

A. Krum

Blanchard et al. [10] proposed an aggregation rule called Krum that is safe for m adversarial clients. In detail, with the assumption of $2m + 2 < K$, the aggregation rule guarantees convergence to the optimum when having no malicious clients while being robust against m malicious clients if the data is i.i.d. across all benign clients. Krum aggregation works as follows:

$$\theta_{t+1} \leftarrow \arg\min_{\theta_{t,i}} \Big(\sum_{j\in S(\theta_{t,i})} \|\theta_{t,i} - \theta_{t,j}\|_p^2 \Big), \tag{5}$$

$$\text{with} \quad S(\theta_{t,i}) = \arg\min_{\substack{S^*\subseteq\{1,\dots,n\}\setminus\{i\} \\ |S^*|=K-m-2}} \Big(\sum_{j\in S^*} \|\theta_{t,i} - \theta_{t,j}\|_p^2 \Big).$$

Where $S(\theta_{t,i})$ denotes the set of indices with their updates being the closest $K - m - 2$ updates to $\theta_{t,i}$, and p is a hyperparameter determining the l_p norm used as similarity measure. Thus, the aggregation rule selects a single $\theta_{t,i}$ as the new global parameter θ_{t+1}. In more detail, this rule consistently selects the update with the smallest summed difference to its closest $K - m - 2$ neighbors. This procedure guarantees convergence to the optimum since the updates of

all benign clients are very similar because, by assumption, the data across these clients is i.i.d. and the majority of clients are benign.

However, Mhamdi et al. [11] demonstrated potential weaknesses of many secure aggregation methods, including KRUM. The authors determined an issue in using an l_p norm for determining similarities between updates. In detail, the objective in a typical FL environment is to optimize in a high-dimensional and non-convex region. An attacker can effectively craft a malicious update that is very close to an honest update by leveraging the leeway these l_p norms present in high-dimension. Let d be the number of dimensions of θ_t. For example, with p being very small, an attacker can easily use an update that is almost identical to an honest one but differs in one coordinate by $\Omega(\sqrt[p]{d})$. This gets even worse when using very large p values or even the infinity norm, as an attacker can craft an update where each component differs by $\Omega(d)$. Since d is usually of large magnitudes in modern neural networks (typically in the range of one hundred billion parameters, e.g., [19]), this gives an attacker a very big leeway in crafting attacks. Based on these insights, the authors concluded that convergence is not enough for a secure aggregation rule and proposed a modification of KRUM, they coined BUYLAN. Due to the complexity of how BUYLAN works, and given the focus of this work on understanding why countermeasures fail in general, we will not investigate BUYLAN any further in this paper.

B. TRIMMEDMEAN *and* MEDIAN

Although, KRUM and especially BUYLAN provide very good robustness against attacks. They are costly to compute in terms of computing time, especially BUYLAN, which can considerably slow down the aggregation step. For this and other reasons, Yin et al. [12] have developed more straightforward aggregation rules that achieve somewhat similar robustness guarantees. The main difference is that they assume strong convexity of the objective function.

The first, so-called TRIMMEDMEAN, method calculates a component-wise mean of the values of the components where the smallest and largest values have been removed. More formally, for a chosen $\beta \in [0, \frac{1}{2})$:

$$\theta_{t+1}[i] \leftarrow \frac{1}{(1-2\beta)K} \sum_{m \in M_i} m, \tag{6}$$

where $\theta_t[i]$ denotes the i.th component of the update θ_t, and $M_i \subseteq \{\theta_{t,1}[i], \ldots, \theta_{t,n}[i]\}$ is the set of all i.th components where the $\lfloor \beta \cdot K \rfloor$ biggest and the $\lfloor \beta \cdot K \rfloor$ smallest values have been removed.

The second method, called component-wise MEDIAN aggregation, does as the name suggests and works as follows:

$$\theta_{t+1}[i] \leftarrow \mathsf{median}(\{\theta_{t,1}[i], \ldots, \theta_{t,n}[i]\}), \tag{7}$$

where median returns the median of a set.

Although they are very simple to compute, their guarantees for being robust requires the objective function to be strongly convex, which is a very hard criterion since it is usually not the case in most practical scenarios, especially in FL [7]. Furthermore, in the work of [20], the authors found that these methods substantially reduce the accuracy of complex models on non-i.i.d. data.

Summary

In summary, all of the mechanisms presented, attempt to identify a majority subset of all received updates by comparing them using a similarity measure and then using this subset as their basis for aggregation. This procedure is common in many defense mechanisms found in the literature. For example, BUYLAN [10], BRUTE [11] and GEOMED [10], [18] all operate similarly.

VI. STATE-OF-THE-ART ATTACKS

In this section, we will present two examples of attacks, with the first one trying to deviate the global model as much as possible from its optimum, while the second attack tries to install a backdoor into the global model. Both attacks are capable of overcoming the countermeasures we have just presented. Throughout both attacks, we will use the following assumption: an attacker has control over m compromised clients, and w.l.o.g, we assume that these are the clients numbered from $1, \ldots, m$.

A. Local Model Poisoning

In [8], the authors proposed several attacks for different countermeasures and different levels of knowledge by the adversary. Here, we will shortly introduce their proposed mechanism to attack an FL architecture that uses KRUM as its aggregation rule to demonstrate how such attacks work.

The main objective of this attack is to alter the global model parameters in a direction opposite to the one they would change without any attacks, effectively deviating them the most [8]. Meaning, for example, if in iteration t, the i.th component should be increased, we aim to find an adversary update that causes the i.th component to decrease as much as possible.

Here, we consider the case of an attacker who has knowledge about the aggregation function of the central server being KRUM and only has access to its compromised clients.

The attack begins by estimating the direction of parameter change, denoted as $\tilde{s}$. To accomplish this, the attack utilizes local updates that were sent by the clients prior to being compromised at iteration t'. The attack calculates an estimate $\tilde{\theta}_{t+1}$ of the global parameters at iteration $t'+1$ by taking the average of all updates:

$$\tilde{\theta}_{t+1} = \frac{1}{m} \sum_{i=1}^{m} \theta_{t',i} \ .$$

It subsequently calculates an estimate $\tilde{s}$ of the direction of change for each parameter, using the current parameters θ_t sent by the server at iteration t:

$$\tilde{s} = \mathsf{sign}(\tilde{\theta}_{t+1} - \theta_t), \tag{8}$$

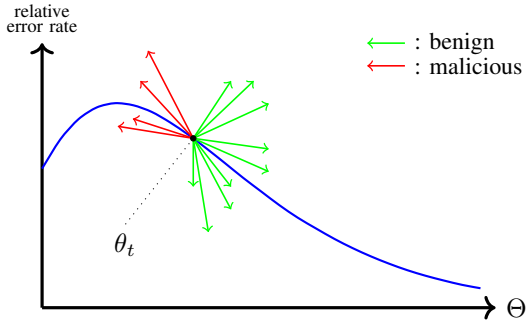

Fig. 2: i.i.d. data

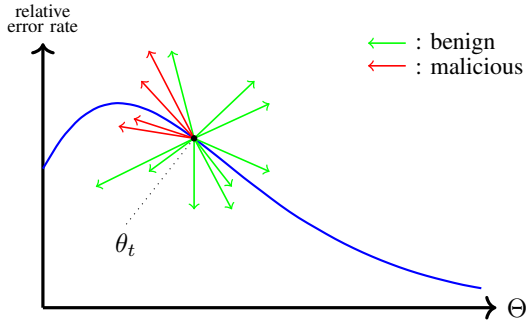

Fig. 3: non-i.i.d. data

Two hypothetical FL scenarios with the key difference being that in Figure 2 the training data is i.i.d. across all benign clients, while in Figure 3 the data is non-i.i.d. The blue curve illustrates the relative error rate when selecting the corresponding parameter value $\theta \in \Theta$. Moreover, the directional updates $\theta_{t,i}$ sent by client i from the prior global model parameters θ_t are displayed with arrows, where the color red indicates that the update was sent from a compromised client, and green from a benign client.

where sign is the signum function, that maps component wise positive values to 1, negative values to -1, and 0 to 0.

As KRUM selects only one update in the end and we only have knowledge about the previous benign updates sent at timestamp t', we can formulate the following optimization problem, where θ^* is the malicious update that will be sent later to the server:

$$\begin{aligned} & \max_{\lambda>0} \lambda \\ \text{subject to} \quad & \theta^* = \textsc{Krum}(\theta^*, \theta_{t',1}, \ldots, \theta_{t',m}) \\ & \theta^* = \theta_t - \lambda \widetilde{s} \end{aligned} \tag{9}$$

To solve this optimization problem efficiently, the attack uses an approximation algorithm that performs a binary search on λ. For each candidate, the algorithm checks whether the current λ satisfies the condition. If it does, it updates the parameter as follows $\lambda_{i+1} \leftarrow \frac{3}{4}\lambda_i$, and $\lambda_{i+1} \leftarrow \frac{1}{2}\lambda_i$ for the case it does not. In the paper [8], the authors also proved an upper bound used to initialize λ_0.

After obtaining a (sub-) optimal λ value, the attack randomly samples $m-1$ updates whose distance is at most ϵ to θ^* by a given l_p norm. Afterward, one randomly chosen compromised client sends out θ^*, while all other compromised clients send one of the samples to the central server. Note that ϵ and p are hyperparameters of the attack.

Additionally, the authors conducted an empirical study about the effectiveness of this attack on several common datasets. Their results are quite dramatic, as they can increase the error rate by relatively $400\,\%$ when using this attack compared to the regular error rate when no attack is present.

B. Backdoor Attack

The objective of the attacks is to achieve high accuracy on both the main task of the global model and an attacker-chosen backdoor subtask. A backdoor task is a small task where the global model produces an output specified by the attacker for specific input sequences chosen by the attacker. For instance, in the hospital scenario introduced in Section I, an adversary might seek to insert a backdoor that causes the global model to produce a positive test result for a specific disease every time an input indicates that the patient has blue eyes.

In its simplest form, the attack works as follows: Each compromised client trains on both its normal data and the backdoor task, such that each client's model learns to be good on its normal task and can differentiate it from the backdoor task and show good performance on it. With this technique, one can already achieve quite a good performance. In an experiment where a backdoor task was installed in a FL architecture designed to learn a word prediction model, this simple algorithm was able to achieve an average accuracy of over $80\,\%$ with the global model on the installed backdoors after 100 FL iterations. This was possible even when the attacker controlled only $10\,\%$ of all clients [7].

The authors of [7] even went a bit further and developed a model replacement technique that can outperform the previous approach while also being aware of possible defense and anomaly detectors used by the central server. Furthermore, they showed that attacking KRUM is fairly easy. As the training process is converging, an update $\theta_{t,i}$ is more likely to be selected if it is very close to the previous global model θ_t. Therefore, an attacker only has to construct model parameters that achieve high accuracy on the backdoor task and is very close to the previous global parameters.

In an empirical study, it was demonstrated that the probability of the central server selecting one of the malicious, close to the previous global model, updates in their test case already with probability greater than $80\,\%$ when the attacker controls only $0.2\,\%$ of all clients [7].

Summary

Despite the presence of countermeasures, both types of attacks have demonstrated their effectiveness. Moreover, as this section has demonstrated, the information about which countermeasure is used is precious as attackers can make

their attacks more efficient. The primary reason for the overall failure of these measures is due to the non-i.i.d. nature of the data, which we will delve into in greater detail in the following section.

VII. Discussion

In this section, we will start discussing our perspective on FL. In particular, we first consider a connection between FL countermeasures and statistical outlier detection. Then, we go on about how these countermeasures can also risk FL's privacy guarantee and, finally, the effect on the convergence behavior of the countermeasures.

A. Statisical Outlier Detection

Upon learning about the susceptibility of one of the simplest FL architectures due to its unbounded aggregation function in Section IV, combined with the more sophisticated defense mechanisms we introduce in Section V that still failed to protect the global model, as demonstrated with the two attacks presented in Section VI, it seems that FL is always vulnerable to adversarial influences. Notably, this vulnerability is not solely attributable to the choice of defense mechanism we made and presented in this paper, as for example the two attacks from the previous section can be modified to more complicated methods like BUYLAN or GEOMED [7], [8].

This raises the question of what all these countermeasures have in common that make them so susceptible for attacks and whether there exists a foolproof defense mechanism against them. To answer this, it is necessary to consider the larger picture and the limited information available to the central server. The only plausible assumption the server can really make is that the majority of clients are benign, otherwise the adversary would always be successful in manipulating the global model and any defense strategy would be pointless. Therefore, almost countermeasures try to compute a new global parameter that is very close by some measure to the majority of the updates received.

In other words, all these methods are just a sort of statistical outlier detection, where the task is to find the outliers that deviate the most of the majority and remove them. For better understanding this level of abstraction, see Figure 2 and Figure 3. The figures depict a hypothetical FL scenario where we plotted how the choice of the global parameter $\theta_t \in \Theta$ corresponds to a relative loss of the overall classifier on a fixed test dataset. The goal of the FL architecture is to derive a parameter that produce the least amount of error, meaning the FL architecture is trying to find the minimum of the blue curve. All parameters between the scenarios depicted in Figure 2 and Figure 3 are identical, with the key exception that the data in Figure 2 is i.i.d., while in Figure 3 it is not.

The central server in these scenarios sees only the current parameter θ_t and all updates by the clients $\theta_{t,i}$, which are indicated with an arrow for their direction and magnitude of change they propose. Based on this, the server must decide which subset of the arrows belongs to the benign majority, and compute a new global parameter based on this subset without wasting too much information. We colored the arrows red if they were sent by a compromised client and green if they were sent by a benign client, but note that the server obviously does not have this information.

In the scenario of Figure 2, it is quite easy to see that the majority of arrows are pointing to the right (the optimal direction), and only a handful of arrows are pointing to the left, such that a good countermeasure, would only use the arrow pointing to the right as its basis for the aggregation.

Unfortunately in practice, the central server will not have it that easy, and the reason for that is not that the amount of dimension increases significantly, since many statistical outlier detection algorithm work on arbitrary dimensions. The problem arises due to the non-i.i.d. nature of the training data used by each client. This presents a significant challenge since the updates received from clients can vary widely, depending on how unbalanced the data is across all clients, see Figure 3. In this scenario, the green arrows are more fanned out and dispersed. However, the crucial point is that the resulting arrow would still point in the right direction if only the green arrows would be averaged, meaning the green arrows alone would lead to the convergence to the global minimum. Unfortunately, the central server does not know which arrows belong to compromised clients and which to benign clients. Thus, the server is most likely to select all red arrows and some green arrows as the majority, as these arrows present the subset that is the closest together and point in a similar direction. Hence, the server will update the global parameter to the left and thereby consequently leading to a worse model. Since all countermeasures presented in Section V assume that the training data are i.i.d. to guarantee the convergence of the model to the optimum, we can safely assume that none of them will converge in the right direction in this hypothetical scenario either.

As a final point, one might ask why the server does not simply use a validation set to check that each update is benign or that the aggregated update leads to a better-performing model. The problem with this idea is that FL is typically used in use cases where the data remains confidential and cannot be shared with third parties, for example, the healthcare settings in [3], [4]. Second, and most importantly, because of their non-i.i.d. nature, a validation set can never represent the entire training data. Thus, if a validation set were to be used, the resulting global model would be biased toward the validation set and disregard any outlier cases present in a client's training data but not in the validation set.

B. Countermeasures as Privacy Concern

One of the most appealing aspects of FL concerning sensitive data is the possibility of not having to share them with the central server while still gaining all the benefits of training on a collective dataset. When discussing attacks, we usually assume that the server remains uncompromised while clients may be compromised. Although an attacker can gain access to all the information stored on a compromised client, it is usually not so easy to intercept other updates exchanged between clients and

the server because of the use of secure cryptographic protocols, such as TLS, between each client and the server.

However, an attacker can still manipulate countermeasures to expose these client updates. The most intriguing example involves the aggregation rule KRUM, which always selects a single update as the new global parameter for the next iteration that is consequently shared with each client. Let $\theta_{t,i}$ from client i be the update that KRUM selects in iteration t and client i is not compromised. An attacker can then send the update $\theta_{t,i}$ as the update of all its compromised clients in the next iteration $t+1$. Since the update sent by client i in iteration $t+1$ will be very similar to its update in iteration t, the attacker artificially increases the number of updates that are close to $\theta_{t+1,i}$, and thereby increases the likelihood that KRUM will select client i's update and share it with all clients again.

This enables an attacker to obtain multiple updates from the same client. Using techniques like those presented in [5], [6], the attacker can recreate the sensitive training data of client i.

C. Impact on the Learning Speed

Another factor we want to discuss is the resulting deceleration of the learning process when employing countermeasures. The gold standard in this regard is the standard averaging aggregation method when there is no Byzantine client, where each of the updates contributes equally to the resulting global parameter. We gave a formalization of this rule in Section II in Equation (2).

For this discussion, we must consider two factors: the time required to compute the aggregation rule and the number of iterations required to converge to an optimum.

First, more complicated aggregation methods like, for example, BUYLAN are more expensive to compute during the aggregation step than standard averaging. However, the authors of [11] developed an efficient implementation of BUYLAN that runs in $\mathcal{O}(n^2d + nd)$, where n represents the number of updates received by the server, and d denotes the dimension of θ. Nonetheless, when recalling that d is typically sufficiently larger than n with $d >> n$, typically in the range of one hundred billion, e.g., [19], and adding that standard averaging can be efficiently computed in $\mathcal{O}(nd)$, the difference in computing time is negligible.

Second, and most importantly, since almost all countermeasures are based on statistical outlier detections and remove a subset of updates that they flag as outliers from the calculation of the new global parameters, these countermeasures lose a lot of information when used only on benign updates. However, in the work of [10], the authors demonstrated that the deceleration of the convergence of the FL architecture when using KRUM with and without Byzantine clients can be controlled by increasing the local batch size for the local training process of the clients. With this insight, they further showed experimentally that the difference in the convergence behavior is negligible with a sufficiently large batch size.

From these results, we conclude that the deceleration of the convergence behavior is less significant than assumed. Thus, combined with the fact that standard averaging is very susceptible to simple attacks as shown in Section IV, and that a distributed system at large scale needs to be robust against Byzantine failures, we conclude that all countermeasures represented in this paper are superior to standard averaging.

VIII. CONCLUSION

Overall, Federated Learning (FL) remains a captivating technology due to its potential for providing privacy and scalability while also allowing for data to be non-i.i.d. This makes this technology easy to deploy, which consequently makes it so appealing. However, as this paper has demonstrated thoughtfully, one must approach FL carefully. We conclude our work by offering two guidelines for those considering using an FL architecture. These guidelines differ depending on whether the data is highly unbalanced or whether the unbalance is kept to a minimum and the data is nearly i.i.d.:

1) In cases where the data is i.i.d. or nearly i.i.d., with the unbalance being contained by some measure, we can be very relaxed about the requirements for joining the FL architecture as a client. As long as the operator of this architecture can guarantee that the majority of clients are benign, the global model will be safe. As shown in Section VII, countermeasures can be quite effective in these scenarios. Examples of such scenarios include [21], [22].
2) In cases where the data is non-i.i.d. and very unbalanced, it is not advisable to allow arbitrary clients to constantly join and leave the system without proper authorization from a trusted third party. As we have demonstrated in Section VII, the ability to detect adversary influences of countermeasure is very limited when the data is non-i.i.d. However, even when clients can be trusted not to intentionally manipulate the global model, they may still be prone to Byzantine failures, leading to updates that are diametrically opposed to the intended direction. In such cases, countermeasures must still be employed to mitigate such risks.

   As an example of such a scenario where the operator of an FL architecture can trust all its participants, we can recall the healthcare example [3], [4] introduced in Section I. Here, the operator must first talk to all participating hospitals. After each of them agrees to participate in the FL architecture, the operator can assume that all of them are benign.

FL architecture operators can gain initial insight from these guidelines, but caution is needed with very unbalanced data. However, the second guideline is more cautious and goes against the vision of FL's inventors. They envisioned FL as a distributed learning technology that could scale perfectly by allowing clients to join and leave at any time, with no constraints on their training data. However, as this work has shown and discussed, we do not believe this vision is liable, as there seems to be no absolute protection against malicious interference in this scenario.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[3] A. Jochems, T. M. Deist, J. van Soest, M. Eble, P. Bulens, P. Coucke, W. Dries, P. Lambin, and A. Dekker, "Distributed learning: Developing a predictive model based on data from multiple hospitals without data leaving the hospital - a real life proof of concept," *Radiother Oncol*, vol. 121, no. 3, pp. 459–467, Oct. 2016.

[4] I. e. a. Dayan, "Federated learning for predicting clinical outcomes in patients with covid-19," *Nature Medicine*, vol. 27, no. 10, pp. 1735–1743, Oct 2021. [Online]. Available: https://doi.org/10.1038/s41591-021-01506-3

[5] J. Zhu and M. B. Blaschko, "R-GAP: recursive gradient attack on privacy," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=RSU17UoKfJF

[6] M. Balunovic, D. I. Dimitrov, R. Staab, and M. Vechev, "Bayesian framework for gradient leakage," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=f2lrIbGx3x7

[7] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2938–2948. [Online]. Available: https://proceedings.mlr.press/v108/bagdasaryan20a.html

[8] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to Byzantine-Robust federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1605–1622. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/fang

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[10] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf

[11] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in Byzantium," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3521–3530. [Online]. Available: https://proceedings.mlr.press/v80/mhamdi18a.html

[12] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 5650–5659. [Online]. Available: https://proceedings.mlr.press/v80/yin18a.html

[13] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, Jun. 2010, pp. 456–464. [Online]. Available: https://aclanthology.org/N10-1069

[14] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," 2015.

[15] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of dnns with natural gradient and parameter averaging," 2015.

[16] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, jul 1982. [Online]. Available: https://doi.org/10.1145/357172.357176

[17] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *CoRR*, vol. abs/2003.02133, 2020. [Online]. Available: https://arxiv.org/abs/2003.02133

[18] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," 2017.

[19] A. Trask, D. Gilmore, and M. Russell, "Modeling order in neural word embeddings at scale," 2015.

[20] X. Chen, T. Chen, H. Sun, S. Z. Wu, and M. Hong, "Distributed training with heterogeneous data: Bridging median- and mean-based algorithms," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 616–21 626. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/f629ed9325990b10543ab5946c1362fb-Paper.pdf

[21] J. Markoff, "How many computers to identify a cat? 16,000," pp. 06–25, 2012.

[22] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017.