Securing Federated Learning against Extreme Model Poisoning Attacks via Multidimensional Time Series Anomaly Detection on Local Updates

Edoardo Gabrielli, Dimitri Belli, Zoe Matrullo, Vittorio Miori, Gabriele Tolomei

***Abstract*—Current defense mechanisms against model poisoning attacks in federated learning (FL) systems have proven effective up to a certain threshold of malicious clients (e.g., 25% to 50%). In this work, we introduce FLANDERS, a novel pre-aggregation filter for FL that is resilient to large-scale model poisoning attacks, i.e., when malicious clients far exceed legitimate participants. FLANDERS treats the sequence of local models sent by clients in each FL round as a matrix-valued time series. Then, it identifies malicious client updates as outliers in this time series by comparing actual observations with estimates generated by a matrix autoregressive forecasting model maintained by the server. Experiments conducted in several non-iid FL setups show that FLANDERS significantly improves robustness across a wide spectrum of attacks when paired with standard and robust aggregation methods.**

***Index Terms*—Federated Learning, Robustness, Model Poisoning Attacks, Deep Learning, Anomaly Detection, Security.**

I. INTRODUCTION

Recently, *federated learning* (FL) has emerged as the leading paradigm for training distributed, large-scale, and privacy-preserving machine learning (ML) systems [1], [2]. The core idea of FL is to allow multiple edge clients to collaboratively train a shared, global model without disclosing their local private training data. A typical FL round involves the following steps: *(i)* the server randomly picks some clients and sends them the current, global model; *(ii)* each selected client locally trains its model with its own private data; then, it sends the resulting local model to the server;[1] *(iii)* the server updates the global model by computing an *aggregation function*, usually the average (FedAvg), on the local models received from clients. This process goes on until the global model converges.

Although its advantages over standard ML, FL also raises security concerns [3]. Here, we focus on *untargeted model poisoning* attacks [4], where an adversary attempts to tweak the global model weights by directly perturbing the local model's parameters of some infected clients before these are sent to the central server for aggregation. In doing so, the adversary aims to jeopardize the global model *indiscriminately* at inference time. Such model poisoning attacks severely impact standard FedAvg; therefore, more robust aggregation functions must be designed to secure FL systems.

Unfortunately, existing defense mechanisms either rely on simple heuristics (e.g., Trimmed Mean and FedMedian by [5]) or need strong and unrealistic assumptions to work effectively (e.g., foreknowledge or estimation of the number of malicious clients in the FL system, as for Krum/Multi-Krum [6] and Bulyan [7], which, however, cannot exceed a fixed threshold). Furthermore, outlier detection methods using K-means clustering [8] or spectral analysis like DnC [9] do not directly consider the temporal evolution of local model updates received. Finally, strategies like FLTrust [10] require the server to collect its own dataset and act as a proper client, thereby altering the standard FL protocol.

This work introduces a novel pre-aggregation *filter* robust to untargeted model poisoning attacks. Notably, this filter (i) operates without requiring prior knowledge or constraints on the number of malicious clients and (ii) inherently integrates temporal dependencies. The FL server can employ this filter as a preprocessing step before applying *any* aggregation function, be it standard like FedAvg or robust like Krum or Bulyan. Specifically, we formulate the problem of identifying corrupted updates as a multidimensional (i.e., matrix-valued) time series anomaly detection task. The key idea is that legitimate local updates, resulting from well-calibrated iterative procedures like stochastic gradient descent (SGD) with an appropriate learning rate, show *higher predictability* compared to malicious updates. This hypothesis stems from the fact that the sequence of gradients (thus, model parameters) observed during legitimate training exhibit regular patterns, as validated in Section IV-B. Inspired by the matrix autoregressive (MAR) framework for multidimensional time series forecasting [11], we propose the FLANDERS (***F**ederated **L**earning meets **AN**omaly **DE**tection for a **R**obust and **S**ecure*) filter. The main advantages of FLANDERS over existing strategies like FLDetector [12] are its resilience to large-scale attacks, where 50% or more FL participants are hostile, and the capability of working under realistic non-iid scenarios. We attribute such a capability to two key factors: (i) FLANDERS works without knowing a priori the ratio of corrupted clients, and (ii) it embodies temporal dependencies between intra- and inter-client updates, quickly recognizing local model drifts caused by evil players. Below, we summarize our main contributions:

Edoardo Gabrielli (corresponding author) is with the Department of Computer, Control and Management Engineering, Sapienza University of Rome, Italy. E-mail: edoardo.gabrielli@uniroma1.it.

Dimitri Belli and Vittorio Miori are with the National Research Council in Pisa, Italy.

Zoe Matrullo is with the Department of Statistics, Ludwig-Maximilians-University of Munich, Germany.

Gabriele Tolomei is with the Department of Computer Science, Sapienza University of Rome, Italy.

[1]Whenever we refer to global/local model, we mean global/local model *parameters*.

(i) We provide empirical evidence that the sequence of models sent by legitimate clients is more predictable than those of malicious participants performing untargeted model poisoning attacks.

(ii) We introduce FLANDERS, the first pre-aggregation filter for FL robust to untargeted model poisoning based on multidimensional time series anomaly detection.

(iii) We integrate FLANDERS into Flower,[2][3] a popular FL simulation framework for reproducibility.

(iv) We show that FLANDERS improves the robustness of the existing aggregation methods under multiple settings: different datasets, client's data distribution (non-iid), models, and attack scenarios.

(v) We publicly release all the implementation code of FLANDERS along with our experiments.[4]

The remainder of the paper is structured as follows. Section II covers background and preliminaries. In Section III, we discuss related work. Section IV and Section V describe the problem formulation and the method proposed. Section VI gathers experimental results. Finally, we conclude in Section VIII.

II. BACKGROUND AND PRELIMINARIES

A. Federated Learning

We consider a typical supervised learning task under a standard FL setting, which consists of a central server S and a set of distributed clients $\mathcal{C}$, such that $|\mathcal{C}| = K$. Each client $c \in \mathcal{C}$ has its own private training set $\mathcal{D}_c$, namely the set of its n_c local labeled examples, i.e., $\mathcal{D}_c = \{\boldsymbol{x}_{c,i}, y_{c,i}\}_{i=1}^{n_c}$.
The goal of FL is to train a global predictive model whose architecture and parameters $\boldsymbol{\theta}^* \in \mathbb{R}^d$ are shared across all clients by solving $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \text{argmin}_{\boldsymbol{\theta}} \sum_{c=1}^{K} p_c \mathcal{L}_c(\boldsymbol{\theta}; \mathcal{D}_c)$, where $\mathcal{L}_c$ is the local objective function for client c. Usually, this is defined as the empirical risk calculated over the training set $\mathcal{D}_c$ sampled from the client's local data distribution: $\mathcal{L}_c(\boldsymbol{\theta}; \mathcal{D}_c) = \frac{1}{n_c} \sum_{i=1}^{n_c} \ell(\boldsymbol{\theta}; (\boldsymbol{x}_{c,i}, y_{c,i}))$, where ℓ is an instance-level loss, e.g., cross-entropy (classification) or squared error (regression). Each $p_c \geq 0$ specifies the relative contribution of each client. Since it must hold that $\sum_{c=1}^{K} p_c = 1$, two possible settings are: $p_c = 1/K$ or $p_c = n_c/n$, where $n = \sum_{c=1}^{K} n_c$.

The generic federated round at each time t is decomposed into the following steps and iteratively performed until convergence, i.e., for each $t \in \{1, 2, \ldots, T\}$:

(i) S randomly selects a subset of clients $\mathcal{C}^{(t)} \subseteq \mathcal{C}$, so that $1 \leq |\mathcal{C}^{(t)}| \leq K$, and sends them the current, global model $\boldsymbol{\theta}^{(t)}$. At $t = 1$, $\boldsymbol{\theta}^{(1)}$ is randomly initalized.

(ii) Each selected client $c \in \mathcal{C}^{(t)}$ trains its local model $\boldsymbol{\theta}_c^{(t)}$ on its own private data $\mathcal{D}_c$ by optimizing the following objective, starting from $\boldsymbol{\theta}^{(t)}$:

$$\boldsymbol{\theta}_c^{(t)} = \text{argmin}_{\boldsymbol{\theta}^{(t)}} \mathcal{L}_c(\boldsymbol{\theta}^{(t)}; \mathcal{D}_c). \tag{1}$$

The value of $\boldsymbol{\theta}_c^{(t)}$ is computed via gradient-based methods (e.g., stochastic gradient descent) and sent to S.

(iii) S computes $\boldsymbol{\theta}^{(t+1)} = \phi(\{\boldsymbol{\theta}_c^{(t)} \mid c \in \mathcal{C}^{(t)}\})$ as the updated global model, where $\phi : \mathbb{R}^{d^m} \mapsto \mathbb{R}^d$ is an *aggregation function*; for example, $\phi = \frac{1}{m} \sum_{c \in \mathcal{C}^{(t)}} \boldsymbol{\theta}_c^{(t)}$, i.e., FedAvg or alike [13].

B. The Attack Model: Federated Aggregation under Model Poisoning

The most straightforward aggregation function ϕ the server can implement is FedAvg, which computes the global model as the average of the local model weights received from clients. FedAvg is effective when all the FL participants behave honestly [2], [14], [15]. Instead, this work assumes that an attacker controls a fraction $b = \lceil r * K \rceil, r \in [0, 1]$ of the K clients, i.e., $0 \leq b \leq K$, known as malicious. This is in contrast to previous works, where $r < 0.5$. Below, we describe our attack model.

Attacker's Goal. Inspired by many studies on poisoning attacks against ML [16]–[22], we consider the attacker's goal is to jeopardize the jointly learned global model *indiscriminately* at inference time with *any* test example. Such attacks are known as *untargeted* [23], as opposed to *targeted* poisoning attacks, where instead, the goal is to induce prediction errors only for some specific test inputs (e.g., via so-called *backdoor triggers* as shown by [24]).

Attacker's Capability. Like Sybil attacks to distributed systems [25], the attacker can inject b fake clients into the FL system or compromise b honest clients. The attacker can arbitrarily manipulate the local models sent by malicious clients to the server S. More formally, let $x \in \mathcal{C}$ be one of the b corrupted clients selected by the server on the generic t-th FL round; it first computes its legitimate local model $\boldsymbol{\theta}_x^{(t)}$ without modifying its private data $\mathcal{D}_x$; then it finds $\tilde{\boldsymbol{\theta}}_x^{(t)}$ by applying a *post hoc* perturbation $\boldsymbol{\varepsilon} \in \mathbb{R}^d$ to $\boldsymbol{\theta}_x^{(t)}$. For example, $\tilde{\boldsymbol{\theta}}_x^{(t)} = \boldsymbol{\theta}_x^{(t)} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a Gaussian noise vector. More advanced attack strategies have been designed, as discussed in Section VI-B.

Attacker's Knowledge. We assume the attacker knows its controlled clients' code, local training datasets, and local models. Moreover, we consider the worst-case scenario, where the attacker is *omniscient*, i.e., it has full knowledge of the parameters sent by honest parties. This allows the attacker to crafting malicious local models that are close to legitimate ones, thus increasing the probability of being chosen by the server for the aggregation.

Attacker's Behavior. In each FL round, the attacker, similar to the server, randomly selects b clients to corrupt out of the K available. Any of these b malicious clients that happen to be among those selected by the server will poison their local models using one of the strategies outlined in Section VI-B. Note that, unlike earlier studies, we do not assume that malicious clients are chosen in the initial round and remain constant throughout the training. We argue that this approach is more realistic, as legitimate clients can be compromised at any point in actual deployments. Therefore, an effective defense in

[2] https://flower.dev/

[3] https://github.com/adap/flower/tree/main/baselines/flanders

[4] https://anonymous.4open.science/r/flanders_exp-7EEB

FL should ideally exclude these clients as soon as they submit their first malicious model.

This category of *untargeted model poisoning* attacks has been extensively explored in previous works [4], [6], [23]. It has been shown that including updates even from a single malicious client can wildly disrupt the global model if the server runs standard FedAvg [5], [6].

III. RELATED WORK

FL is vulnerable to untargeted model poisoning attacks during model aggregation, where malicious clients can manipulate updates to compromise the global model. Existing attacks employ diverse strategies, ranging from noise injection to sophisticated optimization techniques. These approaches represent key threats in FL.

Gaussian Noise Attack. This attack randomly crafts the local models on the compromised clients. Specifically, the attacker samples a random value from a Gaussian distribution $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ and sums it to all the d learned parameters. We refer to this attack as GAUSS.

"A Little Is Enough" Attack [26]**.** This attack shifts the aggregation gradient by carefully crafting malicious values that deviate from the correct ones as far as possible. We call this attack LIE.

Optimization-based Attack [23]**.** This attack is framed as an optimization task, aiming to maximize the distance between the poisoned aggregated gradient and the aggregated gradient under no attack. By using a halving search, one can obtain a crafted malicious gradient. We refer to this attack as OPT.

AGR Attack Series [9]**.** This improves the optimization program above by introducing perturbation vectors and scaling factors. Then, three instances are proposed: AGR-tailored, AGR-agnostic Min-Max, and Min-Sum, which maximize the deviation between benign and malicious gradients. In this work, we experiment with AGR Min-Max, which we call AGR-MM.

Below, we describe the most popular defenses against model poisoning attacks on FL systems, which will serve as the baselines for our comparison. A more comprehensive discussion is in [27], [28].

FedMedian [5]**.** The central server sorts the j-th parameters received from all the m local models and takes the median of those as the value of the j-th parameter of the global model. This process is applied for all the model parameters, i.e., $\forall j \in \{1, \ldots, d\}$.

Trimmed Mean [29]**.** This rule computes a model as FedMedian does, and then it averages the k nearest parameters to the median. If b clients are compromised at most, this aggregation rule achieves an order-optimal error rate when $b \leq \frac{m}{2} - 1$.

Krum [6]**.** It selects one of the m local models received from the clients, which is most similar to *all* other models, as the global model. The rationale behind this approach is that even if the chosen local model is poisoned, its influence would be restricted because it resembles other local models, potentially from benign clients. A variant called Multi-Krum mixes Krum with standard FedAvg.

Bulyan [7]**.** Since a single component can largely impact the Euclidean distance between two high-dimensional vectors, Krum may lose effectiveness in complex, high-dimensional parameter spaces due to the influence of a few abnormal local model weights. Bulyan iteratively applies Krum to select α local models and aggregates them with a Trimmed Mean variant to mitigate this issue.

DnC [9]**.** This robust aggregation uses spectral analysis to detect and filter outliers as proposed by [30]. Similarly, DnC computes the principal component of the set of local updates sent by clients. Then, it projects each local update onto this principal component. Finally, it removes a constant fraction of the submitted model updates with the largest projections.

FLDetector [31]**.** This method is the closest to our approach. It filters out malicious clients by measuring their consistency across N rounds using an approximation of the integrated Hessian to compute the *suspicious scores* for all clients. However, unlike our method, FLDetector struggles with large numbers of malicious clients, and cannot work with highly heterogeneous data.

IV. PROBLEM FORMULATION

A. Time Series of Local Models

At the end of each FL round t, the central server S collects the updated local models $\{\boldsymbol{\theta}_c^{(t)}\}_{c \in \mathcal{C}^{(t)}}$ sent by the subset of selected clients.[5] Without loss of generality, we assume that the number of clients picked at each round is constant and fixed, i.e., $|\mathcal{C}^{(t)}| = m, \ \forall t \in \{1, 2, \ldots, T\}$. Hence, the server arranges the local models received at round t into a $d \times m$ matrix $\boldsymbol{\Theta}_t = [\boldsymbol{\theta}_1^{(t)}, \ldots, \boldsymbol{\theta}_c^{(t)}, \ldots, \boldsymbol{\theta}_m^{(t)}]$, whose c-th column corresponds to the d-dimensional vector of updated parameters $\boldsymbol{\theta}_c^{(t)}$ sent by client c. However, the subset of selected clients may be different at each FL round, i.e., $\mathcal{C}^{(t)} \neq \mathcal{C}^{(t')}$ for $t \neq t'$, although we have assumed their size (m) is the same. More generally, to track the local models sent by *all* clients chosen across $1 \leq w \leq T$ rounds, we can extend each $\boldsymbol{\Theta}_t$ into a $d \times h$ matrix, where $h = |\bigcup_{t=1}^{w} \mathcal{C}^{(t)}|$, such that $m \leq h \leq K$, with K being the total number of clients. Notice that $h = m$ when $w = 1$, whereas $h = K$ if the server selects all K clients at least once over the w rounds considered. At every round t, $\boldsymbol{\Theta}_t$ contains m columns corresponding to the local models sent by the clients *actually* selected for that round. In contrast, the remaining $h - m$ "fictitious" columns refer to the unselected clients. We fill these columns with the current global model at FL round t, i.e., $\boldsymbol{\theta}^{(t)}$, as if this were sent by the $h - m$ unselected clients. Note that this strategy is neutral and will not impact the aggregated global model computed by the server for the next round $(t + 1)$, as this is calculated only from the updates received by the m clients previously selected.

B. Predictability of Local Models Evolution: Legitimate vs. Malicious Clients

We claim that legitimate local updates show *higher predictability* compared to malicious updates. This hypothesis stems from the fact that the sequence of gradients (hence,

[5] A similar reasoning would apply if clients sent their local displacement vectors $\boldsymbol{u}_c^{(t)} = \boldsymbol{\theta}_c^{(t)} - \boldsymbol{\theta}^{(t)}$ or local gradients $\nabla \mathcal{L}_c^{(t)}$ rather than local model parameters $\boldsymbol{\theta}_c^{(t)}$.

model parameters) observed during legitimate training should exhibit "regular" patterns until convergence.

To demonstrate this behavior, we consider two clients $i, j \in \mathcal{C}$. Client i is assumed to be a legitimate participant, while client j acts maliciously. Both are selected by the server over a sequence of consecutive T FL rounds to train a global model on the *MNIST* dataset. Therefore, we examine the local models sent to the server at each round $t \in \{1, 2, \ldots, T\}$ by clients i and j. Specifically, these are d-dimensional vectors of real-valued parameters $\boldsymbol{\theta}_i^{(t)}, \boldsymbol{\theta}_j^{(t)} \in \mathbb{R}^d$, such that $\boldsymbol{\theta}_i^{(t)} = (\theta_{i,1}^{(t)}, \ldots, \theta_{i,d}^{(t)})$ and $\boldsymbol{\theta}_j^{(t)} = (\theta_{j,1}^{(t)}, \ldots, \theta_{j,d}^{(t)})$, respectively. Next, we calculate the average time-delayed mutual information (TDMI) for each pair of observed local models $(\boldsymbol{\theta}_i^{(t)}, \boldsymbol{\theta}_i^{(t')})$ and $(\boldsymbol{\theta}_j^{(t)}, \boldsymbol{\theta}_j^{(t')})$ across $T = 50$ rounds, where $\delta > 0$, for both client i and j. This analysis aims to discern the time-dependent nonlinear correlation and the level of predictability between observations. We expect the TDMI calculated between pairs of models sent by the legitimate client i to be higher than that computed between pairs sent by the malicious client j. Thus, the temporal sequence of local models observed from client i is more predictable than that of client j.

Let $\boldsymbol{\theta}_c^{(t)} = (\theta_{c,1}^{(t)}, \ldots, \theta_{c,d}^{(t)})$ denote the local model sent by the generic FL client $c \in \mathcal{C}$ as an instance of a d-dimensional vector-valued process. We define $\text{TDMI}(\boldsymbol{\theta}_c^{(t)}, \boldsymbol{\theta}_c^{(t')})$ as follows:

$$\text{TDMI}(\boldsymbol{\theta}_c^{(t)}, \boldsymbol{\theta}_c^{(t')}) = \int p(\boldsymbol{\theta}_c^{(t)}, \boldsymbol{\theta}_c^{(t')}) \times \log \left(\frac{p(\boldsymbol{\theta}_c^{(t)}, \boldsymbol{\theta}_c^{(t')})}{p(\boldsymbol{\theta}_c^{(t)}) p(\boldsymbol{\theta}_c^{(t')})} \right) d\boldsymbol{\theta}_c^{(t)} d\boldsymbol{\theta}_c^{(t')}, \tag{2}$$

where $p(\cdot)$ is the probability density function and $t' = t + \delta$. If each realization $\theta_{c,k}^{(t)}$ in the vector $\boldsymbol{\theta}_c^{(t)}$ is *independent* from each other, we can compute the average TDMI as follows [32]:

$$\text{Avg}\left[\text{TDMI}(\boldsymbol{\theta}_c^{(t)}, \boldsymbol{\theta}_c^{(t')})\right] = \frac{1}{d}\left[\sum_{k=1}^{d} \text{TDMI}(\theta_{c,k}^{(t)}, \theta_{c,k}^{(t')})\right], \tag{3}$$

where $\text{TDMI}(\theta_{c,k}^{(t)}, \theta_{c,k}^{(t')})$ calculates TDMI for the univariate case. Indeed, the d model parameters $\theta_{c,k}^{(\cdot)}$ should reasonably be independent. Under this assumption, the joint probability density function of the parameters factors into a product of individual probability density functions, forming a product measure on d-dimensional Euclidean space. Thus, according to Fubini's theorem [33], the integral of each parameter will be independent of the others because the parameters are independent.

To mimic the behavior of a hypothetical optimal FLANDERS filter, and therefore avoid the propagation of poisoned local models sent by client j across the T rounds, we assume that, at each round t, the global model from which both clients start their local training process is polished from any malicious updates received at the previous round $t-1$. Next, we calculate the average time-delayed mutual information (TDMI) for each pair of observed local models $(\boldsymbol{\theta}_i^{(t)}, \boldsymbol{\theta}_i^{(t')})$ and $(\boldsymbol{\theta}_j^{(t)}, \boldsymbol{\theta}_j^{(t')})$ across $T = 50$ rounds, where $t' > t$, for both client i and j. Firstly, we consider the special case where $t' = t + 1$ and compute the average TDMI between each pair of *consecutive* local models sent by the legitimate client and the malicious client, when this runs one of the four attacks considered in this work, namely GAUSS, LIE, OPT, and AGR-MM presented in Section VI-B. In Figure 1, we plot the empirical distributions of the observed average TDMI for the legitimate and malicious clients. To validate our claim that legitimate models are more

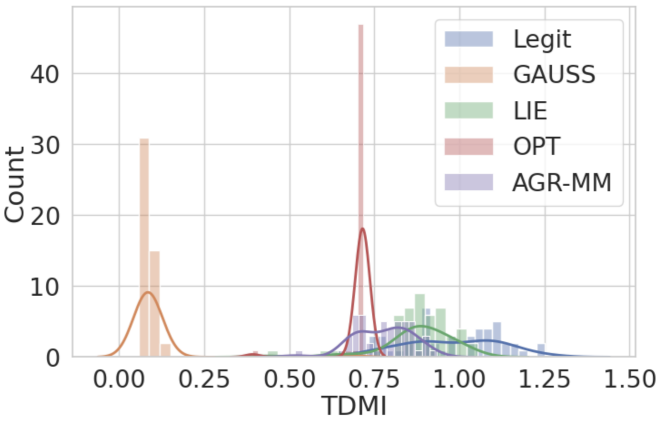

Fig. 1: Empirical distributions of average TDMI computed between each pair of *consecutive* local models sent by the legitimate client i $(\boldsymbol{\theta}_i^{(t)}, \boldsymbol{\theta}_i^{(t+1)})$ and the malicious client j $(\boldsymbol{\theta}_j^{(t)}, \boldsymbol{\theta}_j^{(t+1)})$, when this runs one of the four attacks considered in this work, namely GAUSS, LIE, OPT, and AGR-MM.

predictable than malicious ones, we compute the mean of the empirical distributions for the legitimate and malicious client, $\bar{\boldsymbol{\theta}}_i$ and $\bar{\boldsymbol{\theta}}_j^{atk}$, respectively, where $atk = \{\text{GAUSS, LIE, OPT, AGR-MM}\}$. Then, we run a one-tailed t-test against the null hypothesis $H_0 : \bar{\boldsymbol{\theta}}_i = \bar{\boldsymbol{\theta}}_j^{atk}$, where the alternative hypothesis is $H_a : \bar{\boldsymbol{\theta}}_i > \bar{\boldsymbol{\theta}}_j^{atk}$. The results of these statistical tests are illustrated in Table I, showing that, in all four cases, there is enough evidence to reject the null hypothesis at a confidence level $\alpha = 0.01$.

TABLE I: One-tailed t-test against the null hypothesis $H_0 : \bar{\boldsymbol{\theta}}_i = \bar{\boldsymbol{\theta}}_j^{atk}$. Each cell contains the p-value for the statistical test corresponding to a specific attack. In all four cases, there is enough evidence to reject the null hypothesis at a confidence level $\alpha = 0.01$ (p-value $\ll 0.01$).

	$H_0 : \bar{\boldsymbol{\theta}}_i = \bar{\boldsymbol{\theta}}_j^{atk}$
GAUSS	$5.67 * 10^{-38}$
LIE	$8.33 * 10^{-5}$
OPT	$6.70 * 10^{-18}$
AGR-MM	$4.20 * 10^{-12}$

C. Poisoned Local Models as Matrix-Based Time Series Outliers

We, therefore, formulate our problem as a multidimensional time series anomaly detection task. At a high level, we want to equip the central server with an anomaly scoring function that estimates the degree of each client picked at the current round being malicious, i.e., its *anomaly score*, based on the historical observations of model updates seen so far from the clients selected. Such a score will be used to restrict the set of

trustworthy candidate clients for the downstream aggregation method. Note that unselected clients will *not* contribute to the aggregation; thus, there is no need to compute their anomaly score even if they were marked as suspicious in some previous rounds. On the other hand, we must design a fallback strategy for "cold start" clients – i.e., FL participants who send their local updates for the first time or have not been selected in any of the previous rounds considered – whether honest or malicious.

More formally, let $\hat{\mathbf{\Theta}}_t = f(\mathbf{\Theta}_{t-w:t-1}; \hat{\mathbf{\Omega}})$ be the matrix of local model updates *predicted* by the central server S at the generic FL round t. The forecasting model f depends on a set of parameters $\hat{\mathbf{\Omega}}$ estimated using the past w model updates observed, i.e., $\mathbf{\Theta}_{t-w:t-1}$, where $1 \leq w \leq t-1$. Also, the server sees the actual matrix $\mathbf{\Theta}_t$.

The anomaly score $s_c^{(t)} \in \mathbb{R}$ of the generic client c at round t can thus be defined as follows:

$$s_c^{(t)} = \begin{cases} \delta(\boldsymbol{\theta}_c^{(t)}, \hat{\boldsymbol{\theta}}_c^{(t)}), \text{ if } c \in \mathcal{C}^{(t)} \wedge \exists j \in \{1..w\} \text{ s.t. } c \in \mathcal{C}^{(t-j)} \\ \delta(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}_c^{(t)}), \text{ if } c \in \mathcal{C}^{(t)} \wedge \nexists j \in \{1..w\} \text{ s.t. } c \in \mathcal{C}^{(t-j)} \\ \perp, \text{ if } c \notin \mathcal{C}^{(t)}. \end{cases} \tag{4}$$

The first condition refers to a client selected for round t, which also appeared at least once in the history. In this case, δ measures the distance between the observed vector of weights sent to the server ($\boldsymbol{\theta}_c^{(t)}$) and the predicted vector of weights ($\hat{\boldsymbol{\theta}}_c^{(t)}$) output by the forecasting model f. The second condition, instead, occurs when a client is selected for the first time at round t or does not appear in the previous w historical matrices of observations used to generate predictions. Here, due to the cold start problem, we cannot rely on f and, therefore, the most sensible strategy is to compute the distance between the current global model and the local update received by the new client. Finally, the anomaly score is undefined ($\perp$) for any client not selected for round t.

The server will thus rank all the m selected clients according to their anomaly scores (e.g., from the lowest to the highest). Several strategies can be adopted to choose which model updates should be aggregated in preparation for the next round, i.e., to restrict from the initial set $\mathcal{C}^{(t)}$ to another (possibly smaller) set $\mathcal{C}_*^{(t)} \subseteq \mathcal{C}^{(t)}$ of trusted clients. For example, S may consider only the model updates received from the top-k clients ($1 \leq k \leq m$) with the smallest anomaly score, i.e., $\mathcal{C}_*^{(t)} = \{c \in \mathcal{C}^{(t)} \mid s_c^{(t)} \leq s_k^{(t)}\}$, where $s_k^{(t)}$ indicates the k-th smallest anomaly score at round t. Alternatively, the raw anomaly scores computed by the server can be converted into well-calibrated probability estimates. Here, the server sets a threshold $\rho \in [0, 1]$ and aggregates the weights only of those clients whose anomaly score is below ρ, i.e., $\mathcal{C}_*^{(t)} = \{c \in \mathcal{C}^{(t)} \mid s_c^{(t)} \leq \rho\}$. In the former case, the number of considered clients (k) is bound apriori,[6] whereas the latter does not put any constraint on the size of final candidates $|\mathcal{C}_*^{(t)}|$. Eventually, S will compute the updated global model $\boldsymbol{\theta}^{(t+1)} = \phi(\{\boldsymbol{\theta}_c^{(t)} \mid c \in \mathcal{C}_*^{(t)}\})$, where ϕ is any aggregation function, e.g., FedAvg, Bulyan, or any other strategy.

[6]This may not be true if anomaly scores are not unique; in that case, we can simply enforce $|\mathcal{C}_*^{(t)}| = k$.

Below, we describe how we use the matrix autoregressive (MAR) framework proposed by [11] to implement our multidimensional time series forecasting model f, hence the anomaly score.

V. PROPOSED METHOD: FLANDERS

In Figure 2, we depict how FLANDERS computes the anomaly score vector $\boldsymbol{s}^{(t)}$ at the generic FL round t. Specifically, the server S uses its current MAR(1) forecasting model f whose best parameters $\hat{\mathbf{\Omega}} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}\}$ are estimated from l previous historical observations of local models received in the previous rounds $t-l, \ldots, t-1$. It applies f to the previously observed matrix $\mathbf{\Theta}_{t-1}$ to get the next predicted matrix of local models $\hat{\mathbf{\Theta}}_t$, i.e., $\hat{\mathbf{\Theta}}_t = f(\mathbf{\Theta}_{t-1}; \hat{\mathbf{\Omega}}) = \hat{\boldsymbol{A}}\mathbf{\Theta}_{t-1}\hat{\boldsymbol{B}}$. Then, it compares this predicted matrix $\hat{\mathbf{\Theta}}_t$ with the actual matrix of local updates received $\mathbf{\Theta}_t$. The final anomaly score vector is calculated by measuring the distance δ between each column of those two matrices, according to Eq. 4.

It is worth remarking that, in general, only some clients are selected at every round. In particular, if a client c_{new} – whether it is honest or malicious – is selected by the server for the first time at round t, the MAR forecasting model f will not be able to make any prediction for it due to a cold start problem (i.e., the predicted matrix $\hat{\mathbf{\Theta}}_t$ will not contain a column corresponding to c_{new}). In such a case, we must adopt a fallback strategy. Without any historical information for a client, the most sensible thing to do is to compute the distance δ between the local model update it sent and the current global model.

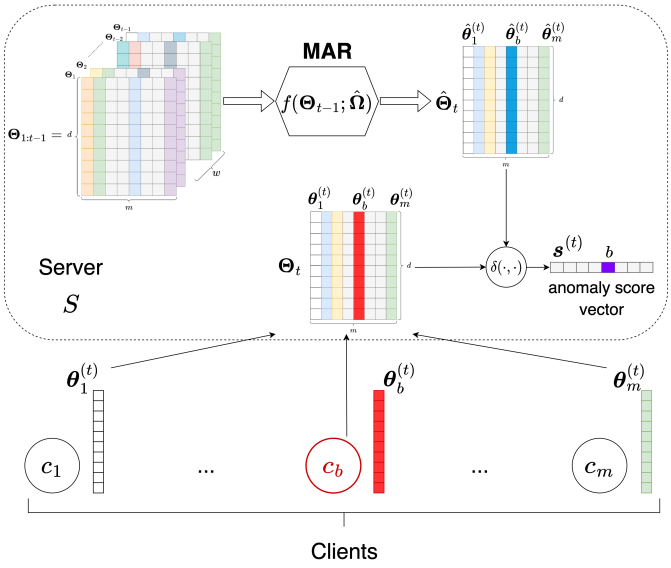

Fig. 2: Overview of FLANDERS.

In Section V-A we formally introduce MAR along with how the training works. Then, in Section V-B we explain how MAR is integrated into the server and how we calculate the anomaly score vector $s^{(t)}$. In Section V-C we illustrate a step-by-step example, in Section V-D we provide the entire pseudocode of FLANDERS, and finally Section V-E contains the computational complexity analysis.

A. Matrix Autoregressive Model (MAR)

We assume the temporal evolution of the local models sent by FL clients at each round is captured by a matrix autoregressive model (MAR). In its most generic form, MAR(w) is a w-order autoregressive model defined as follows:

$$\boldsymbol{\Theta}_t = \boldsymbol{A}_1\boldsymbol{\Theta}_{t-1}\boldsymbol{B}_1 + \cdots + \boldsymbol{A}_w\boldsymbol{\Theta}_{t-w}\boldsymbol{B}_w + \boldsymbol{E}_t, \tag{5}$$

where $\boldsymbol{\Theta}_t$ is the $d \times h$ matrix of observations at time t, $\boldsymbol{\Omega} = \{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^{w}$ are $d\times d$ and $h\times h$ autoregressive coefficient matrices, and $\boldsymbol{E}_t$ is a $d \times h$ white noise matrix. Notice that we drop the assumption made in Section IV-B about the independence of parameters; in fact, the left matrix $\boldsymbol{A}$ reflects row-wise interactions, and the right matrix $\boldsymbol{B}$ captures column-wise dependencies. This work assumes the temporal evolution of the local models sent by clients at each FL round exhibits a bilinear structure captured by a *matrix autoregressive model* of order 1, i.e., a Markovian forecasting model denoted by MAR(1) and defined as follows:

$$\boldsymbol{\Theta}_t = \boldsymbol{A}\boldsymbol{\Theta}_{t-1}\boldsymbol{B} + \boldsymbol{E}_t,$$

where $\boldsymbol{E}_t$ is a white noise matrix, i.e., its entries are iid normal with zero-mean and constant variance. To approximate such a behavior, we consider a parametric forecasting model f, in the form:

$$\widetilde{\boldsymbol{\Theta}}_t = f(\boldsymbol{\Theta}_{t-1}; \widetilde{\boldsymbol{\Omega}}) = \widetilde{\boldsymbol{A}}\boldsymbol{\Theta}_{t-1}\widetilde{\boldsymbol{B}} \approx \boldsymbol{\Theta}_t,$$

where $\widetilde{\boldsymbol{\Theta}}_t$ is the *predicted* matrix of observations at time t according to f when parametrized by coefficient matrices $\widetilde{\boldsymbol{\Omega}} = \{\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}\}$. Thus, the key question is how to estimate the best model f, namely the best coefficient matrices $\hat{\boldsymbol{\Omega}} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}\}$. For starters, we define an instance-level loss function that measures the cost of approximating the true (yet unknown) data generation process with our model f as follows:

$$\begin{aligned}\ell(\widetilde{\boldsymbol{\Omega}}; \boldsymbol{\Theta}_t) &= ||\boldsymbol{\Theta}_t - \widetilde{\boldsymbol{\Theta}}_t||_{\mathrm{F}}^2 \\ &= ||\boldsymbol{\Theta}_t - f(\boldsymbol{\Theta}_{t-1}; \widetilde{\boldsymbol{\Omega}})||_{\mathrm{F}}^2 \\ &= ||\boldsymbol{\Theta}_t - \widetilde{\boldsymbol{A}}\boldsymbol{\Theta}_{t-1}\widetilde{\boldsymbol{B}}||_{\mathrm{F}}^2 \\ &= \ell(\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}; \boldsymbol{\Theta}_t),\end{aligned} \tag{6}$$

where $||\cdot||_{\mathrm{F}}$ indicates the Frobenius norm of a matrix. More generally, if we have access to $l > 0$ historical matrix observations, we can compute the overall loss function below:

$$\mathcal{L}(\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}; \boldsymbol{\Theta}_t, l) = \sum_{j=0}^{l-1} \ell(\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}; \boldsymbol{\Theta}_{t-j}). \tag{7}$$

Notice that l here affects only the size of the training set *not* the order of the autoregressive model. In other words, the forecasting model f will still be MAR(1) and not MAR(l), i.e., the matrix of local updates at time t ($\boldsymbol{\Theta}_t$) depends *only* on the previously observed matrix at time step $t-1$ ($\boldsymbol{\Theta}_{t-1}$).

Eventually, the best estimates $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ can be found as the solutions to the following objective:

$$\begin{aligned}\hat{\boldsymbol{\Omega}} &= \hat{\boldsymbol{A}}, \hat{\boldsymbol{B}} \\ &= \arg\min_{\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}} \Big\{ \mathcal{L}(\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}; \boldsymbol{\Theta}_t, l) \Big\} \\ &= \arg\min_{\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{B}}} \Big\{ \sum_{j=0}^{l-1} ||\boldsymbol{\Theta}_{t-j} - \widetilde{\boldsymbol{A}}\boldsymbol{\Theta}_{t-j-1}\widetilde{\boldsymbol{B}}||_{\mathrm{F}}^2 \Big\}.\end{aligned} \tag{8}$$

A closed-form solution to find $\hat{\boldsymbol{A}}$ can be computed by taking the partial derivative of the loss w.r.t. $\boldsymbol{A}$, setting it to 0, and solving it for $\boldsymbol{A}$. In other words, we search for $\hat{\boldsymbol{A}} = \boldsymbol{A}$, such that:

$$\frac{\partial \mathcal{L}(\boldsymbol{A}, \boldsymbol{B}; \boldsymbol{\Theta}_t, l)}{\partial \boldsymbol{A}} = 0. \tag{9}$$

Using Eq. (7) and Eq. (6), the left-hand side of Eq. (9) can be rewritten as follows:

$$\begin{aligned}&\frac{\partial \mathcal{L}(\boldsymbol{A}, \boldsymbol{B}; \boldsymbol{\Theta}_t, l)}{\partial \boldsymbol{A}} = \frac{\partial \sum_{j=0}^{l-1} ||\boldsymbol{\Theta}_{t-j} - \boldsymbol{A}\boldsymbol{\Theta}_{t-j-1}\boldsymbol{B}||_{\mathrm{F}}^2}{\partial \boldsymbol{A}} = \\ &= -2\sum_{j=0}^{l-1}(\boldsymbol{\Theta}_{t-j} - \boldsymbol{A}\boldsymbol{\Theta}_{t-j-1}\boldsymbol{B}^T)\boldsymbol{B}\boldsymbol{\Theta}_{t-1}^T = \\ &= -2\left[\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j}\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right) - \boldsymbol{A}\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j-1}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right)\right].\end{aligned} \tag{10}$$

If we set Eq. (10) to 0 and solve it for $\boldsymbol{A}$, we find:

$$-2\left[\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j}\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right) - \boldsymbol{A}\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j-1}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right)\right] = 0.$$

By applying basic algebraic rules, the closed-form solution for $\boldsymbol{A}$ is:

$$\boldsymbol{A} = \frac{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j}\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right)}{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j-1}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{\Theta}_{t-j-1}^T\right)}. \tag{11}$$

If we apply the same reasoning, we can also find a closed-form solution to compute $\hat{\boldsymbol{B}}$. That is, we take the partial derivative of the loss w.r.t. $\boldsymbol{B}$, set it to 0, and solve it for $\boldsymbol{B}$:

$$\frac{\partial \mathcal{L}(\boldsymbol{A}, \boldsymbol{B}; \boldsymbol{\Theta}_t, l)}{\partial \boldsymbol{B}} = 0. \tag{12}$$

Eventually, we obtain the following:

$$\boldsymbol{B} = \frac{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j}^T\boldsymbol{A}\boldsymbol{\Theta}_{t-j-1}\right)}{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j-1}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{\Theta}_{t-j-1}\right)}. \tag{13}$$

We now have two closed-form solutions; one for $\boldsymbol{A}$ (see Eq. (11)) and one for $\boldsymbol{B}$ (see Eq. (13)).

However, the solution to $\boldsymbol{A}$ involves $\boldsymbol{B}$, and the solution to $\boldsymbol{B}$ involves $\boldsymbol{A}$. In other words, we must know $\boldsymbol{B}$ to compute $\boldsymbol{A}$ and vice versa.

We can use the standard Alternating Least Squares (ALS) algorithm ([34]) to solve such a problem.

The fundamental idea is to iteratively update the least squares closed-form solution of each variable alternately, keeping the other fixed. At the generic i-th iteration, we compute:

$$\boldsymbol{A}^{(i+1)} = \frac{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j}\boldsymbol{B}^{(i)}\boldsymbol{\Theta}_{t-j-1}^T\right)}{\left(\sum_{j=0}^{l-1}\boldsymbol{\Theta}_{t-j-1}(\boldsymbol{B}^{(i)})^T\boldsymbol{B}^{(i)}\boldsymbol{\Theta}_{t-j-1}^T\right)};$$

$$\boldsymbol{B}^{(i+1)} = \frac{\left(\sum_{j=0}^{l-1} \boldsymbol{\Theta}_{t-j}^{T} \boldsymbol{A}^{(i+1)} \boldsymbol{\Theta}_{t-j-1}\right)}{\left(\sum_{j=0}^{l-1} \boldsymbol{\Theta}_{t-j-1}^{T} (\boldsymbol{A}^{(i+1)})^{T} \boldsymbol{A}^{(i+1)} \boldsymbol{\Theta}_{t-j-1}\right)}.$$

ALS repeats the two steps above until some convergence criterion is met, e.g., after a specific number of iterations N or when the distance between the values of the variables computed in two consecutive iterations is smaller than a given positive threshold, i.e., $d(\boldsymbol{A}^{(i+1)} - \boldsymbol{A}^{(i)}) < \varepsilon$ and $d(\boldsymbol{B}^{(i+1)} - \boldsymbol{B}^{(i)}) < \varepsilon$, where $d(\cdot)$ is any suitable matrix distance function and $\varepsilon \in \mathbb{R}_{>0}$. Eventually, if $\boldsymbol{A}^{(\infty)}$ and $\boldsymbol{B}^{(\infty)}$ are the parameters of the MAR model upon convergence, we set $\hat{\boldsymbol{A}} = \boldsymbol{A}^{(\infty)}$ and $\hat{\boldsymbol{B}} = \boldsymbol{B}^{(\infty)}$ as the best coefficient matrices.

B. MAR-based Anomaly Score

Our approach consists of two primary steps: (i) *MAR estimation* and (ii) *anomaly score computation*.

MAR Estimation. At the first FL round ($t = 1$), the server sends the initial global model $\boldsymbol{\theta}^{(1)}$ to the set of m selected clients $\mathcal{C}^{(1)}$ and collects from them the $d \times m$ matrix of updated models $\boldsymbol{\Theta}_1$. Hence, it computes the *new* global model $\boldsymbol{\theta}^{(2)} = \phi(\{\boldsymbol{\theta}_c^{(1)} \mid c \in \mathcal{C}^{(1)}\})$, where $\phi =$ Krum or any other existing robust aggregation heuristic. For any other FL round $t > 1$, the server can use the past $l > 0$ historical observations $\boldsymbol{\Theta}_{t-l:t-1}$ to estimate the best MAR coefficients $\hat{\boldsymbol{\Omega}} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}\}$ according to (8). In general, $1 \leq l \leq t-1$; however, if we assume l fixed at each round, the server will consider $\boldsymbol{\Theta}_{\max(1,t-l):t-1}$ past observations. Again, independently of the value of l, MAR will learn to predict the current matrix of weights *only* from the previously observed matrix. Therefore, each matrix used for training $\{\boldsymbol{\Theta}_{t-j-1}\}_{j=0}^{l-1}$ has the same size $d \times m$, as it contains exclusively the m updates received at the round $t-j-1$. Of course, employing a higher order MAR(w) model with $w > 1$ would require extending each observed matrix to $d \times h$ ($m < h \leq K$), as detailed in Section IV-A. This is to track the local updates sent by selected clients across multiple historical rounds.

Anomaly Score Computation. At the generic FL round $t > 1$, we compute the anomaly score using the estimated MAR forecasting model as follows. Let $\boldsymbol{\Theta}_t$ be the matrix of observed weights. This matrix may contain one or more corrupted local models from malicious clients. Then, we compute the m-dimensional anomaly score vector $\boldsymbol{s}^{(t)}$, where $\boldsymbol{s}^{(t)}[c] = s_c^{(t)}$, as in (4). A critical choice concerns the function δ used to measure the distance between the observed vector of weights sent by each selected client and the vector of weights predicted by MAR. In this work, we set $\delta(\boldsymbol{u}, \boldsymbol{v}) = ||\boldsymbol{u} - \boldsymbol{v}||_2^2$, where $||\cdot||_2^2$ is the squared L^2-norm. In this work, we set $\delta = ||\cdot||^2$ as L^2-norm; other functions can also be used (e.g., *cosine distance*). Choosing the best δ is outside the scope of this work, and we leave it to future study.

According to one of the filtering strategies discussed in Section IV-C, we retain only the k clients with the smallest anomaly scores. The remaining $m-k$ clients are considered malicious; thus, they are discarded and do not contribute to the aggregation run by the server as if they were never selected.

At the next round $t+1$, we may want to refresh our estimation of the MAR model, i.e., to update the coefficient matrices $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$. We do so by considering the latest observed $\boldsymbol{\Theta}_t$ and the other $l-1$ previous matrices of local updates, using the same sliding window of size l. Since the observed matrix $\boldsymbol{\Theta}_t$ contains $m-k$ potentially malicious clients, we cannot use it as-is. Otherwise, *if any of the spotted malicious clients is selected again at round* $t+1$, we may alter the estimation of $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ with possibly corrupted matrix columns. To overcome this problem, we replace the original $\boldsymbol{\Theta}_t$ with $\boldsymbol{\Theta}'_t$ *before*, feeding it to train the new MAR model. Specifically, $\boldsymbol{\Theta}'_t$ is obtained from $\boldsymbol{\Theta}_t$ by substituting the $m-k$ anomalous columns either with the parameter vectors from the same clients observed at time $t-1$, which are supposed to be still legitimate *or* the current global model. The advantage of this solution is twofold. On the one hand, a client labeled as malicious at FL round t would likely still be considered so at $t+1$ if it keeps perturbing its local weights, thus improving robustness. On the other hand, our solution allows malicious clients to alternate legitimate behaviors without being banned, speeding up model convergence. Notice that the two considerations above might not be valid if we updated the MAR model using the original, partially corrupted $\boldsymbol{\Theta}_t$. Indeed, in the first case, the distance between two successive poisoned models by the same client would reasonably be small. So, the client's anomaly score will likely drop to non-alarming values, thereby increasing the number of false negatives. In the second case, the distance between a corrupted and a legitimate model would likely be large. Thus, a malicious client will maintain its anomaly score high even if it acts honestly, impacting the number of false positives.

C. A Step-by-Step Example

To better clarify how FLANDERS works, consider the following practical example.

Suppose an FL system consists of a centralized server and 10 clients $c_1, \ldots, c_{10}$; furthermore, at each round, 4 of those clients are randomly chosen for training. At the very first round ($t = 1$), let $\mathcal{C}^{(1)} = \{c_2, c_3, c_7, c_9\}$ be the set of 4 clients selected by the server. Hence, the server sends the current global model $\boldsymbol{\theta}^{(1)} \in \mathbb{R}^d$ to each of those clients and collects the $d \times 4$ matrix of updated local models $\boldsymbol{\Theta}_1 = [\boldsymbol{\theta}_2^{(1)}, \boldsymbol{\theta}_3^{(1)}, \boldsymbol{\theta}_7^{(1)}, \boldsymbol{\theta}_9^{(1)}]$. Therefore, it computes the *new* global model $\boldsymbol{\theta}^{(2)} = \phi(\{\boldsymbol{\theta}_c^{(1)} \mid c \in \mathcal{C}^{(1)}\})$. Notice that, at this stage, no anomaly score can be computed as FLANDERS cannot take advantage of any historical observations of local model updates. As such, if one (or more) selected clients in the very first round are malicious, plain FedAvg may not detect those. To overcome this problem, FLANDERS should be paired with one of the existing robust aggregation heuristics at $t = 1$. For example, $\phi = \{$Trimmed Mean, Krum, Bulyan$\}$.

At the next round ($t = 2$), FLANDERS can start using past observations (i.e., only $\boldsymbol{\Theta}_1$) to estimate the best MAR coefficients $\hat{\boldsymbol{\Omega}} = \{\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}\}$ according to Eq. (8) above. Suppose $\mathcal{C}^{(2)} = \{c_1, c_3, c_6, c_9\}$ is the set of 4 clients selected by the server at the second round. Let $\boldsymbol{\Theta}_2 = [\boldsymbol{\theta}_1^{(2)}, \boldsymbol{\theta}_3^{(2)}, \boldsymbol{\theta}_6^{(2)}, \boldsymbol{\theta}_9^{(2)}]$ be the local models sent by the clients to the server. Moreover,

$\hat{\boldsymbol{\Theta}}_2 = f(\boldsymbol{\Theta}_1; \hat{\boldsymbol{\Omega}}) = [\hat{\boldsymbol{\theta}}_2^{(2)}, \hat{\boldsymbol{\theta}}_3^{(2)}, \hat{\boldsymbol{\theta}}_7^{(2)}, \hat{\boldsymbol{\theta}}_9^{(2)}]$ are the local models predicted by MAR, using the previous set of observations.

It is worth noting that $\mathcal{C}^{(1)} \cap \mathcal{C}^{(2)} = \{c_3, c_9\}$; the other two clients, c_1 and c_6, are considered "cold-start" since they are selected for the first time or, in any case, they do not appear in the historical observations used by MAR for predictions (i.e., in the previous $w = 1$ matrices). Thus, we calculate the following anomaly scores, according to Eq. 5:

- $s_3^{(2)} = \delta(\boldsymbol{\theta}_3^{(2)}, \hat{\boldsymbol{\theta}}_3^{(2)})$ and $s_9^{(2)} = \delta(\boldsymbol{\theta}_9^{(2)}, \hat{\boldsymbol{\theta}}_9^{(2)})$, i.e., we measure the distance between the models actually sent by c_3 and c_9 and those predicted by MAR using the previous observations (*first condition*);
- $s_1^{(2)} = \delta(\boldsymbol{\theta}^{(2)}, \boldsymbol{\theta}_1^{(2)})$ and $s_6^{(2)} = \delta(\boldsymbol{\theta}^{(2)}, \boldsymbol{\theta}_6^{(2)})$, i.e., we measure the distance between the models actually sent by c_1 and c_6 and the current global model at time $t = 2$, as those clients were never picked before (*second condition*);
- $s_2^{(2)} = s_4^{(2)} = s_5^{(2)} = s_7^{(2)} = s_8^{(2)} = s_{10}^{(2)} = \perp$, i.e., these clients were not selected at round $t = 2$, and therefore they will not contribute to computing the new global model $\boldsymbol{\theta}^{(3)}$ anyway (*third condition*).

Let us assume that c_3 is a malicious client controlled by an attacker. Moreover, suppose that this client started sending poisoned models since the very first round, i.e., $\boldsymbol{\theta}_3^{(1)}$ was already corrupted. In such a case, it is evident that if we had used plain FedAvg at $t = 1$, this would have likely polluted the global model $\boldsymbol{\theta}^{(2)}$ and, therefore, FLANDERS might fail to recognize this as a malicious client due to a relatively low anomaly score $s_3^{(2)}$. As stated before, the consequences of this edge situation in which one or more malicious clients are picked at the beginning of the FL training can be mitigated by replacing FedAvg with one of the robust aggregation strategies available in the literature, such as Trimmed Mean, Krum, or Bulyan.

However, suppose c_3 is correctly spotted as malicious at the end of round $t = 2$ due to its high anomaly score $s_3^{(2)}$. Therefore, c_3 (actually, $\boldsymbol{\theta}_3^{(2)}$) will be discarded from the aggregation at the server's end. Hence, FedAvg can now safely be used; more generally, the server can restart running FedAvg from $t = 2$ on, i.e., once FLANDERS can compute *valid* anomaly scores. The server can now compute the updated global model as $\boldsymbol{\theta}^{(3)} = \phi(\{\boldsymbol{\theta}_c^{(2)} \mid c \in \mathcal{C}_*^{(2)}\})$, where $\mathcal{C}_*^{(2)} \subset \mathcal{C}^{(2)}$ contains the clients with the k smallest anomaly scores. For instance, if $k = 2$ and $\mathcal{C}_*^{(2)} = \{c_1, c_9\}$, $\boldsymbol{\theta}^{(3)} = 1/2 * (\boldsymbol{\theta}_1^{(2)} + \boldsymbol{\theta}_9^{(2)})$.

At the next round ($t = 3$), FLANDERS can use the previous two observations $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$ to refine the estimation of the best MAR coefficients, again solving Eq. (8).[7] However, $\boldsymbol{\Theta}_2$ cannot be fed as-is to re-train MAR since one of its components – i.e., the local model $\boldsymbol{\theta}_3^{(2)}$ sent by client c_3 – has been flagged as malicious. Otherwise, the resulting updated MAR coefficients could be unreliable due to the propagation of local poisoned models.

To overcome this problem, FLANDERS replaces the local model marked as suspicious $\boldsymbol{\theta}_3^{(2)}$ with either one of the previously observed models from the same client that is supposedly legitimate *or* the current global model. Since, in this example, we assume c_3 has been malicious from the first round, we use the latter approach. Specifically, we change $\boldsymbol{\Theta}_2$ with $\boldsymbol{\Theta}_2' = [\boldsymbol{\theta}_1^{(2)}, \underline{\boldsymbol{\theta}^{(2)}}, \boldsymbol{\theta}_6^{(2)}, \boldsymbol{\theta}_9^{(2)}]$, where $\boldsymbol{\theta}^{(2)}$ substitutes $\boldsymbol{\theta}_3^{(2)}$. As discussed at the end of Section V-B, this fix allows FLANDERS to work even when a malicious client is picked for two or more rounds consecutively.

We can now compute $\hat{\boldsymbol{\Theta}}_3 = f(\boldsymbol{\Theta}_2'; \hat{\boldsymbol{\Omega}}) = [\hat{\boldsymbol{\theta}}_1^{(3)}, \hat{\boldsymbol{\theta}}_3^{(3)}, \hat{\boldsymbol{\theta}}_6^{(3)}, \hat{\boldsymbol{\theta}}_9^{(3)}]$ using the updated MAR, leveraging the previous set of amended observations. Let $\mathcal{C}^{(3)} = \{c_2, c_3, c_4, c_5\}$ denote the set of clients selected at round 3 and $\boldsymbol{\Theta}_3 = [\boldsymbol{\theta}_2^{(3)}, \boldsymbol{\theta}_3^{(3)}, \boldsymbol{\theta}_4^{(3)}, \boldsymbol{\theta}_5^{(3)}]$ be the local models sent by the clients to the server. Thus, $\mathcal{C}^{(2)} \cap \mathcal{C}^{(3)} = \{c_3\}$. The remaining three clients – c_2, c_4, and c_5 – are treated as "cold-start." Specifically, c_4 and c_5 are selected for the first time, while c_2, even though previously selected in the first round, was not chosen at the previous round (2). As c_2 was not part of the historical observations used by MAR(1) to make predictions, we need to treat it as if it were a cold-start client.

Therefore, anomaly scores are updated as follows:

- $s_3^{(3)} = \delta(\boldsymbol{\theta}_3^{(3)}, \hat{\boldsymbol{\theta}}_3^{(3)})$, i.e., we measure the distance between the models actually sent by c_3 and those predicted by MAR using the previous observations (*first condition*);
- $s_2^{(3)} = \delta(\boldsymbol{\theta}^{(3)}, \boldsymbol{\theta}_2^{(3)})$, $s_4^{(3)} = \delta(\boldsymbol{\theta}^{(3)}, \boldsymbol{\theta}_4^{(3)})$, and $s_5^{(3)} = \delta(\boldsymbol{\theta}^{(3)}, \boldsymbol{\theta}_5^{(3)})$ i.e., we measure the distance between the models actually sent by c_2, c_4, and c_5 and the current global model at time $t = 3$, as those clients were never picked before or did not appear in the previous observations used to make predictions (*second condition*);
- $s_1^{(3)} = s_6^{(3)} = s_7^{(3)} = s_8^{(3)} = s_9^{(3)} = s_{10}^{(3)} = \perp$, i.e., these clients were not selected at round $t = 3$, and therefore they will not contribute to computing the new global model $\boldsymbol{\theta}^{(4)}$ anyway (*third condition*).

Generally, the process above continues until the global model converges.

D. FLANDERS' Pseudocode

We present the pseudocode of a hypothetical FL server that integrates FLANDERS into the model aggregation stage. The main server-side loop is shown in Algorithm 1, whereas Algorithm 2 details the computation of anomaly scores, which is at the heart of the FLANDERS filter.

E. Computational Complexity Analysis

To train our MAR forecasting model, we first need to run the ALS algorithm, which estimates the coefficient matrices $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$. ALS iteratively performs two steps to compute $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$, respectively. At the generic i-th iteration, computing the i-th $d \times d$ matrix $\hat{\boldsymbol{A}}$ costs $O(d^3) + O(dm^2)$; similarly, computing the i-th $m \times m$ matrix[8] $\hat{\boldsymbol{B}}$ costs $O(m^3) + O(d^2 m)$. Thus, a single iteration costs $O(d^3) + O(dm^2) + O(m^3) + O(d^2 m)$, namely $O(d^3)$ *or* $O(m^3)$,[9] depending what term dominates

[7] In general, FLANDERS uses l past observations $\boldsymbol{\Theta}_{\max(1,t-l):t-1}$.

[8] It is worth remarking that, using MAR(1), $h = m$ and thus $\hat{\boldsymbol{B}}$ has size $m \times m$.

[9] $O(d^{2.376})$ *or* $O(m^{2.376})$ using the Coppersmith-Winograd algorithm [35].

Algorithm 1 FLANDERS-SERVER

Input: The aggregation function (ϕ); the number of randomly selected clients at each FL round (m); the number of local models to keep as legitimate (k); the autoregressive order of MAR (w); the number of historical observations used to train MAR (l); the number of MAR training iterations (N); the number of total FL rounds (T).
Output: The global model $\boldsymbol{\theta}^{(T)}$.

procedure FLANDERS-SERVER(ϕ, m, k, w, l, N, T)
  $\boldsymbol{\theta}^{(1)} \leftarrow$ A randomly initialized model
  for all $t \in \{1, 2, \dots T\}$ **do**
    $\mathcal{C}^{(t)} \leftarrow$ sample a subset of m clients from $\mathcal{C}$
    Send the global model $\boldsymbol{\theta}^{(t)}$ to every $c \in \mathcal{C}^{(t)}$
    $\boldsymbol{\Theta}_t \leftarrow [\boldsymbol{\theta}_1^{(t)}, \dots, \boldsymbol{\theta}_m^{(t)}]$ ▷ *Receive the m local models trained by each $c \in \mathcal{C}^{(t)}$*
    if $t == 1$ **then**
      ▷ *At the very first round, use the designated fallback aggregation strategy (e.g., FedAvg)* ◁
      $\mathcal{C}_*^{(t)} \leftarrow$ FALLBACK($\boldsymbol{\Theta}_t$)
    else
      $\boldsymbol{\Theta}_t \leftarrow$ COLDSTART ($C^{(t)}, \boldsymbol{\Theta}_t$)
      ▷ *Returns the set of clients classified as legitimate* ◁
      $\mathcal{C}_*^{(t)} \leftarrow$ FLANDERS($\boldsymbol{\Theta}_{t-l:t-1}, \mathcal{C}^{(t)}, k, w, N$)
    $\boldsymbol{\theta}^{(t+1)} \leftarrow \phi(\{\boldsymbol{\theta}_c^{(t)} \mid c \in \mathcal{C}_*^{(t)}\})$

Algorithm 2 FLANDERS-FILTER

Input: The tensor containing the local models of selected clients observed from l past FL rounds ($\boldsymbol{\Theta}_{t-l:t-1}$); the clients selected at round t ($\mathcal{C}^{(t)}$); the autoregressive order of MAR (w); the number of local models to keep as legitimate (k); the number of MAR training iterations (N).
Output: The set of k clients classified as legitimate in round t $\mathcal{C}_*^{(t)}$.

procedure FLANDERS($\boldsymbol{\Theta}_{t-l:t-1}, \mathcal{C}^{(t)}, k, w, N$)
  ▷ *MAR(w) estimation via ALS training (N iterations)* ◁
  $\hat{\boldsymbol{\Theta}}_t \leftarrow$ MAR($\boldsymbol{\Theta}_{t-l:t-1}, w, N$)
  $\boldsymbol{s}^{(t)} \leftarrow \delta(\boldsymbol{\Theta}_t, \hat{\boldsymbol{\Theta}}_t)$ ▷ *Anomaly score vector*
  $\mathcal{C}_*^{(t)} \leftarrow \{c \in \mathcal{C}^{(t)} \mid s_c^{(t)} \leq s_k^{(t)}\}$ ▷ *$s_k^{(t)}$ is the k-th smallest anomaly score*
  return $\mathcal{C}_*^{(t)}$

between d (the number of weights) and m (the number of clients). ALS performs the two steps above for N iterations (e.g., in this work, we set $N = 100$). Second, the computation of the anomaly score costs $O(md)$ as it measures the distance between observed and predicted local updates from all the m selected clients. Third, every time the MAR model is refreshed, we need to rerun ALS; in the worst-case scenario, we might want to re-estimate $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ at every single FL round.

It is worth noticing that performing ALS directly on high dimensional parameter space (d) in standard FL settings with loads of clients (m) may be unfeasible (e.g., when d and m range from 10^6 to 10^9). To keep the computational cost

Algorithm 3 COLDSTART

Input: The list of clients $C^{(t)}$; the list of local models $\boldsymbol{\Theta}_t$.
Output: The list of local models $\boldsymbol{\Theta}_t$ modified according to the cold-start strategy.

procedure COLDSTART($C^{(t)}$, $\boldsymbol{\Theta}_t$)
  for all $c \in C^{(t)}$ **do**
    if $\boldsymbol{\theta}_c^{t-1} \neq \perp$ **then**
      $\boldsymbol{\theta}_c^t \leftarrow \boldsymbol{\theta}_c^{t-1}$
    else if $\boldsymbol{\theta}_c^{t-j} \neq \perp$ where $j \in \{1..w\}$ **then**
      $\boldsymbol{\theta}_c^t \leftarrow \boldsymbol{\theta}_c^{t-j}$ ▷ *It can be either the global model or the last update of client c*
    else if $\boldsymbol{\theta}_c^t = \perp$ **then** ▷ *Cold start for c*
      $\boldsymbol{\theta}_c \leftarrow \boldsymbol{\theta}^{t-1}$ ▷ *Assign the current global model*
  return $\boldsymbol{\Theta}_t$

tractable (and limit the impact of the curse of dimensionality), in our experiments, where $d \gg m$, we reduce dimensionality via random sampling on the parameter space, as proposed by [9]. Specifically, we sample a fixed subset of $\tilde{d} < d$ model parameters for ALS, with $\tilde{d}$ set to 500. We adopt a task-agnostic approach, keeping this number constant across all experiments to ensure a fair comparison and demonstrate the general applicability of our method without task-specific tuning. The choice of 500 is not a critical hyperparameter, as our results are robust across a range of values; our ablation study in Section VI-D (see Table XIV) shows that FLANDERS' detection performance remains perfect even when reducing the number of sampled parameters to as low as 100. While we use random sampling for simplicity, neural networks often exhibit correlated parameters, more sophisticated solutions may remove all parameters that show obvious dependency from other ones.

VI. EXPERIMENTS

A. Experimental Setup

Datasets, Tasks, and FL Models. We consider four public datasets for image classification: *MNIST*, *Fashion-MNIST*, *CIFAR-10*, and *CIFAR-100*. All datasets are randomly shuffled and partitioned into two disjoint sets: 80% is used for training and 20% for testing. We train a Multilayer Perceptron (MLP) on *MNIST* and *Fashion-MNIST* and a Convolutional Neural Network (CNN) on *CIFAR-10* and *CIFAR-100* (MobileNet [36]). All models are trained by minimizing cross-entropy loss.
In Table II, we report the full details of our experimental setup concerning the datasets used, their associated tasks, and the models (along with their hyperparameters) trained on the simulated FL environment, i.e., Flower[10]. Table III shows the description of the hyperparameters.

The MLP for the *MNIST* dataset is a 2-layer fully connected feed-forward neural network, whereas the MLP for the *Fashion-MNIST* dataset is a 4-layer fully connected feed-forward neural network. Both MLPs are trained by minimizing multiclass cross-entropy loss using Adam optimizer with batch size equal to 32 [40]. The CNN used for *CIFAR-10* is a 6-layer convolutional

[10]https://flower.dev/

TABLE II: Experimental setup: datasets, tasks, and FL models considered.

Dataset	N. of Instances (training/test)	N. of Features	Task	FL Model	Hyperparameters
MNIST [37]	$60,000/10,000$	28x28 (numerical)	multiclass classification	MLP	{batch=32; layers=2; opt=Adam; η=10^{-3}}
Fashion-MNIST [38]	$60,000/10,000$	28x28 (numerical)	multiclass classification	MLP	{batch=32; layers=4; opt=Adam; η=10^{-3}}
CIFAR-10 [39]	$50,000/10,000$	32x32x3 (numerical)	multiclass classification	CNN	{batch=32; layers=6; opt=SGD; η=10^{-2}; μ=0.9}
CIFAR-100 [39]	$50,000/10,000$	32x32x3 (numerical)	multiclass classification	CNN (MobileNet)	{batch=32; layers=28; opt=SGD; η=10^{-2}; μ=0.9}

TABLE III: Description of hyperparameters.

Hyperparameter	Description
η	learning rate
μ	momentum
opt	optimizer: stochastic gradient descent (SGD); Adam
batch	batch size
layers	number of neural network layers

TABLE IV: Main properties of our FL environment simulated on Flower.

Total N. of Clients	$K = 100$
N. of Selected Clients (at each round)	$m = K = 100$
Ratio of Malicious Clients	$r = \{0, 0.2, 0.6, 0.8\}$
Total N. of FL Rounds	$T = 50$
Autoregressive Order of MAR	$w = 1$
Historical Window Size of Past FL Rounds	$l = 2$
Non-IID Dataset Distribution across Clients	$\alpha_D = 0.5$

neural network, while the CNN used for *CIFAR-100* is the well-known MobileNet architecture [36]. Both CNNs are trained by minimizing multiclass cross-entropy loss via stochastic gradient descent (SGD) with batch size equal to 32 [41].

At each FL round, every client performs one training epoch of Adam/SGD, which corresponds to the number of iterations needed to "see" all the training instances once, when divided into batches of size 32. In any case, the updated local model is sent to the central server for aggregation.

We run our experiments on a machine equipped with an AMD Ryzen 9, 64 GB RAM, and an NVIDIA 4090 GPU with 24 GB VRAM.

FL Simulation Environment. To simulate a realistic FL environment, we integrate FLANDERS into Flower [42]. Moreover, we implement in Flower any defense baseline considered that the framework does not natively provide. In Table IV we synthetically report the hyperparameters.

We set the number K of FL clients to 100, and we assume that the server selects *all* these clients in every FL round ($m = K$). We suppose the attack starts at $t = 1$; then, we monitor the performance of the global model until $t = T = 50$. Recall that we select the malicious clients randomly across all the participating ones; this implies that a single client can alternate between legitimate and malicious behavior over successive FL rounds.

Non-IID Local Training Data. We simulate non-iid clients' training data distributions following [43]. We assume every client training example is drawn independently with class labels following a categorical distribution over N classes parameterized by a vector $\boldsymbol{q}$ such that $q_i \geq 0, i \in [1, \ldots, N]$ and $||\boldsymbol{q}||_1 = 1$. To synthesize a population of non-identical clients, we draw $\boldsymbol{q} \sim \text{Dir}(\alpha_D, \boldsymbol{p})$ from a Dirichlet distribution, where $\boldsymbol{p}$ characterizes a prior class distribution over N classes, and $\alpha_D > 0$ is a *concentration* parameter controlling the identicalness among clients. With $\alpha_D \to \infty$, all clients have identical label distributions; on the other extreme, with $\alpha_D \to 0$, each client holds examples from only one class chosen at random. In our experiments, we set $\alpha_D = 0.5$.

B. Attacks

We assess the robustness of FLANDERS under the attacks introduced in III. For each attack, we vary the number $b = \lceil r * m \rceil, r \in [0, 1]$ of malicious clients, where $r = \{0, 0.2, 0.6, 0.8\}$. Ultimately, we analyze an attacker's performance embedded with a MAR model.

Below, we describe the critical parameters for each attack considered, which are also summarized in Table V.

GAUSS. This attack has only one parameter: the magnitude σ of the perturbation to apply. We set $\sigma = 10$ for all the experiments.

LIE. This method has no parameters to set.

OPT. The parameter τ represents the minimum value that λ can assume. Below this threshold, the halving search stops. As suggested by the authors, we set $\tau = 10^{-5}$.

AGR-MM. In addition to the threshold τ, AGR-MM uses the perturbation vectors ∇^p in combination with the scaling coefficient γ to optimize. We set $\tau = 10^{-5}$ and $\nabla^p = \{std\}$, which is the vector obtained by computing the parameters' inverse of the standard deviation. For Krum, we set $\nabla^p = \{uv\}$, which is the inverse unit vector perturbation.

TABLE V: Key parameter settings for each attack strategy considered.

Attack	Parameters
GAUSS	$\sigma = 10$
LIE	N/A
OPT	$\tau = 10^{-5}$
AGR-MM	$\tau = 10^{-5}; \nabla^p = \{uv, std\}; \gamma = 5$

C. Evaluation

We evaluate four key aspects of FLANDERS. Firstly, we test its ability to detect malicious clients against the best-competing filtering strategy, FLDetector. Secondly, we measure the accuracy improvement of the global model when FLANDERS is paired with "vanilla" FedAvg and the most popular robust aggregation baselines: FedMedian, Trimmed Mean, Multi-Krum, Bulyan, DnC. Thirdly, we analyze the cost-benefit trade-off of FLANDERS. Lastly, we test the robustness of our method against adaptive attacks.

Malicious Detection Accuracy. Table VI shows the precision (P) and recall (R) of FLDetector and FLANDERS in filtering out malicious clients across different datasets and attacks, with $r = 0.2$ (20% evil participants in the FL systems). Remarkably, FLANDERS successfully detects *all and only* malicious clients across every attack setting except for OPT, outperforming its main competitor, FLDetector. This is further confirmed by Table VII, which shows that our method generally provides much higher protection than FLDetector when combined with standard FedAvg.

Better results are observed for extreme attack settings ($r = 0.8$) in Tables VIII, and IX, where OPT maximizes the distance between the legitimate models and the malicious ones by erroneously thinking that the high number of its controlled clients pierces through the aggregation functions.

FLDetector fails to distinguish malicious updates from legitimate ones because *(i)* the anomaly score is based on approximating a single Hessian matrix that must be close to all legitimate clients, making it inadequate for highly non-iid settings, where malicious updates can be similar to legitimate ones; and *(ii)* the suspicious scores are computed as the average normalized Euclidean distance of the past iterations. Although this is coherent and works well with a threat model where the malicious clients are always the same across all rounds, FLANDERS does not make such an assumption, making FLDetector unable to recognize malicious updates arriving for the first time from an ex-legitimate client.

Aggregation Robustness Lift. We proceed to evaluate the enhancement that FLANDERS provides to the robustness of the global model. Specifically, we measure the best accuracy of the global model under several attack strengths using all the baselines *without* and *with* FLANDERS as a pre-filtering strategy. Table X shows that FLANDERS keeps high accuracy for the best global model under extreme attacks ($r = 0.8$) when used before *every* aggregation method, including Multi-Krum and Bulyan, which would otherwise be inapplicable in such strong attack scenarios.

In Figure 3 we compare the accuracy of the global model when using FLANDERS (left) and when using FedAvg (right) across the whole FL training process. The left figure shows how the evolution of the accuracy over multiple rounds remains stable and similar to the one without any attack (dashed line), while on the right the accuracy drops irremediably.

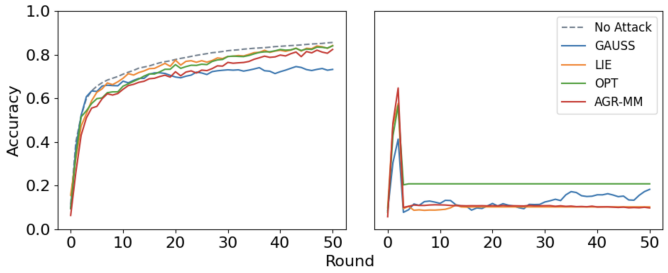

Fig. 3: FedAvg with FLANDERS (left) vs. "vanilla" FedAvg (right). Accuracy of the global model in each FL round under all attack strategies on the *MNIST* dataset, with 80% of malicious clients. Attack starts at round $t = 3$.

Cost-Benefit Analysis. Obviously, the robustness guaranteed by FLANDERS under extreme attack scenarios comes with costs, especially due to the MAR estimation stage. Fig. 4 depicts two scatter plots for the *MNIST* and *CIFAR-10* datasets, focusing on a specific attack scenario (AGR-MM). Each data point on a scatter plot represents a method under one of two attack strengths considered ($r = 0.2$ and $r = 0.6$). These data points are specified by two coordinates: the overall training time on the x-axis and the maximum accuracy of the global model obtained after $T = 50$ FL rounds on the y-axis.

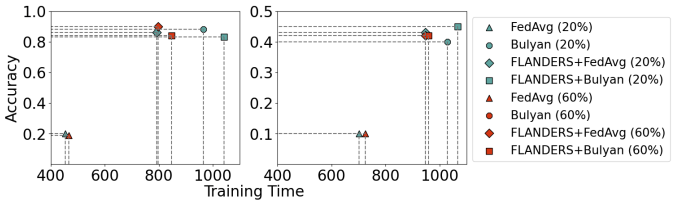

Fig. 4: Accuracy vs. *total* Training Time (in seconds) of FedAvg and Bulyan compared with their corresponding versions with FLANDERS as a filter for the *MNIST* (left) and *CIFAR-10* (right) datasets.

Overall, the take-home message is as follows. In scenarios with low attack strength ($r = 0.2$), Bulyan demonstrates superior accuracy, while FLANDERS + FedAvg offers comparable performance with notably shorter training times. However, as the attack strength increases ($r = 0.6$), Bulyan becomes impractical, FedAvg alone proves ineffective, and FLANDERS emerges as the optimal choice for achieving the best accuracy vs. cost trade-off.

Robustness against Adaptive Attacks. In this section, we further validate the robustness of FLANDERS against adaptive attacks. We consider a scenario where malicious clients are aware that the FL server uses our method as a pre-aggregation filter. Specifically, we focus on two levels of knowledge. The first scenario assumes that malicious clients tentatively

TABLE VI: *Precision* (P) and *Recall* (R) of FLDetector and FLANDERS in detecting malicious clients across various datasets and attacks ($r = 0.2$; $T = 50$ rounds).

	FLDetector								**FLANDERS**							
	GAUSS		LIE		OPT		AGR-MM		GAUSS		LIE		OPT		AGR-MM	
	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
MNIST	0.20	0.20	0.22	0.22	**0.20**	**0.20**	0.19	0.19	**1.0**	**1.0**	**1.0**	**1.0**	0.13	0.13	**1.0**	**1.0**
Fashion-MNIST	0.21	0.21	0.18	0.18	**0.21**	**0.21**	0.20	0.20	**1.0**	**1.0**	**1.0**	**1.0**	0.13	0.13	**1.0**	**1.0**
CIFAR-10	0.21	0.21	0.21	0.21	0.22	0.22	0.19	0.19	**1.0**	**1.0**	**1.0**	**1.0**	**0.58**	**0.58**	**1.0**	**1.0**
CIFAR-100	0.20	0.20	0.19	0.19	0.20	0.20	0.21	0.21	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

TABLE VII: Accuracy of the global model using FedAvg with FLDetector and FLANDERS ($r = 0.2$).

	FedAvg				**FLDetector + FedAvg**				**FLANDERS + FedAvg**			
	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM
MNIST	0.18	0.12	0.63	0.34	0.20	0.11	**0.68**	0.43	**0.86**	**0.83**	0.62	**0.85**
Fashion-MNIST	0.25	0.10	0.56	0.16	0.28	0.10	**0.60**	0.17	**0.69**	**0.64**	0.58	**0.63**
CIFAR-10	0.10	0.10	0.23	0.10	0.10	0.10	0.26	0.12	**0.38**	**0.37**	**0.28**	**0.36**
CIFAR-100	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.02	**0.07**	**0.05**	**0.05**	**0.06**

TABLE VIII: *Precision* (P) and *Recall* (R) of FLDetector and FLANDERS in detecting malicious clients across various datasets and attacks ($r = 0.8$; $T = 50$ rounds).

	FLDetector								**FLANDERS**							
	GAUSS		LIE		OPT		AGR-MM		GAUSS		LIE		OPT		AGR-MM	
	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
MNIST	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Fashion-MNIST	0.80	0.80	0.80	0.80	0.79	0.79	0.80	0.80	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
CIFAR-10	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
CIFAR-100	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

TABLE IX: Accuracy of the global model using FedAvg with FLDetector and FLANDERS ($r = 0.8$).

	FedAvg				**FLDetector + FedAvg**				**FLANDERS + FedAvg**			
	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM
MNIST	0.18	0.11	0.21	0.11	0.15	0.13	0.23	0.12	**0.75**	**0.84**	**0.84**	**0.82**
Fashion-MNIST	0.24	0.10	0.19	0.10	0.19	0.10	0.03	0.10	**0.68**	**0.70**	**0.66**	**0.66**
CIFAR-10	0.10	0.10	0.11	0.10	0.10	0.10	0.10	0.10	**0.33**	**0.32**	**0.32**	**0.32**
CIFAR-100	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	**0.09**	**0.11**	**0.11**	**0.10**

TABLE X: Accuracy of the global model using all the baseline aggregations without and with FLANDERS ($r = 0.8$).

	MNIST				*Fashion-MNIST*				*CIFAR-10*				*CIFAR-100*			
	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM	GAUSS	LIE	OPT	AGR-MM
FedAvg	0.18	0.11	0.21	0.11	0.24	0.10	0.19	0.10	0.10	0.10	0.11	0.10	0.01	0.01	0.01	0.01
+ FLANDERS	**0.75**	**0.84**	**0.84**	**0.82**	**0.68**	**0.70**	**0.66**	**0.66**	**0.33**	**0.32**	**0.32**	**0.32**	**0.09**	**0.11**	**0.11**	**0.10**
FedMedian	0.34	0.19	0.13	0.23	0.29	0.10	0.17	0.10	0.13	0.10	0.10	0.12	0.01	0.01	0.01	0.01
+ FLANDERS	**0.81**	**0.84**	**0.85**	**0.81**	**0.70**	**0.71**	**0.71**	**0.68**	**0.29**	**0.29**	**0.31**	**0.28**	**0.10**	**0.11**	**0.11**	**0.10**
TrimmedMean	0.15	0.13	0.20	0.18	0.21	0.10	0.23	0.10	0.10	0.10	0.10	0.10	0.01	0.01	0.01	0.01
+ FLANDERS	**0.81**	**0.83**	**0.83**	**0.85**	**0.71**	**0.70**	**0.71**	**0.69**	**0.30**	**0.29**	**0.30**	**0.29**	**0.11**	**0.10**	**0.11**	**0.11**
Multi-Krum	0.80	0.12	0.17	0.25	0.64	0.10	0.20	0.10	0.34	0.10	0.10	0.10	0.08	0.01	0.01	0.01
+ FLANDERS	**0.87**	**0.90**	**0.88**	**0.89**	**0.69**	**0.68**	**0.72**	**0.68**	**0.38**	**0.38**	**0.39**	**0.40**	**0.11**	**0.10**	**0.10**	**0.11**
Bulyan	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
+ FLANDERS	**0.89**	**0.85**	**0.88**	**0.82**	**0.68**	**0.66**	**0.68**	**0.70**	**0.40**	**0.43**	**0.40**	**0.41**	**0.11**	**0.11**	**0.11**	**0.11**
DnC	0.21	0.11	0.17	0.11	0.25	0.10	0.14	0.10	0.10	0.10	0.10	0.10	0.01	0.01	0.01	0.01
+ FLANDERS	**0.85**	**0.87**	**0.89**	**0.87**	**0.71**	**0.69**	**0.68**	**0.68**	**0.41**	**0.40**	**0.39**	**0.40**	**0.10**	**0.11**	**0.11**	**0.12**

guess the subset of parameters ($\widetilde{d}$) used by the FL server to estimate the MAR forecasting model. We refer to this setting as *non-omniscient*. The second, more challenging as well as unrealistic scenario assumes that malicious clients know *exactly* which parameters are used by the FL server. We call this second scenario *omniscient*. Obviously, the latter penalizes FLANDERS way more than the former.

Specifically, we perform our experiments over $T = 20$ rounds with $m = 20$ clients, of which $r = 0.2$ ($b = 5$) are malicious. The attacker constructs a matrix $M = b \times \widetilde{d}$ using the local models generated by the corrupted clients. This matrix M is then passed as input to the same forecasting model, MAR, that the server uses to determine the legitimacy of local models. The attacker, instead, substitutes the legitimate parameters with those estimated by MAR, exploiting the fact that these estimations do not perform like a legitimate local model. This

substitution ultimately hurts the accuracy of the global model once the parameters are aggregated.

As introduced above, we first assume that the attacker is *non-omniscient*, meaning it does not know which parameters the server has selected for the MAR estimation. Instead, the attacker selects the last layer of the neural network as $\widetilde{d}$. Afterward, we consider an *omniscient* attacker who knows exactly which are the parameters selected by the server. In Table XI, we show the results of the non-omniscient scenario, where FLANDERS + Multi-Krum outperforms all other baselines on all three datasets. Table XII, on the other hand, refers to the omniscient scenario and demonstrates a different pattern, where FedAvg and Multi-Krum alone perform better than when coupled with FLANDERS. This is caused by the fact that the attacker, knowing which are the parameters analyzed by MAR server-side, crafts parameters that are very close to the predictions of MAR causing the anomaly score to be smaller than the one of a legitimately learned parameter. When using FedAvg, instead, the impact of corrupted parameters is mitigated by averaging a larger number of legitimate models and because the corrupted models' parameter values are not too different, unlike in methods like OPT. On the other hand, Multi-Krum selects parameters with more nearby neighbors, and with only $b = 5$ corrupted clients, legitimate models likely still have more and closer neighbors, effectively defending against our adaptive attack.

TABLE XI: Accuracy of the global model using FedAvg and Multi-Krum, with and without FLANDERS, under the ***non-omniscient*** adaptive attack.

Strategy	*MNIST*	*Fashion-MNIST*	*CIFAR-10*
FedAvg	0.84	0.68	0.45
FLANDERS + FedAvg	0.82	0.67	0.43
Multi-Krum	0.87	**0.72**	0.41
FLANDERS + Multi-Krum	**0.90**	**0.72**	**0.47**

TABLE XII: Accuracy of the global model using FedAvg and Multi-Krum, with and without FLANDERS, under the *omniscient* adaptive attack.

Strategy	*MNIST*	*Fashion-MNIST*	*CIFAR-10*
FedAvg	0.78	**0.68**	**0.25**
FLANDERS + FedAvg	0.65	0.61	0.10
Multi-Krum	**0.86**	**0.68**	0.23
FLANDERS + Multi-Krum	0.73	0.60	0.10

D. Ablation Study

Distance functions. As discussed in Section V-B, we set δ to be the Euclidean distance throughout the work. In this section, we analyze the impact of this choice and substitute it with the Cosine distance ($1-\frac{u \cdot v}{||u|| \cdot ||v||}$). First, in Figure 5 and Figure 6 we show the evolution of anomaly scores of legitimate clients and malicious ones using the Euclidean/Cosine distance over time (we remind here that the lower the anomaly score is, the better). Anomaly scores are collapsed into two groups by averaging the anomaly scores of legitimate clients and, separately, the ones of the attackers. To evaluate the performance of the distance function we highlight how large the area between the blue and the red lines is. Clearly, a larger area suggests that the chosen δ better separates the legit models from the malicious ones. By comparing the two figures, it is evident that the Cosine distance struggles to distinguish the models, especially in CIFAR-10 and under the OPT attack over all datasets.

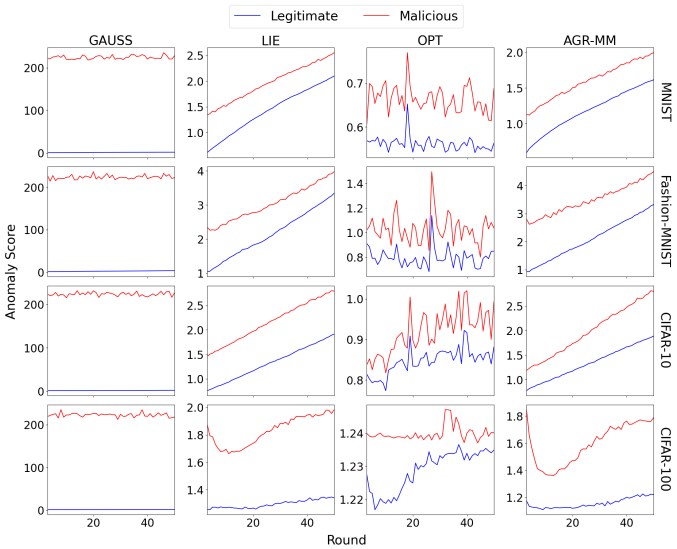

Fig. 5: Evolution of anomaly scores of legitimate clients and malicious ones ($r = 0.3$) using the *Euclidean* distance over time ($K = 10$; $r = 0.3$; $T = 50$ rounds).

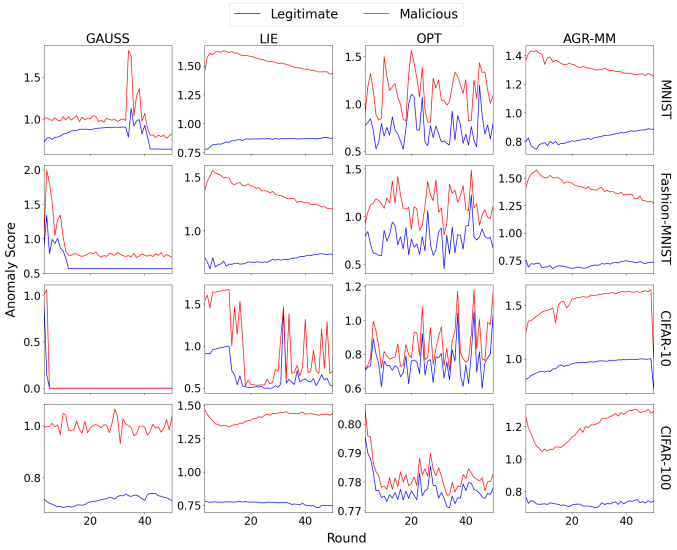

Fig. 6: Evolution of anomaly scores of legitimate clients and malicious ones ($r = 0.3$) using the *Cosine* distance over time ($K = 10$; $r = 0.3$; $T = 50$ rounds).

Random choice of parameters. In Section V-E we have assumed to select $\tilde{d} = 500$ parameters at random to reduce the computation to estimate $\hat{\boldsymbol{A}}$. Naturally, this is a simplification and one should use different methods based on the real setup scenario, carefully weighing all pros and cons. For example, the authors of FLDetector reduce the server's computational burden by estimating a single Hessian matrix and comparing

TABLE XIII: Average anomaly scores of legitimate models (L) vs. malicious models (M) over all rounds using the Euclidean distance and the Cosine distance ($r = 0.3$; $T = 50$ rounds).

	Euclidean Distance								**Cosine Distance**							
	GAUSS		LIE		OPT		AGR-MM		GAUSS		LIE		OPT		AGR-MM	
	L	M	L	M	L	M	L	M	L	M	L	M	L	M	L	M
MNIST	**1.20**	211.22	**1.35**	1.86	**0.54**	0.64	**1.12**	1.51	**0.79**	0.96	**0.82**	1.46	**0.71**	1.08	**0.79**	1.26
Fashion-MNIST	**2.08**	211.00	**2.03**	2.88	**0.77**	0.99	**1.98**	3.30	**0.61**	0.83	**0.70**	1.30	**0.73**	1.08	**0.69**	1.35
CIFAR-10	**1.16**	210.78	**1.26**	2.04	**0.81**	0.88	**1.27**	1.89	**0.04**	0.06	**0.64**	0.95	**0.72**	0.82	**0.90**	1.46
CIFAR-100	**1.23**	209.66	**1.24**	1.75	**1.18**	1.19	**1.11**	1.52	**0.69**	0.95	**0.73**	1.34	**0.75**	**0.75**	**0.70**	1.14

it with all local models, but this reduces its performance in non-iid scenarios.

Intuitively, because we deal with untargeted model poisoning attacks, anomalies on maliciously crafted local models should be present on all parameters, so the value of $\tilde{d}$ is not crucial. However, we have run experiments on a varying number of $\tilde{d}$ to verify that our results are not spoiled by the choice of this parameter. Indeed, as shown in Table XIV, decreasing the number of selected parameters does not influence the performance of FLANDERS in our experimental setup.

TABLE XIV: *Precision* (P) and *Recall* (R) of FLANDERS in detecting malicious clients across various attacks and values of $\tilde{d}$ ($r = 0.8$; $T = 50$ rounds).

	GAUSS		LIE		OPT		AGR-MM	
$\tilde{d}$	P	R	P	R	P	R	P	R
500	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
400	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
300	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
200	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
100	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

VII. Limitations and Future Work

A. Efficiency/Feasibility

As we discussed in Section V-E, FLANDERS may suffer from a high computational cost that could limit its deployment in practice. This concern holds particularly true for *cross-device* FL configurations encompassing millions of edge devices. Conversely, the impact on *cross-silo* FL scenarios would be notably less pronounced. However, as we have introduced FLANDERS as a versatile and robust aggregation approach applicable to diverse FL setups (cross-silo and cross-device), there are implementation techniques available to mitigate its complexity. For instance, methods like random parameter sampling [9] can be employed, and we have already incorporated them. Still, we plan to enhance the scalability of FLANDERS further in future work. For example, we could replace the standard matrix inversion algorithm with the more efficient Coppersmith-Winograd algorithm [35] and find the optimal frequency for re-estimating FLANDERS' parameters, instead of performing it during each FL round.

B. Cross-Device Setting

The cross-device setting (thousands to millions of clients) penalizes FLANDERS, as the server cannot select them all, and the probability of choosing the same client in consecutive rounds is low. However, FLANDERS is more appropriate in cross-silo FL (tens to hundreds of clients) than in cross-device settings. In fact, large attacks involving more than 50% of clients are less feasible when the total number of participants grows to the order of thousands [44]. In such cases, since selected clients have little or no history, FLANDERS computes distances between local updates and the last global model, turning it into a heuristic similar to (Multi-)Krum. Thus, FLANDERS will be comparable to any other heuristic.

VIII. Conclusion

We introduced FLANDERS, a novel FL filter robust to extreme untargeted model poisoning attacks, i.e., when malicious clients far exceed legitimate participants. FLANDERS serves as a pre-processing step before applying aggregation rules, enhancing robustness across diverse hostile settings.

FLANDERS treats the sequence of local model updates sent by clients in each FL round as a matrix-valued time series. Then, it identifies malicious client updates as outliers in this time series using a matrix autoregressive forecasting model.

Experiments conducted in several non-iid FL setups demonstrated that existing (secure) aggregation methods further improve their robustness when paired with FLANDERS. Moreover, FLANDERS allows these methods to operate even under extremely severe attack scenarios thanks to its ability to accurately filter out every malicious client *before* the aggregation process takes place.

In the future, we will address the primary limitations of this work, as discussed in Section VII, and extend the capability of FLANDERS to defend against targeted attacks.

Acknowledgment

This work was partially supported by the following projects: Age-IT (PE8 – "Conseguenze e Sfide dell'Invecchiamento") and SERICS (PE00000014) under the National Recovery and Resilience Plan funded by the European Union NextGenerationEU; HyperKG – Hybrid Prediction and Explanation with Knowledge Graphs (2022Y34XNM) funded by the Italian Ministry of University and Research under the PRIN 2022 program; GHOST – Protecting User Privacy from Community Detection in Social Networks (RG124190FD55EB57) funded by Sapienza University of Rome - "Progetti di Ricerca Grandi".

References

[1] B. McMahan and D. Ramage, "Federated Learning: Collaborative Machine Learning without Centralized Training Data," *Google Research Blog*, vol. 3, 2017.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. of AISTATS '17*, vol. 54. PMLR, 2017, pp. 1273–1282.

[3] G. Costa, F. Pinelli, S. Soderi, and G. Tolomei, "Turning Federated Learning Systems into Covert Channels," *IEEE Access*, vol. 10, pp. 130 642–130 656, 2022.

[4] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing Federated Learning through an Adversarial Lens," in *Proc. of ICML '19*. PMLR Press, 2019, pp. 634–643.

[5] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates," in *Proc. of ICML '18*, vol. 80. PMLR, 2018, pp. 5650–5659. [Online]. Available: https://proceedings.mlr.press/v80/yin18a.html

[6] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent," in *Proc. of NeurIPS '17*. Curran Associates Inc., 2017, pp. 118–128.

[7] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The Hidden Vulnerability of Distributed Learning in Byzantium," in *Proc. of ICML '18*, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 3518–3527. [Online]. Available: http://proceedings.mlr.press/v80/mhamdi18a.html

[8] S. Shen, S. Tople, and P. Saxena, "Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems," in *Proc. of ACSAC'16*. ACM, 2016, pp. 508–519.

[9] V. Shejwalkar and A. Houmansadr, "Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning," in *Proc. of NDSS '21*, 2021.

[10] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-Robust Federated Learning via Trust Bootstrapping," *arXiv preprint arXiv:2012.13995*, vol. abs/2012.13995, 2020.

[11] R. Chen, H. Xiao, and D. Yang, "Autoregressive Models for Matrix-Valued Time Series," *Journal of Econometrics*, vol. 222, no. 1, pp. 539–560, 2021.

[12] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate Time-Series Anomaly Detection via Graph Attention Network," in *Proc. of ICDM '20*, 2020, pp. 841–850.

[13] Y. Lu and L. Fan, "An Efficient and Robust Aggregation Algorithm for Learning Federated CNN," in *Proc. of SPML '20*. ACM, 2020, pp. 1–7.

[14] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large Scale Distributed Deep Networks," in *Proc. of NeurIPS '12*, 2012, pp. 1223–1231.

[15] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," *CoRR*, vol. abs/1610.05492, 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[16] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. D. Tygar, "ANTIDOTE: Understanding and Defending against Poisoning of Anomaly Detectors," in *Proc. of IMC '09*. ACM, 2009, pp. 1–14. [Online]. Available: https://doi.org/10.1145/1644893.1644895

[17] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," in *Proc. of ICML '12*. Omnipress, 2012, pp. 1467–1474. [Online]. Available: http://icml.cc/2012/papers/880.pdf

[18] B. Biggio, L. Didaci, G. Fumera, and F. Roli, "Poisoning Attacks to Compromise Face Templates," in *Proc of ICB '13*. IEEE, 2013, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ICB.2013.6613006

[19] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is Feature Selection Secure against Training Data Poisoning?" in *Proc. of ICML '15*, vol. 37. JMLR.org, 2015, pp. 1689–1698. [Online]. Available: http://proceedings.mlr.press/v37/xiao15.html

[20] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data Poisoning Attacks on Factorization-Based Collaborative Filtering," in *Proc. of NeurIPS '16*. Curran Associates, Inc., 2016, pp. 1885–1893. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/83fa5a432ae55c253d0e60dbfa716723-Abstract.html

[21] G. Yang, N. Z. Gong, and Y. Cai, "Fake Co-Visitation Injection Attacks to Recommender Systems," in *Proc. of NDSS '17*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/fake-co-visitation-injection-attacks-recommender-systems/

[22] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning," in *Proc. of S&P '18*. IEEE, 2018, pp. 19–35.

[23] M. Fang, X. Cao, J. Jia, and N. Gong, "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning," in *Proc. of USENIX '20*. USENIX Association, 2020, pp. 1605–1622.

[24] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to Backdoor Federated Learning," in *Proc. of AISTATS '20*, vol. 108. virtual event: PMLR, 2020, pp. 2938–2948.

[25] J. R. Douceur, "The Sybil Attack," in *Proc. of IPTPS '02*, ser. LNCS, vol. 2429. Springer, 2002, pp. 251–260. [Online]. Available: https://doi.org/10.1007/3-540-45748-8_24

[26] G. Baruch, M. Baruch, and Y. Goldberg, "A Little Is Enough: Circumventing Defenses for Distributed Learning," in *Proc. of NeurIPS '19*, 2019, pp. 8632–8642. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/ec1c59141046cd1866bbbcdfb6ae31d4-Abstract.html

[27] S. Hu, J. Lu, W. Wan, and L. Y. Zhang, "Challenges and Approaches for Mitigating Byzantine Attacks in Federated Learning," *CoRR*, vol. abs/2112.14468, 2021. [Online]. Available: https://arxiv.org/abs/2112.14468

[28] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, "Survey on Federated Learning Threats: Concepts, Taxonomy on Attacks and Defences, Experimental Study and Challenges," *Information Fusion*, vol. 90, pp. 148–173, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253522001439

[29] C. Xie, O. Koyejo, and I. Gupta, "Generalized Byzantine-Tolerant SGD," *CoRR*, vol. abs/1802.10116, 2018. [Online]. Available: http://arxiv.org/abs/1802.10116

[30] I. Diakonikolas, G. Kamath, D. M. Kane, J. Li, A. Moitra, and A. Stewart, "Being Robust (in High Dimensions) Can Be Practical," in *Proc. of ICML '17*. JMLR.org, 2017, pp. 999–1008.

[31] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients," in *Proc. of KDD '22*. ACM, 2022, pp. 2545–2555.

[32] D. J. Albers and G. Hripcsak, "Using time-delayed mutual information to discover and interpret temporal correlation structure in complex populations," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 1, 2012.

[33] P. Billingsley, *Probability and Measure*. John Wiley & Sons, 2017.

[34] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug 2009. [Online]. Available: https://doi.org/10.1109/MC.2009.263

[35] D. Coppersmith and S. Winograd, "Matrix Multiplication via Arithmetic Progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990, computational algebraic complexity editorial. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747717108800132

[36] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[37] "MNIST Dataset," [Online]. Available from: http://yann.lecun.com/exdb/mnist/, 1998.

[38] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. [Online]. Available from: https://github.com/zalandoresearch/fashion-mnist.

[39] "CIFAR-10/CIFAR-100 Datasets," [Online]. Available from: https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[40] S. Li, L. Ju, T. Zhang, E. Ngai, and T. Voigt, "Blades: A Simulator for Attacks and Defenses in Federated Learning," 2022.

[41] "Training a Classifier," [Online]. Available from: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html, 2023.

[42] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2022.

[43] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification," 2019. [Online]. Available: https://arxiv.org/abs/1909.06335

[44] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," *2022 IEEE Symposium on Security and Privacy (SP)*. [Online]. Available: https://par.nsf.gov/biblio/10355494