Eduard Josep Bel Ribes

LEVERAGING INTER- AND INTRA-CLASS DISTANCES FOR POISONING ATTACKS

MASTER'S THESIS

Directed by Dr. Alberto Blanco Justicia

Master's Degree in Computer Security Engineering and Artificial Intelligence

UNIVERSITAT ROVIRA i VIRGILI

Tarragona
2023

Acknowledgements

I want to thank Alberto for being so patient during the course of this thesis. Also for having taught me so much during this time.

I also want to thank my mother for her support and for always being there for me.

I want to thank my father for having taught me so many things that I am passionate about now.

Finally, I want to thank Toni and Natalia for giving me their support during difficult times.

Abstract

In the interconnected world we live in, Artificial Intelligence (AI) and Machine Learning (ML) have revolutionised our interactions with technology. Among emerging paradigms, Federated Learning (FL) is a new approach to train ML models in a decentralised way. FL allows ML models to obtain responses from users' data without compromising their privacy, making it essential for applications such as predictive text keyboards, speech recognition systems, and even disease diagnostic models.

However, the intrinsic decentralisation of FL also exposes it to security vulnerabilities. This research is motivated by the need to understand and address these vulnerabilities as FL is increasingly integrated into real-world applications, including critical systems such as autonomous driving vehicles.

The main objective of this study is to investigate the vulnerabilities faced by FL systems and identify strategies to effectively mitigate possible attacks. Specifically, we explore the feasibility of intelligent label-flipping techniques compared to brute force methods when attacking FL systems. Our goal is to determine whether a strategic selection of samples for label-flipping can produce more successful attacks than indiscriminate label-flipping.

In this thesis, we have conducted experiments on label-flipping attacks and can draw two key conclusions. First, we found that the effectiveness of label-flipping attacks increases as the number of samples with flipped labels rises, particularly in scenarios with numerous attackers and weak defences. Second, our proposed stealthier attacks exhibit greater resilience against defence mechanisms compared to the standard attack.

Resumen

En el mundo interconectado en el que vivimos, la Inteligencia Artificial (IA) y el Aprendizaje Automático (AA) han revolucionado nuestras interacciones con la tecnología. Entre los paradigmas emergentes, el Aprendizaje Federado (AF) es un nuevo enfoque para el entrenamiento de modelos de IA de forma descentralizada. El AF permite que los modelos de AA puedan obtener respuestas de los datos de los usuarios sin comprometer la privacidad de éstos, lo que lo hace esencial para aplicaciones como los teclados de texto predictivo, los sistemas de reconocimiento de voz, e incluso en modelos de diagnóstico de enfermedades.

Sin embargo, la descentralización intrínseca del AF también le expone a vulnerabilidades de seguridad. Esta investigación se ve motivada en la necesidad de comprender y abordar estas vulnerabilidades ya que el AF se integra cada vez más en aplicaciones del mundo real, incluidos sistemas críticos como los vehículos con conducción autónoma.

El objetivo principal de este estudio es investigar las vulnerabilidades a las que se enfrentan los sistemas de AF e identificar estrategias para mitigar posibles ataques de forma efectiva. Específicamente, exploramos la viabilidad de técnicas inteligentes de manipulación de etiquetas en comparación con los métodos de fuerza bruta cuando se atacan sistemas de AF. Nuestro objetivo es determinar si una selección estratégica de muestras para la manipulación de etiquetas puede producir ataques más exitosos que la manipulación indiscriminada de etiquetas.

En esta tesis, hemos realizado experimentos sobre ataques de manipulación de etiquetas y podemos sacar dos conclusiones clave. Primero, descubrimos que la efectividad de los ataques de manipulación de etiquetas aumenta a medida que aumenta el número de muestras con etiquetas cambiadas, particularmente en escenarios con numerosos atacantes y defensas débiles. En segundo lugar, nuestros ataques propuestos, que son más sigilosos, muestran una mayor resistencia contra los mecanismos de defensa en comparación con el ataque estándar.

Resum

En el món interconnectat en el que vivim, la Intel·ligència Artificial (IA) i l'Aprenentatge Automàtic (AA) han revolucionat les nostres interaccions amb la tecnologia. Entre els paradigmes emergents, l'Aprenentatge Federat (AF) és un nou enfocament per a l'entrenament de models d'IA de forma descentralitzada. L'AF permet que els models d'AA puguin obtenir respostes de les dades dels usuaris sense comprometre la privacitat d'aquests, la qual cosa el fa essencial per a aplicacions com els teclats de text predictiu, els sistemes de reconeixement de veu, i fins i tot en models de diagnòstic de malalties.

No obstant això, la descentralització intrínseca de l'AF també l'exposa a vulnerabilitats de seguretat. Aquesta investigació es veu motivada en la necessitat de comprendre i abordar aquestes vulnerabilitats ja que l'AF s'integra cada vegada més en aplicacions del món real, inclosos sistemes crítics com els vehicles amb conducció autònoma.

L'objectiu principal d'aquest estudi és investigar les vulnerabilitats a les quals s'enfronten els sistemes d'AF i identificar estratègies per mitigar possibles atacs de forma efectiva. Específicament, explorem la viabilitat de tècniques intel·ligents de manipulació d'etiquetes en comparació amb els mètodes de força bruta quan s'ataquen sistemes d'AF. El nostre objectiu és determinar si una selecció estratègica de mostres per a la manipulació d'etiquetes pot produir atacs més reeixits que la manipulació indiscriminada d'etiquetes.

En aquesta tesi, hem realitzat experiments sobre atacs de manipulació d'etiquetes i podem treure dues conclusions clau. Primer, descobrim que l'efectivitat dels atacs de manipulació d'etiquetes augmenta a mesura que augmenta el nombre de mostres amb etiquetes canviades, particularment en escenaris amb nombrosos atacants i defenses febles. En segon lloc, els nostres atacs proposats, que són més sigilosos, mostren una major resistència contra els mecanismes de defensa en comparació amb l'atac estàndard.

Contents

List of code snippets

List of Figures

List of Tables

1 Introduction

In today's age of information and connectivity, advances in Artificial Intelligence (AI) and Machine Learning (ML) have transformed the way users interact with technology and process data.

Among the emerging paradigms in the field of ML, Federated Learning (FL) appeared as an innovative approach to train AI models in a distributed and decentralized environment.

In the last decade, ML has revolutionized the way in which we face complex problems in different areas, from computer vision to natural language processing. The last two years have been filled with news about promising new paradigms of image generation, classification, chatbots, and speech recognition, among other AI-powered systems. As time goes by, the applications of AI are becoming more present in our daily lives.

We can find examples of applications using FL in examples such as the text predictive keyboard that we can find on our mobile phones (Google's Android Keyboard [1]). We also find FL in Apple's assistant Siri voice recognition. This technology helps distinguish whether it is the main user of the smartphone saying "Hey, Siri", or another iPhone user attempting to activate Siri on their phone. As a final example, FL is used in more complex applications such as autonomous driving systems. In all three cases, the use of FL allows the machine learning models to be trained with the users' data without them having to share their data with third parties. In the case of Google's predictive keyboard, the model is trained with the users' data locally on the device, while in the case of the autonomous driving systems, the users' data is used to train the model in a distributed way among the vehicles in the fleet.

Despite its advantages in terms of privacy, scalability and efficiency, FL presents significant challenges. One of them is its vulnerability to attacks: these attacks can exploit the distributed nature of the learning process, aiming to compromise integrity, confidentiality, and the model's efficiency. As seen in the real-world examples, if an attacker exploited a vulnerability of the autonomous driving system by modifying the action that the car takes when recognising a STOP sign, changing it to the action taken when recognising 120km/h sign, and causing the vehicle to accelerate at full capacity, it could lead to multiple car accidents. In the context of FL this is known as a label-flipping attack.

As Federated Learning systems are increasingly integrated in real-world applications, it becomes necessary to understand and address these challenges to guarantee a successful and secure deployment of this technology.

1.1 Motivation

Since the paper proposing FL got published in 2017, this new paradigm has been increasingly used over the years as we can see in Figure 1. With its increase in popularity and its implementation in sensitive applications such as autonomous driving systems, our concerns about the security of this technology also increase. From this concern, the motivation of detecting possible vulnerabilities so they can be addressed arises.

1.2 Objectives

The main theoretical objective of this thesis is to explore and examine in detail the kind of attacks that can be directed towards Federated Learning systems, as well as identifying

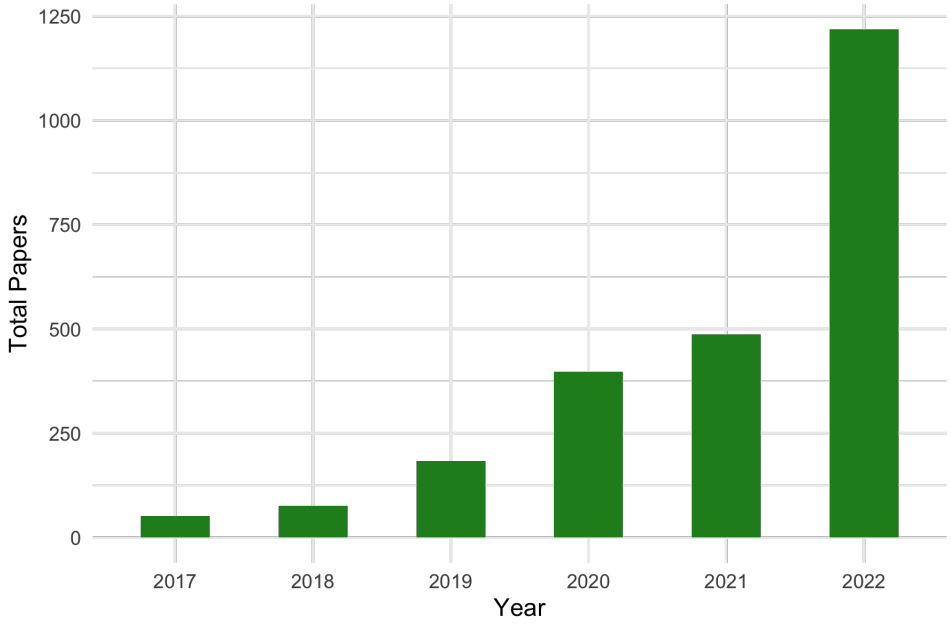

Figure 1: Published papers per year since FL was proposed. Source: *Web of Science*

strategies and solutions to mitigate these attacks.

The practical objective is to examine the effectiveness and implications of employing a sophisticated label-flipping technique compared to a straightforward approach when targeting a Federated Learning system. Specifically, the focus is on evaluating whether a more strategic and intelligent choice of samples to flip can lead to greater success for potential attackers compared to indiscriminately flipping all labels. This investigation represents an essential step towards comprehending the vulnerabilities and potential weak points within the Federated Learning paradigm. Furthermore, we aim to provide results on some of the most used aggregation rules in Federated Learning systems.

The research may be able to identify patterns and insights that conventional methods of attack might miss by examining the results of strategically manipulated label-flipping and comparing them with the brute-force method. The results of this thesis will be valuable in gaining a deeper understanding of the security implications of FL and to develop more robust and secure FL systems in case the conceived attacks succeed.

In summary, this thesis conducts a critical examination of the viability of using intelligent label-flipping techniques in comparison to a brute-force approach when attacking a Federated Learning system.

1.3 Outline

The remainder of this thesis organized in the following sections:

- Section 2, Background: Provides a brief overview of the Federated Learning paradigm, its advantages and disadvantages, and the different types of attacks that can be directed towards it.
- Section 3, Design of the proposed attacks: It describes the different attack hypotheses that are proposed.

- Section 4, Implementation: Explains the implementation of the new code.
- Section 5, Results: Presents the results obtained from the experiments conducted with the new code.
- Section 6, Conclusions: Summarizes the conclusions drawn from the results obtained in the experiments.

2 Background

In this Section, we present the essential background to understanding the details of model poisoning through label-flipping attacks on FL. We begin by presenting some relevant ideas required to understand its security landscape.

We examine the state of the art regarding adversarial attacks and defence mechanisms to get a practical perspective on security challenges. By exploring the landscape of model poisoning attacks, we will detail attacker tactics that manipulate model updates, potentially compromising the integrity of federated learning.

Finally, we explore the different (robust) aggregation techniques that are going to be employed on the practical side of this thesis.

2.1 Deep Neural Networks

Deep Neural Networks (DNNs) are a specific type of ML algorithms, similar to Artificial Neural Networks (ANN), but characterized by their incorporation of multiple hidden layers situated between the input and output layers. These hidden layers allow DNNs to acquire intricate and refined data representations during training. This inherent capability enhances their performance across diverse domains, encompassing healthcare [2], game playing [3], speech recognition [4], analysis of molecular structures and predicting chemical properties [5], and recommendation systems [6], among many others.

Formally, a DNN applies a series of nested non-linear operations between the inputs and a collection of tunable parameters, called weights and biases. Given an input vector x, a weight vector w, and a bias vector b, a neuron computes $o = \sigma(\sum_i x_i w_i + b)$, where $\sigma(\cdot)$ is a non-linear function, such as the sigmoid function, the hyperbolic tangent function, or the Rectified Linear Unit (which equals 0 for negative inputs and the identity for non-negative inputs). These outputs are forwarded to the following layers of the network until the output layer. The output of a DNN (or any ANN, for that matter) when presented with a data point is a vector of probabilities for that data point to correspond to each possible class. The highest probability is taken as the model's prediction. During training, an optimization algorithm, such as stochastic gradient descent, is used to minimize a loss function, that is, a function that scores how close are the predictions of the model with respect to the expected ones. These optimization algorithms alternate forward and backward passes in which the loss function is computed on some training examples and the expected results, the derivative of the loss function with respect to the model parameters is computed, and the weights and biases are adjusted according to the computed gradients and a configurable learning rate. These forward and backward passes are repeated until the model converges or reaches an acceptable level of loss or predictive power. Figure 2 shows a representation of a DNN that classifies images into classes of animals.

Specialized architectures, such as convolutional neural networks, recurrent neural networks, or transformers use special layers to solve particular problems, such as image recognition, time series analysis, or natural language processing.

For instance, a DNN trained to recognise plant species will examine the supplied image and determine whether the plant in the image is a member of a particular species. The user can then evaluate the outcomes and select the probabilities that the network should present (those that are higher than a certain threshold, etc.), resulting in the suggested

label.

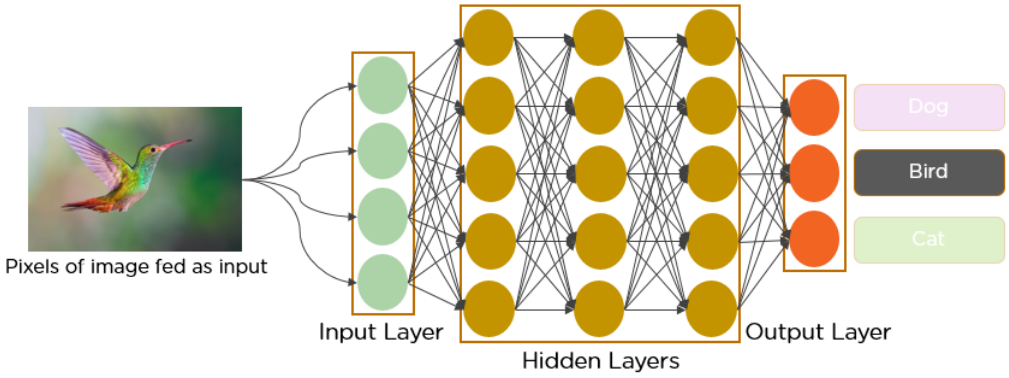

Figure 2: Deep Neural Network representation. Source: *Analytics Vidhya*

2.2 Federated Learning

Federated Learning, first proposed by McMahan et al. in 2017 [7], has become recognised as an innovative paradigm in the context of ML. In the age of distributed computing and data privacy concerns, FL offers an innovative approach of model training. By allowing ML models to be trained collaboratively across decentralised devices while protecting the confidentiality of specific data sources, it addresses the issue of centralised data ownership and the privacy implications it has.

The FL training procedure is an excellent representation of a distributed and privacy-protecting mechanism. Using their private data, participating devices or clients refine specific models in this scheme. The global model is used as the starting point for each client's training process and is initialised and distributed by a central server. Clients produce incremental model updates by iteratively optimising this global model through their local datasets. The central server receives these updates, aggregates them into a new global model (usually by just computing the average of the provided models), and distributes the new global model again to the clients. This method's elegance lies in its capacity to combine information from various data sources while protecting private data at the source.

Due to its collaborative and iterative structure, FL has special qualities that set it apart from conventional centralised learning paradigms. The training dynamics are made more complex by the presence of device-specific data distributions and a variety of client computational capabilities. Therefore, FL encompasses challenges that go beyond those of conventional ML, calling for the investigation of reliable communication protocols, secure aggregation techniques, and methods for dealing with potential adversarial threats.

Compared to conventional centralised ML approaches, FL offers several compelling advantages:

- The training computational load is effectively distributed across the participating peers' devices, which is particularly important for large-scale ML tasks.
- Joint training on various data sources improves model accuracy and yields more accurate insights for both peers and the central server.
- By eliminating the requirement to share local data with a central server, FL crucially protects individual privacy.

Due to the latter advantage, FL is particularly suitable for scenarios involving sensitive data, such as those in location-based services, voice assistants, healthcare, facial recognition, and voice assistants. Additionally, FL is extremely useful in scenarios where data processing and collection are limited by privacy protection laws like the General Data Protection Regulation (GDPR [8]) by the European Commission or the Spanish "*Ley Orgánica de Protección de Datos*" (LOPD).

2.3 Privacy and security issues of Federated Learning

Despite the numerous advantages that FL offers over centralised learning, the decentralised nature of this approach also creates vulnerabilities to security and privacy attacks. In fact, the distributed architecture that empowers FL, can increase the impact of these attacks, surpassing the risks associated with more conventional centralised learning [9, 10].

Numerous security and privacy issues can still affect FL. On the security front, the vulnerabilities include Byzantine attacks, that aim to obstruct model convergence, and poisoning attacks, that are deliberately designed to influence convergence in the wrong direction.

2.3.1 Privacy attacks

While FL works to prevent the direct sharing of private data, the process of exchanging local updates creates the possibility of sensitive information leakage to malicious actors.

The gradients computed on individual devices have the potential to unintentionally make adversaries aware of subtle aspects of the training data. Deep Learning models have an extraordinary capacity to retain information beyond what is strictly required for their primary task, and this tendency can unintentionally reveal unintended properties of the data they were trained on, opening the door to membership inference, attribute inference, and reconstruction attacks [11–14].

Local updates from peers reflect the understanding gained from their different training datasets. As a result, these updates have the unintentional potential to reveal personal information, such as class distributions, membership details, and inherent properties of the local training data. This gives adversaries the ability to infer private information without having any prior knowledge of the underlying data.

While these attacks are also possible in centralized ML, FL allows the central entity aggregating updates to access the individual updates in a white-box manner, which might increase the risks against the aforementioned attacks. Ensuring good model generalization [11] and using defences such as differential privacy [15] can limit the vulnerability to such attacks.

2.3.2 Poisoning attacks against Federated Learning

Given the distributed nature of FL, where the central server does not have access to the training data in order to assess their quality and does not have control over the behaviour of the peers, it is vulnerable to poisoning attacks, a type of attack designed to sabotage the learning process by introducing malicious data or model updates [16, 17]. These attacks within FL systems can be divided into two categories: untargeted and targeted.

Understanding the landscape of poisoning attacks against FL is crucial to safeguarding

the integrity of the learning process. During FL's training phase, both targeted and untargeted poisoning attacks can be executed, influencing either the local model or the local data. The act of injecting manipulated samples into the training dataset is known as "data poisoning attack," capable of introducing distortions by feeding the model with inaccurate or biased data. In contrast, model poisoning attacks involve the manipulation of model parameters during the local model training, either directly or indirectly.

2.3.3 Untargeted poisoning attacks

Untargeted poisoning attacks in the context of FL focus on degrading the model's overall performance rather than aiming for particular misclassifications [17]. Without following a predetermined pattern, these attacks introduce noise or perturbations into the training process. The result is a compromised global model with decreased prediction accuracy and reliability.

Byzantine attacks are a subset of untargeted attacks that involve malicious devices deliberately sending false updates to the central server during the model aggregation phase. In order to prevent the convergence of the global model from happening, these malicious devices act dishonestly by transmitting updated models that has been altered or corrupted. Unaware of the adversarial behaviour, the central server aggregates these updates, creating a distorted model that does not accurately reflect the underlying data.

Due to the hidden nature of the malicious actions, which can closely resemble legitimate participation, detecting Byzantine attacks can be challenging. Standard aggregation techniques might unintentionally include these contaminated updates, which would make the global model perform poorly on unobserved data.

2.3.4 Targeted poisoning attacks

Targeted poisoning attacks, on the other hand, have a specific goal: inducing the global model to misclassify a chosen set of samples into a target class chosen by the attacker. The specific targeted attack that holds relevance for this thesis is the label-flipping attack [18].

In a label-flipping attack, attackers employ their local dataset to carry out their poisoning strategy in the following way: for each instance in the dataset that originally belongs to an attacker-chosen source class, they change their label to an also attacker-chosen target one. These attackers then proceed to train their local models after manipulating their training data. The parameters used in this training process are the same as those provided by the central server. As a result, the model learns from these incorrectly labelled examples, which causes it to produce inaccurate predictions in the future when presented with images of a similar nature.

Consider a scenario for a medical diagnosis where an FL model is trained to distinguish between samples that are healthy and those that present some disease. A label-flipping attack could be carried out by an attacker by changing the labels of some healthy samples to read "diseased." The model then gains knowledge from these falsified data, misclassifying real healthy samples as diseased during inference. In real-world applications, incorrect diagnoses or treatment recommendations might have serious consequences.

2.3.5 Defences against poisoning attacks

The way for a malicious actor to compromise the integrity of the learning process is, as seen in the previous sections 2.3.3 and 2.3.4, to inject poisoned data on local model updates. The attacker must also overcome the combined influence of benign clients during the aggregation process in order to ensure a successful attack. This can be done in a number of ways, including:

- Applying scaling factors to boost the impact of their own updates.
- Colluding with other malicious clients, whether they are additional accounts linked to the same attacker or different attackers themselves.
- Applying a combination of the aforementioned strategies.

A useful strategy to prevent attacks of this nature would be, given that the server possesses the unique identifiers (ID) of peers participating in the current training round, and in the case that the server has access to public data of the same distribution, to compare the global model's accuracy with those of prior rounds. In rounds of salient disparities, the algorithm could store the IDs of participants from that specific round. In subsequent rounds where discrepancies occur, this stored list could be cross-referenced with the current round's IDs, allowing for the removal of IDs not actively participating in the ongoing round. This strategy would eventually facilitate the detection of a single malicious peer. Notably, this identification does not require to ban the detected malicious peers, as this might lead them to create new accounts to evade detection. For scenarios involving multiple malicious peers, an iterative process could take place, persisting until additional attackers are detected. To establish a trusted first global round for future comparisons, the server could initiate this round with the assurance of a lack of attackers at the chosen peers set.

A less drastic approach to blacklisting is using reputations, rewarding good updates and penalizing bad ones, and, during aggregation, weighting updates based on the peers' reputations. Thus, updates from suspicious peers carry less weight than those from trustworthy ones. The issue with this mechanism is that it assumes the central server has access to sufficient testing data for model evaluation. In a FL setting, this does not always hold true, which is why the mechanisms discussed below have been proposed.

The strategies for defending against poisoning attacks that have been suggested in the literature follow one of the following principles [9, 19]:

- Update Aggregation: In this method, local model updates are aggregated using methods that are robust to outliers. The impact of inaccurate updates on the final global model is reduced by using aggregation techniques that are resilient to extreme values. This ensures that the effects of potentially harmful updates are minimised, allowing to produce an aggregated model that is more reliable and accurate.
- Evaluation Metrics: The central idea of this approach is to evaluate the quality of local updates using evaluation metrics connected to the global model. The model aggregation process may exclude or penalise a local update if it negatively impacts a certain metric, such as accuracy.

- Update Clustering: In a different approach, updates are split into two clusters, with the smaller cluster being marked as potentially malicious and ignored during model learning. This idea helps filter out potentially harmful updates.
- Peers' Behaviour: This approach makes the assumption that malicious peers behave similarly, which makes their updates more similar than those of honest peers. In order to reduce the impact of potentially harmful updates, penalization is therefore based on the similarity between updates.
- Differential Privacy (DP): Using DP, each update parameter is altered by being clipped to a maximum threshold and then introducing calibrated random noise. As a result, there is a compromise between the ability of the aggregated model to perform its main task and the mitigation of potential attacks through added noise.

The strategies implemented in the base code that is used in this thesis [20] are:

- Federated Averaging (FedAvg) [7]: The standard aggregation method used in FL. It works by aggregating the local updates by averaging them. This method is not designed to counter poisoning attacks.
- Median [21]: This strategy involves collecting local model updates from participating devices and determining the median value for each parameter across these updates. After sorting the parameter values, the median strategy selects the middle value while ignoring extreme outliers. Since a single adversarial device cannot significantly impact the final aggregated model, this method makes the median aggregation resistant to malicious or inaccurate updates.
- Trimmed Mean (TMean) [21]: This method decreases the impact of outliers by excluding a certain percentage of extreme values. By reducing the impact of potentially malicious updates or noisy data from individual devices, this strategy improves the robustness of the aggregation process.
- Multi-Krum (MKrum) [22]: The updates chosen for aggregation using this method are the most agreeable ones. The algorithm isolates potentially malicious or abnormal updates by selecting a subset of updates with the highest consensus among participating devices. The MKrum strategy improves the robustness of aggregation against adversarial behaviour and data anomalies by focusing on the level of agreement among multiple devices.
- FoolsGold (FGold) [23]: By taking into account the similarity of their contributions, the method adapts the learning rate of clients. The central idea of this strategy is based on the notion that when a group of sybils[1] manipulates a shared model, their updates over the course of training will align towards a particular malicious objective, displaying a higher degree of similarity than expected. In FL, it serves as a robust defence mechanism against assaults planned by any number of sybils.

[1] A term used in computer security to describe a scenario in which a user or entity generates several different entities.

- Tolpegin [24]: The weights associated with the potentially targeted source class are examined by this method using Principal Component Analysis (PCA)[2]. Within those weight distributions, it selectively eliminates potential adversarial updates that deviate from the prevailing trend.
- FLAME [25]: Model clustering and weight clipping are two techniques FLAME employs to reduce the required noise infusion. This method accomplishes two goals through this method: successfully closing any potential adversarial backdoors while maintaining the aggregate model's desirable performance.
- LFighter [19]: This method effectively filters out updates that might be harmful before model aggregation by extracting gradients corresponding to potential source and target classes from local updates, clustering them. It's noteworthy that this proposed defence is robust across various data distributions and model dimensions.

[2]PCA is a statistical technique used to reduce the dimensionality of data while preserving its essential features.

3 Design of the proposed attacks

In this Section, we present the rationale and the design of new, smarter, and stealthier label-flipping attacks that are capable of bypassing defences in the literature. To push the limits of effectiveness and applicability, each hypothesis is motivated by a strategic vision that combines theoretical insights with practical considerations.

3.1 Standard label-flipping

This is the label-flipping algorithm that is used in the base code [20] and is usually considered in the literature.

As described above in Section 2.3.4, this is a very simple attack algorithm that changes the labels of all the samples of the source class to the target class's label prior to training the local model. Given that the attacker changes all source class samples, the training objective of the attackers clearly differs from honest peers, and thus their computed model updates will show inconsistencies with respect to the updates computed by honest peers.

As an example, a malicious peer who wants to misclassify $dogs$ as $cats$ will produce updates that push the probability of $dogs$ being classified as $dogs$ (the correct answer) $\mathbf{down}$ while pushing the probability of $dogs$ being classified as $cats$ $\mathbf{up}$. This is in direct contrast to the models computed with the correct data and labels and, as we will show in Section 5 it allows defence mechanisms to easily detect and discard or suppress such updates.

The next subsections present some modifications to the standard label-flipping attack aimed at reducing these big discrepancies between malicious and honest updates with the goal of making the poisoned models stealthier and more difficult to detect.

3.2 Entropy-based label-flipping

The first algorithm we propose is based on the entropy of the classification probability vectors output by the models. Entropy is a measure of the uncertainty of a random variable. In this case, the random variable is the label of the sample. Figure 3 shows the clear difference between classification probability vectors with high and low entropy. The samples that produce vectors with a high entropy are the ones that are more difficult to classify (possibly because they are outliers within their class or are at the boundaries of the classification function computed by the model), while the samples that produce probability vectors with a low entropy are the ones that are easier to classify.

We focus on samples that produce high-entropy classification probability vectors to flip their labels. As described above, label-flipping attacks produce updates that push classification probabilities up and down from source to target classes in a very different and discernible way compared with the updates produced by honest peers. Focusing on classification probability vectors that have high entropy (vectors that are $flatter$) makes the malicious pushing of probabilities smaller, and thus the computed updates are less noticeable (or more similar to the updates computed by honest peers).

Again, in Figure 3, we can see that an attacker does not have to introduce too many changes to the model to increase the classification probability of cat and reducing that of dog so that cat has the highest probability in the high entropy setting. On the other hand, attackers would need to introduce major modifications in the low entropy setting if they

wanted to misclassify *cat* as *dog*.

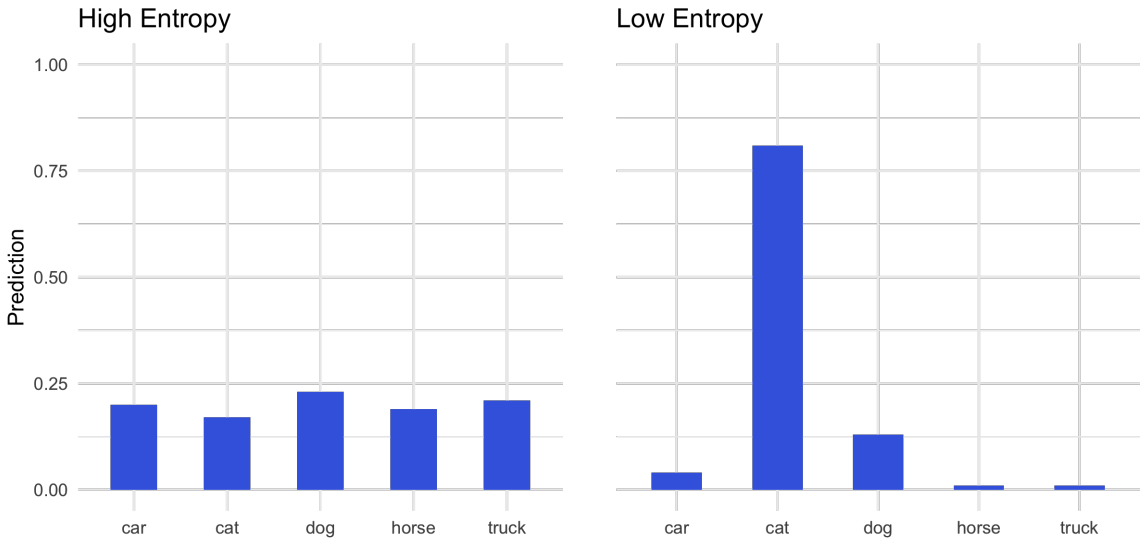

Figure 3: High vs. low entropy

3.3 Closeness-based label-flipping

The second proposed algorithm uses the concept of sample classification probability closeness, which serves as a metric to determine how closely a given sample aligns with classification as either the source class or the target class (how close the probabilities of belonging to either of those classes are). Figure 4 shows the clear difference between a high and low closeness. The samples with a high closeness are the ones that are likely to be classified as either the source or the target class, while the samples with a low closeness are the ones that are more likely to be classified as one of them.

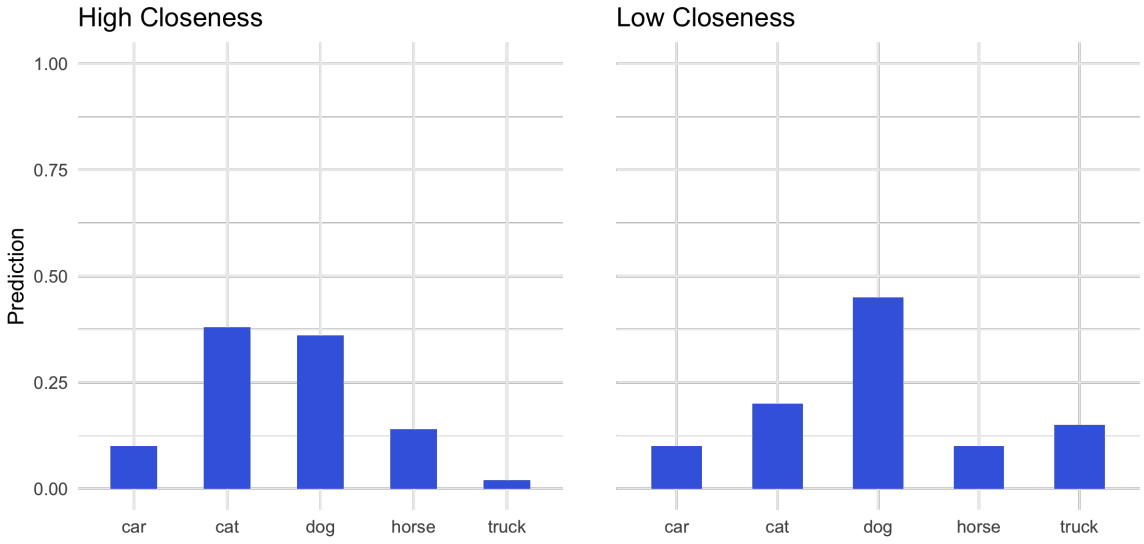

Figure 4: High vs. low closeness between classes *dog* and *cat*

Hence, our strategy involves flipping the labels of samples exhibiting high closeness, as they are more likely to suffer potential misclassification. Another reason to choose high

closeness is that the samples that have the higher closeness between the source and the target classes, may be also the ones more difficult to classify for the other peers. That could be, a dog with a short snout and hairy limbs that at first sight could be thought of as a cat. Thus, the modifications introduced by the attack on the model updates computed by the malicious peers are rather small and hence difficult to detect when comparing them to the updates computed by honest peers.

While in the entropy-based attack we targeted samples with $flat$ classification vectors (which might correspond to outliers), in this attack we do not focus on the entropy of the probability vectors but only on how close the source and target probabilities are (either if these two are much higher than for the rest of the classes, which would result also in a low-entropy vector).

3.4 Adaptive label-flipping

The idea behind this approach is to use the previous two algorithms to create a more sophisticated and effective attack. The difference resides in the fact that, instead of using a fixed number of samples to flip, we will flip a percentage of the samples depending on how far the global model has been refined, taking into account the global rounds. The rationale behind this is that, since the global model is initialised randomly, the first updates computed by peers are inherently more diverse and thus, more susceptible to attacks. The closer the model is to convergence, the more similar will be the updates of honest peers and stealthier attacks will be more beneficial, so fewer labels are flipped.

4 Implementation

In this Section, we describe the implementation of the proposed attacks, but we will first explore the base FL benchmarking framework in the GitHub repository [20], which we use to test our proposed attacks. We will discuss the structural elements that constitute the basis of our work, elucidating the positioning of the newly crafted code that serves the thesis' purpose. The goal is to demonstrate how the intricate nature of a typical FL environment is mirrored in the architecture of this implementation by drawing comparisons to real-world FL scenarios and the base code structure.

We will also dive deep into our label-flipping algorithms using a methodical approach that starts with a study of the base code dissected in Section 4.1. In order to lay the foundation for our attacks, this first step involves disassembling the existing components and exploring the rationale behind their design decisions.

The proposed attacks, which range from new ideas to small modifications, collectively aim to refine and enhance the overall performance of the label-flipping attacks. Through careful implementation and rigorous testing, we seek to demonstrate the efficacy of our approach and highlight the evolution of label-flipping algorithms into more sophisticated and effective tools for adversarial scenarios. The implemented code can be accessed on the project's GitHub repository [26].

For the upcoming results, which will be detailed in Section 5, our focus revolves around devising attacks that optimize four parameters:

- Test error (TE). Error resulting from the loss function used in training. The lower TE, the better.
- 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.
- Source class accuracy (Src-Acc). Number of the source class examples correctly predicted divided by the total number of the source class examples. The greater Src-Acc, the better.
- Attack success rate (ASR). Proportion of the source class examples incorrectly classified as the target class. Since we aim to attack the system, the higher ASR, the better.

4.1 Base code structure

The base code is implemented in Python and structured as follows:

- Python notebooks (.ipynb files): There are three notebooks, one for each dataset used in the developer's experiments. The included datasets are:
  - MNIST [27]: This dataset contains samples of handwritten digits. It consists of a training set of 60,000 samples, and a test set of 10,000 samples. Each sample is a 28x28 bit grayscale image, associated with a label from 10 classes. The task is to classify the images into their respective digit classes.

  - CIFAR-10 [28]: This dataset contains 60,000 32x32 bit colour images in 10 different classes. These classes encompass a diverse array of objects, including but not limited to animals, vehicles, and everyday items. The objective underlying this dataset is to accurately classify each image into its designated class.
  - IMDB [29]: This dataset contains 50,000 movie reviews from the Internet Movie Database. The task is to classify the reviews into positive or negative sentiment.

  The notebooks are used to run the experiments using the different aggregation functions commented in section 2.3.5 at user's will. This is done by importing the libraries and defining global variables in one executable section, and arranging the tests with different aggregation methods into separate sections. These notebooks are also used to visualize the results and checking the process.

- *experiment_federated.py*: This python file contains a sole function (*run_exp()*) that is called by one of the Python notebooks whenever we wish to start a new experiment with an aggregation function. This function merely prints by console information concerning the parameters used in the current experiment, initializes the FL environment, and calls a more complex function.

- *environment_federated.py*: This is the most complex file in the base code. It contains the *run_experiment()* function, which is the one called by the previous file. This function is responsible for the execution of the experiment, and it is where the whole FL setting is initialized and simulated. The file is divided into two main classes:

  - Peer: When a Peer object is created, the Peer class initializes all its variables, such as the peer's ID, its local data, etc. This class contains a single function, named *participant_update()*, which is called when a peer has to update its local model. This function is responsible for the training of the local model, and it is where, depending on the value of the parameter "*attack_type*", the execution of the attack is determined. The function returns the updated local model.
  - FL: This class is responsible for the initialization of the FL environment, from the most simple parameters such as the global rounds, to the more complex tasks such as creating the peers' instances and setting the global model up. It contains four functions:
    * *test()*: This function is used to test the model's accuracy.
    * *test_label_predictions()*: As the name suggests, this function is used to test the label predictions of the model received as a parameter. It is responsible for returning a list with the actual labels and another with the predicted ones in order to obtain the accuracy of the model.
    * *choose_peers()*: This function selects *n* random peers from the list of peers. The value of *n* is determined by the total amount of peers and the malicious rate, which is a floating-point variable.
    * *run_experiment()*: This is the function that simulates the whole FL environment. The function is built as follows:
      1. It begins by copying the global model into a local variable.
      2. After that, it iterates through a loop a total of "*global_rounds*" times. Inside this loop, for each global round, it selects the peers that will participate in the current round by calling the *choose_peers()* function,

it also reinitializes the utility tables (weights, local models, etc.) of the peers that will participate in the current round, and then, for each peer participating:

(a) It defines the peer as attacker or regular depending on the output of *choose_peers()*, calls the *participant_update()* function of the peer.

(b) It updates the utility tables of the peer with the new values.

3. After all peers have been updated, it aggregates the local models of the peers that participated in the current round by the selection of the aggregation function dependant on a series of conditional statements.
4. It updates the global model with the aggregated model and the process is repeated until the global rounds are completed.
5. The *test()* function is called to obtain the model's accuracy.
6. Finally, it returns the system's state by console.

The location for the attackers' code is in the *environment_federated.py* file. The exact placement for these functions is inside the *participant_update()* function, contained by the Peer class. This is because it is the single point of entry for the execution of the attack, and it is where the local model is updated.

4.2 Real-world FL vs. base code

In this section, we state the similarities between a real FL system and the base code described in the previous section, 4.1. As an aid as we navigate through the details of these two distinct environments, Figure 5 provides a diagram of an actual FL scenario to further illuminate this comparison.

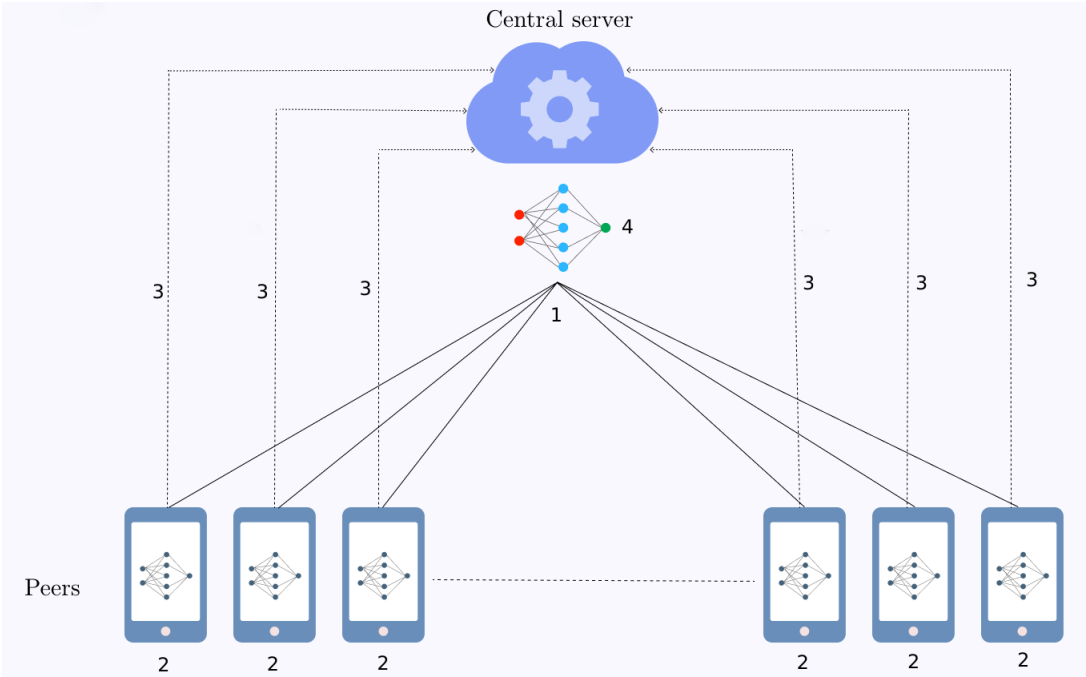

Figure 5: Federated Learning process. Source: *Devfi*

Next, the steps comprising the diagram are briefly outlined:

1. The server sends the global model to the clients. This step is mirrored in the base code at the beginning of the *run_experiment()* function, which is responsible for the initialization of the FL environment, and it is where the global model is initialized and sent to the clients.

2. The clients train the model with their local data. The second step exhibits its equivalence in the *participant_update()* function, which is called for each peer participating in the current round. This function is responsible for the training of the local model.

3. The clients send the updated models to the server. This step is difficult to locate in the base code, as it is not explicitly stated. However, it is possible to infer that the updated models are sent to the server in the *run_experiment()* function, where the local models are located in table structures managed by the code.

4. The server aggregates the models and sends the updated global model to the clients. This step is mirrored in the base code at the series of if statements mentioned in the previous section, 4.1, which are responsible for the aggregation of the local models.

The process is repeated until the global model converges. This step is mirrored in the loop defined in the *run_experiment()* function, which is dependent on the value of the parameter "*global_rounds*".

4.3 Chosen parameters and modifications

The first step after dissecting the base code [20] is to choose the parameters that we use for our experiments:

- Independent and Identically Distributed (IID) data: In an IID case, all the peers have a uniform sample of the data, meaning each peer possesses a similar proportion of each class. Thus, each of them represents a proportion of the global data. As a result, every update they compute serves as an unbiased estimate of the global model. In the non-IID scenario, this is not the case. Each user could have different percentages of classes or contain a lot of outliers or various other scenarios. In the most extreme case, each user might only have data corresponding to a single class. In such instances, the updates computed by peers can be significantly dissimilar from one another.

  IID is the simpler case for defences, and therefore more challenging for attackers. If an attack performs moderately well in the IID case, it should perform better in the non-IID case. Or, at the very least, it should be harder to detect. That is why it makes sense to start here. The non-IID case would be the most favourable scenario, as many defences will likely fail directly. The reality is that peers typically fall somewhere in between: neither fully IID nor completely non-IID. The IID case serves as a good starting point for research into these topics.

- Global rounds: This variable is set to 100. This means that the global model is updated 100 times.

- Local epochs[3]: This variable is set to 3. This means that each peer trains its model

[3]Epochs refer to the quantity of iterations a ML algorithm performs on the entire training dataset. Each epoch involves the algorithm making incremental adjustments to its model parameters based on the training data, aiming to improve its performance over time.

for 3 epochs before sending the update to the server.

- Number of peers: There are 20 peers in the system.

Another minimal change made to the base code is adding a condition to check if the device being used for the experiments is a Graphics Processing Unit (GPU) or the device's Central Processing Unit (CPU). This change is made because the GPU is much faster than the CPU when employing tensors[4]. The code used to perform this check is shown in Code 1.

```
import torch
...
if torch.cuda.is_available():
        DEVICE = "cuda"
        print('GPU is available:', torch.cuda.get_device_name(0))
else:
        DEVICE = "cpu"
        print('GPU is not available, CPU will be used')
DEVICE = torch.device(DEVICE)
...
```

Code 1: Computing device selection and information

Finally, we also added a section after the FL system has finished the global rounds to print the results obtained in a confusion matrix format. This matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. The code used to print this matrix is shown in Code 2.

```
from matplotlib import pyplot as plt
...
actuals, predictions = self.test_label_predictions(simulation_model,
↪  self.device, self.test_loader, dataset_name=self.dataset_name)
plt.matshow(confusion_matrix(actuals, predictions))
plt.colorbar()
plt.show()
...
```

Code 2: Printing the confusion matrix

4.4 Implementation of the entropy-based label-flipping attack

The general structure that this attack follows is shown next, and the code that implements the entropy-based label-flipping attack and serves as a reference for other attacks

[4]Tensors are multi-dimensional arrays commonly used in mathematics and ML to represent data. In ML, tensors are used to store and manipulate data, such as images, sequences, and more complex structures. Tensors are particularly well-suited for ML tasks due to their ability to efficiently handle large volumes of data and perform operations in parallel. GPUs are highly effective for tensor operations, as they are optimized for parallel computing.

implemented is shown in Code 3.

1. Initialize empty lists to store the indices of the samples of the source class and the entropy associated with each sample.
2. Iterate through the local model's samples using the DataLoader[5].
   (a) For each sample, we have to check if the sample's label is the source class's label. If it is, we append its index to the list. Since we iterate through batches of samples, it is important to keep the absolute index and not the relative one.
   (b) Then, we introduce the samples into a tensor so we can predict the output the model would return.
   (c) To obtain this output, we call a function that predicts the output of the model.
   (d) After that, we must transform the output to NumPy[6] format so we can calculate the entropy.
   (e) Finally, we compute the entropy of the outputs and store them in a list.
3. We merge the indices and entropies lists into a single list.
4. The next step is sorting the list by **highest entropy**.
5. We select the top 50% of the list.
6. We sort it again by index, so we can flip the labels of the samples in the correct order as we iterate in the following step.
7. To iterate through the samples in the local dataset, we create a *index_label_flip()* function, shown in Code 4. This function works as follows:
   (a) The function receives the dataset, the sorted indices and entropies list, and the target class (its ID).
   (b) It begins with the creation of an empty list to store the poisoned data.
   (c) Then, it iterates through the dataset.
   (d) For each sample, it checks if the sample's index is in the sorted indices list.
   (e) If it is, it appends a structure with the data of the sample and the target class's label to the poisoned data list.
   (f) If it is not, it appends a structure with the data of the sample and the original label to the poisoned data list.
   (g) Finally, it returns the poisoned data list.
8. Next, we create a new DataLoader object with the poisoned dataset, ready to be used to train the local model.

[5] A "DataLoader" in PyTorch refers to a utility class that simplifies the process of loading and managing data for training and testing machine learning models.

[6] NumPy is a fundamental package in the Python programming language used for numerical computations and data manipulation. It provides support for working with large, multi-dimensional arrays and matrices, along with an extensive collection of mathematical functions to operate on these arrays efficiently.

9. Finally, we increment a counter that keeps track of the number of attacks performed and print the information about the attack.

It is important to note that the iterations through the DataLoader are not deterministic if the *shuffle* parameter is set to true because each time we iterate through the DataLoader, the samples are shuffled. Our solution is to set the parameter to false before the first access so we can keep the indices of the samples in the same order to flip them. Then, after the dataset poisoning is completed, as we create the new DataLoader, we set the parameter to true again so the model does not learn the order of the samples.

This attack's structure sets the base that is used for the rest of the attacks. Therefore, the main structure for the rest of the attacks is almost the same.

Given that the execution time for an attack with the parameters mentioned in Section 4.3 is approximately 4 hours for an Nvidia RTX2060 graphics card, and that we are executing eight attacks per poisoning method, as seen in Section 2.3.5, we are not able to try all the possible modifications of the entropy-based label-flipping attack. Therefore, we have to choose the ones that we think are the most promising. The modifications that we have chosen to implement and test are the following:

- Keeping the highest 25% entropies. This modification is based on the idea that the samples with the highest entropy are the ones that are more likely to be misclassified. Therefore, if we are more specific and only flip the labels of the samples with the highest entropy, we should be able to increase the attack success rate by avoiding detection from the server. To modify the code, we only have to change line 23 of Code 3 to *num_elements_to_keep = len(sorted_entropy_list) // **4***.
- Keeping the highest 75% entropies. This modification is based on the opposite idea, that is, if the attack is avoiding detection when poisoning 50% of the samples, we can try to increase the attack success rate by poisoning more samples.
- Keeping the lowest 50% entropies. The thought for this idea is that the samples with the lowest entropy are the ones that are easier to classify. Therefore, if we flip these labels, we may be able to confuse the server in the first training rounds, and indirectly lean benevolent peers to misclassify the samples according to our will. To modify the code, we only have to change line 22 of Code 3 to *sorted_entropy_list = sorted(entropy_list, key=lambda x: x[1], reverse=**False**)* so the list is ordered by ascending entropy.
- Keeping the lowest 25% entropies. The reason for this modification is the same as the previous one, but being more specific and only flipping the labels from the samples that better represent the source class.
- Applying a scaling factor to the weights before the update. This modification is based on literature proposals to ensure a successful attack, commented in Section 2.3.5. The code used to implement this modification is shown in Code 5. The code is located at the end of the *participant_update()* function to receive the trained local model.

```
if (attack_type == 'entropy_label_flipping') and (self.peer_type ==
↪  'attacker'):
    train_loader = DataLoader(self.local_data, self.local_bs, shuffle =
    ↪  False, drop_last=True)
    model.eval()  # Set the model to evaluation mode
    entropies = []  # To store entropies for each data sample
    kept_indices=[] #to store the indices of the data samples with
    ↪  label==target_class
    with torch.no_grad():
        for batch_idx, (data, labels) in enumerate(train_loader):
            # Find the indices of data samples with label of the source
            ↪  class
            source_mask = labels == source_class
            # Keep the positions of the data samples with
            ↪  label==source_class
            kept_indices_batch = (batch_idx * train_loader.batch_size + i
            ↪  for i in range(len(source_mask)) if source_mask[i])
            kept_indices.extend(kept_indices_batch)
            # Get the data samples with label target
            data_source_batch = data[source_mask]
            # Send the data samples with label=source_class to the device
            data_source_batch = data_source_batch.to(self.device)
            # Obtain the predicted outputs for data samples with label
            ↪  source_class
            output = model(data_source_batch)  # Get the model's output for
            ↪  the source class data samples
            predictions_np=(output.cpu().numpy())   # Convert the output to
            ↪  numpy
            entropies.extend(stats.entropy(predictions_np, axis=1)) #
            ↪  Entropy is calculated by row
    entropy_list = list(zip(kept_indices, entropies))
    sorted_entropy_list = sorted(entropy_list, key=lambda x: x[1],
    ↪  reverse=True)    # Sort it by entropy value
    num_elements_to_keep = len(sorted_entropy_list) // 2   # Number of
    ↪  elements that represents the 50% of the data that we can attack
    top_50_percent = sorted_entropy_list[:num_elements_to_keep] # Extract
    ↪  the top 50% of the data
    sorted_entropy_list = sorted(top_50_percent, key=lambda x: x[0])  # Sort
    ↪  by index to keep the order of the data

    poisoned_data = index_label_flip(train_loader.dataset,
    ↪  sorted_entropy_list, target_class)
    # Create a new DataLoader with the updated dataset and with a shuffle
    train_loader = DataLoader(poisoned_data, self.local_bs, shuffle = True,
    ↪  drop_last=True)
    self.performed_attacks+=1
    print('Entropy-based label-flipping attack launched')
```

Code 3: Entropy-based label-flipping algorithm

```
def index_label_flip(dataset, sorted_list, target_class):
        poisoned_data = []
        for i, (data, label) in enumerate(dataset):
        if i in [index for index, _ in sorted_list]:
                poisoned_data.append((data, target_class))  # Change the
                ↪  label to target_class
        else:
                poisoned_data.append((data, label))  # Keep the original
                ↪  label
        return poisoned_data
```

Code 4: *index_label_flip()* function

```
def scale_model(update, scale_factor):
        for key in update.keys():
        update[key] = (update[key].float() * scale_factor).long()
        return update
...
if self.peer_type == 'attacker' and (attack_type == 'entropy_label_flipping'
↪  or attack_type == 'closeness_label_flipping'):
        update = scale_model(model.state_dict(), scale_factor = 1.1)
        model.load_state_dict(update)
```

Code 5: Scaling factor implementation

The verification process to ensure the correct functionality of the entropy-based label-flipping algorithm has been carried out with meticulous attention to detail. Extensive testing and validation procedures have been executed to ensure that the algorithm operates as intended. The different evaluations that have been used to evaluate the algorithm's behaviour and performance are listed below:

- Printing the entropies list to ensure that the range of values is correct. Then, printing the length of the entropies list and the indices list to ensure that they have the same length.
- Printing the sorted entropy list to ensure that the list is sorted by descending entropy. Then, when the list is trimmed, printing it again to make sure it is ordered by index.
- Counting and printing the amount of source class labels before and after the poisoning to ensure that the number of samples with the source class label has been reduced by the number of elements in the sorted entropy list. Then, performing the same operation with the target class label to ensure that the number of samples with the target class label has been increased by the number of elements in the sorted entropy list.

4.5 Implementation of the closeness-based label-flipping attack

The implementation of the closeness-based label-flipping attack is very similar to the entropy-based label-flipping attack. The code that implements the closeness-based label-flipping attack is shown in Code 6.

The set of different modifications that we have chosen to implement and test for the closeness-based label-flipping attack are the following:

- Keeping the highest 25% closeness. This modification is based on the idea that the samples with the highest closeness are the ones that are more likely to be misclassified as the target class.
- Applying a threshold instead of keeping a percentage. The rationale behind this modification is that, as the global rounds progress, the global model is more refined. Therefore, the top percentage may include samples that are not as close to the target class as they were in the first global rounds. In order to mitigate this, we can try to keep the samples with a closeness difference, lower than a threshold. The code used to implement this modification is shown in Code 7. The code is located before the call to the *index_label_flip()* function.

```
1  if (attack_type == 'closeness_label_flipping') and (self.peer_type ==
   ↪  'attacker'):
2  ...
3  closeness = []  # To store the prob differences for each data sample
4  kept_indices=[] #to store the indices of the data samples with
   ↪  label==target_class
5  with torch.no_grad():
6  ...
7          predictions_np=(output.cpu().numpy())    # Convert the output to
           ↪  numpy
8          temp = []
9          for i in range(len(predictions_np)):
10                 temp.append(abs(predictions_np[i][source_class] -
                   ↪  predictions_np[i][target_class]))
11         closeness.extend(temp)
12 ...
```

Code 6: Closeness-based label-flipping algorithm

The verifications to test the correct functionality of the closeness-based label-flipping algorithm are the same as the ones used for the entropy-based label-flipping algorithm.

4.6 Implementation of the adaptive label-flipping attack

As described in Section 3.4, the adaptive label-flipping attack is a combination of using the entropy-based or closeness-based label-flipping attacks, while flipping a different number of labels depending on the global round the system is in. The code that implements the adaptive label-flipping attack is shown in Code 8.

```
...
sorted_closeness_list = sorted(closeness_list, key=lambda x: x[1])     # Sort
↪  it by entropy value
threshold=0.02
count=1
actual=sorted_closeness_list[0][1]
while actual <= threshold and count < len(sorted_closeness_list):
        count += 1
num_elements_to_keep = count    # Number of elements that to keep
top_closeness = sorted_closeness_list[:num_elements_to_keep]
...
```

Code 7: Threshold implementation

```
if (attack_type == 'stealthy_closeness_label_flipping') and (self.peer_type
↪  == 'attacker'):
        ...
        if global_epoch > 25:
                with torch.no_grad():
                        ...
                sorted_entropy_list = sorted(entropy_list, key=lambda x:
                ↪  x[1], reverse=True)
                num_elements_to_keep=0
                if global_epoch <= 50:  # round 25 to 50, flip 75% of the
                ↪  data
                        num_elements_to_keep = len(sorted_entropy_list) // 2
                        ↪  + len(sorted_entropy_list) // 4
                if global_epoch <= 75:  # round 50 to 75, flip 50% of the
                ↪  data
                        num_elements_to_keep = len(sorted_entropy_list) // 2
                if global_epoch <= 100: # round 75 to 100, flip 25% of the
                ↪  data
                        num_elements_to_keep = len(sorted_entropy_list) //4
                ...
        else:   # round 0 to 25, flip all the data
                poisoned_data = label_filp(self.local_data, source_class,
                ↪  target_class)
                train_loader = DataLoader(poisoned_data, self.local_bs,
                ↪  shuffle = True, drop_last=True)
        ...
```

Code 8: Adaptive label-flipping algorithm

The first segment does not follow the usual code because, since all the labels are flipped there is no point in computing the entropy or the closeness of the samples.

The set of different modifications that we have chosen to implement and test for the adaptive label-flipping attack are the following:

- Testing with entropy and closeness-based label-flipping to flip all the labels during the first half of the global rounds. During the second half, flip 50% of the samples.
- Testing with either entropy or closeness-based label-flipping to flip all the labels during the first half of the global rounds. During the second half, flip 25% of the samples.

The verifications to test the correct functionality of the adaptive label-flipping algorithm are the same as the ones used for the previous label-flipping algorithms.

5 Results

In this Section, we present the findings from the thorough tests that were discussed in the previous section. The significant amount of time needed for the code to run across various aggregation systems is the main obstacle to carrying out these tests, as was previously mentioned. To ensure extensive evaluations of the proposed strategies' effectiveness, a substantial amount of time is required.

Additionally, as discussed in the previous section, our main goal is to optimise the ASR, while also maintaining an acceptable value of overall accuracy.

This guarantees that our attack only affects the targeted classes while maintaining a reasonable performance across the full range of classifications. The upcoming findings shed light on the performance of the hypotheses discussed in Sections 3.2, 3.3, and 3.4.

We will also comment the results obtained from the experiments that were carried out to evaluate the effectiveness of the strategies for the MNIST dataset.

5.1 Results for the MNIST dataset

The experiments for this dataset were carried out using the following values for the specified parameters:

- The data distribution type was changed from non-IID to IID.
- The number of peers was changed from 100 to 50.
- the amount of global rounds was changed from 200 to 50.

The first step is to evaluate which are the results for the baseline model[7]. The baseline for a FL setting is FedAvg, since it is the aggregation method specified in the proposal of this system [7]. We also evaluate the performance for the standard label-flipping attack to be able to compare it with the proposed strategies.

5.1.1 MNIST results with an attacker's ratio of 10%

The first experiment is performed for the standard label-flipping attack. The results are shown in Table 1. As it can be seen, we do not perform all the possible experiments, since at this point of the project we are interested in evaluating the performance of our hypotheses with the specified attacker's ratio. For future reference, Figure 6 shows the confusion matrix for the baseline model, where there have been no attacks and the model is almost perfectly trained.

	FedAvg (No Attackers)	FedAvg	Median	TMean	MKrum
TE	0.0519	0.055	0.055	0.0539	0.0522
All-Acc	98.35	98.34	98.36	98.36	98.41
Src-Acc	97.28	96.4	96.89	96.98	97.67
ASR	0.29	0.39	0.29	0.29	0.29

Table 1: MNIST. Standard label-flipping, attacker's ratio of 10%

[7]The baseline model is the model that is trained without any attack.

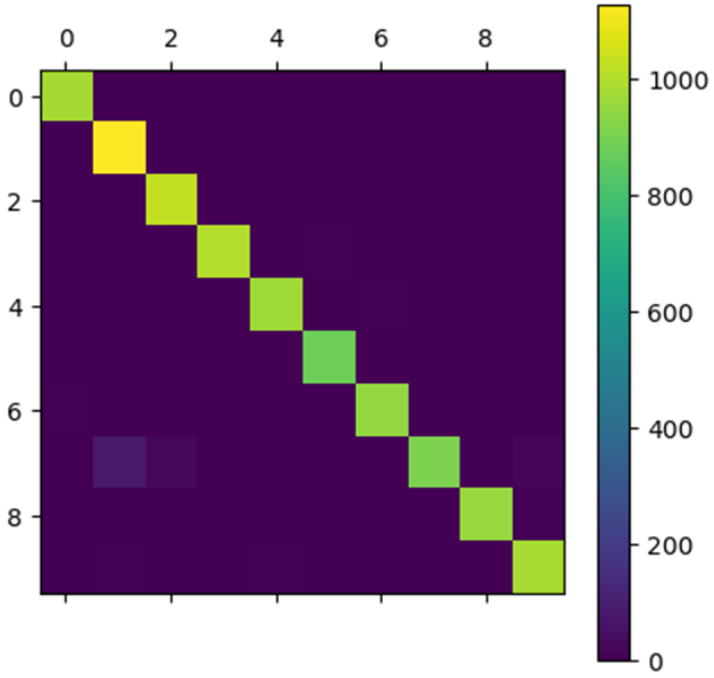

Figure 6: Confusion matrix for the baseline model

Table 2 shows the results obtained when testing the top 25% entropy and an attacker's ratio of 10%. The obtained results are similar to the results for the standard label-flipping attack, except for FedAvg, where the ASR drops by 0.20%.

	FedAvg	Median	TMean	MKrum
TE	0.0509	0.0546	0.0539	0.0516
All-Acc	98.42	98.31	98.36	98.38
Src-Acc	97.57	97.47	97.47	97.57
ASR	0.19	0.29	0.29	0.29

Table 2: MNIST. Entropy-based label-flipping, top 25% entropy, attacker's ratio of 10%

5.1.2 MNIST results with an attacker's ratio of 40%

Since the results for an attacker's ratio of 10% are not very promising, we decided to increase it to 40% to see if the results improve. Table 3 shows the results obtained for the standard label-flipping attack and an attacker's ratio of 40%. Note that for this experiment, the baseline is not tested, and only two aggregation methods are evaluated. As seen in the table, the ASR improves for both aggregation methods.

	FedAvg	TMean
TE	0.0985	0.0653
All-Acc	97.56	98.11
Src-Acc	87.94	94.55
ASR	7.78	1.46

Table 3: MNIST. Standard label-flipping, attacker's ratio of 40%

Finally, Table 4 shows the results obtained for the entropy-based label-flipping attack

and an attacker's ratio of 40%. The results are not promising when compared to the results for the standard label-flipping attack.

	FedAvg	TMean
TE	0.0533	0.0556
All-Acc	98.46	98.38
Src-Acc	97.57	97.57
ASR	0.19	0.19

Table 4: MNIST. Entropy-based label-flipping, top 25% entropy, attacker's ratio of 40%

We first began using the MNIST dataset for our experiments because it is the most common dataset used in the literature. However, we quickly realized that the results were not very interesting. The reason for this is that the MNIST dataset is too simple. The model is able to achieve a very high accuracy, and the label-flipping attacks are not able to reduce it significantly. Therefore, we decided to use the CIFAR-10 dataset instead. This dataset is more complex, and the model is not able to achieve such a high accuracy. This means that the label-flipping attacks are able to reduce the accuracy significantly.

5.2 Results for the CIFAR-10 dataset

Since the results obtained with the MNIST dataset are not promising, we decided to evaluate the performance of the proposed strategies with the CIFAR-10 dataset. MNIST offers a relatively easy problem to solve, since the images are very simple. This is why the aggregation methods are able to achieve a very high accuracy.

Our aim will be to flip the labels of the source class (dog) to the target class label (cat). The purpose of these experiments is finding which attack obtains the best results in order to evaluate the performance of the proposed strategies on lower attacker's ratio. This is why, for some strategies' modifications, we only evaluate the performance for some aggregation methods.

5.2.1 CIFAR-10 results with an attacker's ratio of 30%

We begin these experiments with an attacker's ratio of 30%, using the parameters commented in 4.3. Table 5 shows the results obtained for the standard label-flipping attack. As it can be seen, the ASR is very high for almost all the aggregation methods, this way it will be easier to detect if the proposed strategies are effective.

	FedAvg (NA)	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8058	0.8935	0.8462	0.872	1.0101	0.8821	0.9063	1.1157	0.8891
All-Acc	76.56	75.02	74.8	75.35	72.7	75.85	74.77	72.97	74.8
Src-Acc	65.6	35.4	36.0	35.0	17.5	63.9	60.4	21.3	62.2
ASR	14.7	43.2	41.6	45.3	62.7	15.8	15.4	55.5	15.2

Table 5: Standard label-flipping, attacker's ratio of 30%

The results for the entropy-based label-flipping attack, which flips the 25% highest entropies are shown in Table 6. As it can be seen, the ASR is lower than the ASR obtained for the standard label-flipping attack. The next step is to increment the amount of labels flipped to see if the results improve.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8074	0.7735	0.7821	0.8867	0.8059	0.8379	0.9565	0.8709
All-Acc	76.7	76.4	76.37	76.45	76.72	76.39	76.24	76.85
Src-Acc	61.2	61.7	61.2	58.3	60.2	59.8	58.8	61.8
ASR	17.4	18.5	18.7	20.7	19.5	19.4	19.8	17.7

Table 6: Entropy-based label-flipping, top 25% entropy, attacker's ratio of 30%

Table 7 shows the results obtained for the entropy-based label-flipping attack, which flips the 50% highest entropies. This attack is the one selected to represent entropy in experiments with lower attacker's ratio, since it obtains good results without incrementing too much the amount of labels flipped.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8301	0.8092	0.8146	0.8978	0.8402	0.8421	1.0441	0.8724
All-Acc	76.38	75.48	75.55	75.4	76.62	76.38	73.88	76.13
Src-Acc	56.3	54.6	54.5	51.2	55.1	53.5	46.9	62.6
ASR	22.3	25.3	24.1	25.7	23.0	25.8	26.5	15.6

Table 7: Entropy-based label-flipping, top 50% entropy, attacker's ratio of 30%

The results for the entropy-based label-flipping attack, which flips the 75% highest entropies are shown in Table 8. We can observe higher results when compared with the strategy flipping 50% of the samples. Given that the results do not improve by a huge factor, we will not use this strategy in the experiments with lower attacker's ratio since the amount of extra labels flipped is considered to be too high.

	FedAvg	TMean
TE	0.8353	0.8277
All-Acc	76.08	75.59
Src-Acc	48.6	47.3
ASR	29.1	28.3

Table 8: Entropy-based label-flipping, top 75% entropy, attacker's ratio of 30%

Table 9 shows the results obtained for the entropy-based label-flipping attack, which flips the 50% lowest entropies. The results are close to those obtained when flipping the highest 50% entropies, but the ASR is lower while flipping the same amount of labels.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8273	0.8166	0.8263	0.907	0.8453	0.826	1.0302	0.8851
All-Acc	76.55	75.46	75.82	75.67	76.32	76.08	74.37	75.44
Src-Acc	55.4	54.8	57.8	53.7	57.0	56.5	51.9	61.6
ASR	22.9	23.4	22.1	25.6	21.0	22.9	25.1	16.3

Table 9: Entropy-based label-flipping, lowest 50% entropy, attacker's ratio of 30%

The results for the entropy-based label-flipping attack, which flips the 25% lowest entropies are shown in Table 10. As the table shows, the ASR is lower than the ASR obtained for the strategy flipping the 50% lowest entropies.

The results for an entropy-based label-flipping attack, keeping the 50% highest entropies, and boosted with a scaling factor of 1.2 are shown in Table 11. As can be seen, the

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.824	0.7864	0.7639	0.8805	0.8068	0.8118	1.0406	0.902
All-Acc	76.7	76.51	76.88	76.65	77.07	76.54	74.8	76.42
Src-Acc	60.1	58.8	62.4	60.1	61.2	61.3	57.7	58.4
ASR	18.8	19.9	16.9	18.1	18.7	18.6	18.4	18.3

Table 10: Entropy-based label-flipping, lowest 25% entropy, attacker's ratio of 30%

scaling factor either improves or lowers the ASR, depending on the aggregation method. The scaling factor also affects the overall accuracy, since we are scaling all weights instead of only the weights of the targeted classes. Figure 7 shows the confusion matrix for FoolsGold when applying a scaling factor of 1.2. As can be seen, the overall accuracy is lowered. In this case, the effect is that most classes are classified as class 4, which is not the source nor the target class.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	1.7627	1.2765	1.3963	0.9382	1.7487	0.8719	1.1691	0.872
All-Acc	36.02	54.27	49.37	74.87	38.21	76.65	73.39	76.28
Src-Acc	28.5	36.4	10.0	64.0	33.9	64.3	62.1	64.4
ASR	14.0	44.9	73.9	16.2	16.7	14.6	16.5	14.4

Table 11: Entropy-based label-flipping, top 50% entropy, scaling factor of 1.2, attacker's ratio of 30%

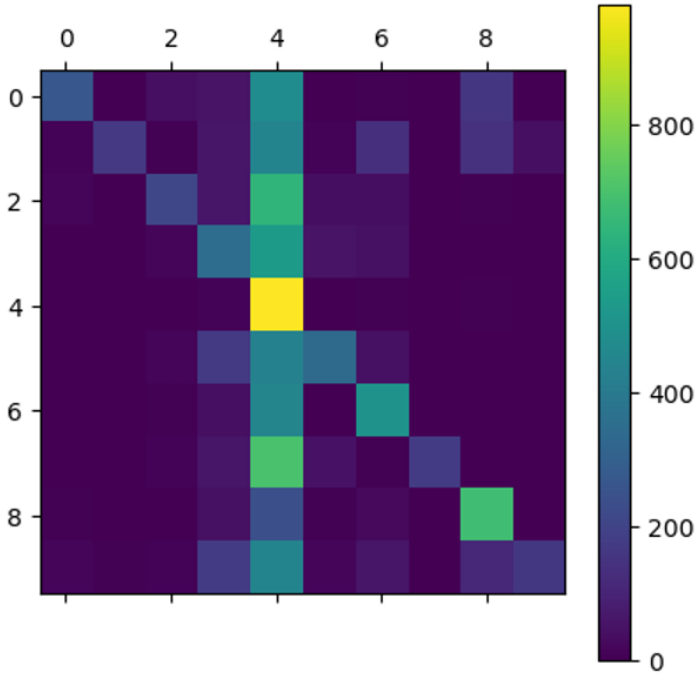

Figure 7: Confusion matrix for FoolsGold when applying a scaling factor of 1.2

Table 12 shows the results obtained for the entropy-based label-flipping attack, which flips the 50% highest entropies, and boosted with a scaling factor of 1.1. From these results we can extract that modifying the scaling factor while flipping the top 50% entropy samples is not improving the results. Our next step is to try applying a scaling factor after flipping the highest 25% entropy samples.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	2.1662	1.2913	1.4383	0.9225	2.101	0.8495	1.0811	0.8697
All-Acc	21.26	52.55	47.58	74.61	23.59	76.88	73.46	75.3
Src-Acc	9.6	24.3	13.9	64.8	18.7	64.4	61.7	61.4
ASR	15.1	56.2	63.5	15.0	17.5	14.4	16.3	16.6

Table 12: Entropy-based label-flipping, top 50% entropy, scaling factor of 1.1, attacker's ratio of 30%

The final experiment with scaling factors is using again a factor of 1.1 and changing the entropy method to flip the top 25% highest entropies. The results are shown in Table 13. Since applying a scaling factor does not improve the results, we will not use it in the experiments with lower attacker's ratio.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	2.1276	1.3397	1.3826	0.941	2.1069	0.8886	1.1464	0.8919
All-Acc	23.79	52.91	50.76	75.55	23.18	75.5	73.11	76.37
Src-Acc	14.0	40.0	32.4	63.2	14.3	63.5	62.9	62.0
ASR	12.8	40.7	46.4	14.3	14.6	14.9	15.3	15.9

Table 13: Entropy-based label-flipping, top 25% entropy, scaling factor of 1.1, attacker's ratio of 30%

With the experiments of the entropy-based label-flipping concluded for an attacker's ratio of 30%, we proceed to perform the proposed experiments for the closeness-based label-flipping attack.

We begin by testing the closeness-based label-flipping attack, which flips the 25% closest images. The results are shown in Table 14. Knowing that for entropy the results improved when flipping the 50% highest entropy samples, we test only two aggregation methods for a closeness attack where we flip the 25% highest closeness'. This is because the amount of time taken to test the whole set of aggregation methods is very high and, if it is not a promising attack, that time can be used to test other attacks.

	FedAvg	TMean
TE	0.8299	0.8235
All-Acc	76.63	76.48
Src-Acc	60.4	59.5
ASR	18.2	20.1

Table 14: Closeness-based label-flipping, top 25% closest images, attacker's ratio of 30%

Table 15 shows the results obtained for the closeness-based label-flipping attack, which flips the 50% closest images. This attack is the one selected to represent closeness in experiments with lower attacker's ratio, since it obtains good results without incrementing too much the number of labels flipped. As can be seen, we have taken a good choice not testing all the aggregation methods for the 25% closest images.

To conclude with the closeness-based label-flipping, we perform the experiments for different thresholds. Table 16 shows the results obtained for the closeness-based label-flipping attack, which flips the samples that have a closeness value lower than 0.3. The

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8205	0.8455	0.806	0.9058	0.8301	0.8467	1.0287	0.9029
All-Acc	76.23	76.68	76.08	75.83	76.7	76.47	74.24	75.05
Src-Acc	52.0	55.4	53.2	47.9	54.0	55.6	43.8	64.0
ASR	26.3	23.6	25.7	31.8	23.6	23.2	32.9	13.8

Table 15: Closeness-based label-flipping, top 50% closest images, attacker's ratio of 30%

rationale behind not testing all the aggregation methods is the same as the one explained for the closeness-based label-flipping attack that flips the 25% closest images.

	FedAvg	TMean
TE	0.82	0.7903
All-Acc	76.56	76.5
Src-Acc	63.0	65.5
ASR	15.9	14.8

Table 16: Closeness-based label-flipping, threshold of 0.3, attacker's ratio of 30%

The results for a threshold of 0.6 are shown in Table 17. As can be seen, the results improve by using a higher threshold. However, flipping samples with a closeness value lower than 0.6 is not a good strategy, since the ASR is lower than the ASR obtained for the strategy flipping the 50% closest images.

	FedAvg	TMean
TE	0.825	0.7901
All-Acc	76.67	76.52
Src-Acc	64.4	64.6
ASR	15.8	16.1

Table 17: Closeness-based label-flipping, threshold of 0.6, attacker's ratio of 30%

Since the experiments carried out using a threshold do not improve the results, we will not use this strategy in the experiments with lower attacker's ratio.

The final experiments for an attacker's ratio of 30% are performed with the adaptive label-flipping attack, using entropy. The first tested strategy is the one that flips the 100% of the samples during the first 25 rounds, 75% during the rounds in the range of 25-50, 50% during the rounds in the range of 50-75, and 25% during the last 25 rounds. The results are shown in Table 18. The obtained results are not promising when compared to the results obtained for the entropy or closeness-based label-flipping attacks.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8327	0.787	0.8044	0.8875	0.8272	0.8427	1.0105	0.8576
All-Acc	76.56	76.93	76.44	76.46	76.33	76.45	74.17	75.95
Src-Acc	59.2	63.1	60.9	58.4	59.1	61.4	54.1	58.7
ASR	19.5	18.2	18.6	20.1	19.8	18.6	23.7	19.1

Table 18: Adaptive label-flipping (entropy), 100 75 50 25, attacker's ratio of 30%

Table 19 shows the results obtained for the adaptive label-flipping attack, which flips the 100% of the samples during the first 50 rounds, and the 50% samples with highest entropy during the last 50 rounds. The results are better than those obtained for the entropy and

closeness-based label-flipping attacks, but following the rationale of the top 75% entropies, we will not use this strategy in the experiments with lower attacker's ratio since the amount of extra labels flipped is considered to be too high.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.825	0.7872	0.8134	0.8913	0.8538	0.8856	1.0319	0.8958
All-Acc	76.08	76.41	76.08	75.95	76.15	75.85	75.69	74.92
Src-Acc	55.0	55.3	55.3	53.6	63.2	51.6	51.5	63.0
ASR	23.4	22.9	23.4	26.3	15.7	26.3	27.6	15.9

Table 19: Adaptive label-flipping (entropy), 100 50, attacker's ratio of 30%

The results for the adaptive label-flipping attack, which flips the 100% of the samples during the first 50 rounds, and the 25% samples with highest entropy during the last 50 rounds are shown in Table 20. The results are not improved comparing with the previous strategy and neither when compared to the entropy-based label-flipping attack that flips the 50% highest entropies.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8215	0.7819	0.7953	0.8924	0.8532	0.8611	1.0247	0.8865
All-Acc	76.71	76.15	76.87	76.11	76.17	76.24	74.81	75.91
Src-Acc	61.6	59.3	62.2	58.7	63.4	60.1	55.8	62.0
ASR	19.1	21.4	18.4	18.9	15.0	18.8	21.3	16.9

Table 20: Adaptive label-flipping (entropy), 100 25, attacker's ratio of 30%

Our final experiment with adaptive label-flipping is using closeness and flipping the 100% of the samples during the first 50 rounds, and the 50% samples with highest closeness during the last 50 rounds. The results are shown in Table 21. The results are highly similar to the ones obtained for the closeness-based label-flipping attack that flips the 50% closest samples. Since the results are similar and the amount of labels flipped is the higher, we will not continue to perform experiments with adaptive label-flipping.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8352	0.7974	0.8395	0.8999	0.8707	0.8659	1.0128	0.8738
All-Acc	76.25	75.88	75.71	75.84	75.92	76.47	74.62	75.04
Src-Acc	50.9	54.1	55.1	48.3	62.7	53.0	42.4	63.7
ASR	26.9	24.2	23.6	29.2	16.1	24.9	33.9	14.5

Table 21: Adaptive label-flipping (closeness), 100 50, attacker's ratio of 30%

Based on the obtained results, the selection of the best attack for each strategy is the following:

- Entropy-based label-flipping: 50% highest entropies.
- Closeness-based label-flipping: 50% closest samples.
- Adaptive label-flipping: Due to the similar performance with other methods that flip less labels, we will not continue to perform experiments with adaptive label-flipping.

5.2.2 CIFAR-10 results with an attacker's ratio of 20%

As seen in the results of Section 5.2.1, closeness-based label-flipping performs slightly better than entropy-based label-flipping. In this section and the following one, we will continue comparing the results for our main hypotheses to test if this is a mere coincidence or turns out that closeness is a better approach than entropy when referring to label-flipping attacks in FL. We will also verify if the difference between the standard method and ours varies or it keeps outperforming them.

The obtained results for the CIFAR-10 dataset with an attacker's ratio of 20% are shown in this section. Table 22 shows the results obtained for the standard label-flipping attack. As can be seen, the ASR for the standard label-flipping attack drops when lowering the attacker's ratio. This is normal since there are less attackers poisoning their local models.

	FedAvg (NA)	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8058	0.8412	0.8204	0.817	0.9174	0.8602	0.8851	1.0738	0.8552
All-Acc	76.56	76.22	75.46	75.82	75.3	75.75	75.47	73.95	75.52
Src-Acc	65.6	44.8	50.2	44.3	39.1	59.8	63.5	33.9	62.8
ASR	14.7	32.6	28.4	32.4	37.8	18.3	14.2	41.6	15.2

Table 22: Standard label-flipping, attacker's ratio of 20%

The results obtained for the entropy-based label-flipping attack, which flips the 50% highest entropies are shown in Table 23. Because of the same reason as for the standard label-flipping, the ASR drops, but in this case, it only drops a few points.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8345	0.7777	0.8126	0.8647	0.8323	0.8181	0.9417	0.8683
All-Acc	76.18	76.05	76.24	76.12	76.22	76.63	76.69	75.65
Src-Acc	59.5	59.4	59.3	56.7	56.3	60.3	61.3	64.8
ASR	22.0	19.7	19.5	21.7	21.8	18.8	17.9	14.1

Table 23: Entropy-based label-flipping, attacker's ratio of 20%

Table 24 shows the results obtained for the closeness-based label-flipping attack, which flips the 50% closest images. The results are similar to those of the entropy-based label-flipping attack, but in this case, the closeness attack keeps outperforming the entropy attack for some aggregation methods.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8448	0.8127	0.8118	0.8914	0.8194	0.842	1.0035	0.8622
All-Acc	76.31	76.33	76.3	75.92	76.56	76.72	75.85	75.73
Src-Acc	57.4	59.0	56.1	54.0	59.3	59.3	51.7	64.1
ASR	20.0	20.0	22.6	26.2	19.7	19.9	26.7	15.5

Table 24: Closeness-based label-flipping, attacker's ratio of 20%

As can be seen, when lowering the attacker's ratio, the results for the standard label-flipping attack decrease much more than the results for the proposed attacks, which decrease only a few points. Probably, the reason why this happens is because, with a smaller attacker's ratio, the aggregation functions can better detect the malevolent peers that are flipping all the labels than the ones that are flipping only a few of them, which are stealthier.

5.2.3 CIFAR-10 results with an attacker's ratio of 10%

When lowering the attacker's ratio to 10%, we approach to a real-world situation where, there are millions of users participating in the FL system. It is not realistic to think of a scenario where, with that many users, even 10% of them are attackers.

Table 25 shows the results obtained using the standard label-flipping attack with an attacker's ratio of 10%. The standard solution for label-flipping keeps dropping its ASR when lowering the attacker's ratio.

	FedAvg (NA)	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8058	0.8081	0.7828	0.7996	0.8545	0.8332	0.874	0.991	0.8581
All-Acc	76.56	77.05	75.98	76.3	76.51	76.56	75.87	75.28	76.05
Src-Acc	65.6	51.8	54.9	55.8	53.4	66.5	63.9	49.2	63.8
ASR	14.7	21.7	23.1	21.5	24.7	13.9	13.4	27.8	13.7

Table 25: Standard label-flipping, attacker's ratio of 10%

The results obtained for the entropy-based label-flipping attack, which flips the 50% highest entropies are shown in Table 26. Our first hypothesis keeps a similar ASR when lowering the attacker's ratio, which is a good sign.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.8277	0.7724	0.8035	0.8533	0.826	0.8196	1.0331	0.8319
All-Acc	76.78	76.17	76.65	76.13	76.46	76.69	73.07	76.02
Src-Acc	61.8	62.3	63.1	62.9	60.3	61.7	56.2	61.8
ASR	17.8	18.6	17.4	17.7	17.6	18.4	20.4	16.8

Table 26: Entropy-based label-flipping, attacker's ratio of 10%

Finally, Table 27 shows the results obtained for the closeness-based label-flipping attack, which flips the 50% closest images. Our second hypothesis is also keeping a similar ASR when lowering the attacker's ratio. The closeness-based solution is no longer outperforming the entropy-based solution in many aggregation methods.

	FedAvg	Median	TMean	MKrum	Foolsgold	Tolpegin	FLAME	LFighter
TE	0.816	0.8238	0.787	0.8378	0.8141	0.8409	0.9718	0.8166
All-Acc	76.82	76.24	76.99	76.88	76.59	76.66	76.05	76.45
Src-Acc	63.6	62.4	61.3	60.3	61.1	63.2	57.6	63.7
ASR	17.5	18.4	19.1	19.4	18.5	16.9	22.0	14.9

Table 27: Closeness-based label-flipping, attacker's ratio of 10%

As can be seen in Figure 8, Figure 9, and Figure 10 when lowering the attacker's ratio to a 10%, the ASR for all kinds of label-flipping drop. It is important to note that, when comparing the drop of the results for the standard label-flipping attack against the proposed attacks, the drop from 30% to 10% attacker's ratio is much more significant for the standard label-flipping attack. This means than even though the proposed attacks do not obtain such a high ASR as the standard label-flipping attack, they keep a higher stability when lowering the attacker's ratio, as well as when tested against different defences. As commented before, a real-world scenario is not likely to have such a high attacker's ratio among the peers. Thus, our hypotheses perform similarly to the standard label-flipping attack in a situation resembling a real-world scenario, while flipping half of the labels.

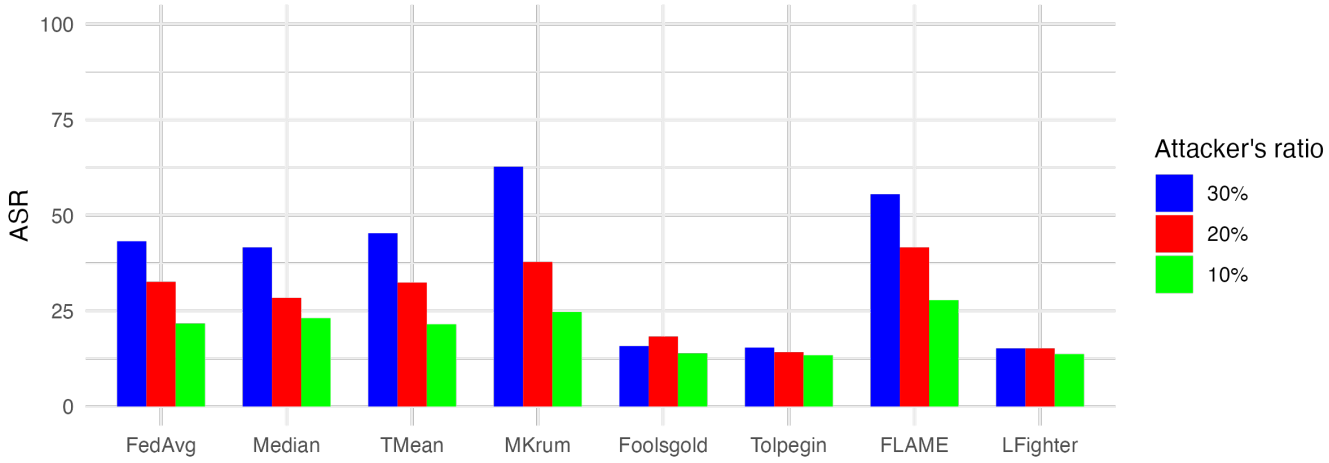

Figure 8: ASR for standard label-flipping

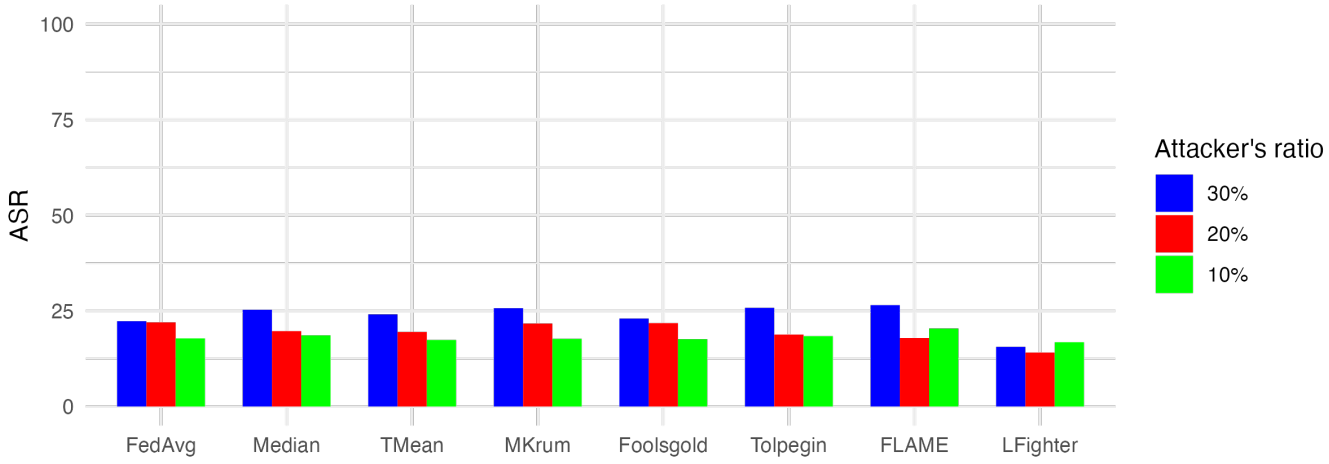

Figure 9: ASR for Entropy-based label-flipping

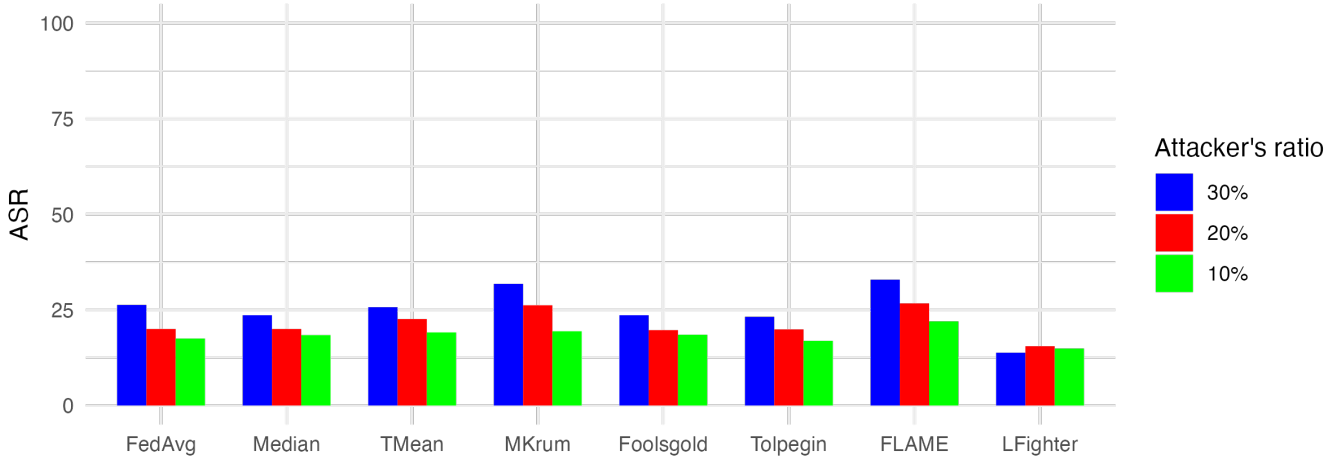

Figure 10: ASR for Closeness-based label-flipping

6 Conclusions

In this study, we presented a comprehensive analysis of various label-flipping algorithms' performance within the context of Federated Learning. The proposed hypotheses to create a more sophisticated label-flipping attack are the following:

- Entropy-based label-flipping: This attack is based on the idea of flipping the labels of those samples that have a high entropy among classification confidence vectors. Thus, being more difficult to classify.
- Closeness-based label-flipping: The idea behind this attack is to flip the labels of those samples that have similar probabilities to be classified as either the source or the target class.
- Adaptive label-flipping: The hypothesis of this attack is that by varying the number of samples that are flipped in each global round, the attack will be more difficult to detect. The logic behind this idea is that, as the global model is refined, lowering the number of samples that are flipped will make the attack more difficult to detect.

We explored how these hypotheses perform against the performance of a standard label-flipping attack in the context of their impact on different aggregation methods.

Label-flipping attacks pose a significant threat to the integrity of Machine Learning models. Our findings demonstrate that certain algorithms, such as Foolsgold[23], Tolpegin[24] and LFighter[18], exhibit robustness against label-flipping attacks, effectively mitigating their impact on model accuracy (LFighter being the one defence which maintains the lower Attack Success Rate (ASR) and thus, being the most effective one analysed). These algorithms leverage the collective intelligence of the participating devices in the Federated Learning framework to adaptively adjust model parameters and effectively counteract the adversarial manipulation of labels.

The results of our experiments allow us to draw two main conclusions. First, the more samples whose labels are flipped, the more effective the label-flipping attack is, especially in settings with a high number of attackers and no (or bad) defences. This is exemplified by the standard label-flipping attack, which changes all instances of the source class to the target class. This is also supported by our attacks when we choose to flip a high percentage of the samples. Note, however, that our ASR measurements are based on the whole source class, and our attacks only flip a fraction of samples from the source class. Focusing only on the samples that we do change would bring our ASR values higher. Our second conclusion, which partly supports our hypotheses (and thus merits further experimentation) is that our proposed stealthier attacks drop proportionally much less than the standard attack when confronted with defence mechanisms in terms of ASR. While the ASR for the standard attack drops significantly when using different detection mechanisms and different ratios of attacker peers, our proposed attacks remain quite stable, surpassing on some occasions the results from the standard attack. Thus, further experiments are required to confirm (or dismiss) the effectiveness of our proposed attacks.

6.1 Future work

Future research directions could explore different approaches to sophisticated label-flipping attacks and perform more extensive evaluations of their performance. Including

evaluating the ASR on only the attacked samples.

Additionally, our work has focused on IID data, that is, identically and independently distributed data, which means that all peers have a uniform sample of the data. This makes deviations in the computed models easier to detect since all models computed by honest peers are unbiased estimators of the global model. In real settings, the distribution of data tends to be non-IID, with peers having unbalanced data, biased data, or even data where samples of some classes are completely missing. This diversity in the local data makes computed local models more diverse and therefore more difficult to assess. Non-IID settings make detecting attackers much more difficult. We plan to continue our experiments on different regimes of non-IID distributions.

It would be interesting to assess the performance of the evaluations proposed in this section using more powerful equipment. This would allow us to perform more tests in a shorter period of time, thus allowing us to perform more extensive evaluations of the algorithms' performance.

References

[1] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[2] Md. Omaer Faruq Goni, Fahim Md. Sifnatul Hasnain, Md. Abu Ismail Siddique, Oishi Jyoti, and Md. Habibur Rahaman. Breast cancer detection using deep neural network, 2020.

[3] Dennis JNJ Soemers, Vegard Mella, Cameron Browne, and Olivier Teytaud. Deep learning for general game playing with ludii and polygames. *ICGA Journal*, 43(3): 146–161, 2021.

[4] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7: 19143–19165, 2019.

[5] Mykola Galushka, Chris Swain, Fiona Browne, Maurice Mulvenna, Raymond Bond, and Darren Gray. Prediction of chemical compounds properties using a deep learning model. *Neural Computing and Applications*, 33, 10 2021. doi: 10.1007/s00521-021-05961-4.

[6] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[8] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), 2016. URL https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[9] Alberto Blanco-Justicia, Josep Domingo-Ferrer, Sergio Martínez, David Sánchez, Adrian Flanagan, and Kuan Eeik Tan. Achieving security and privacy in federated learning systems: Survey, research challenges and future directions. *Engineering Applications of Artificial Intelligence*, 106:104468, 2021.

[10] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[11] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

[12] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[13] Benjamin Zi Hao Zhao, Aviral Agrawal, Catisha Coburn, Hassan Jameel Asghar, Raghav Bhaskar, Mohamed Ali Kaafar, Darren Webb, and Peter Dickinson. On the (in) feasibility of attribute inference attacks on machine learning models. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 232–251. IEEE, 2021.

[14] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. {Updates-Leak}: Data set inference and reconstruction attacks in online learning. In *29th USENIX security symposium (USENIX Security 20)*, pages 1291–1308, 2020.

[15] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ml: A practical guide to machine learning with differential privacy. *Journal of Artificial Intelligence Research*, 77:1113–1201, 2023.

[16] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.

[18] Najeeb Moharram Jebreel, Josep Domingo-Ferrer, David Sánchez, and Alberto Blanco-Justicia. Defending against the label-flipping attack in federated learning, 2022.

[19] Najeeb Moharram Salim Jebreel et al. Protecting models and data in federated and centralized learning.

[20] Najeeb Jabreel. Lfighter, 2023. URL https://github.com/NajeebJebreel/LFighter.

[21] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2021.

[22] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent, 2017.

[23] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings, 2020.

[24] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems, 2020.

[25] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al. {FLAME}: Taming backdoors in federated learning, 2022.

[26] Eduard Bel. Master's thesis code, 2023. URL https://github.com/EduardBel/MastersThesisCode.

[27] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[28] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[29] Aditya Pal, Abhilash Barigidad, and Abhijit Mustafi. Imdb movie reviews dataset, 2020. URL https://dx.doi.org/10.21227/zm1y-b270.