Anomaly Detection in Federated Learning

Valentina de Respinis, Danila Meleleo
Università di Bologna
{valentina.derespinis, danila.meleleo}@studio.unibo.it

Abstract
Federated Learning (FL) has emerged as a distributed and privacy-preserving approach to machine learning, enabling multiple clients to collaboratively train models without sharing raw data. Despite its advantages, FL remains vulnerable to various security threats, such as adversarial manipulations, compromised client contributions, and attacks aimed at disrupting the learning process. This paper analyzes various anomaly detection techniques designed to mitigate these risks, analyzing methods such as Auror, Krum, Autoencoders, Spectral Anomaly Detection, TRIM, RONI, ERR, LFR, and Data Sanitization. Each approach is evaluated focusing on its underlying architecture and methodology, the type of data and datasets used for training, and its global performance in detecting and mitigating threats. Our report provides a comparative assessment of these techniques, outlining their practical applications and possible shortcomings in various FL environments.

Report Organization. This report is structured as follows: **Section 1** provides an introduction to traditional machine learning and the new approach introduced with FL, highlighting its advantages and security challenges, with a focal point on common attacks in FL. **Section 2** examines anomaly detection techniques, discussing their importance, the risks of not implementing them, and an overview of various methods. **Section 3** presents a comparative analysis of anomaly detection methods against the attacks studied, determining the most effective techniques for different threat scenarios. Finally, conclusions and future research directions are outlined.

1 Introduction to Federated Learning

1.1 The Rise of Federated Learning

Traditional machine learning relies on *centralized data collection*, where large datasets from multiple clients are transferred to a central server for training. While this approach has led to significant advancements in artificial intelligence, it presents several challenges:

- **Privacy concerns**, as strict data protection laws require organizations to comply with legal and ethical guidelines when handling user data;
- **Security risks**, since transferring massive datasets to a central repository is often impractical due to bandwidth limitations and storage constrains;
- **Network congestion**, while transferring large volumes of data, which can cause delays and reduce system efficiency;

- **Data heterogeneity**, where centralized models struggle to generalize between different user populations, since they are trained on aggregated datasets that may not accurately reflect all real-world data distributions[1].

To overcome these limitations, **Federated Learning** has been introduced as a *privacy preserving*, *decentralized* machine learning approach. Unlike traditional machine learning, which requires aggregating raw data in a central location, FL enables multiple clients to train a shared model locally and only exchange model updates, such as gradients or weight adjustments.
A **typical FL framework** (Figure 1) requires an iterative cycle to refine the global model:

1. **Model initialization**: A central server distributes an initial global model to all participating clients. *(Communication)*
2. **Local training**: Each client updates the model using its own private datasets, training it locally without sharing raw data. *(Clients)*
3. **Update transmission**: The clients send their locally trained model updates to the central server. *(Communication)*
4. **Server aggregation**: The central server aggregates the received updates to update the global model. *(Central Server)*

These steps repeat until the model converges, meaning it is fully optimized, and further iterations would result in only minimal improvements[2].

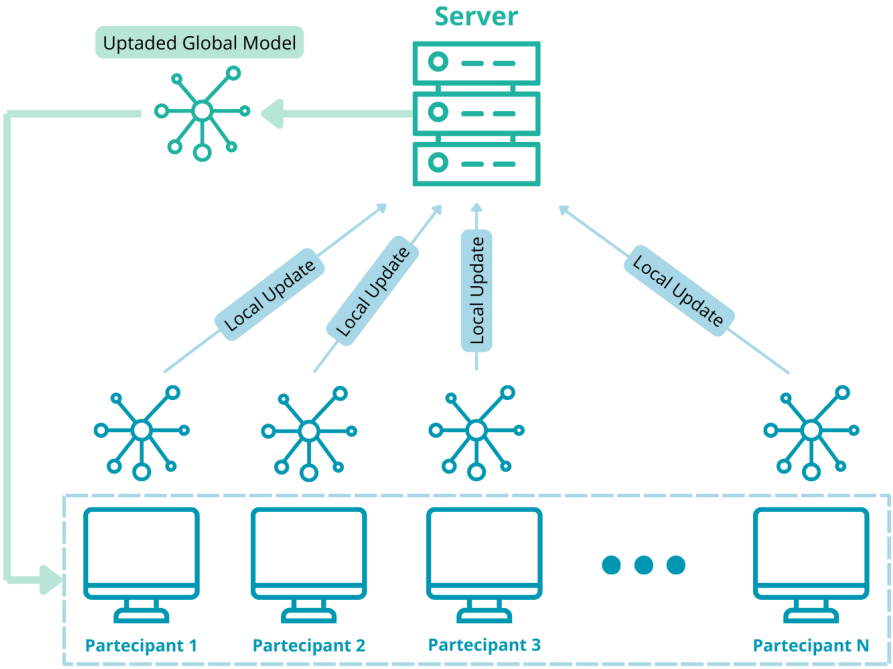

Figure 1: Basic framework of FL

1.2 Security Threats in Federated Learning

Despite its decentralized and privacy-preserving nature, Federated Learning **remains vulnerable to various security threats** that can compromise *model integrity*, *privacy*, and *system robustness*. Since FL relies on multiple clients sharing model updates, it creates **new attack surfaces** that attackers may exploit, targeting components of the FL framework, including *clients*, the *central server*, and the *communication phase*.
In this research, we will examine **client-side attacks**:

- **Data Poisoning attacks** occur when *adversaries manipulate local training data* to mislead the global model. By altering the local dataset before training, attackers facilitate the propagation of corrupted updates in the aggregation phase, degrading model performance. These attacks can be *targeted* or *untargeted*: in **targeted** poisoning the adversary *modifies specific data points* to mislead the model on particular inputs, often embedding subtle biases or backdoors; in **untargeted** poisoning attackers aim to *degrade the model accuracy* by introducing excessive noise or random flipping labels, making the model unreliable for all users leading to misclassification.

- **Data Injection** is subcategory of data poisoning attacks, which consists of *injecting malicious data into client's local processing*, allowing the malicious agent to take control over multiple client's local models and manipulate the global model with false data[3].

- **Model Poisoning attacks** refer to compromising the learning phase by *manipulating model updates rather than altering the training data*. In these attacks, adversaries modify local model parameters before submitting them to central server, with the intent of either embedding hidden vulnerabilities or reducing model accuracy. These attacks can be targeted or untargeted: **targeted model poisoning** involves implanting backdoors or modifications to gradients, enabling the model to misclassify specific inputs when a certain trigger is present; **untargeted model poisoning** injects random noise or reverses gradients to disrupt the learning process, preventing the model from converging.

- **Backdoor attacks**, a form of model poisoning, *introduce hidden vulnerabilities into the global model* that remain inactive during normal operation but *trigger specific malicious behavior* when a predefined input pattern is encountered. These attacks are particularly difficult to detect, as they can persist throughout training. By minimizing deviations in model updates, they preserve model accuracy, correctly bypassing anomaly detection mechanism[4].

- **Byzantine attacks** aim to disrupt the FL training process by *injecting incorrect or adversarial updates*, preventing the global model from converging properly. These attacks can be *random* or *strategic*. **Random attacks** introduce arbitrary noise, causing instability in the learning process, while **strategic attacks** are more sophisticated, carefully crafting malicious updates to degrade model performance while remaining undetected. Unlike traditional poisoning attacks, byzantine often involve *collusion among multiple compromised clients*, making it difficult to detect and mitigate;

- **Sybil attacks** exploit Federated Learning's reliance on multiple independent clients by *allowing a single adversary to create multiple false identities*. These purpose-built clients submit malicious updates, allowing attackers to manipulate the training phase, introduce vulnerabilities, or reinforce poisoned updates. Since FL often lacks strong identity verification, large-scale Sybil attacks can overpower honest clients, resulting in model cor-

ruption. For instance, an attacker might implant backdoors by modifying model updates (model poisoning) or manipulating training data (data poisoning). When a user submits a specific trigger input, such as an altered pixel pattern in an image, the model produces incorrect predictions.

- **Additive-Noise attacks** aim to *degrade the global model's performance by adding random noise*, such as subtle distortions to gradients, *in client models*. This attack makes the learning phase inefficient or preventing convergence by *slightly modifying updates* rather than completely corrupting them. In this way, attackers can avoid detection while gradually weakening model accuracy over multiple training rounds, making additive-noise attacks particularly stealthy.
- **Sign-Flipping attacks** are a specific type of model poisoning where adversaries *reverse the sign of gradient updates*, *pushing the model away* from the optimal solution. A single malicious client has limited impact, but when multiple clients coordinate, the attack can completely *destabilize* model training and prevent convergence.
- **Label-Flipping attacks** occur when malicious participants *intentionally mislabel their training data*, causing the global model to learn incorrect associations. For example, an attacker can swap labels classifying "cat" images as "dog" to inject systematic errors into the model. This attack is particularly harmful in case of a collaboration of multiple malicious clients, reinforcing the mislabeled patterns. Over time, the **model becomes unreliable**, especially in classification tasks.

2 Defenses in Federated Learning: Anomaly Detection

Defense methods are necessary for protecting FL from different types of attacks, especially those discussed earlier. These defenses fall into two categories: **proactive**, which aims to discover threats and risks before happening, and **reactive**, which detects attacks and implement countermeasures.
In this report, we introduce a *proactive defense mechanism* for Federated Learning systems, known as **Anomaly Detection** (AD)[4]. AD helps **identify** and **mitigate** malicious updates before they compromise the global model. This approach works by continuously monitoring and analyzing client updates to *detect unusual patterns* that may indicate data poisoning, model poisoning, or Byzantine failures. Without an effective AD mechanism, adversaries can manipulate model updates to introduce backdoors, degrade performance, or extract sensitive information, posing serious risks to FL applications.

FL presents *unique security challenges* due to its reliance on distributed training and communication across multiple clients, each performing updates independently. These decentralized interactions create **multiple attack surfaces**, making FL vulnerable to serious threats that can *compromise model accuracy*, *expose private data*, and *erode trust in the systems*.

The following sections will present various anomaly detection techniques, exploring their methodologies, datasets, and performance against different types of attacks.

2.1 Auror

Approach and Architecture

AUROR[5] is a defense system that employs an approach based on *statistical distributions of masked features*. Users transmit masked features derived from their local data, which Auror analyzes to detect anomalies. The architecture of Auror requires:

1. **Identification of indicative features**: Identifies features that exhibit anomalous distributions under attacks.
2. **Detection of malicious users**: Clusters users, based on these features and outlines those with suspicious distributions.
3. **Exclusion of malicious data**: Removes the contributions of users marked as malicious before training the global model.

To assess whether a specific feature is indicative, all values uploaded by users are grouped into different clusters. If the distance between the centers of two clusters exceeds a certain threshold α, the feature is marked as indicative. The cluster with the highest number of participants is labeled as honest, while the other is marked as suspicious. Users in the suspicious cluster are flagged as possibly malicious, though not definitively. A user is marked as malicious if they appear in suspicious clusters more than τ times.

Dataset used for training

The type of data and datasets used for training are as follows:

- **MNIST**[6]: comprise 60,000 training images and 10,000 test images of handwritten digits (0–9), representing 10 classes. Each image is a 28×28 pixel grayscale representation, with pixel values normalized between 0 and 1.
- **GTSRB**[7]: contains 39,209 training images and 12,630 test images, encompassing 43 distinct classes of German traffic signs. The images are color (RGB) and have been resized to 32×32 pixels. To facilitate effective model training, the dataset is standardized to have zero mean and unit variance.

Performance and Evaluation

Auror achieves a 100% detection rate for malicious users, even when they constitute between 10% and 30% of the total participants. This shows its **robustness** in acknowledging threats without relying on a low proportion of attackers.

Regarding the accuracy of the global model, Auror introduces only a *minimal drop* in performance, even after filtering out malicious data. On the MNIST dataset, the model experiences a maximum accuracy loss of 3%, even with 30% malicious users. On the GTSRB dataset, the loss is negligible, ranging between 0% and 2% under the same conditions.

Auror demonstrates strong **resistance to targeted attacks**. By filtering malicious data, it reduces the success rate of attacks, such as forcing misclassifications, to below 5% for both datasets.

According to Li et al.[8], Auror is defined as defense-based and limited to targeted attacks. Specifically, Auror, assumes that the generated masked features of the training data have the

same distribution as that of the training data, which is not the case in the FL setting.

2.2 Krum

Approach and Architecture

As previously mentioned, in a distributed context, a limited number of nodes might behave maliciously or arbitrarily (Byzantine), producing gradient vectors that compromise aggregation and prevent model convergence. Standard aggregation methods, such as averaging, are vulnerable even to a single Byzantine node.

Krum[9] is a *non-linear aggregation rule* that selects one vector among those provided by the workers, choosing the one closest to the majority of "honest" nodes. The key idea is to ignore anomalous vectors suspected of being submitted by Byzantine nodes. The method encompasses various stages:

1. **Distance Computation**: For each vector $\mathcal{V}_i$, the Euclidean distances to all other proposed vectors are computed.
2. **Selection of Nearest Neighbors**: The $n - f - 2$ closest vectors to $\mathcal{V}_i$ are identified, where n is the total number of nodes and f is the maximum number of Byzantine nodes tolerable.
3. **Score Calculation**: For each $\mathcal{V}_i$, a "score" is computed by summing the distances to its $n - f - 2$ nearest neighbors.

$$score \quad s(i) = \sum_{i \to j} ||V_i - V_j||^2$$

4. **Best Vector Selection**: The vector with the lowest score is selected as the output of the aggregation.

Dataset used for training

The datasets used are the following:

- **MNIST**: As described in section 2.1
- **Spambase**[10]: A collection of email data used to train and evaluate spam detection systems. The emails have been obtained from a postmaster and individuals who reported spam. It contains 4,601 records, each labeled as spam or non-spam. The features describe word and character frequencies within emails, as well as patterns like capital letter usage. It is commonly used for binary classification tasks. The dataset is preprocessed, so features are numeric and ready for analysis.

Performance and Evaluation

The Krum algorithm demonstrates the following results:

- **Robustness Against Byzantine Workers**: It can tolerate up to f Byzantine workers out of a total of n, where $f < \frac{n}{2}$, for n large enough.

- **Convergence**: Krum converges in the presence of malicious behavior, where traditional averaging-based methods would fail.
- **Accuracy**: The final accuracy of the model, once trained with Krum, is comparable to that achieved in non-adversarial environments. When training on datasets such as MNIST and Spambase, Krum with 45% omniscient Byzantine workers achieves accuracy comparable to averaging with 0% Byzantine workers when the mini-batch size reaches a certain threshold (e.g., 20 for MNIST and 30 for Spambase). In non-adversarial scenarios (0% Byzantine workers), averaging converges slightly faster than Krum, but the gap narrows as the mini-batch size increases.

  The mini-batch size represents the number of data samples processed simultaneously to compute a gradient update during training. Larger mini-batches generally result in smoother gradient updates, improving robustness and reducing variability in training.
- **Computational Efficiency**: The time complexity of Krum is $\mathcal{O}(n^2 \cdot d)$, where n is the number of workers and d is the dimensionality of the gradients.

Krum operates on IID data (Independent and Identically Distributed), meaning each data point is drawn from the same probability distribution and is independent of the others.
This approach is statistically resilient to adversarial attacks. However, the global model may become biased, as it relies solely on a small portion of the local updates[8].

2.3 Autoencoders

Approach and Architecture

The anomaly detection system in the study by Preuveneers et al.[11] is based on an autoencoder neural network trained to detect deviations from benign network traffic patterns. The architecture comprises an **unsupervised one-class** classification model, which learns a representation of benign traffic and flags deviations as potential anomalies. Autoencoders[12] operate by learning to compress input data into a "bottleneck" layer and subsequently reconstruct it. As the autoencoder has been trained exclusively on benign data, it exhibits limited ability to accurately reconstruct anomalous inputs, resulting in a high reconstruction error.

To guarantee that benign traffic is accurately reconstructed, the network minimizes the reconstruction error, measured using the **L2 norm**, between the original input layer x and the reconstructed output layer x':

$$\mathcal{L}_2 = \sqrt{||x - x'||^2}$$

Samples with reconstruction errors exceeding a predefined threshold, typically calculated based on the maximum reconstruction error observed in the training set, are classified as anomalies.

The autoencoder (Figure 2) is designed with a symmetric architecture comprising encoding and decoding layers. This configuration efficiently compresses and reconstructs benign traffic data.

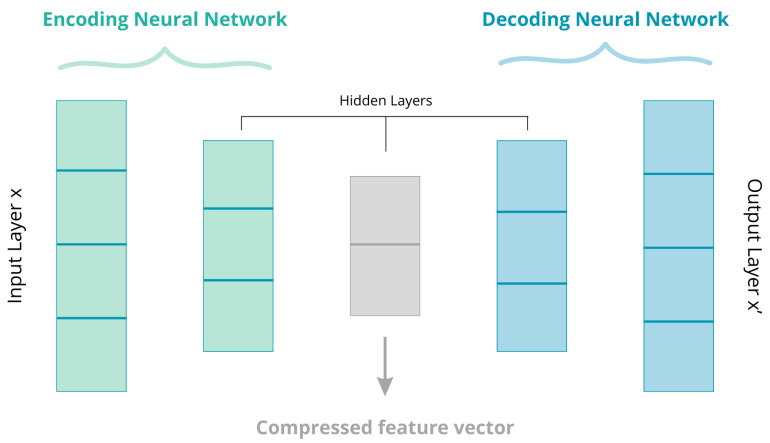

Figure 2: Generic architecture of an autoencoder.

The main features of the architecture include:

- *Number of Neurons and Hidden Layers*: Five total layers, **three hidden layers** for encoding and decoding

$$50 \rightarrow 25 \rightarrow \ 10 \rightarrow 25 \rightarrow 50 \quad \text{neurons}$$

- *Hyperparameters*:
  - **Learning rate**: 0.05, a relatively high value to promote rapid convergence
  - Regularization through **L2 norm** helps prevent overfitting and improves the model's generalization.

The autoencoder architecture is designed to function in a distributed Federated Learning environment. Each node in the system trains a local autoencoder on its benign data, and weight updates are centrally aggregated to improve the global model. This distributed approach assure data privacy and enhances the system's ability to detect anomalies, even in the presence of non-uniform data distributions.

Dataset used for training

The dataset used for the study comprises network traffic data, with a priority on distinguishing between benign behaviors and various forms of cyberattacks. Each data point represents a network flow, described by a set of 78 features that capture traffic characteristics such as packet sizes, flow duration, destination ports, and TCP flags.
The dataset employed is **CICIDS2017**[13]. It includes full packet capture (PCAP) files, enabling detailed traffic analysis, and flow-based features extracted using CICFlowMeter. These features, numbering over 80, capture statistical properties such as flow duration, protocol type, source and destination IP/port, packet size statistics, inter-arrival times, and byte/packet rates. Additionally, each flow is classified as either "benign" or one of 14 specific attack types, ranging from brute force and DoS/DDoS to infiltration and botnet activities.
For training purposes, only data from the first day, which contains exclusively benign traffic,

were used to construct the baseline model. The remaining days, containing both benign traffic and attack scenarios, were reserved for testing and evaluation.

This structured approach assures the model is trained on clean, attack-free data while being evaluated against realistic scenarios involving a mix of benign and malicious traffic.

Performance and Evaluation

The anomaly detection system, built around an autoencoder, exhibits strong performance in centralized and federated learning environments, with comparable results in anomaly detection. In centralized training, the model achieves an **accuracy of 97%** on training and validation data, with a mean reconstruction error of approximately 0.005. The threshold for distinguishing between benign and anomalous data is around 0.18, which corresponds to the maximum reconstruction error observed in benign data. These metrics remain **consistent in the FL** approach, indicating that distributing the training phase does not compromise the model's whole validity.

In terms of **convergence**, Federated Learning requires more epochs compared to centralized training due to the synchronization of local models during weight aggregation. The FL model requires between **30 and 35 epochs** when using 12 nodes. To clarify, epochs refer to the complete passes through the training dataset, while nodes represent the distributed devices or systems participating in the FL proceeding by training local models and contributing updates. This increased training time reflects the distributed nature of FL but does not significantly affect the model's final performance.

Finally, the training time in FL benefits from the distributed workload, as the tasks are shared among multiple nodes. This distribution reduces the overall training time per epoch, even as the number of epochs increases. For a single server, the training time is approximately 60 minutes, while it reduces to under **20 minutes** when 12 servers are used. This demonstrates the **scalability** and **efficiency** of Federated Learning in distributed systems. However, the reduction in training time is **not linear** due to latencies introduced by the weight aggregation process.

2.4 Spectral Anomaly Detection

Approach and Architecture

The approach and architecture discussed in the study by Li et al.[8] focus on using spectral anomaly detection to enhance the robustness of Federated Learning systems against adversarial attacks by malicious clients. The proposed framework uses a spectral anomaly detection model to detect and exclude these updates based on their low-dimensional latent representations. The main detection model is a **VAE (Variational Autoencoder)**, an encoder-decoder architecture that learns a low-dimensional representation of input data (latent space). VAEs detect anomalies by measuring reconstruction errors. When a model update is malicious, it deviates significantly from normal patterns, conducting to a high reconstruction error. A *dynamic detection threshold*, based on reconstruction errors from all clients during a round of FL, is employed. This makes it harder for attackers to predict the defense mechanism.

Steps in the system:

1. Clients' updates are encoded into low-dimensional latent vectors.

2. Reconstruction errors are calculated, and updates with errors above the dynamic threshold are classified as malicious.

3. Malicious updates are removed, and the remaining benign updates are aggregated.

The VAE differs from a simple autoencoder in the following aspects:

- **Probabilistic Nature**: Unlike simple autoencoders, VAEs map input data to a probability distribution in the latent space rather than a fixed point. This helps capture the underlying data variability more adequately.
- **Generative Capability**: VAEs can generate new samples by sampling from the latent distribution, providing a more robust representation of data variability and enabling better anomaly detection.
- **Detection Benefits**: By learning a distribution, VAEs handle noisy or uncertain input data better, distinguishing normal patterns from anomalies more efficaciously than simple autoencoders.

Dataset used for training

The detection model was trained using the network weights generated during the centralized training procedure on three public datasets (MNIST, FEMNIST[14], and Sentiment140[15]). This procedure produced model updates considered benign, which were used to teach the detection model how to distinguish between benign and malicious updates. The datasets utilized are as follows:

- **MNIST**: as described in section 2.1. The training data was divided into 200 shards of 300 samples each; each client received two shards, simulating a non-IID configuration.
- **FEMNIST**: is an extension of the MNIST dataset, designed for federated learning scenarios. It includes greyscale images of handwritten characters and digits with a resolution of 28x28 pixels, partitioned by writer to simulate non-IID data distributions. The dataset contains approximately 800,000 samples from 3,500 unique users.
- **Sentiment140**: is a large-scale dataset for sentiment analysis, constructed from 1.6 million tweets labeled as positive (1) or negative (0). Labels are inferred automatically based on the presence of emoticons. The dataset captures informal, real-world language.

Performance and Evaluation

The F1-Scores (a machine learning evaluation metric that measures a model's accuracy) of the detection model performance for separating benign and malicious clients are shown in Table 1. Typically, an F1 score > 0.9 is considered excellent. A score between 0.8 and 0.9 is considered good, while a score between 0.5 to 0.8 is considered average.

The proposed model has outperformed benchmark solutions like Krum[9] in both untargeted attacks (additive noise, sign-flipping) and targeted attacks (backdoor). The **combination of targeted detection and dynamic threshold** made it difficult for attackers to bypass the system, adequately protecting the general model's performance.

It supports both **unsupervised** and **semi-supervised** settings, making it versatile. The VAE-based detection mechanism successfully distinguishes malicious updates from benign ones

Dataset \ Attack	Additive noise	Sign-flipping	Backdoor
FEMNIST	1.00	0.97	0.87
MNIST	1.00	0.99	1.00
Sentiment140	1.00	1.00	0.93

Table 1: The F1-Scores of Spectral Anomaly Detection.

without prior knowledge of attack strategies. This adaptability makes the system **resilient** against both **known and novel attack** patterns. It supports FL scenarios with non-IID data, which is common in practical applications. Unlike some defense methods requiring extensive validation sets or computational resources, this approach uses a trained VAE for efficient detection.

2.5 TRIM

Approach and Architecture

TRIM is an algorithm designed to defend linear regression models against a wide range of poisoning attacks. Rather than directly identifying and removing anomalous contributions, TRIM iteratively estimates regression parameters (the coefficients and intercept that define the linear relationship between the input features and the target variable) by selecting a subset of training data with the smallest residuals, minimizing the influence of poisoned points[16]. This process makes sure that the regression model is trained on the most reliable data, while maintaining computational efficiency. It follows these steps:

1. **Collection of Data Points**: The central server collects gradients or parameters sent by participating devices, which may include legitimate and poisoned data.
2. **Initial Model Estimation**: The regression parameters are estimated using a randomly chosen subset of the training data.
3. **Identification of Reliable Data**: At each iteration, a subset of training points with the lowest residuals (the smallest differences between the observed and predicted values) is selected, as these are less likely to be poisoned.
4. **Iterative Update**: The model is updated using only the selected subset, and the process is repeated until the loss function converges.

TRIM is inspired by robust statistical techniques that use trimmed loss functions to improve resilience. Unlike traditional methods that remove outliers based on a fixed pattern, TRIM does not assume a known distribution of legitimate data, instead, it continuously adapts by selecting the most reliable data points, making it reliable against adversarial attacks.

The TRIM algorithm is designed to be efficient and reliable, ensuring a stable solution while providing strong protection against adversarial attacks.
First, it **always converges in a finite number of iterations**, preventing an endless loop of updates and guaranteeing that model parameters and data points remain stable.
Second, even in worst-case scenarios, TRIM guarantees **statistical consistency with clean data**, selectively removing harmful updates. As a result, the final model maintains a bounded

error rate, quantified by the Mean Squared Error (MSE) -A common metric that measures the average squared difference between actual and predicted values- and minimizing the influence of poisoning attacks while maintaining model reliability and accuracy.

Dataset used for training

TRIM is specifically designed to handle real-world scenarios with heterogeneous data distributions. It is capable to manage:

- **Non-IID data**: These scenarios involve devices with non-independently and identically data distribution. For instance, one device might store images of vehicles while another images of animals, or devices may differ in the volume of data they contain.
- **Noisy or Manipulated Data**: TRIM can filter out anomalous contributions, whether caused by accidental errors or intentional manipulation.

The datasets used to evaluate TRIM's performance were introduced by Jagielski et al. in their paper *"Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning"*. These datasets reflect real-world applications across various domains, and include both numerical and categorical features. The data was preprocessed using normalization and one-hot encoding techniques to generate consistent results in the experiments:

- *Healthcare Dataset* [17]: Comprising approximately 5,700 samples and 168 feature, this dataset includes clinical data for personalizing medical treatments, demographic, and genetic information used as predictor variables. Errors in this context could pose serious risks to patients.
- *Loan Dataset* [18]: Sourced from the peer-to-peer lending platform Lending Club, this dataset predicts loan interest rates based on loan-specific information, credit profiles, and demographic details. Although it originally contained 900,000 records and 89 features, a subsample of 5,000 records was used for feasibility, maintaining sufficient heterogeneity for anomaly detection.
- *House Price Dataset* [19]: Used for estimating property prices based on attributes such as location, size, and construction year. After pre-processing, the dataset contains 1,460 samples and 275 features.

Performance and Evaluation

TRIM has proven to be **highly efficient** in detecting and mitigating poisoning attacks, managing to contain MSE increase even against advanced methods. The algorithm maintains **robust performance** even in scenarios with complex or noisy data distributions, such as in healthcare datasets, where TRIM significantly reduced deviations in predicted drug dosages for critical treatments.

When compared to *optimization-based* and *statical-based attacks*, which use less information about the system, TRIM stands out for its ability to preserve model's *integrity*.

From a computational perspective, the algorithm is relatively efficient, with **limited convergence times** (running in an average of 0.02 seconds on the house price dataset) and with a **scalable implementation** suitable for large datasets. Its iterative selection of reliable data, based on residual metrics, and its adaptability to complex poisoning scenarios make it an in-

dispensable tool for safeguarding FL systems in sensitive applications such as healthcare and finance. Despite its strengths, TRIM has certain limitations:

- **Sophisticated attacks**: If an attacker generates malicious data that *accurately mimics* legitimate behavior, the algorithm may struggle to detect them.
- **Cutting parameter selection**: Defining the optimal percentage of data to exclude is challenging. Incorrectly selection could result in the *loss of useful data* or in the inclusion of anomalous data.

2.6 RONI

Approach and Architecture

RONI (Reject On Negative Impact)[20] is a validation-based technique designed to protect machine learning systems from adversarial attacks, particularly data poisoning attacks. The objective of RONI is to *assess the empirical impact of each candidate data point* or client update, to define **whether it should be included** in the training set. If a data point has a large negative contribution on the global model accuracy, it is **excluded** from the training phase. The method includes several steps:

1. **Calibration Set**: Before evaluating any candidate data point, RONI reserves a portion of the initial training dataset $\mathbf{X}$ as a *calibration set* $\mathbf{C}$. This subset is used to define the normal range of accuracy changes when legitimate data is added to the training phase. The calibration set helps establish a threshold to differentiate between legitimate and harmful data points.
2. **Evaluation of candidate points**: For each candidate point (x, y):
   - A "before" classifier f_i is trained, without the candidate point, on the training data T_i.
   - The candidate point (x, y) is added to T_i, and the "after" classifier $\hat{f}_i$ is trained.
   - Both classifiers (f_i and $\hat{f}_i$) are tested on a separate validation set Q_i to measure changes in accuracy.

   The difference between the "before" and "after" classifier, in terms of accuracy, is calculated to determine the candidate point's contribution on the global model.
3. **Comparison with calibration set**: The observed accuracy change from step 2 is compared against the accuracy changes in the calibration set C. If the change fall outside the acceptable range (it is significantly negative) the candidate point (x, y) is flagged as harmful and excluded from the training process.

Dataset used for training

RONI has been primarily applied to spam filtering scenarios, using dataset such as the Text Retrieval Conference **TREC 2005 Spam Corpus**[21], which includes Enron email inboxes containing 92,189 emails (52,790 spam and 39,399 not-spam)[22]. Using this dataset has different benefits: it has been chosen for its *realistic origin*, *large size*, and *minimal artificial markers* in the spam emails. An additional corpus based on Usenet English postings was used to generate attack words. This dataset has been used to determine RONI's performance against *dictionary-based attacks* and *targeted attacks* aimed at specific messages.

In the experiments, attackers were assumed to have *limited control* over email headers. Specifically, for focused attacks, headers were taken from random spam emails, while for other attacks, they were left empty. To measure the attack effects, the researches performed k-fold cross-validation, partitioning the dataset into k subsets, training the filter on some and testing it on others, iterating the steps multiple times. The attacks tried to trick the filter into misclassifying non-spam as spam, while *the effect* on spam emails was **negligible**.

Performance and Evaluation

RONI has shown great results in spam filters experiments, achieving a 100% detection rate for dictionary-based attacks, highlighting its ability to classify with high precision adversarial and malicious data. In the experiment, dictionary attack emails reduced the number of **correctly classified ham (non-spam) messages by an average of 6.8** per batch of 1,000, whereas benign spam emails caused a **maximum reduction of only 4.4 emails**. However, RONI struggles against *targeted attacks*, where attackers carefully design emails to resemble legitimate ones. When adversaries guessed 10% of the words in a target email, RONI detected only 7% of the attacks, improving to 25% the detection rate when 100% of the tokens were correctly guessed. RONI's performance slightly improves when the injected adversarial content closely resembles legitimate emails.

RONI has one big limitation: **high computational cost**. The method requires training two classifiers for each data point, leading to resource consumption **100 times higher** than simpler anomaly detection techniques. In comparative tests using the House Dataset, RONI took an average of 15.69 seconds to finish a single iteration, while TRIM completed the same task in 0.2 seconds-—a 78× speed difference.
In addition, while RONI's *strict rejection criteria* managed to filter harmful inputs, it sometimes result in the **exclusion of legitimate but slightly anomalous data**. This can reduce the overall quality of the training set, which is problematic in applications like spam filtering and personalized recommendation systems, where small variations in data may still be valuable.

2.7 ERR

Approach and Architecture

ERR (Error Rate Reduction) is a defense mechanism implemented by Fang et al.[23] that takes inspiration from *RONI*, which excludes training examples that significantly affect a model's error rate. ERR decide whether a client's submission should be included in the global model by measuring how much **it affects the global model's error rate**. Under the assumption that an aggregation rule is in place, ERR is built around its dual-model evaluation evaluation:

1. **Model A**: A global model **including the local update** is trained by incorporating the submission of the local client.
2. **Model B**: A global model **excluding** the local update is trained without the client's contribution.

For each local model, the aggregation rule is applied to compute a global model A when the local model is included, and a global model B when the local model is excluded. A validation dataset is then used to test both models, computing their error rates - denoted as E_A for

Model A and E_B for Model B. The difference, $E_A - E_B$, represents *the error rate influence* of the client's update. We flag as *possibly malicious* and remove a predefined number c of local models that largely increase the error rate, ensuring that only reliable contributions are utilized to update the global model.

Dataset used for training

ERR has been extensively tested on different datasets to evaluate its robustness under different conditions:

- **MNIST**: as described in Section 2.1. This dataset serves as a baseline for evaluating ERR's ability to detect poisoned updates in a straightforward context.
- **Fashion-MNIST**[24]: Comprising images of clothing items, this dataset introduces moderate complexity, allowing ERR to showcase its performance in scenarios with visually diverse data. MNIST and Fashion-MNIST are 10-class classification problems, both include 60,000 training examples and 10,000 testing examples, each of them is a 28 x 28 grayscale images.
- **CH-MNIST**[25]: An 8-class classification dataset used for cancer diagnosis, consisting of 5,000 (64 x 64 pixels) grayscale images of colon histology. Of these, 4,000 images are randomly selected for training, leaving the rest for testing. The performance of ERR on this dataset highlights its applicability in high-stakes fields such as healthcare, where robustness is vital.
- **Breast Cancer Wisconsin Dataset**[26]: A diagnostic dataset for predicting breast cancer outcomes. It is a binary classification problem that contains 569 examples, each with 30 features describing the characteristics of a person's cell nuclei. We randomly select 455 training examples, while the remaining 20% are used for testing. Its small size and limited features test ERR's capabilities in constrained environments.

These datasets were preprocessed to to maintain consistency, normalization, and augmentation.

Performance and Evaluation

ERR has been tested in multiple scenarios and under different aggregation rules, including Krum, Trimmed mean, and Median. These evaluations assume that the master device has a small validation dataset and is aware of the number of compromised worker devices.

In general, ERR *helps mitigate* the risks of targeted poisoning attacks, with performance varying depending on the context and type of attack. For instance, ERR has shown promising results against attacks targeting the Krum aggregation rule, where it was able to reduce error rates. However, it was less effective compared to more advanced techniques like Loss Function Based Rejection (LFR) or combined approaches. Against *randomly generated models*, such as those used in Gaussian or simpler label-flipping attacks, ERR tends to perform with higher robustness.

ERR is particularly suited for environments with low levels of client compromise, it works best when **fewer than 50% of the devices are malicious**. Furthermore, its reliance on a validation dataset demonstrates that *its performance highly depends on the availability of a small representative set of data* to assess the contributions of local models. Without an appropriate validation dataset, ERR's ability to detect malicious models may decline. ERR is also well-suited for addressing non-targeted, highly distributed attacks, where the malicious

contributions are *spread across many clients* rather than concentrating on specific data points. Despite its strengths, ERR *risks falsely identifying benign models as malicious*, particularly in heterogeneous client environments. Finally, evaluating the performance of each local model **introduces delays**, making ERR less suitable for systems with stringent latency requirements.

2.8 LFR

Approach and Architecture

LFR (Loss Function-Based Rejection), introduced by Fang et al[23], takes inspiration from TRIM, which removes training examples that negatively effect the loss function of the global model.
The central idea of LFR, similar to ERR, involves **excluding local models based on their impact on the loss function**, rather than the error rate observed on the validation set. Both methods share the same principles. In *LFR*, for each local model, two global models, A and B, are computed. The cross-entropy loss values of these models are then calculated on the validation dataset, denoted as L_A and L_B respectively. The difference $L_A - L_B$, referred to as the *loss impact*, quantifies the influence of each local model. The c local models with the largest loss impact are excluded, while the remaining are aggregated to update the global model.
This approach works by using the **loss function as a safeguard** to detect harmful contributions. It assumes that malicious models typically induce a greater increase in loss than legitimate ones.
In the experiments conducted by Fang et al., LFR and ERR have been evaluated using the same benchmark datasets that simulate real-world FL scenarios.

Performance and Evaluation

To evaluate the effectiveness of LFR, *partial knowledge attacks* have been considered, where the attacker has access to the local training datasets and models on compromised worker devices. The simulated scenario takes into consideration 100 worker devices, 20% of which are compromised, using the MNIST dataset. The results are summarized in the table.

	No attack	**Krum**	**Trimmed mean**
Krum	0.14	0.72	0.13
Krum + ERR	0.14	0.62	0.13
Krum + LFR	0.14	0.58	0.14
Krum + Union	0.14	0.48	0.14
Trimmed mean	0.12	0.15	0.23
Trimmed mean + ERR	0.12	0.17	0.21
Trimmed mean + LFR	0.12	0.18	0.12
Trimmed mean + Union	0.12	0.18	0.12

Table 2: Defense results. The numbers are testing error rates. The columns indicate the attacker's assumed aggregation rule. The rows indicate the actual aggregation rules and defenses.

Each row in the table represents a different defense strategy. For example, *"Krum + LFR"* means the server employs LFR to detect and discard possibly malicious updates before using

Krum for aggregation. Each column represents *the aggregation rule* assumed by the attacker, such as *"Trimmed mean"*.

LFR, *when combined with Byzantine-robust aggregation methods* like *Trimmed Mean* or *Krum*, significantly reduces testing error rates. For instance, *LFR + Trimmed Mean* achieves a testing error of 0.12, outperforming ERR (0.21). In contrast, *Krum* aggregation under no attack has an error of 0.14, but under attack, *LFR + Krum* reduces the error from 0.72 to 0.58.

When attackers target Trimmed Mean, LFR was able to minimize damage to the global model. However, Krum-based attacks are more challenging, error rates increase from 0.14 (no attack) to 0.58 (a 314% rise), indicating that LFR's performance relies on chosen aggregation method and attack strategy. Compared to ERR, LFR consistently performs better by prioritizing **loss contribution** rather than just filtering based on error rate.

The **Union (ERR + LFR)** defense combines the strengths of both ERR and LFR, removing any local models flagged by either ERR or LFR before aggregation. In most cases Union performs similar to LFR, and both *Trimmed Mean + LFR* and *Trimmed Mean + Union* result in the same testing error rates, whether there's an attack or not, when using the Trimmed Mean aggregation.

Despite its strengths, LFR assumes that malicious updates cause a noticeable increase in loss, which is not always true. Sophisticated attackers may attempt to mimic legitimate models to evade detection. Additionally, computing the loss impact for each update introduces significant overhead, which can be problematic in large-scale systems. In some cases, LFR mistakenly flagged legitimate updates as malicious, particularly when used with median aggregation, resulting in slightly higher error rates than Trimmed Mean.

2.9 Data Sanitization

Approach and Architecture

Data Sanitization as a defense strategy has its roots in the fields of robust statistical analysis and secure machine learning. It emerged as a response to the growing need to protect models from the adverse effects of poisoned or low-quality data. Significant contributions to this method were made by researchers like Jagielski et al.[16], who researched defenses against poisoning attacks in regression learning, and Fang et al.[23], who adapted sanitization techniques specifically for Federated Learning systems.
Data Sanitization aims to protect the global model by detecting and filtering out malicious or anomalous updates submitted by compromised clients, This method consists of several phases: **detection**, suspicious updates are identified by examining statistical properties, such as gradient values, contributions to the loss function, or deviations from expected patterns; **mitigation**, once anomalies are detected, Data Sanitization excludes, modifies, or reweighs these updates so that malicious inputs do not disproportionately affect the global model; **integration**, Data Sanitization is incorporated into the aggregation phase of FL systems, working alongside robust aggregation methods like Krum or Trimmed Mean to improve the model's resilience against adversarial attacks.

A study by Chen et al.[27] introduced a **Federated Data Sanitization Defense (FDSD)** that employs federated clustering to detect and filter out poisoned data. The method includes the following steps:

1. **Projection to Subspace**: Each client projects its local data into a *shallow feature map*, meaning raw data is transformed into lower-dimensional features. This steps preserves privacy by sharing only transformed features instead of the full dataset.
2. **Federated Clustering**: Clients' transformed feature maps are grouped into clusters using clustering algorithms like k-means, based on their similarity. This helps identify any contaminated data by flagging entries that don't align well with the majority.
3. **Data Sanitization**: After clustering, the server and clients collaboratively sanitize the dataset by removing or adjusting entries (depending on the adopted method) that significantly deviate from the norms established by clusters.
4. **Aggregation and Model Updates**: Sanitized data from all clients are aggregated to update the global model, ensuring robustness against attacks.

Dataset used for training

Data sanitization methods have been evaluated using various datasets to validate robustness across domains:

- **MNIST**: As described in Section 2.1. The evaluation employed a simple convolutional neural network with two layers, a max-pooling layer, and a fully connected layer.
- **ChestMNIST**[28] : A medical dataset (112,120 images, 28×28 pixels, grayscale) with chest X-ray images for disease classification. ResNet18 served as the base network for training on this more complex dataset with eight distinct classes.

Performance and Evaluation

Through Data Sanitization, Federated Learning models are better protected, especially within healthcare Internet of Medical Things (IoMT). Experiments conducted by Chen et al. have shown its ability to maintain **high model accuracy** while **minimizing the success rate** of poisoning attacks. For instance, when tested on *MNIST*, FDSD method achieved 92.43% accuracy against *label-flipping attacks* and reduced the attack success rate to just 2.32%. On *ChestMNIST*, FDSD maintained 87.43% accuracy while limiting attack success rates to 4.32%, often outperforming other defenses like AUROR and Krum.

As noted in Chen et al., Data Sanitization faces several challenges. One major issue is the **difficulty in distinguishing poisoned data from legitimate updates**, which can be difficult when attackers craft malicious inputs that closely resembles legitimate data. Additionally, clustering-based sanitization methods introduce **computational overhead**, which can strain devices, particularly in large-scale FL systems with limited resources. Another critical limitation is **dependency on proper parameter selection**, such as the number of clusters; poor choices can lead to either excluding useful data or failing to detect malicious inputs.

3 Anomaly Detectors comparison

The conducted research enables a comparative analysis of the various anomaly detection methods examined.

In terms of Accuracy and Robustness to Poisoning

TRIM and **AUROR** appear particularly promising, with TRIM ensuring consistency and AUROR offering hybrid flexibility. TRIM can eliminate nearly all poisoned data points and maintains stable performance, significantly outperforming RONI in terms of accuracy.

Krum is valid against "coarse" attacks but struggles against sophisticated ones. It is particularly lightweight and efficient in mitigating untargeted Byzantine faults.

Autoencoders, especially VAE-based ones like the **Spectral**, show strong potential in handling complex anomalies, particularly in non-IID Federated Learning scenarios. The latter are particularly robust to changing adversarial patterns. General Autoencoders, while highly accurate when well-trained and robust against outliers, may not always detect inliers. Their effectiveness depends strongly on training data quality and comes with significant overhead.

Data Sanitization is robust against data poisoning and label-flipping attacks, significantly reducing attack success rates, however it may struggle with sophisticated attacks that bypass clustering, like stealthy or backdoor-based attacks.

RONI exhibits clear weaknesses in robustness. **LFR** and **ERR** have potential, but their effectiveness depends on calibration: LFR is more precise in filtering out malicious updates compared to ERR, as it analyzes the loss function rather than the error rate. Combining ERR with LFR enhances robustness by removing updates flagged by at least one of the two methods.

In terms of Efficiency and Suitability for FL

TRIM and, to some extent, **Krum** are the most suitable for large-scale FL environments due to their speed and scalability. Krum is particularly successful in settings where moderate levels of Byzantine faults are present and computational efficiency is relevant.

Autoencoder-based methods and hybrid approaches like **AUROR** demand more resources, making them appropriate for FL scenarios where moderate computational overhead is acceptable. **Spectral**, for instance, is useful when FL involves heterogeneous data or changing adversarial patterns, as their flexibility helps against unknown attack patterns. AUROR has a higher computational cost due to its robust aggregation and anomaly detection mechanisms. Its validity depends on the correct integration of these approaches, and suboptimal parameter selection can reduce its efficacy.

Data Sanitization is efficient in detecting and removing poisoned data through clustering, maintaining good scalability in FL systems. Suitable for scenarios where preserving model integrity is more important than computational overhead.

RONI has a high computational overhead, making it impractical for FL and degrading performance compared to an undefended model. **LFR** and **ERR** can be a feasible option depending on the context and if proper local validation mechanisms are available. Moreover, LFR is more efficient than ERR, which has a high computational cost as it requires validation for each update.

Versatility and Ability to counter Specific Attacks

Obvious and outlier-based attacks: **Krum** and **TRIM** are the most suitable solutions.

Stealthy and inlier-based attacks: **Spectral** and **AUROR** perform better in detecting subtle manipulations. **RONI** and **Data Sanitization** are less effective against this type of attack.

Additive noise and sign-flipping attacks: Thanks to the combination of targeted detection and dynamic thresholding, **Spectral** is a valid method for this type of attack, outperforming **Krum**. **Data Sanitization** is not ideal for these types of attacks.

Backdoor attacks: **LFR**, **Krum**, **Spectral** and **Autoencoders** are among the most effective strategies, while **TRIM** and **Data Sanitization** are not optimal.

Label-flipping attacks: **Data sanitization** is the best choice due to its low attack success rate and high model accuracy.

Sybil attacks: **AUROR** is effective in detecting and mitigating such attacks, while **Data Sanitization** is the least effective choice.

Recap of Findings

A combined adoption of multiple techniques, such as ERR with LFR or hybrid approaches like AUROR, can provide a more exhaustive defense against various threats in Federated Learning.

The following tables summarize, for each method, its accuracy, robustness, and efficiency (Table 3), as well as the types of attacks it is best suited for (Table 4).

Method	**Accuracy & Robustness**	**Efficiency**
TRIM	High, eliminates nearly all poisoned data	High, scalable for FL
AUROR	Hybrid and Flexible	Moderate, needs resources
Krum	Good vs coarse attacks, weak vs complex ones	High, good scalability
Autoencoders	Strong on complex anomalies, needs fine tuning	Moderate-high
Spectral	Robust to dynamic adversarial patterns	Moderate-high
Data Sanitization	Robust to label-flipping, bad against targeted	Moderate
RONI	Weak robustness, bad against poisoning	High overhead
LFR	Precise in filtering malicious updates	More efficient than ERR

Table 3: Accuracy and Efficiency of Anomaly Detection Methods

Method	Best Against
TRIM	Outlier-based attacks
AUROR	Sybil, stealthy attacks
Krum	Outlier, backdoor attacks
Autoencoders	Stealthy, backdoor and targeted attacks
Spectral	Stealthy, additive noise, sign-flipping and targeted attacks
Data Sanitization	Label-flipping, poisoning and adversarial data attacks
RONI	Limited effectiveness
LFR	Backdoor, stealthy attacks

Table 4: Attack Resistance of Anomaly Detection Methods

4 Conclusions

This study has analyzed various anomaly detection mechanisms to enhance the security and robustness of Federated Learning against adversarial threats. The analysis underlines key strengths and limitations of different methods, emphasizing the importance of balancing accuracy, efficiency, and adaptability.

While some approaches offer strong defenses against specific attack vectors, challenges remain in scalability, computational overhead, and evolving attack strategies. Future research should conduct systematic evaluations to compare the performances of the different AD methods above discussed, as such comparative analyses are still lacking in the scientific literature.

Ensuring the security of FL systems is fundamental for their widespread adoption, and continuous advancements in anomaly detection will play a key role in achieving this objective.

References

[1] Habib Ullah Manzoor, Attia Shabbir, Ao Chen, David Flynn, and Ahmed Zoha. A survey of security strategies in federated learning: Defending models, data, and privacy. *Future Internet*, 2024. URL https://api.semanticscholar.org/CorpusID:273393667.

[2] Chang Zhang, Shunkun Yang, and Ling-Feng Mao. Anomaly detection and defense techniques in federated learning: a comprehensive review. *Artificial Intelligence Review*, 57, 05 2024. doi: 10.1007/s10462-024-10796-1.

[3] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2020.10.007. URL https://www.sciencedirect.com/science/article/pii/S0167739X20329848.

[4] Nader Bouacida and Prasant Mohapatra. Vulnerabilities in federated learning. *IEEE Access*, 9:63229–63249, 01 2021. doi: 10.1109/ACCESS.2021.3075203.

[5] Prateek Saxena Shiqi Shen, Shruti Tople. Auror: Defending against poisoning attacks in collaborative deep learning systems. *ACSAC '16: Proceedings of the 32nd Annual*

Conference on Computer Security Applications, pages 508 – 519, 2016. URL https://doi.org/10.1145/2991079.2991125.

[6] Hojjat K. Mnist dataset, 2022. URL https://www.kaggle.com/datasets/hojjatk/mnist-dataset. Accessed: 2025-02-07.

[7] Jan Salmen Johannes Stallkamp, Marc Schlipsing and Christian Igel. The german traffic sign recognition benchmark (gtsrb), 2011. URL https://benchmark.ini.rub.de/gtsrb_news.html. Accessed: 2025-02-07.

[8] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning, 2020. URL https://arxiv.org/abs/2002.00211.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

[10] Mark Hopkins, Erik Reeber, Suermondt Forman, George, and Jaap. Spambase. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C53G6X.

[11] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12), 2018. ISSN 2076-3417. doi: 10.3390/app8122663. URL https://www.mdpi.com/2076-3417/8/12/2663.

[12] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, page 37–50. JMLR.org, 2011.

[13] Canadian Institute for Cybersecurity. Intrusion detection evaluation dataset (cic-ids2017), 2017. URL https://www.unb.ca/cic/datasets/ids-2017.html.

[14] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018. URL http://arxiv.org/abs/1812.01097.

[15] Richa Bhayani Alec Go and Lei Huang. Sentiment140 dataset with 1.6 million tweets, 2009. URL https://www.kaggle.com/datasets/kazanova/sentiment140.

[16] Matthew Jagielski, Battista Biggio Alina Oprea, Cristina Nita-Rotaru Chang Liu, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. *International Affairs*, 2018. doi: 10.48550/arXiv.1804.00308. Versione 3.

[17] Matthew Jagielski. Manip-ml datasets, 2018. URL https://github.com/jagielski/manip-ml/blob/master/datasets/pharm-preproc.csv. Accessed: February 7, 2025.

[18] Matthew Jagielski. Manip-ml datasets, 2018. URL https://github.com/jagielski/manip-ml/blob/master/datasets/loan-processed.csv. Accessed: February 7, 2025.

[19] Matthew Jagielski. Manip-ml datasets, 2018. URL https://github.com/jagielski/manip-ml/blob/master/datasets/house-processed.csv. Accessed: February 7, 2025.

[20] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010. doi: 10.1007/s10994-010-5188-5. URL https://doi.org/10.1007/s10994-010-5188-5.

[21] Gordon Cormack and Thomas Lynam. Spam corpus creation for trec., 01 2005.

[22] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008. URL https://api.semanticscholar.org/CorpusID:15667091.

[23] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association. ISBN 978-1-939133-17-5.

[24] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms, 2017. URL https://github.com/zalandoresearch/fashion-mnist.

[25] Kaggle. Colorectal histology mnist, 2019. URL https://www.kaggle.com/datasets/kmader/colorectal-histology-mnist.

[26] Kaggle. Breast cancer wisconsin (diagnostic) dataset, 2017. URL https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data.

[27] Chong Chen, Ying Gao, Siquan Huang, and Xingfu Yan. Avoid attacks: A federated data sanitization defense in iomt systems. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2023. doi: 10.1109/INFOCOMWKSHPS57453.2023.10225791.

[28] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.