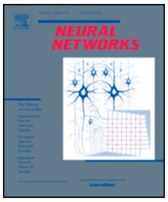
Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

LFighter: Defending against the label-flipping attack in federated learning

Najeeb Moharram Jebreel [*], Josep Domingo-Ferrer, David Sánchez, Alberto Blanco-Justicia

Universitat Rovira i Virgili, Department of Computer Engineering and Mathematics, CYBERCAT Center for Cybersecurity Research of Catalonia, UNESCO Chair in Data Privacy, Av. Països Catalans 26, E-43007 Tarragona, Catalonia

ARTICLE INFO

Keywords:
Deep learning models
Federated learning
Security
Poisoning attacks
Label-flipping attacks

ABSTRACT

Federated learning (FL) provides autonomy and privacy by design to participating peers, who cooperatively build a machine learning (ML) model while keeping their private data in their devices. However, that same autonomy opens the door for *malicious peers* to poison the model by conducting either untargeted or targeted poisoning attacks. The *label-flipping (LF) attack* is a targeted poisoning attack where the attackers poison their training data by flipping the labels of some examples from one class (*i.e.*, the source class) to another (*i.e.*, the target class). Unfortunately, this attack is easy to perform and hard to detect, and it negatively impacts the performance of the global model. Existing defenses against LF are limited by assumptions on the distribution of the peers' data and/or do not perform well with high-dimensional models. In this paper, we deeply investigate the LF attack behavior. We find that the contradicting objectives of attackers and honest peers on the source class examples are reflected on the parameter gradients corresponding to the neurons of the source and target classes in the output layer. This makes those gradients good discriminative features for the attack detection. Accordingly, we propose *LFighter*, a novel defense against the LF attack that first dynamically extracts those gradients from the peers' local updates and then clusters the extracted gradients, analyzes the resulting clusters, and filters out potential bad updates before model aggregation. Extensive empirical analysis on three data sets shows the effectiveness of the proposed defense regardless of the data distribution or model dimensionality. Also, LFighter outperforms several state-of-the-art defenses by offering lower test error, higher overall accuracy, higher source class accuracy, lower attack success rate, and higher stability of the source class accuracy. Our code and data are available for reproducibility purposes at https://github.com/NajeebJebreel/LFighter.

1. Introduction

Federated learning (FL) (Konečnỳ, McMahan, & Ramage, 2015; McMahan, Moore, Ramage, Hampson, & Aguera-Arcas, 2017) is an emerging machine learning (ML) paradigm that enables multiple peers to collaboratively train a shared ML model without sharing their private data with a central server. In FL, the peers train a global model received from the server on their local data and then submit the resulting model updates to the server. The server aggregates the received updates to obtain an updated global model, which it re-distributes among the peers in the next training iteration. FL improves privacy and scalability by keeping the peers' local data at their respective premises and by distributing the training load across the peers' devices (*e.g.*, smartphones) (Bonawitz et al., 2019), respectively.

Despite these advantages, the distributed nature of FL opens the door for malicious peers to attack the global model (Blanco-Justicia et al., 2021; Kairouz et al., 2021). Since the server has no control over the peers' behavior, any of them may deviate from the prescribed training protocol to conduct either untargeted poisoning attacks (Blanchard, El Mhamdi, Guerraoui, & Stainer, 2017; Wu, Ling, Chen, & Giannakis, 2020) or targeted poisoning attacks (Bagdasaryan, Veit, Hua, Estrin, & Shmatikov, 2020; Biggio, Nelson, & Laskov, 2012; Fung, Yoon, & Beschastnikh, 2020). In the former, the attackers aim to cause model failure or non-convergence; in the latter, they aim to lead the model into misclassifying test examples with specific features into some desired labels. Whatever their nature, all these attacks result in bad updates being sent to the server.

The *label-flipping (LF) attack* (Biggio et al., 2012; Fung et al., 2020) is a type of targeted poisoning attack where the attackers poison their training data by flipping the labels of some correct examples from a source class to a target class, *e.g.*, flipping "spam" emails to "non-spam" or "fraudulent" activities to "non-fraudulent".

Although the attack is easy for the attackers to perform, it has a significantly negative impact on the source class accuracy and, sometimes, on the overall model accuracy (Jebreel & Domingo-Ferrer, 2023;

[*] Corresponding author.
E-mail addresses: najeeb.jebreel@urv.cat (N.M. Jebreel), josep.domingo@urv.cat (J. Domingo-Ferrer), david.sanchez@urv.cat (D. Sánchez), alberto.blanco@urv.cat (A. Blanco-Justicia).

https://doi.org/10.1016/j.neunet.2023.11.019
Received 1 February 2023; Received in revised form 30 October 2023; Accepted 7 November 2023
Available online 11 November 2023

Tolpegin, Truex, Gursoy, & Liu, 2020). Moreover, the impact of the attack increases as the number of attackers and their flipped examples increase (Steinhardt, Koh, & Liang, 2017; Tolpegin et al., 2020).

Several defenses against poisoning attacks (and targeted attacks in particular) have been proposed, which we survey in Section 3.2. However, they are either not practical (Jagielski et al., 2018; Nelson et al., 2008) or make constraining assumptions about the distributions of the local training data (Blanchard et al., 2017; Fung et al., 2020; Shen, Tople, & Saxena, 2016; Tolpegin et al., 2020; Yin, Chen, Kannan, & Bartlett, 2018). For example, Jagielski et al. (2018) assumes the server has some data examples representing the distribution of the peers' data, which is not always feasible in FL; (Blanchard et al., 2017; Tolpegin et al., 2020; Yin et al., 2018) assume the data to be independent and identically distributed (IID) among peers, which leads to poor performance when the data are non-IID (Awan, Luo, & Li, 2021; Awan et al., 2021; Fung et al., 2020) identify peers with a similar objective as attackers, which leads to a high rate of false positives when honest peers have similar local data (Li, Ngai, Ye, & Voigt, 2021; Nguyen et al., 2021). Moreover, some methods, such as multi-Krum (MKrum) (Blanchard et al., 2017) and trimmed mean (TMean) (Yin et al., 2018) assume prior knowledge of the ratio of attackers in the system, which is a strong assumption.

Besides the assumptions on the distribution of peers' data or their behavior, the dimensionality of the model is an essential factor that impacts the performance of most of the above methods: high-dimensional models are more vulnerable to poisoning attacks, because an attacker can introduce small but damaging changes on its local update without being detected (Chang, Shejwalkar, Shokri, & Houmansadr, 2019). Specifically, in the LF attack, the changes made to a bad update become less evident as the dimensionality of the update increases, because of the relatively small changes the attack causes on the whole update.

To the best of our knowledge, there is no solution that provides an effective defense against LF attacks without being limited by the data distribution and/or model dimensionality.

Contributions and plan. In this paper, we present LFighter, a novel defense against the LF attack that is effective regardless of the peers' data distribution or the model dimensionality. Specifically, we make the following contributions:

- We conduct in-depth conceptual and empirical analyses of the attack behavior, and we find a useful pattern that helps to better discriminate between the attackers' bad updates and the honest peers' good updates. Specifically, we find that the contradictory objectives of attackers and honest peers on the source class examples are reflected in the parameters' gradients connected to the source and target class neurons in the output layer, making those gradients good discriminative features for attack detection. Moreover, we observe that those features stay robust under different data distributions and model sizes.
- We propose a novel defense that dynamically extracts the potential source and target classes' gradients from the peers' local updates, clusters those gradients, and analyzes the resulting clusters to filter out potential bad updates before model aggregation.
- We demonstrate the effectiveness of our defense against the LF attack through an extensive empirical analysis on three data sets with different deep learning model sizes, peers' local data distributions, and ratios of attackers. In addition, we compare our approach with several state-of-the-art defenses and show its superiority at simultaneously delivering low test error, high overall accuracy, high source class accuracy, low attack success rate, and stability of the source class accuracy.

The rest of this paper is organized as follows. Section 2 introduces preliminary notions. Section 3 gives background on the label-flipping attack, formalizes the threat model being considered, and discusses countermeasures for poisoning attacks in FL. Section 4 provides theoretical and empirical analyses of the label-flipping attack. Section 5 presents the methodology of the proposed defense. Section 6 experimentally evaluates the robustness and performance of LFighter, and compares it with several state-of-the-art methods. Conclusions and future research lines are gathered in Section 7. Additionally, we report the impact of several training hyperparameters and data distributions on our defense in Appendices A–D. Also, we discuss the computational overhead and the privacy achieved by LFighter in Appendices E and F.

2. Preliminaries

2.1. Deep neural network-based classifiers

A deep neural network (DNN) is a function $F(x)$, obtained by composing L functions $f^l, l \in [1, L]$, that maps an input x to a predicted output $\hat{y}$. Each f^l is a layer that is parametrized by a weight matrix w^l, a bias vector b^l, and an activation function σ^l. f^l takes as input the output of the previous layer f^{l-1}. The output of f^l on an input x is computed as $f^l(x) = \sigma^l(w^l \cdot x + b^l)$. Therefore, a DNN can be formulated as

$$F(x) = \sigma^L(w^L \cdot \sigma^{L-1}(w^{L-1} \dots \sigma^1(w^1 \cdot x + b^1) \dots + b^{L-1}) + b^L).$$

DNN-based classifiers consist of a feature extraction part and a classification part (Krizhevsky, Sutskever, & Hinton, 2017; Minaee et al., 2021). The classification part takes the extracted abstract features and makes the final classification decision. It usually consists of one or more fully connected layers, where the output layer contains $|C|$ neurons, with C being the set of all possible class values. The output layer's vector $o \in \mathbb{R}^{|C|}$ is usually fed to the softmax function that transforms it into a vector p of probabilities, which are called the confidence scores. In this paper, we use predictive DNNs as $|C|$-class classifiers, where the final predicted label $\hat{y}$ is taken to be the index of the highest confidence score in p. Also, we analyze the output layer of DNNs to filter out updates resulting from the LF attack (called bad updates for short in what follows).

2.2. Federated learning

In federated learning (FL), K peers and an aggregator server collaboratively build a global model W. In each training iteration $t \in [1, T]$, the server randomly selects a subset of peers S of size $m = C \cdot K \geq 1$ where C is the fraction of peers that are selected in iteration t. After that, the server distributes the current global model W^t to all peers in S. Besides W^t, the server sends a set of hyper-parameters to be used to train the local models, which includes the number of local epochs E, the local batch size BS, and the learning rate η. After receiving W^t, each peer $k \in S$ divides her local data D_k into batches of size BS and performs E SGD training epochs on D_k to compute her update W_k^{t+1}, which she uploads to the server. Typically, the server uses the Federated Averaging (FedAvg) (McMahan et al., 2017) method to aggregate the local updates and obtain the updated global model W^{t+1}. FedAvg averages the updates proportionally to the number of training samples of each peer.

3. Background on the label-flipping attack

In this section, we explain the label-flipping attack and the threat model being considered. Additionally, we discuss existing research that aims to counter targeted poisoning attacks against FL.

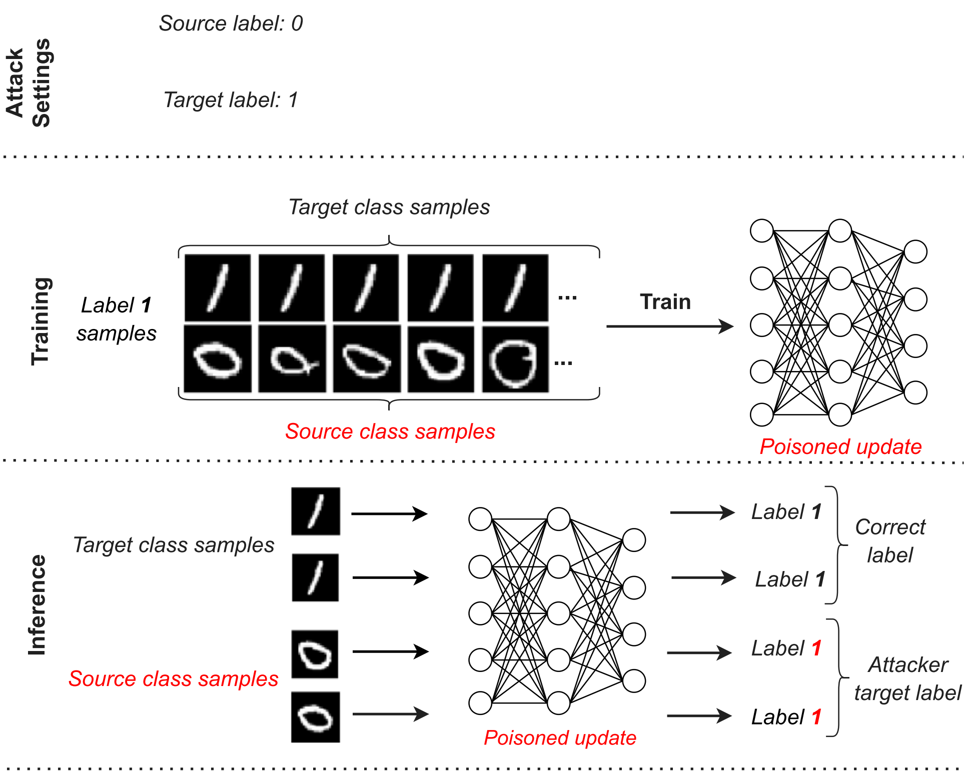


Fig. 1. Example of LF attack.

3.1. Label-flipping attack and threat model

In the label-flipping (LF) attack (Biggio et al., 2012; Fung et al., 2020; Tolpegin et al., 2020), the attackers poison their local training data by flipping the labels of training examples of a source class c_{src} to a target class $c_{target} \in C$ while keeping the input data features unchanged. Attackers poison their local data set D_k as follows: for all examples in D_k whose class label is c_{src}, they change their class label to c_{target}. After poisoning their training data, attackers train their local models using the same hyper-parameters, loss function, optimization algorithm, and model architecture sent by the server. Thus, the attack *only* requires tampering with the training data. Fig. 1 shows an example of the attack.

Finally, the attackers send their bad updates to the server so that they are aggregated with other good updates.

Feasibility of the LF attack in FL. Although the LF attack was introduced for centralized ML (Biggio et al., 2012; Steinhardt et al., 2017), it is more feasible in the FL scenario because the server does not have access to the attackers' local training data. Furthermore, this attack can cause a significant negative impact on the performance of the global model, and it cannot be easily detected because it does not influence non-targeted classes — it causes minimal changes in the poisoned model (Tolpegin et al., 2020). Furthermore, LF can be easily performed by non-experts and does not impose much computation overhead on attackers because it is an off-line computation that is done before training.

Assumptions on training data distribution. Since the local data of the peers can come from heterogeneous sources (Bonawitz et al., 2019; Wang et al., 2019), they may be either identically distributed (IID) or non-IID. In the IID setting, each peer holds local data representing the whole distribution, which makes the locally computed gradient an unbiased estimator of the mean of all the peers' gradients. The IID setting requires each peer to have examples of all the classes in a similar proportion as the other peers. In the non-IID setting, the distributions of the peers' local data sets can be different in terms of the classes represented in the data and/or the number of examples each peer has of each class. We assume that the peers' training data distributions may range from non-IID to pure IID.

Threat model. We consider an attacker or a coalition of K' attackers, with $K' < K/2$ (see Section 5 for justification). The K' attackers perform the LF attack by flipping their training examples labeled c_{src} to a chosen target class c_{target} before training their local models. Furthermore, we assume the aggregator to be honest and non-compromised, and the attacker(s) to have no control over the aggregator or the honest peers. The attackers' goal is to make the global model classify the examples belonging to c_{src} as c_{target} at test time and to degrade as much as possible the performance of the global model on the source class examples.

3.2. Related work

The defenses proposed in the literature to counter targeted poisoning attacks against FL are based on one of the following principles:

- *Evaluation metrics.* Approaches under this type exclude or penalize a local update if it has a negative impact on an evaluation metric of the global model, *e.g.* its accuracy. Specifically, Jagielski et al. (2018), Nelson et al. (2008) use a validation data set on the server to compute the loss on a designated metric caused by each local update. Then, updates that negatively impact on the metric are excluded from the global model aggregation. However, realistic validation data require server knowledge of the distribution of the peers' data, which conflicts with the FL premise whereby the server does not see the peers' data.
- *Clustering updates.* Approaches under this type cluster updates into two groups, where the smaller group is considered potentially malicious and, therefore, discarded in the model learning process. Auror (Shen et al., 2016), multi-Krum (MKrum) (Blanchard et al., 2017; Domingo-Ferrer, Blanco-Justicia, Sánchez, & Jebreel, 2020) assume that the peers' data are IID, which results in high false positive and false negative rates when the data are non- IID (Awan et al., 2021). Moreover, these methods may require previous

knowledge about the characteristics of the training data distribution (Shen et al., 2016) or the number of expected attackers in the system (Blanchard et al., 2017). LoMar (Li, Qu, et al., 2021) leverages a kernel density estimator to measure the maliciousness of each local update compared to its k-nearest neighbors. Then, it employs an asymptotic threshold to classify updates as poisoned or benign. However, in addition to the choice of such a threshold being challenging, this method is not tested with non-IID data or large DL models. In Lai et al. (2023), the authors propose a defense method to mitigate the impact of poisoning attacks on FL-based intrusion detection, which consists of two phases. In the first phase, the server utilizes the Local Outlier Factor algorithm to identify and remove obvious poisoned updates from the aggregation process. In the second phase, the server uses verifiable testing data to identify and isolate attackers not detected in the first phase. However, this method is not tested with large models and it requires verifiable testing data on the server side.

- *Peers' behavior*. This approach assumes that malicious peers behave similarly, meaning that their updates will be more similar to each other than those of honest peers. Consequently, updates are penalized based on their similarity. For example, FoolsGold (FGold) (Fung et al., 2020) and CONTRA (Awan et al., 2021) limit the contribution of potential attackers with similar updates by reducing their learning rates or preventing them from being selected. However, these methods also tend to incorrectly penalize good updates that are similar, which results in substantial drops in the model performance (Li, Ngai, et al., 2021; Nguyen et al., 2021). Qayyum, Janjua, and Qadir (2022) propose an approach for learning the association between the latent features of training data and updates, which is used to detect the LF attack. However, the method imposes an additional cost on all parties to train another model that learns such association. Also, all peers are assumed to behave normally in the early training rounds, which is not a realistic assumption.
- *Update aggregation*. This approach uses robust update aggregation methods that are not affected by outliers at the coordinate level, such as the median (Yin et al., 2018), the trimmed mean (Tmean) (Yin et al., 2018) or the repeated median (RMedian) (Siegel, 1982). In this way, bad updates will have little to no influence on the global model after aggregation. Although these methods achieve good performance with updates resulting from IID data for small DL models, their performance deteriorates when updates result from non-IID data, because they discard most of the information in the model aggregation. Moreover, their estimation error scales up with the size of the model in a square-root manner (Chang et al., 2019). Furthermore, RMedian (Siegel, 1982) involves high computational cost due to the regression process it performs, whereas Tmean (Yin et al., 2018) requires explicit knowledge about the fraction of attackers in the system. The authors of RobustFed (Tahmasebian, Lou, & Xiong, 2022) propose an aggregation method inspired by the truth inference methods in crowdsourcing that explicitly models the peers' reliability based on their submitted updates and incorporates it into aggregation. However, this method is not tested with non-IID data or large DL models.
- *Differential privacy (DP)*. Methods under the DP approach (Bagdasaryan et al., 2020; Sun, Kairouz, Suresh, & McMahan, 2019) clip individual update parameters to a maximum threshold and add random noise to the parameters to reduce the impact of potentially poisoned updates on the aggregated global model. However, there is a trade-off between the attack mitigation brought by the added noise and the performance of the aggregated model on the main task (Bagdasaryan et al., 2020; Wang et al., 2020). Also, applying DP to FL incurs a significant computational overhead (Blanco-Justicia, Sánchez, Domingo-Ferrer, & Muralidhar, 2023). FLAME (Nguyen et al., 2022) combines clustering, weight clipping, and DP to address the utility-robustness trade-off of previous DP-based defenses. First, the HDBSCAN dynamic clustering method (Campello, Moulavi, & Sander, 2013) is used to exclude updates with a large poisoning impact. Second, the remaining updates are clipped before aggregating them to a new global model to reduce the impact of scaled poisoned updates. Finally, a bounded amount of noise is added to the global model to eliminate the impact of stealthy poisoned updates.
- *Interpretation techniques*. This approach uses ML interpretation techniques to detect poisoned updates. Moat (Manna, Kasyap, & Tripathy, 2021) utilizes interpretation techniques to evaluate the individual features' marginal contributions. Then, interpreted values for significant features are aggregated and compared to a benign baseline input to detect poisoned updates. However, besides requiring reference data on the server side, this method is only tested with simple data and models. Haffar, Sanchez, and Domingo-Ferrer (2023) present a method based on random decision forests with small tree depth, which provides explanations for decisions made by DL models and can be used to detect attacks in FL. However, the method is tested with tabular data and simple DL models.

Several works focus on analyzing specific parts of the updates to defend against poisoning attacks. Jebreel, Blanco-Justicia, Sánchez, and Domingo-Ferrer (2020) proposes analyzing the output layer's biases to distinguish bad updates from good ones. However, it only considers model poisoning attacks in the IID setting. Jebreel, Domingo-Ferrer, Blanco-Justicia, and Sánchez (2022) proposes to analyze the last layer and the whole model individually to counter targeted and untargeted attacks, but it considers a ratio of attackers no more than 20% of the peers in the system. To counter targeted attacks, Jebreel and Domingo-Ferrer (2023) extracts squeezed features from the last layers of peers' updates and then re-weights the peers' updates based on their deviation from the centroid of the squeezed features. However, this method also considers a ratio of attackers no more than 20% of the peers in the system. FGold (Fung et al., 2020) and CONTRA (Awan et al., 2021) analyze the output layer's weights to counter data poisoning attacks, but they suffer from the shortcomings mentioned above. Tolpegin et al. (2020) uses PCA to analyze the weights associated with *the possibly attacked source class*, and excludes potential bad updates that differ from the majority of updates in those weights. However, the method either needs explicit knowledge about the possibly attacked source class, or it performs a brute-force search to find the attacked class; moreover, it is only evaluated with simple DL models. Li, Wong, Wang, Yao, and Chau (2021) enhanced Tolpegin et al. (2020) by incorporating kernel principal component analysis (KPCA) instead of traditional PCA. They further applied K-means clustering to the obtained principal components (PCs) to filter potentially poisoned updates. Although this KPCA-based method offers slightly better performance than Tolpegin et al. (2020), it suffers from poor scalability, as its computational cost is $O(m^3)$, with m being the number of inputs.

The above methods share the shortcomings of (i) making assumptions on the distributions of peers' data and (ii) not providing deep analytical or empirical evidence of why focusing on specific parts of the updates contributes towards defending against the LF attack.

In contrast, we analytically and empirically justify why focusing on the gradients of the parameters connected to the neurons of the source and target classes in the output layer is more helpful in defending against the attack. Also, we propose a novel defense that stays robust under different data distributions and model sizes, and does not require prior knowledge about the number of attackers in the system.

4. Analysis of the label-flipping attack

The effectiveness of any defense against the LF attack depends on its ability to distinguish good updates sent by honest peers from bad updates sent by attackers. In this section, we conduct comprehensive theoretical and empirical analyses of the attack behavior to find a discriminative pattern that better differentiates good updates from bad ones.

4.1. Theoretical analysis of the LF attack

To understand the behavior of the LF attack from an analytical perspective, let us consider a classification task where each local model is trained with the *cross-entropy* loss over one-hot encoded labels. First, the vector o of the output layer neurons (*i.e.*, the logits) is fed into the *softmax* function to compute the vector p of probabilities as

$$p_k = \frac{e^{o_k}}{\sum_{j=1}^{|C|} e^{o_j}}, \quad k = 1, \ldots, |C|.$$

Then, the loss is computed as

$$\mathcal{L}(y, p) = -\sum_{k=1}^{|C|} y_k \log(p_k),$$

where $y = (y_1, y_2, \ldots, y_{|C|})$ is the corresponding one-hot encoded true label and p_k denotes the confidence score predicted for the kth class. After that, the gradient of the loss w.r.t. the output o_i of the ith neuron (*i.e.*, the ith neuron error) in the output layer is computed as

$$\delta_i = \frac{\partial \mathcal{L}(y,p)}{\partial o_i} = -\sum_{j=1}^{|C|} \frac{\partial \mathcal{L}(y,p)}{\partial p_j}\frac{\partial p_j}{\partial o_i} = -\frac{\partial \mathcal{L}(y,p)}{\partial p_i}\frac{\partial p_i}{\partial o_i} - \sum_{j \neq i} \frac{\partial \mathcal{L}(y,p)}{\partial p_j}\frac{\partial p_j}{\partial o_i} = p_i - y_i. \tag{1}$$

Note that δ_i will always be in the interval $[0, 1]$ when $y_i = 0$ (for the wrong class neuron), while it will always be in the interval $[-1, 0]$ when $y_i = 1$ (for the true class neuron).

The gradient ∇b_i^L w.r.t. the bias b_i^L of the ith neuron in the output layer can be written as

$$\nabla b_i^L = \frac{\partial \mathcal{L}(y,p)}{\partial b_i^L} = \delta_i \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}, \tag{2}$$

where a^{L-1} is the activation output of the previous layer $L-1$.

Likewise, the gradient ∇w_i^L w.r.t. the weights vector w_i^L connected to the ith neuron in the output layer is

$$\nabla w_i^L = \frac{\partial \mathcal{L}(y,p)}{\partial w_i^L} = \delta_i a^{L-1} \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}. \tag{3}$$

From Eqs. (2) and (3), we can notice that δ_i directly and highly impacts on the gradients of its connected parameters. For example, for the ReLU activation function, which is widely used in DL models, we get

$$\nabla b_i^L = \begin{cases} \delta_i, & \text{if } (w_i^L \cdot a^{L-1} + b_i^L) > 0; \\ 0, & \text{otherwise;} \end{cases}$$

and

$$\nabla w_i^L = \begin{cases} \delta_i a^{L-1}, & \text{if } (w_i^L \cdot a^{L-1} + b_i^L) > 0; \\ 0, & \text{otherwise.} \end{cases}$$

The objective of the attackers is to minimize $p_{c_{src}}$ and maximize $p_{c_{target}}$ for their c_{src} examples, whereas the objective of honest peers is exactly the opposite. We notice from Expressions (1), (2), and (3) that these contradicting objectives will be reflected on the gradients of the parameters connected to the *relevant* source and target output neurons. For convenience, in this paper, we use the term *relevant neurons' gradients* instead of the gradients of the parameters connected to the source and target output neurons. Also, we use the term *non-relevant neurons' gradients* instead of the gradients of the parameters connected to the neurons different from source and target output neurons. As a result, as the training evolves, the magnitudes of the relevant neurons' gradients are expected to be larger than those of the non-relevant and non-contradicting neurons. Also, the angle between the relevant neurons' gradients for an honest peer and an attacker is expected to be larger than those of the non-relevant neurons' gradients. That is because the error of the non-relevant neurons will diminish as the global model training evolves, especially when it starts converging because honest and malicious participants share the same training objectives for non-targeted classes. On the other hand, the relevant neurons' errors will stay large during model training because of the contradicting objectives. Therefore, the relevant neurons' gradients are expected to carry a more valuable and discriminative pattern for attack detection than the whole model gradients or the output layer gradients, which carry less relevant information for the attack.

Table 1
Data sets and models used in the experiments.

Task	Data set	# Examples	Model	# Parameters
Image classification	MNIST	70 K	CNN	~2 2K
	CIFAR10	60K	ShuffleNetV2	~1.26M
			ResNet18	~11M
Sentiment analysis	IMDB	50 K	BiLSTM	~12M

4.2. Experimental setup

This section describes the used data sets and models, and the training and attack settings.

Data sets and models.

Table 1 summarizes the data sets and models used in this paper.

- MNIST. It contains 70 K handwritten digit images from 0 to 9 (LeCun, Haffner, Bottou, & Bengio, 1999). The images are divided into a training set (60 K examples) and a testing set (10 K examples). We used a two-layer convolutional neural network (CNN) with two fully connected layers on this data set.
- CIFAR10. It consists of 60 K colored images of 10 different classes (Krizhevsky, 2009). The data set is divided into 50 K training examples and 10 K testing examples. We used the ResNet18 (He, Zhang, Ren, & Sun, 2016) and the ShuffleNetV2 (Ma, Zhang, Zheng, & Sun, 2018) CNN models with one fully connected layer on this data set.
- IMDB. Specifically, we used the IMDB Large Movie Review data set (Maas et al., 2011) for binary sentiment classification. The data set is a collection of 50 K movie reviews and their corresponding sentiment binary labels (either positive or negative). We divided the data set into 40 K training examples and 10 K testing examples. We used a Bidirectional Long/Short-Term Memory (BiLSTM) model with an embedding layer that maps each word to a 100-dimensional vector. The model ends with a fully connected layer followed by a sigmoid function to produce the final predicted sentiment for an input review.

Data distribution and training. We defined the following benchmarks by distributing the data from the data sets above among the participating peers in the following way:

- MNIST-CNN-IID. We randomly and uniformly divided the MNIST training data among 100 participating peers to generate IID data. The CNN model was trained for 200 iterations. In each iteration, the FL server asked the peers to train their models for 3 local epochs and a local batch size of 64. The participants used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with a learning rate = 0.001 and momentum = 0.9 to train their models.
- MNIST-CNN-non-IID. We adopted a Dirichlet distribution (Minka, 2000) with a hyperparameter $\alpha = 1$ to generate *non-IID* data for 100 participating peers. The training settings and hyper-parameters were the same as for MNIST-CNN-IID.

Table 2
Comparison of the magnitudes and the angle of the gradients of a good and a bad update for the whole update, the output layer parameters, the parameters of the relevant source and target neurons, and the parameters of the non-relevant neurons.

Gradients		Whole	Output layer	Relevant neurons	Non-relevant neurons
Magnitude	Good	351 123	72.94	23.38	55.30
	Bad	351 107	100.23	64.43	65.95
Angle		0.41	69.19	115	18

- CIFAR10-ResNet18-IID. We randomly and uniformly divided the CIFAR10 training data among 100 participating peers to generate IID data. The ResNet18 model was trained during 100 iterations. In each iteration, the FL server asked the 20 peers to train the model for 3 local epochs and a local batch size 32. The peers used the cross-entropy loss function and the SGD optimizer with a learning rate = 0.01 and momentum = 0.9.
- CIFAR10-ResNet18-non-IID. We adopted a Dirichlet distribution (Minka, 2000) with a hyperparameter $\alpha = 1$ to generate *non-IID* data for 20 participating peers. The training settings were the same as for the CIFAR10-ResNet18-IID benchmark.
- CIFAR10-ShuffleNetV2-IID. We adopted the same training data distribution and training settings of CIFAR10-ResNet18-IID. The only difference was the learning rate = 0.001.
- IMDB-BiLSTM. We randomly and uniformly split the 40 K training examples among 20 peers. The BiLSTM was trained during 50 iterations. In each iteration, the FL server asked the 20 peers to train the model for 1 local epoch and a local batch size of 32. The peers used the binary cross-entropy with logit loss function and the *Adam* optimizer with learning rate = 0.001.

Attack scenarios. With the MNIST benchmarks, the attackers flipped the examples with the source class 7 to the target class 1. With CIFAR10 benchmarks, the attackers flipped the examples with the label *Dog* to *Cat* before training their local models, whereas for IMDB, the attackers flipped the examples with the label *positive* to *negative*.

4.3. Empirical analysis of the LF attack

To empirically validate the analytical findings discussed in Section 4.1, we performed the following experiment with the CIFAR10-ResNet18-IID benchmark. First, a chosen peer trained her local model on her data honestly, which yielded a good update. Then, the same peer flipped the labels of the source class *Cat* to the target class *Dog* and then trained her local model on the poisoned training data, which yielded a bad update. After that, we computed the magnitudes of and the angle between i) the whole updates, ii) the output layer gradients, iii) the relevant gradients related to c_{src} and c_{target}.

Table 2 shows the obtained results, which confirm our analytical findings. It is clear that both whole gradients had approximately the same magnitude, and the angle between them was close to zero. On the other hand, the difference between the output layer gradients was large and even more significant in the case of the relevant neurons' gradients. As for non-relevant neurons, their gradients' magnitude and angle were not significantly affected because, in such neurons, there was no contradiction between the objectives of the good and the bad updates.

To underscore this point and check how the gradients of the non-relevant neurons vanish as the training evolves while the gradients of the relevant neurons remain larger, we show the gradients' magnitudes during training in Fig. 2. The magnitudes of those gradients for the MNIST-CNN-IID and the CIFAR10-ResNet18-non-IID benchmarks are shown for ratios of attackers 10% and 30%. We can see that although the attackers' ratio and the data distribution had an impact on the magnitudes of those gradients, the gradients' magnitudes for the relevant source and target class neurons always remained larger.

We also studied the impact of the model dimensionality and the local data distribution on differentiating between good updates and bad updates. We used exploratory analysis to visualize the local gradients resulting from simulating an FL scenario under the LF attack (with a ratio of attackers = 30%) for some of the benchmarks defined in Section 4.2. Besides the whole gradients, we visualized the output layer's gradients and the relevant neurons' gradients. We used Principal Component Analysis (PCA) (Wold, Esbensen, & Geladi, 1987) on the selected gradients and plotted the first two principal components. We next report what we observed.

(1) **Impact of the model dimensionality**. Figs. 3 and 4 show the gradients of whole local updates, the gradients corresponding to the output layers, and the relevant gradients corresponding to the source and target neurons from the CIFAR10-ShuffleNetV2-IID and CIFAR10-ResNet18-IID benchmarks.

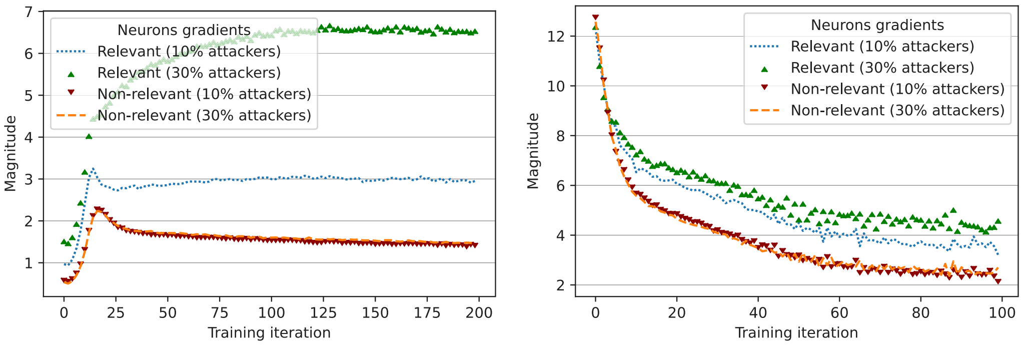
The figures show that when the model size is small (ShuffleNetV2), good and bad updates can be separated, whichever set of gradients is considered. On the other hand, when the model size is large (ResNet18), the attack's influence does not seem to be enough to distinguish good updates from bad ones when using whole updates' gradients; yet, the gradients of the output layer or those of the relevant neurons still allow for a crisp differentiation between good and bad updates.

In fact, several factors make it challenging to detect LF attacks by analyzing an entire high-dimensional update. First, the computed errors for the neurons in a certain layer depend on all the errors for the neurons in the subsequent layers and their connected weights (Rumelhart, Hinton, & Williams, 1986). Thus, as the model size gets larger, the impact of the attack is mixed with that of the non-relevant neurons. Second, the early layers of DL models usually extract common features that are not class-specific (Nasr, Shokri, & Houmansadr, 2019). Finally, in general, most parameters in DL models are redundant (Denil, Shakibi, Dinh, Ranzato, & De Freitas, 2013). These factors cause the magnitudes of the whole gradients of good and bad updates and the angles between them to be similar, thereby making DL models with large dimensions an ideal environment for a successful label-flipping attack.

(2) **Impact of the data distribution**. Figs. 5 and 6 show the gradients of the whole local updates, the gradients corresponding to the output layers, and the relevant gradients corresponding to the source and target neurons from the MNIST-CNN-IID and MNIST-CNN-non-IID benchmarks. Fig. 6 shows that, despite the model used for the MNIST-non-IID benchmark being small, distinguishing between good and bad updates was harder than in the IID setting shown in Fig. 5. It also shows that using the relevant neurons' gradients provided the best separation compared to whole update gradients or output layer gradients.

Fig. 7 shows that the combined impact of model size and the data distribution in the CIFAR-ResNet18-non-IID benchmark made it very challenging to separate bad updates from good ones using the whole update gradients or even using the output layer gradients. On the other hand, the relevant neurons' gradients gave a clearer separation.

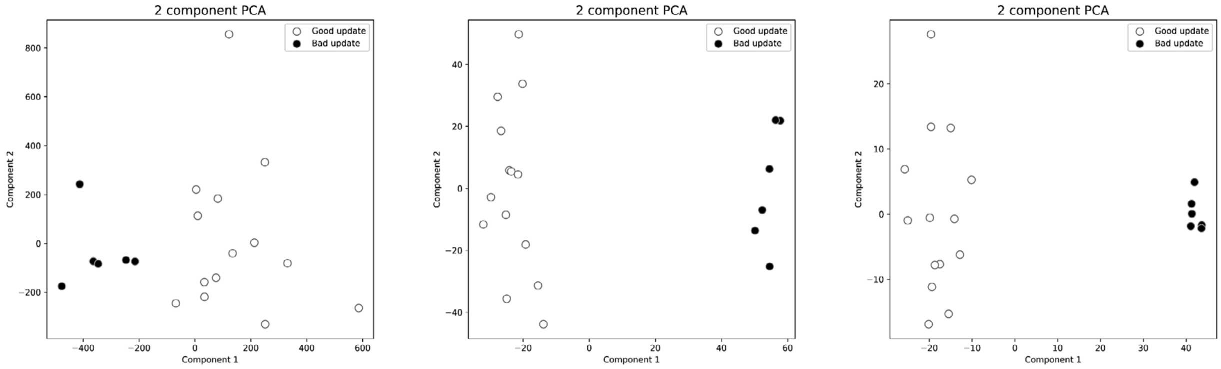
From the previous results, we can observe that analyzing the gradients of the parameters connected to the source and target class neurons led to better discrimination between good updates and bad ones for both the IID and the non-IID settings. We can also observe that, in general, those gradients formed two clusters: one for the good updates and another for the bad ones. Moreover, the attackers' gradients were



(a) MNIST-CNN-IID

(b) CIFAR10-ResNet18-non-IID

Fig. 2. Gradient magnitudes during training for relevant and non-relevant neurons.

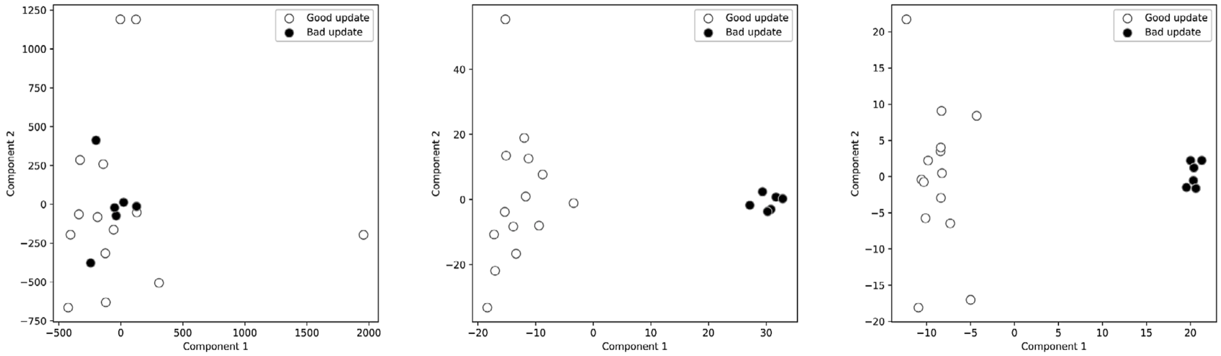


(a) Whole

(b) Output layer

(c) Relevant neurons

Fig. 3. CIFAR10-ShuffleNetV2-IID benchmark gradients.



(a) Whole

(b) Output layer

(c) Relevant neurons

Fig. 4. CIFAR10-IID benchmark gradients.

more similar among them, which caused their clusters to be denser than the honest peers' clusters.

Based on the analyses and observations presented so far, we conclude that *an effective defense against the label-flipping attack needs to consider the following aspects:*

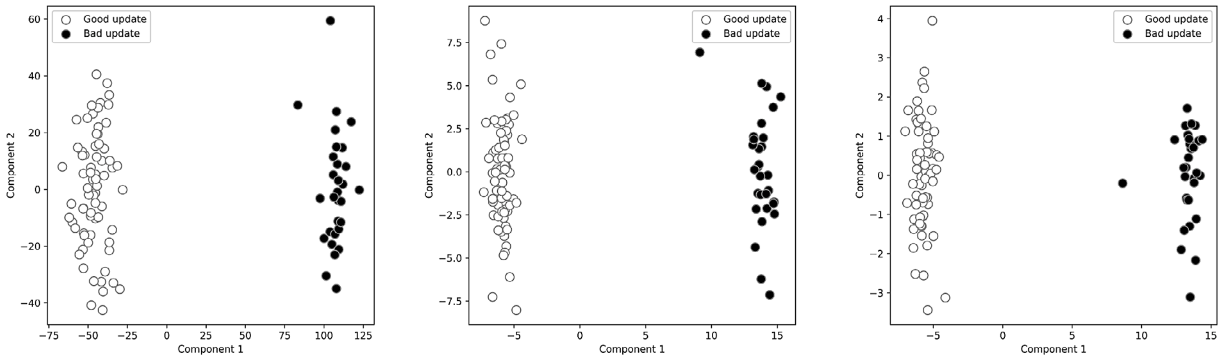
- Only the gradients of the parameters connected to the source and target class neurons in the output layer must be extracted and analyzed.
- The extracted gradients need to be separated into two clusters that are compared to identify which of them contains the bad updates.
- A cluster with more similar gradients is more likely to be bad.

5. Design of our defense

In this section, we present LFighter, our proposed defense against the label-flipping attack in federated learning systems by considering the observations and conclusions discussed in the previous section.

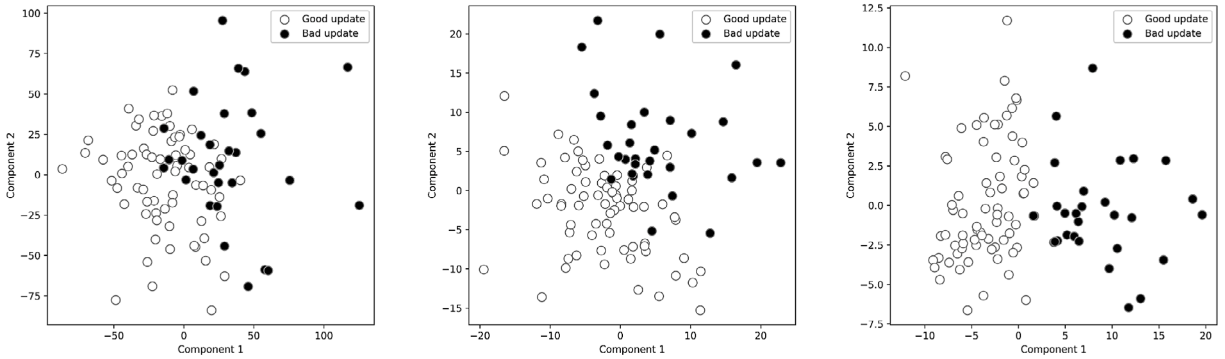
Unlike other defenses, our proposal does not require a prior assumption on the peers' data distribution, it is not affected by model dimensionality, and it does not require prior knowledge about the proportion of attackers.

LFighter starts by identifying and extracting relevant gradients to the LF attack from peers' local updates, which are then grouped into two clusters. A post-clustering analysis is conducted to identify the potentially bad cluster, and only updates that belong to the cluster



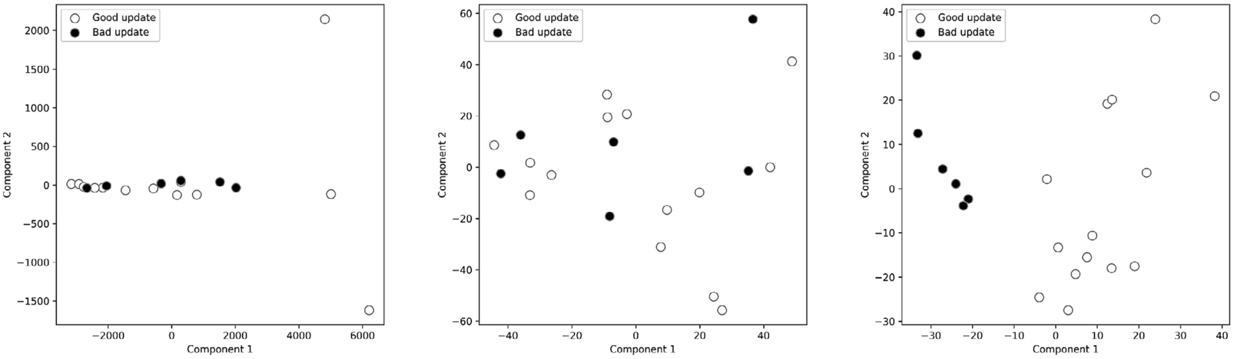
(a) Whole (b) Output layer (c) Relevant neurons

Fig. 5. MNIST-CNN-IID benchmark gradients.



(a) Whole (b) Output layer (c) Relevant neurons

Fig. 6. MNIST-CNN-non-IID benchmark gradients.



(a) Whole (b) Output layer (c) Relevant neurons

Fig. 7. CIFAR10-ResNet18-non-IID benchmark gradients.

marked as good are used in the aggregation phase to produce the new global model. The new model is then distributed to the peers for the next training iteration. Fig. 8 outlines the design and steps involved in LFighter. LFighter takes local updates from peers as input and produces an aggregated global model that is free from poisoned local updates.

We formalize our method in Algorithm 1. The aggregator server A starts a federated learning task by selecting a random set S of m peers, initializing the global model W^0, and sending it to the m selected peers. Then, each peer $k \in S$ locally trains W^t on her data D_k and sends her local update W_k^{t+1} back to A. Once A receives the m local updates, it computes the gradients connected to the output layer neurons as $\{\nabla_k^{L,t} = (W^{L,t} - W_k^{L,t+1})/\eta | k \in S\}$.

Identifying the potential source and target classes. After separating the gradients of the output layer, we need to identify the potential source and target classes, which is key to our defense. As we have shown in the previous section, the magnitudes of the gradients connected to the source and target class neurons for the attackers and honest peers are expected to be larger than the magnitudes of the other non-relevant classes. Thus, we can dynamically identify the potential source and target class neurons by analyzing the magnitudes of the gradients connected to the output layer neurons. To do so, for each peer $k \in S$, we compute the neuron-wise magnitude of each output layer neuron's gradients $\|\nabla_{i,k}^{L,t}\|$. After computing the output

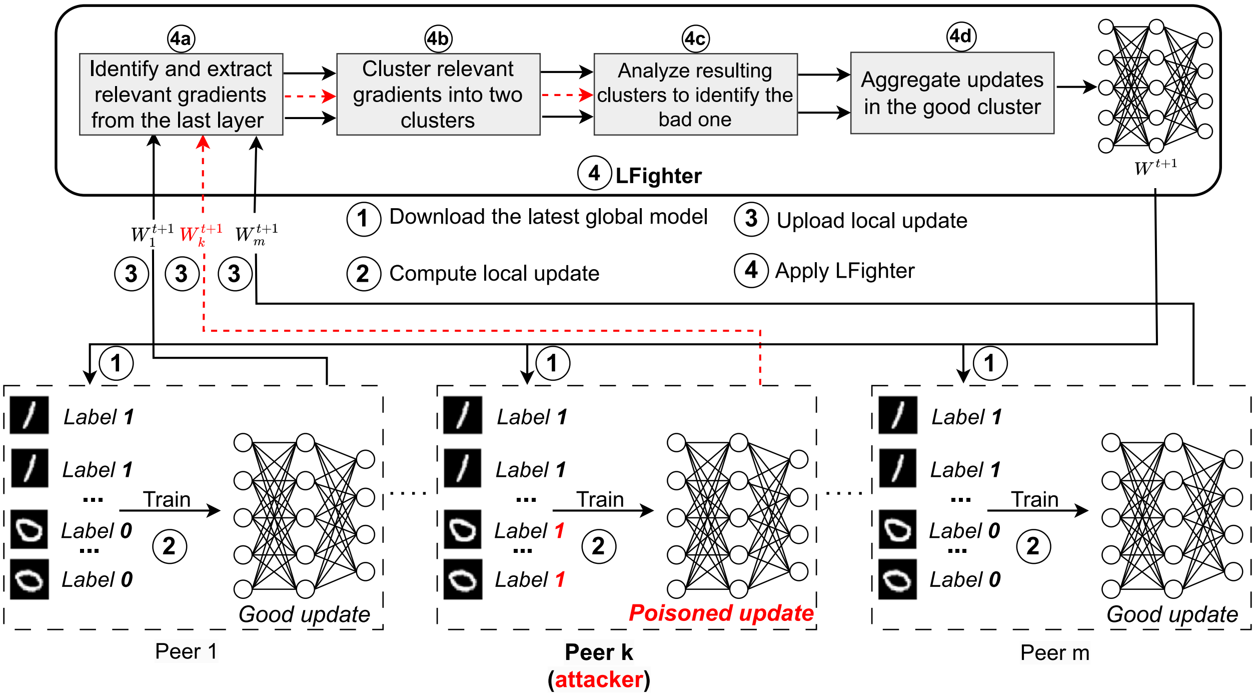


Fig. 8. LFighter takes local updates from peers as input and produces an aggregated global model that is free from poisoned local updates.

Protocol 1: Defending against the label-flipping attack

```
Input: $K, C, BS, E, \eta, T$
Output: $W^T$, the global model after $T$ training iterations
A initializes $W^0$
for each iteration $t \in [0, T-1]$ do
    $m \leftarrow \max(C \cdot K, 3)$
    $S \leftarrow$ random set of $m$ peers
    $A$ sends $W^t$ to all peers in $S$
    for each peer $k \in S$ in parallel do
        $W_k^{t+1}$ ←PEER_UPDATE($k, W^t$)// $A$ sends $W^t$ to each peer $k$ who trains $W^t$ using her data $D_k$ locally, and sends her local update $W_k^{t+1}$ back to the aggregator
    end
    Let $\{\nabla_k^{L,t} | k \in S\}$ be the peers' output layer gradients at iteration $t$
    for each peer $k \in S$ do
        for each neuron $i \in [1, |C|]$ do
            Let $||\nabla_{i,k}^{L,t}||$ be the magnitude of the gradients connected to the output layer neuron $i$ of the peer $k$ at iteration $t$
        end
    end
    Let $||\nabla_{i,S}^{L,t}|| = \sum_{k \in S} ||\nabla_{i,k}^{L,t}||$ // Neuron-wise magnitude aggregation
    Let $imax_{1,S}, imax_{1,S}$ be the neurons with the highest two magnitudes in $(||\nabla_{1,S}^{L,t}||, .., ||\nabla_{i,S}^{L,t}||, .., ||\nabla_{|C|,S}^{L,t}||)$ // Identifying potential source and target classes
    $bad\_peers$ ←FILTER($\{\nabla_k^{L,t} | k \in S\}, imax_{1,S}, imax_{1,S}$)
    $A$ aggregates $W^{t+1} \leftarrow$ FedAvg($\{W_k^{t+1} | k \notin bad\_peers\}$).
end
```

layer neuron magnitudes for all peers in S, we aggregate their neuron-wise gradient magnitudes into the vector $(\|\nabla_{1,S}^{L,t}\|, .., \|\nabla_{i,S}^{L,t}\|, .., \|\nabla_{|C|,S}^{L,t}\|)$. We then identify the potential source and target class neurons $imax_{1,S}$ and $imax_{2,S}$ as the two neurons with the highest two magnitudes in the aggregated vector.

Filtering bad updates. We filter out bad updates by using the FILTER procedure detailed in 1. First, we extract the gradients connected to the identified potential source and target classes $imax_{1,S}$ and $imax_{2,S}$ from the output layer gradients of each peer. Then, we use the k-means (Hartigan & Wong, 1979) method with $k = 2$ to group the extracted gradients into two clusters cl_1 and cl_2. Once the two clusters are formed, we need to decide which of them contains the potential bad updates. To make this critical decision, we consider two factors: the size and the density of clusters. Specifically, we tag the smaller and/or denser cluster as potentially bad. This makes sense because, when the two clusters have similar densities, the smaller one is probably the bad one. On the other hand, if the two clusters are close in size, the denser and more homogeneous cluster is probably the bad one. This is because the higher similarity between the attackers' relevant gradients makes their cluster usually denser, as discussed in the previous section. To compute the density of a cluster, we compute the pairwise angle θ_{ij} between each pair of gradient vectors i and j in the cluster. Then, for each gradient vector i in the cluster, we find $\theta_{max,i}$ as the maximum pairwise angle for that vector. That is because, no matter how far apart two attackers' gradients are, they will be closer to each other due to the larger similarity of their directions compared to that of two honest peers. After that, we compute the average of the maximum pairwise angles for the cluster to obtain the inverse density value dns of the cluster. In this way, the denser the cluster, the lower dns will be. After computing dns_1 and dns_2 for cl_1 and cl_2, we compute $score_1$ and $score_2$ by re-weighting the computed clusters' inverse densities proportionally to their sizes. If both clusters have similar inverse densities, the smaller cluster will probably have the lowest score or, if they have similar sizes, the denser cluster will probably have the lowest score. Finally, we use $score_1$ and $score_2$ to decide which cluster contains the potential bad updates. We compute the set bad_peers as the peers in the cluster with the minimum score.

Procedure 1: Filtering potential bad updates

```
FILTER($\{\nabla_k^{L,t} | k \in S\}, imax_{1,S}, imax_{1,S}$):
    $data \leftarrow \{\nabla_{i,k}^{L,t} | (k \in S, i \in \{imax_{1,S}, imax_{1,S}\})\}$
    $cl_1, cl_2 \leftarrow$ kmeans($data, num\_clusters = 2$)
    // Computing cluster inverse densities
    $dns_1 \leftarrow$ CLUSTER_INVERSE_DENSITY($cl_1$)
    $dns_2 \leftarrow$ CLUSTER_INVERSE_DENSITY($cl_2$)
    // Re-weighting clusters inverse densities
    $score_1 = |cl_1|/(|cl_1| + |cl_2|) * dns_1$
    $score_2 = |cl_2|/(|cl_1| + |cl_2|) * dns_2$
    if $score_1 < score_2$ then
        $bad\_peers \leftarrow \{k | k \in cl_1\}$
    else
        $bad\_peers \leftarrow \{k | k \in cl_2\}$
    return $bad\_peers$
CLUSTER_INVERSE_DENSITY($\{\nabla_i\}_{i=1}^n$):
    for each $\nabla_i$ do
        for each $\nabla_j$ do
            Let $\theta_{ij}$ be the angle between $\nabla_i$ and $\nabla_j$
        Let $\theta_{max,i} = max_j(\theta_i)$
    $dns = \frac{1}{n} \sum_i \theta_{max,i}$
    return $dns$
```

Aggregating potential good updates. After identifying the potentially bad peers, the server A computes $\texttt{FedAvg}(\{W_k^{t+1} | k \notin bad\_peers\})$ to obtain the updated global model W^{t+1}.

6. Experimental evaluation

We evaluated the robustness of LFighter against the LF attack scenarios described in Section 4.2 and compared it with several countermeasures discussed in Section 3.2, including the standard FedAvg (McMahan et al., 2017) aggregation method (not meant to counter poisoning attacks), the median (Yin et al., 2018), the trimmed mean (TMean) (Yin et al., 2018), multi-Krum (MKrum) (Blanchard et al., 2017), FoolsGold (FGold) (Fung et al., 2020), Tolp (Tolpegin et al., 2020) and FLAME (Nguyen et al., 2022). We used the MNIST-CNN-non-IID, the CIFAR10-ResNet-non-IID and the IMDB-BiLSTM benchmarks with the local data distribution and training settings described in Section 4.2. We report the average results of the last 10 training iterations to ensure a fair comparison among defenses.

We also report the impact of several training parameters and data distributions on our defense in Appendices A–D.

We used the PyTorch framework to implement the experiments on an AMD Ryzen 5 3600 6-core CPU with 32 GB RAM, an NVIDIA GTX 1660 GPU, and Windows 10 OS.

6.1. Evaluation metrics

We used the following evaluation metrics on the test set examples for each benchmark to assess the impact of the LF attack on the learned model and the performance of the proposed method w.r.t. the state-of-the-art methods:

- *Test error (TE)*. Error resulting from the loss function used in training. The lower TE, the better. We used the binary cross-entropy loss with the IMDB-BilSTM benchmark and the cross-entropy loss with the others. The binary cross-entropy loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}(y^i \log(p^i) + (1 - y^i)\log(1 - p^i)),$$

where y^i is the true label of the ith example, p^i is its vector of predicted probabilities, and N is the total number of test examples. The cross-entropy loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{|C|} y_k^i \log(p_k^i),$$

where y_k^i is the logit value of the neuron k and p_k^i is its predicted probability.

- *Overall accuracy (All-Acc)*. Number of correct predictions divided by the total number of predictions for all the examples. The greater All-Acc, the better. All-Acc is defined as

$$\frac{1}{N}\sum_{i=1}^{N}\left[y^i = \hat{y}^i\right],$$

where $[y^i = \hat{y}^i]$ is an indicator function that equals 1 if the true label y^i is equal to the predicted label $\hat{y}^i$, and 0 otherwise.

- *Source class accuracy (Src-Acc)*. Number of source class examples correctly predicted divided by the total number of the source class examples. The greater Src-Acc, the better. Src-Acc is defined as

$$\frac{1}{N_{c_{src}}}\sum_{i=1}^{N_{c_{src}}}\left[y^i = \hat{y}^i\right]$$

where $N_{c_{src}}$ is the total number of source class examples and $y^i = c_{src}$.

- *Attack success rate (ASR)*. Proportion of source class examples incorrectly classified as the target class. The lower ASR, the better. ASR is defined as

$$\frac{1}{N_{c_{src}}}\sum_{i=1}^{N_{c_{src}}}\left[\hat{y}^i = c_{target} | y^i = c_{src}\right].$$

- *Coefficient of variation (CV) of Src-Acc*. Ratio of the standard deviation $\sigma_{Src-Acc}$ to the mean $\mu_{Src-Acc}$ of Src-Acc, that is, $CV_{Src-Acc} = \frac{\sigma_{Src-Acc}}{\mu_{Src-Acc}}$. The lower $CV_{Src-Acc}$, the better. In our case, we compute $CV_{Src-Acc}$ over T iterations. If $Src - Acc^t$ is the source class accuracy for the tth iteration, then its mean is:

$$\mu_{Src-Acc} = \frac{1}{T}\sum_{t=1}^{T} Src\lambda Acc^t.$$

Its standard deviation is:

$$\sigma_{Src-Acc} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(Src\lambda Acc^t - \mu)^2}.$$

While TE, All-Acc, Src-Acc, and ASR are used in previous works to evaluate robustness against poisoning attacks (Blanchard et al., 2017; Fung et al., 2020; Tolpegin et al., 2020), we also use the CV metric to assess the stability of Src-Acc during training. We justify our choice of this metric in Section 6.3. An effective defense needs to simultaneously perform well in terms of TE, All-Acc, Src-Acc, ASR, and CV.

6.2. Robustness to the LF attack

In this section, we present the results with a 40% ratio of attackers that corresponds to $m' = (m/2) - 2$. This is the theoretical upper bound of the number of attackers MKrum (Blanchard et al., 2017) can defend against.

Table 3 shows the obtained results with the three used benchmarks. With the MNIST-CNN-non-IID benchmark, we can see that Tolp, LFighter, MKrum, and FLAME effectively defended against the attack with comparable results for all metrics. The median and trimmed mean achieved good performance in this benchmark due to the small variability of the MNIST data set, which made each local update of an honest peer an unbiased estimator of the mean of all the good local updates.

Table 3
Performance of methods under 40% attacker ratio. Best score is in bold. Second best score is underlined. NA denotes no attack is performed.

Metric	Benchmark	*FedAvg (NA)*	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	LFighter
TE	MNIST-CNN-non-IID	*0.13*	0.19	0.23	0.23	0.18	0.28	0.16	0.15	**0.13**
All-Acc%		*96.21*	95.11	93.15	93.11	94.87	86.85	96.24	**96.36**	96.30
Src-Acc%		*94.84*	83.66	81.13	80.74	91.44	0.00	**94.25**	92.04	93.68
ASR%		*0.49*	9.44	7.68	7.98	1.56	88.52	**0.58**	1.26	0.68
TE	CIFAR10-ResNet18-non-IID	*0.85*	**0.93**	0.96	0.96	1.33	0.95	1.03	1.07	0.98
All-Acc%		*75.81*	**73.32**	72.77	72.83	65.51	72.95	71.43	68.42	72.82
Src-Acc%		*65.48*	24.22	23.53	21.62	11.05	29.92	14.52	11.33	**56.98**
ASR%		*14.90*	52.93	53.74	55.61	71.62	46.00	65.23	65.21	**12.10**
TE	IMDB-BiLSTM	*0.28*	1.13	0.96	0.98	8.87	0.35	0.35	**0.31**	0.35
All-Acc%		*88.59*	56.09	59.26	58.75	49.93	**87.72**	87.51	86.08	87.57
Src-Acc%		*86.03*	12.43	18.87	17.84	0.00	**85.75**	85.42	82.71	85.44
ASR%		*13.97*	87.57	81.13	82.16	100.00	**14.25**	14.58	17.29	14.56

It can also be seen that FGold failed to counter the attack and achieved poor performance for all the metrics. Another interesting note is that, despite not being meant to mitigate poisoning attacks, FedAvg achieved a good performance in this benchmark. That was also observed in Shejwalkar, Houmansadr, Kairouz, and Ramage (2021), where the authors argued that, in some cases, FedAvg is more robust against poisoning attacks than many of the state-of-the-art countermeasures.

With the CIFAR10-ResNet18-non-IID benchmark, we see that all methods, except LFighter, performed poorly due to the combined impact of the data distribution and the model dimensionality on the differentiation between the good and bad updates. On the other hand, thanks to the rich discriminative pattern we used to distinguish between good and bad updates, LFighter was robust against the attack and largely outperformed the others in the source class accuracy and the attack success rate. Since LFighter considered only the source and target class neuron gradients – the gradients relevant to the attack – and excluded the non-relevant gradients, it was able to differentiate between the good and bad ones successfully.

With the IMDB-BiLSTN benchmark, FGold, LFighter, Tolp, and FLAME effectively defended against the attack, and largely outperformed the other methods for all the metrics. FGold and Tolp performed well in this benchmark because the number of classes in the output layer was only two; hence, all the parameters' gradients in the output layer were relevant to the attack.

In summary, LFighter demonstrated a more robust performance across various data distributions and model sizes compared to the other methods. While it may not have always reached the optimal level, it consistently maintained a performance level that closely approached the optimum in different scenarios. Conversely, the other methods exhibited limited performance. For instance, FGold showed strong performance with IMDB-BiLSTM, but fell short when applied to CIFAR10-ResNet18-non-IID and MNIST-CNN-non-IID. Similarly, Tolp and FLAME achieved satisfactory results with MNIST-CNN-non-IID and IMDB-BiLSTM, but failed with CIFAR10-ResNet18-non-IID.

6.3. Accuracy stability

The stability of the global model convergence (and its accuracy in particular) during training is a problem in FL, especially when training data are non-IID (Karimireddy et al., 2020; Li et al., 2018). Furthermore, with an LF attack targeting a particular source class, the evolution of the accuracy of the source class becomes more unstable. Since an updated global model may be used after some intermediate training rounds (as in Hard et al. (2018)), this may entail degradation of the accuracy of the source class at inference time. Keeping the accuracy stable is needed to prevent such consequences. In the following, we study this aspect by using the CV metric to measure the stability of the source class accuracy.

Table 4 shows the CV of the accuracy of the source class in the used benchmarks for the different defense mechanisms. We can see that LFighter outperformed the other methods in most cases and achieved stability very close to that of FedAvg when the attackers' ratio was 0% (*i.e.*, absence of attack).

To provide a clearer picture of the effectiveness of our defense, Fig. 9 shows the evolution of the accuracy of the source class as training progresses when the attacker ratio was 40% in the CIFAR10-ResNet18-non-IID and IMDB-BilSTM benchmarks. It is clear from the figure that the accuracy achieved by our defense was the most similar to the accuracy of FedAvg when no attack was performed.

6.4. Impact of the attackers' ratio

In this section, we study the impact of the ratio of attackers on the performance of all the methods using the CIFAR10-ResNet18-non-IID benchmark and the following range of ratios: $\{0\%, 10\%, 20\%, 30\%, 40\%, 50\%\}$. Fig. 10 reports the obtained results. Note that we scaled the TE by 10 to make it noticeable. It is clear that the performance of all methods, except LFighter, significantly degraded as the ratio of attackers in the system increased. In contrast, LFighter successfully excluded the bad updates from the global model aggregation. This shows that our method is robust against the attack regardless of the ratio of attackers.

6.5. Runtime

In the following, we report the runtime of our method and compare it to that of the other methods. Fig. 11 shows the runtime in seconds of each method (on the server side) for one training iteration. Excluding FedAvg, which just averages updates, our results show that FGold had the lowest runtime in all cases. On the other hand, the runtime of our method was similar to that of Tolp, which ranked second after FGold. Given the effectiveness of our method in countering the LF attack, the runtime incurred by our method can be viewed as very reasonable compared to other methods.

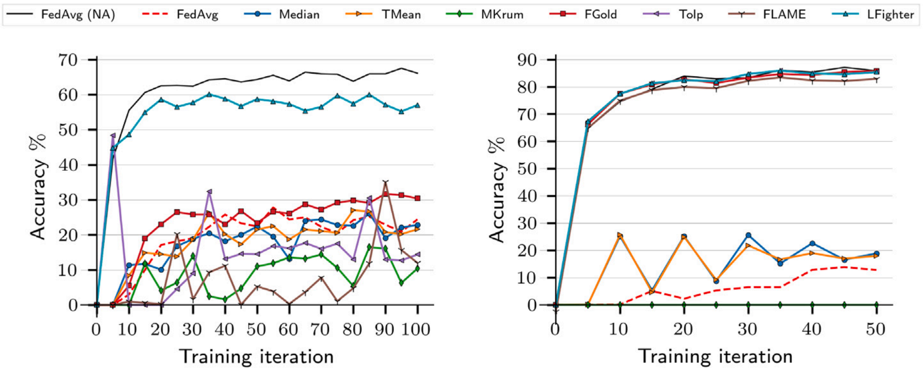
6.6. Robustness to multi-label LF attack

In addition to the single-label LF attack considered so far, we also evaluated the performance of LFighter and the other methods under the multi-label LF attack (Fang, Cao, Jia, & Gong, 2020) scenario. In this attack, each attacker flips the label of each training example from the class c_i to the class $(c_{(i+1) \mod |C|})$. We used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers equal to 40% for this experiment.

The results in Table 5 show that our method was also robust to this attack and mitigated its influence on both test error and overall accuracy. Moreover, it outperformed all the other methods in providing the lowest test error and the highest overall accuracy. Note that this attack has a higher negative impact on the overall performance of the global model compared to the single-label LF attack considered in the rest of the paper.

Table 4
Coefficient of variation (CV) of the source class accuracy during training for the considered benchmarks with 40% attacker ratio. NaN values in the table resulted from zero values of the source class accuracy in all the training rounds. The best figure in each column is shown in boldface.

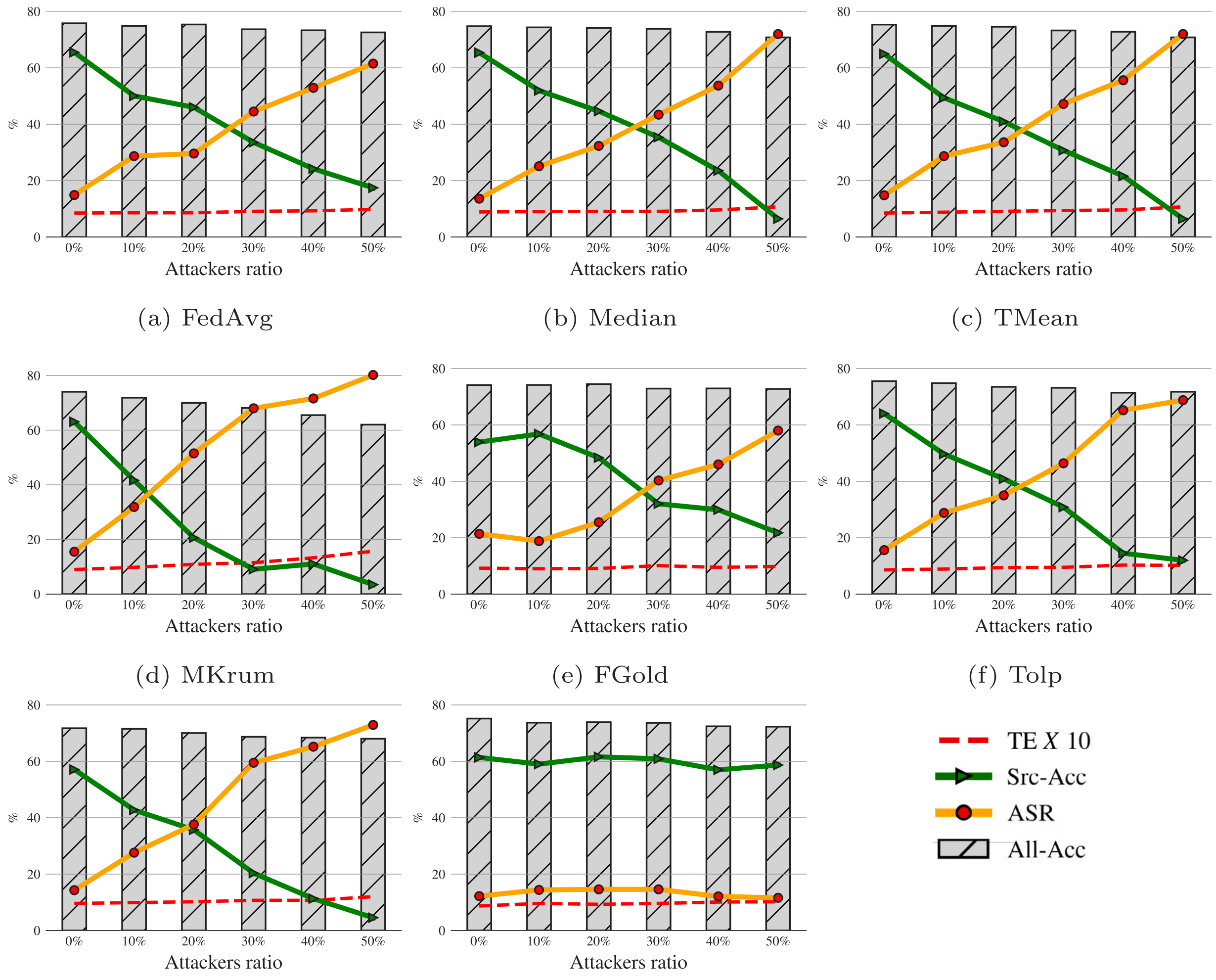
Benchmark	*FedAvg (NA)*	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	LFighter
MNIST-CNN-non-IID	*0.081*	0.275	0.433	0.426	0.437	1.35	0.098	0.103	**0.097**
CIFAR10-ResNet18-non-IID	*0.142*	0.281	0.278	0.277	0.45	0.261	0.507	0.582	**0.152**
IMDB-BiLSTM	*0.102*	0.352	0.266	0.267	NaN	0.095	**0.094**	0.098	**0.094**



(a) CIFAR10-ResNet18-non-IID

(b) IMDB-BiLSTM

Fig. 9. Evolution of the source class accuracy with 40% attackers ratio.



(a) FedAvg

(b) Median

(c) TMean

(d) MKrum

(e) FGold

(f) Tolp

(g) FLAME

(h) LFighter

Fig. 10. Robustness against the label-flipping attack with the CIFAR10-Mild benchmark.

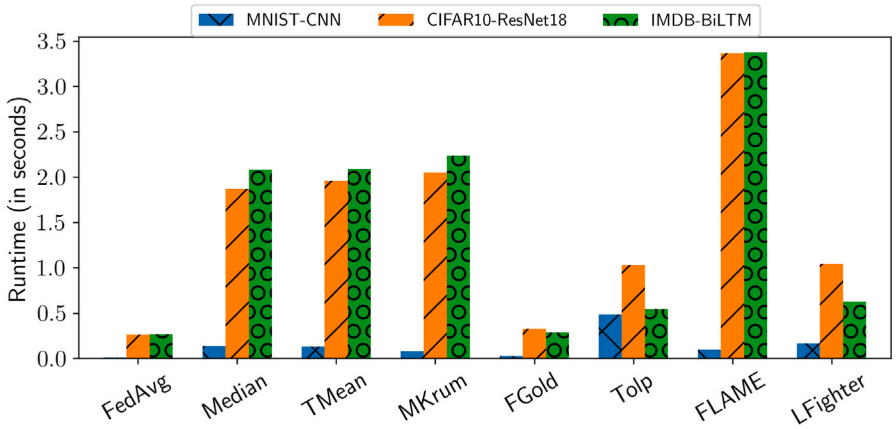


Fig. 11. Runtime overhead in seconds.

Table 5
Robustness to multi-LF attack. Best score is in bold. Second best score is underlined. NA denotes no attack is performed.

Metric	*FedAvg (NA)*	FedAvg	Median	Tmean	MKrum	FGold	Tolp	FLAME	LFighter
TE	*0.85*	1.47	1.17	1.14	1.54	1.16	1.09	1.19	**0.94**
All-Acc%	*75.81*	62.61	62.72	64.30	48.88	63.03	66.72	61.81	**72.32**

7. Conclusions and future work

We have conducted a comprehensive analysis of the label-flipping attack behavior. From this, we have observed that the contradictory objectives of attackers and honest peers turn the parameter gradients connected to the source and target class neurons into robust discriminative features to detect the attack. Accordingly, we have presented LFighter, a novel defense that dynamically extracts those gradients and uses them as input features to an adapted clustering method in order to detect attackers. The empirical results we report show that our defense is effective and simultaneously achieves better test error, overall accuracy, source class accuracy, and attack success rate than related works. Moreover, unlike the related works, our defense was still robust to significantly large ratios of attackers.

As future work, we intend to investigate the behavior of other effective untargeted and targeted poisoning attacks against federated learning, and then develop a comprehensive defense against them. Also, we plan to leverage deep multi-view techniques (Du, Liu, Chen, Yang, & Wang, 2021; Fang et al., 2023; Wang, Chen, Du, & Lin, 2021) to extract multi-view features from local updates in order to enhance clustering results of FL updates.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Josep Domingo-Ferrer reports financial support was provided by European Commission. David Sanchez reports financial support was provided by European Commission. Najeeb Jebreel reports financial support was provided by Government of Catalonia. Josep Domingo-Ferrer reports financial support was provided by Government of Catalonia. David Sanchez reports financial support was provided by Government of Catalonia. Josep Domingo-Ferrer reports a relationship with Rovira i Virgili University that includes: employment.

Data availability

All data and code are publicly available on a GitHub repository. We provide the link to the repository in the manuscript.

Acknowledgments

This research was funded by the European Commission (projects H2020-871042 "SoBigData++" and H2020-101006879 "MobiData Lab"), the Government of Catalonia (ICREA Acadèmia Prizes to J.Domingo-Ferrer and to D. Sánchez, FI grant to N. Jebreel), MCIN/AEI/ 10.13039/ 501100011033 and "ERDF A way of making Europe" under grant PID2021-123637NB-I00 "CURLING", and INCIBE-NextGeneration EU (project "HERMES" and INCIBE-URV cybersecurity chair). The authors are with the UNESCO Chair in Data Privacy, but the views in this paper are their own and are not necessarily shared by UNESCO.

Appendix A. Impact of feature similarity between the source and the target classes

We report here the impact of the feature similarity between the source and the target class samples on the performance of both the LF attack and our defense. We used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40%. The following three attack scenarios were implemented:

1. Source and target classes with high feature similarity: the attackers flipped the examples with the label *Dog* to *Cat*.
2. Source and target classes with moderate feature similarity: the attackers flipped the examples with the label *Truck* to *Car*.
3. Source and target classes with high feature dissimilarity: the attackers flipped the examples with the label *Frog* to *Plane*.

Results reported in Table A.1 reveal that the higher the feature similarity between the two classes, the more effective the attack is. This is because when the features of the two classes are more similar, their decision boundaries overlap more, making the attacker's task easier. This also explains why ASR was about 15% with the *Dog-Cat* scenario in the absence of attacks. Nevertheless, our defense was effective in all scenarios and achieved a similar ASR to FedAvg when no attack was present.

Appendix B. Impact of the local data distribution

We studied the impact of the local data distribution on the performance of our defense using the CIFAR10-ResNet18-non-IID benchmark

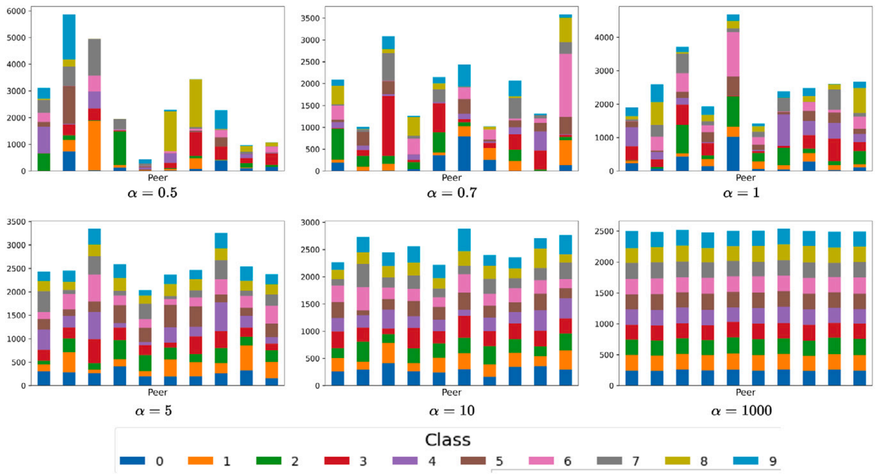


Fig. B.1. Local data distributions of 10 peers generated using the Dirichlet distribution with different α values using the CIFAR10 training data.

Table A.1
Impact of feature similarity between the source and target class.

Metric	Method	Source-Target		
		Dog-Cat	Truck-Car	Frog-Plane
All-Acc%	*FedAvg (NA)*	*75.81*	*75.81*	*75.81*
	FedAvg	73.32	73.78	75.21
	LFighter	72.82	72.90	74.62
Src-Acc%	*FedAvg (NA)*	*65.48*	*81.62*	*87.40*
	FedAvg	24.22	56.75	75.10
	LFighter	56.98	78.73	81.80
ASR%	*FedAvg (NA)*	*14.90*	*8.41*	*0.50*
	FedAvg	52.93	29.90	6.10
	LFighter	12.10	8.50	0.70

Table B.1
Impact of the non-IIDness of the local data.

Metric	Method	α					
		0.5	0.7	1	5	10	1 K
All-Acc%	*FedAvg (NA)*	*74.22*	*74.57*	*75.81*	*76.41*	*76.44*	*77.87*
	FedAvg	73.06	72.56	73.32	73.06	74.72	75.23
	LFighter	72.07	72.34	72.82	74.03	74.71	75.31
Src-Acc%	*FedAvg (NA)*	*60.61*	*61.72*	*65.48*	*64.81*	*65.23*	*67.13*
	FedAvg	21.32	23.63	24.22	21.32	24.22	24.44
	LFighter	54.14	56.83	56.98	59.24	61.32	63.15
ASR%	*FedAvg (NA)*	*10.91*	*12.54*	*14.92*	*16.63*	*15.32*	*14.57*
	FedAvg	57.30	54.62	52.93	57.33	54.14	54.18
	LFighter	14.81	14.62	12.10	12.11	11.93	11.97

with a ratio of attackers = 40%. We generated local data for 20 peers with different degrees of non-IIDness using different values of the Dirichlet distribution parameter $\alpha \in \{0.5, 0.7, 1, 5, 10, 1000\}$. Fig. B.1 shows the per quantity and per class data distribution of the first 10 peers. We selected just 10 random peers to make it easier to read the figure. It can be seen that lower values of α correspond to higher non-IIDness.

Table B.1 shows that high non-IIDness of the peers' local data negatively impacted the overall accuracy of the final global model. In fact, the non-IIDness of the local data of peers is a challenge to the performance and convergence of FL, as it has been observed in FL studies such as (Zhao et al., 2018). It can also be seen that higher non-IIDness (lower α) of the local data distribution corresponds to higher LF ASRs. This is because the larger divergence in local updates caused by high non-IIDness makes it more challenging to differentiate between good and poisoned updates. Nevertheless, it appears that our defense was able to effectively mitigate the impact of the attack without significantly reducing the accuracy of the main task.

Appendix C. Impact of the local learning rate

The learning rate is one of the most important hyper-parameters affecting the performance of DL models in both centralized and federated learning. Choosing an appropriate value for the learning rate depends on several factors, such as model dimensionality and training data complexity (Roy, 1994). We used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40% and different learning rates.

Table C.1 shows that larger learning rate values (up to 0.05) led to better global model performance. We can also note that with larger learning rates, which resulted in better global model performance, our method defended better against the attack.

Table C.1
Impact of the learning rate.

Metric	Method	Learning rate				
		0.0001	0.001	0.01	0.05	0.1
All-Acc%	*FedAvg (NA)*	*55.24*	*64.75*	*75.81*	*76.77*	*74.94*
	FedAvg	54.23	63.31	73.32	73.81	71.56
	LFighter	53.79	62.81	72.82	73.82	72.84
Src-Acc%	*FedAvg (NA)*	*46.72*	*56.13*	*65.48*	*64.52*	*64.72*
	FedAvg	17.93	19.91	24.22	19.64	8.03
	LFighter	39.74	49.33	56.98	60.65	60.01
ASR%	*FedAvg (NA)*	*19.41*	*17.14*	*14.92*	*13.12*	*11.41*
	FedAvg	47.42	49.42	52.93	58.70	69.15
	LFighter	21.31	19.04	12.10	9.53	10.40

Appendix D. Impact of the local batch size

The batch size determines how many training examples the model sees at once during training. Larger batch sizes can provide computational speed-ups but usually lead to poor generalization. The reason for this is not fully understood in the field (Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2016). We studied the impact of the local batch size on the performance of our defense using the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40% and different batch sizes.

Table D.1 shows that smaller batch sizes correspond to better global model performance in the used benchmark. Also, as long as the global performance was good, our method defended effectively against the attack.

Table D.1
Impact of the local batch size.

Metric	Method	Local batch size			
		16	32	64	128
All-Acc%	*FedAvg (NA)*	*77.03*	*75.81*	*72.58*	*65.70*
	FedAvg	74.76	73.32	69.41	64.35
	LFighter	74.60	72.82	70.42	66.46
Src-Acc%	*FedAvg (NA)*	*64.70*	*65.48*	*60.03*	*56.93*
	FedAvg	25.70	24.22	19.32	20.11
	LFighter	60.72	56.98	56.43	47.76
ASR%	*FedAvg (NA)*	*14.10*	*14.92*	*18.81*	*17.61*
	FedAvg	51.70	52.93	56.80	53.84
	LFighter	12.21	12.10	12.81	19.10

Table E.1
Comparison of the computational overhead on the server.

Framework	Computational overhead
LFighter	$\mathcal{O}(m^2 d_i)$
Median	$\mathcal{O}(m \log md)$
TMean	$\mathcal{O}(m \log md)$
MKrum	$\mathcal{O}(m^2 d)$
FGold	$\mathcal{O}(m^2 d_L)$
Tolp	$\mathcal{O}(m^3 d_i)$
FLAME	$\mathcal{O}(m^2 d)$

Appendix E. Computational overhead of LFighter

We analyze the computational overhead introduced by LFighter versus standard FL on the server side for one iteration, and we provide a comparison with related works.

To quantify the computational overhead, we use d, d_L, and d_i to represent the dimensionalities of the entire model, the last layer, and the parameters connected to the $i^t h$ neuron in the last layer, respectively.

It is important to note that the last layer of a DNN model usually has a much lower dimensionality and contains a very small number of parameters compared to the number of parameters of the whole model. Moreover, the number of parameters connected to one output neuron is even smaller than that of the last layer. As an example, in our experiments, we use the ResNet18 model, which contains approximately 11 million parameters. However, its last layer only has 5130 parameters, while one neuron has only 513 connected parameters.

The computation overhead on the server side can be broken down into the following parts: (1) computing the last-layer gradients from m local updates, which costs $\mathcal{O}(md_L)$; (2) computing the per-neuron magnitudes for $|C|$ output neurons and m peers, which costs $\mathcal{O}(m|C|d_i)$; (3) aggregating the computed per-neuron magnitudes for m peers, which costs $\mathcal{O}(m|C|)$; (4) identifying the neurons with the highest two magnitudes from the aggregated magnitudes vector, which costs $\mathcal{O}(|C| \log |C|)$; (5) using k-means (with $k = 2$) to cluster m relevant gradients, which costs $\mathcal{O}(md_i)$; (6) computing the density inverse of the two formed clusters cl_1, cl_2, which costs $\mathcal{O}(m^2 d_i)$.

Since the quadratic term dominates as the number of peers increases, the computational overhead of the server can be expressed as $\mathcal{O}(m^2 d_i)$.

Table E.1 presents a comparison of the computation overhead of LFighter with that of the following methods: the median (Yin et al., 2018), the trimmed mean (TMean) (Yin et al., 2018), multi-Krum (MKrum) (Blanchard et al., 2017), FoolsGold (FGold) (Fung et al., 2020), Tolp (Tolpegin et al., 2020) and FLAME (Nguyen et al., 2022).

Considering the effectiveness of LFighter against the LF attack, it strikes a better balance between the attack detection and the computational overhead, making it more practical for large-scale FL systems.

Appendix F. Privacy preservation of LFighter

Although FL reduces privacy risks by not sharing the peers' local data with the server, an *honest-but-curious* server might still infer sensitive information from the peers' local updates, as reported in several studies (Ganju, Wang, Yang, Gunter, & Borisov, 2018; Geiping, Bauermeister, Dröge, & Moeller, 2020; Hitaj, Ateniese, & Perez-Cruz, 2017; Melis, Song, De Cristofaro, & Shmatikov, 2019; Nasr et al., 2019). Fortunately, our defense can be used in conjunction with any existing FL privacy defense that enables the server to analyze individual updates, such as Domingo-Ferrer, Blanco-Justicia, Manjón, and Sánchez (2021), Ma et al. (2022). Specifically, Domingo-Ferrer et al. (2021) proposes a co-utile FL architecture to solve the accuracy-privacy-security conflict by breaking the link between local updates and their originators before sending the updates to the server. On the other hand, Ma et al. (2022) integrates local differential privacy on the peer's side with an intermediate shuffler between the peers and the server to allow countering poisoning attacks while protecting the privacy of peers.

References

Awan, S., Luo, B., & Li, F. (2021). CONTRA: Defending against poisoning attacks in federated learning. In *European symposium on research in computer security* (pp. 455–475). Springer.

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., & Shmatikov, V. (2020). How to backdoor federated learning. In *International conference on artificial intelligence and statistics* (pp. 2938–2948). PMLR.

Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., & Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 118–128).

Blanco-Justicia, A., Domingo-Ferrer, J., Martínez, S., Sánchez, D., Flanagan, A., & Tan, K. E. (2021). Achieving security and privacy in federated learning systems: Survey, research challenges and future directions. *Engineering Applications of Artificial Intelligence*, *106*, Article 104468.

Blanco-Justicia, A., Sánchez, D., Domingo-Ferrer, J., & Muralidhar, K. (2023). A critical review on the use (and misuse) of differential privacy in machine learning. *ACM Computing Surveys*, *58*(8), 160:1–160:16.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., et al. (2019). Towards federated learning at scale: System design. arXiv preprint arXiv:1902.01046.

Campello, R. J., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 160–172). Springer.

Chang, H., Shejwalkar, V., Shokri, R., & Houmansadr, A. (2019). Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. arXiv preprint arXiv:1912.11279.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., & De Freitas, N. (2013). Predicting parameters in deep learning. arXiv preprint arXiv:1306.0543.

Domingo-Ferrer, J., Blanco-Justicia, A., Manjón, J., & Sánchez, D. (2021). Secure and privacy-preserving federated learning via co-utility. *IEEE Internet of Things Journal*, *9*(5), 3988–4000.

Domingo-Ferrer, J., Blanco-Justicia, A., Sánchez, D., & Jebreel, N. (2020). Co-utile peer-to-peer decentralized computing. In *2020 20th IEEE/ACM international symposium on cluster, cloud and internet computing* (pp. 31–40). IEEE.

Du, S., Liu, Z., Chen, Z., Yang, W., & Wang, S. (2021). Differentiable bi-sparse multi-view co-clustering. *IEEE Transactions on Signal Processing*, *69*, 4623–4636.

Fang, M., Cao, X., Jia, J., & Gong, N. (2020). Local model poisoning attacks to Byzantine-robust federated learning. In *29th USENIX security symposium* (pp. 1605–1622).

Fang, Z., Du, S., Lin, X., Yang, J., Wang, S., & Shi, Y. (2023). DBO-Net: Differentiable bi-level optimization network for multi-view clustering. *Information Sciences*, *626*, 572–585.

Fung, C., Yoon, C. J., & Beschastnikh, I. (2020). The limitations of federated learning in sybil settings. In *23rd International symposium on research in attacks, intrusions and defenses* (pp. 301–316).

Ganju, K., Wang, Q., Yang, W., Gunter, C. A., & Borisov, N. (2018). Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security* (pp. 619–633).

Geiping, J., Bauermeister, H., Dröge, H., & Moeller, M. (2020). Inverting gradients–How easy is it to break privacy in federated learning? arXiv preprint arXiv:2003.14053.

Haffar, R., Sanchez, D., & Domingo-Ferrer, J. (2023). Explaining predictions and attacks in federated learning via random forests. *Applied Intelligence*, *53*(1), 169–185.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., et al. (2018). Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *28*(1), 100–108.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hitaj, B., Ateniese, G., & Perez-Cruz, F. (2017). Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security* (pp. 603–618).

Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., & Li, B. (2018). Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE symposium on security and privacy* (pp. 19–35). IEEE.

Jebreel, N., Blanco-Justicia, A., Sánchez, D., & Domingo-Ferrer, J. (2020). Efficient detection of Byzantine attacks in federated learning using last layer biases. In *International conference on modeling decisions for artificial intelligence* (pp. 154–165). Springer.

Jebreel, N. M., & Domingo-Ferrer, J. (2023). FL-defender: Combating targeted attacks in federated learning. *Knowledge-Based Systems*, *260*, Article 110178.

Jebreel, N. M., Domingo-Ferrer, J., Blanco-Justicia, A., & Sánchez, D. (2022). Enhanced security and privacy via fragmented federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, *14*(1–2), 1–210.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S., & Suresh, A. T. (2020). Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning* (pp. 5132–5143). PMLR.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.

Konečnỳ, J., McMahan, B., & Ramage, D. (2015). Federated optimization: Distributed optimization beyond the datacenter. arXiv preprint arXiv:1511.03575.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* [Master's thesis], University of Toronto.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Lai, Y. C., Lin, J. Y., Lin, Y. D., Hwang, R. H., Lin, P. C., Wu, H. K., et al. (2023). Two-phase defense against poisoning attacks on federated learning-based intrusion detection. *Computers & Security*, *129*, Article 103205.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision* (pp. 319–345). Springer.

Li, S., Ngai, E., Ye, F., & Voigt, T. (2021). Auto-weighted robust federated learning with corrupted data sources. arXiv preprint arXiv:2101.05880.

Li, X., Qu, Z., Zhao, S., Tang, B., Lu, Z., & Liu, Y. (2021). Lomar: A local defense against poisoning attack on federated learning. *IEEE Transactions on Dependable and Secure Computing*.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2018). Federated optimization in heterogeneous networks. arXiv preprint arXiv:1812.06127.

Li, D., Wong, W. E., Wang, W., Yao, Y., & Chau, M. (2021). Detection and mitigation of label-flipping attacks in federated learning systems with KPCA and K-means. In *2021 8th International conference on dependable systems and their applications* (pp. 551–559). IEEE.

Ma, X., Sun, X., Wu, Y., Liu, Z., Chen, X., & Dong, C. (2022). Differentially private Byzantine-robust federated learning. *IEEE Transactions on Parallel and Distributed Systems*, *33*(12), 3690–3701.

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision* (pp. 116–131).

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 142–150).

Manna, A., Kasyap, H., & Tripathy, S. (2021). Moat: Model agnostic defense against targeted poisoning attacks in federated learning. In *Information and communications security: 23rd International conference, ICICS 2021, Chongqing, China, November 19-21, 2021, proceedings, Part I 23* (pp. 38–55). Springer.

McMahan, B., Moore, E., Ramage, D., Hampson, S., & Aguera-Arcas, B. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (pp. 1273–1282). PMLR.

Melis, L., Song, C., De Cristofaro, E., & Shmatikov, V. (2019). Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy* (pp. 691–706). IEEE.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys*, *54*(3), 1–40.

Minka, T. (2000). *Estimating a Dirichlet distribution*. Technical Report, MIT.

Nasr, M., Shokri, R., & Houmansadr, A. (2019). Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy* (pp. 739–753). IEEE.

Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., et al. (2008). Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st usenix workshop on large-scale exploits and emergent threats* (pp. 16–17).

Nguyen, T. D., Rieger, P., Chen, H., Yalame, H., Möllering, H., Fereidooni, H., et al. (2022). FLAME: Taming backdoors in federated learning. In *31st USENIX security symposium* (pp. 1415–1432).

Nguyen, T. D., Rieger, P., Yalame, H., Möllering, H., Fereidooni, H., Marchal, S., et al. (2021). FLGUARD: Secure and private federated learning. arXiv preprint arXiv:2101.02281.

Qayyum, A., Janjua, M. U., & Qadir, J. (2022). Making federated learning robust to adversarial attacks by learning data and model association. *Computers & Security*, *121*, Article 102827.

Roy, S. (1994). Factors influencing the choice of a learning rate for a backpropagation neural network. In *Proceedings of 1994 IEEE international conference on neural networks, vol. 1* (pp. 503–507). IEEE.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Shejwalkar, V., Houmansadr, A., Kairouz, P., & Ramage, D. (2021). Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. arXiv preprint arXiv:2108.10241.

Shen, S., Tople, S., & Saxena, P. (2016). Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd annual conference on computer security applications* (pp. 508–519).

Siegel, A. F. (1982). Robust regression using repeated medians. *Biometrika*, *69*(1), 242–244.

Steinhardt, J., Koh, P. W., & Liang, P. (2017). Certified defenses for data poisoning attacks. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 3520–3532).

Sun, Z., Kairouz, P., Suresh, A. T., & McMahan, H. B. (2019). Can you really backdoor federated learning? arXiv preprint arXiv:1911.07963.

Tahmasebian, F., Lou, J., & Xiong, L. (2022). Robustfed: A truth inference approach for robust federated learning. In *Proceedings of the 31st ACM international conference on information & knowledge management* (pp. 1868–1877).

Tolpegin, V., Truex, S., Gursoy, M. E., & Liu, L. (2020). Data poisoning attacks against federated learning systems. In *European symposium on research in computer security* (pp. 480–501). Springer.

Wang, S., Chen, Z., Du, S., & Lin, Z. (2021). Learning deep sparse regularizers with applications to multi-view clustering and semi-supervised classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(9), 5042–5055.

Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., & Chen, M. (2019). In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, *33*(5), 156–165.

Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J. y., et al. (2020). Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, *33*, 16070–16084.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *2*(1–3), 37–52.

Wu, Z., Ling, Q., Chen, T., & Giannakis, G. B. (2020). Federated variance-reduced stochastic gradient descent with robustness to Byzantine attacks. *IEEE Transactions on Signal Processing*, *68*, 4583–4596.

Yin, D., Chen, Y., Kannan, R., & Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. In *International conference on machine learning* (pp. 5650–5659). PMLR.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., & Chandra, V. (2018). Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.