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...
blablabla

Resum

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat.

Resumen

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna.

Abstract

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna.

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 promissing new paradigms of image generation, classification, chatbots, speech regonition and other AI's and, 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 wether 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 the Tesla autonomous driving system. 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 Tesla, the users' data is used to train the model in a distributed way among the vehicles in the Tesla 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 Tesla autonomous driving system by modifying the action that the car takes when recognizing a STOP sign, changing it to accelerating at full capacity, it could lead to multiple car accidents.

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.

The GitHub repository "LFighter" [2] by Najeeb Jabreel is this thesis inspiration. It already offers a working FL simulator, where the programmer can modify the environment's parameters and obtain results of attacks against a variety of servers with different rules. With the explicit purpose of formulating and deploying attacks against FL systems to assess their robustness, this simulator serves as the foundation for the development of this thesis.

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 Tesla's autonomous driving system, 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.

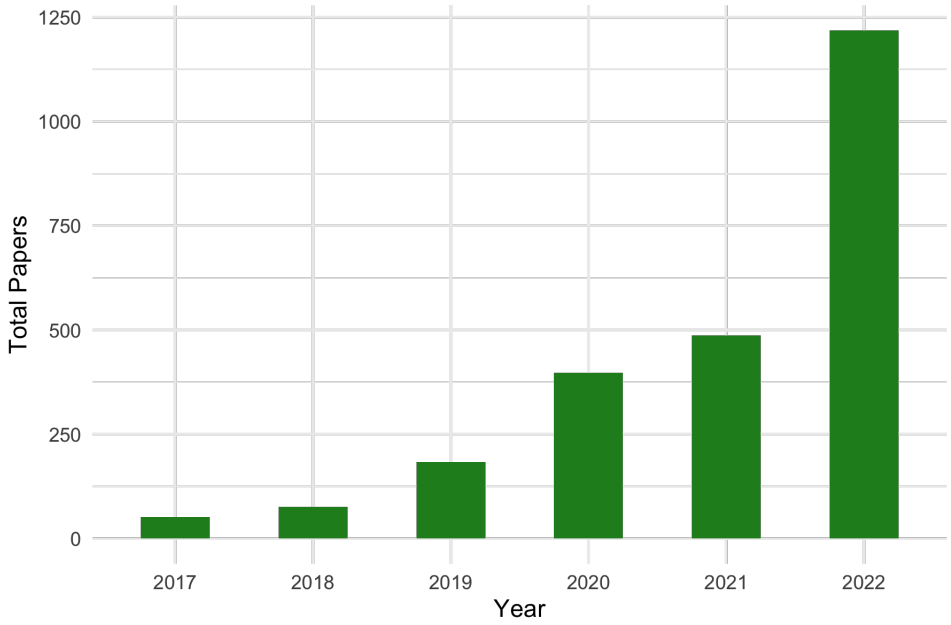

Figure 1: Published papers per year since FL was proposed. Source: *Web of Science*

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 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 suffer a label flipping 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

This thesis is organized as follows: . IDJQWOIJIOQ

. DHUIQWIHDIUQWUHI

2 Background

In this section, we establish the essential groundwork for comprehending the intricacies of model poisoning through label flipping on FL. We begin by presenting important ideas required to understand its security landscape.

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

Finally, we explore the different 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 Machine Learnig (ML) algorithms 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 [3], game playing, speech recognition, analysing molecular structures and predicting chemical properties, recommendation systems, and many others.

The DNN first creates a map of virtual neurons and, assigns to the connections linking them initial random "weights", or numerical values. These weights interact with the inputs to produce an output that can range from 0 to 1. An algorithm intervenes to adjust these weights if the network is unable to recognise a particular pattern with sufficient accuracy. Through this procedure, the algorithm can gradually increase the influence of specific parameters while iteratively determining the most effective mathematical operations necessary to efficiently process the input data. Figure 2 shows a representation of a DNN that classifies images into classes of animals.

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.

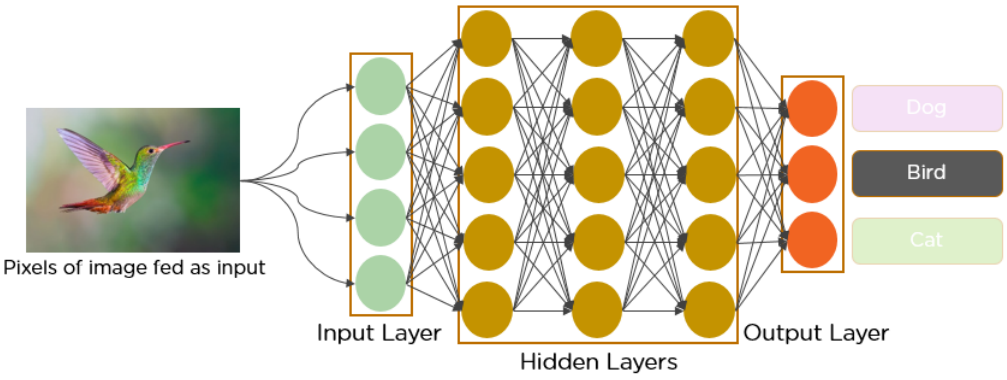

Figure 2: Deep Neural Network representation. Source: *Analytics Vidhya*

2.2 Federated Learning

Federated Learning, first proposed by McMahan et al. in 2017 [4], 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, compiles them, 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 a number of 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 [5]) by the European Commission or the Spanish "*Ley Orgánica de Protección de Datos*" (LOPD).

2.3 State of the art

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

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 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.

Peers' local updates 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 unintentionally gives adversaries the ability to infer class labels by reconstructing training samples without having any prior knowledge of the underlying data.

2.3.2 Poisoning attacks against Federated Learning

FL is vulnerable to poisoning attacks, a technique designed to sabotage the learning process by introducing malicious data or model updates. 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. 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 model data 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 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.

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 has the source class label, they meticulously change the label to the 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 on 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 a FL model is trained to distinguish between samples that are healthy and those that are diseased. 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, such as incorrect diagnoses or treatment recommendations, such a scenario might have serious consequences.

2.3.5 Defenses 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, by injecting poisoned data on local model updates. The attacker must also overcome the combined influence of benign clients during the aggregation process in order 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 some unique identifier (ID) of peers participating in the current training round, contrasting the model's accuracy results with those of prior rounds. In rounds of noteworthy disparities, the algorithm could store the IDs of participants from that specific round. In subsequent rounds marked by discrepancies, this stored list could be cross-referenced with the current round 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 peer, as this might lead her to create a new account to evade detection. For scenarios involving multiple malicious peers, an iterative process would take place, persisting until the identification of additional attackers is achieved. To establish a trustable 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 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 doesn't 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:

- 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 the DP method, each update parameter is altered by being clipped to a maximum threshold and then introducing 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 [2] are:

- Federated Averaging (FedAvg) [4]: 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 [6]: 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) [6]: 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) [7]: 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) [8]: 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.

- Tolpegin [9]: 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 [10]: 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 [11]: 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.

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

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

3 Architecture

In this section dedicated to the architecture of the thesis, we will delve into the organizational framework of the foundational code derived from the GitHub repository [2]. 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.

3.1 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 [12]: 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 [13]: This dataset contains 60,000 32x32 bit color 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 [14]: 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 logical location for the newly created code, explained in section 4 is inside 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.

3.2 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, 3.1. As a navigational aid as we navigate through the details of these two distinct environments, Figure 3 provides a diagram of an actual FL scenario to further illuminate this comparison.

Next, the steps comprising the diagram are briefly outlined:

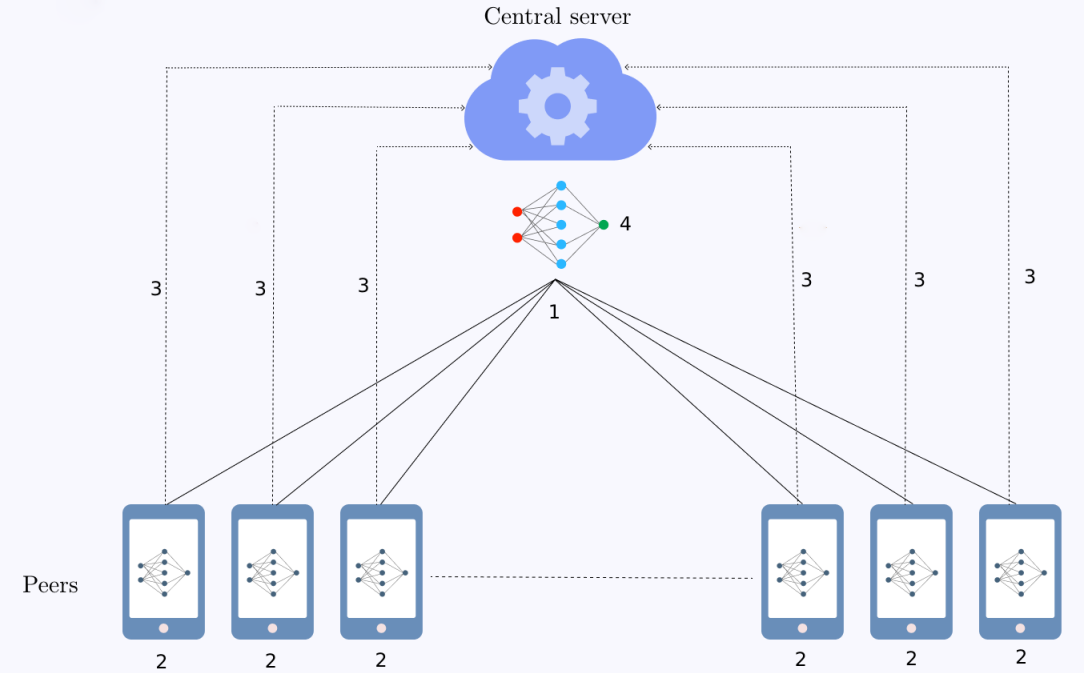

Figure 3: Federated Learning process. Source: *Devfi*

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, 3.1, 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 dependant on the value of the parameter "*global_rounds*".

4 Implementation

intro to the section, what are we going to talk about?

4.1 Dataset

we began by using the MNIST dataset, but we had to change to the CIFAR-10 dataset because of the size of the MNIST dataset.
explain

• MNIST. It contains 70K grayscale images of handwritten digits ranging from 0 to 9 with a size of 28fi 28fi 1 pixels (LeCun et al., 1999). It is split into a training set of 60K examples and a testing set of 10K examples. • CIFAR10. It is made up of 60K color images belonging to 10 dif- ferent classes (Krizhevsky, 2009). The images have a size of 32 fi 32 fi 3 pixels and are split into a training set of 50K examples and a testing set of 10K examples.

both of them are defined as image classification datasets

4.2 Creating the flipping functions

What is it? where can it be found? pros?

4.2.1 Example subsubtitle

5 Results

intro to the section, what are we going to talk about?

5.1 Results for entropy-based label flipping

What is it? where can it be found? pros?

5.2 Security attacks on Federated Learning

blabla

5.2.1 Example subsubtitle

6 Example title

6.1 Example subtitle

6.1.1 Example subsubtitle

7 Text examples

Example text example text

7.1 Bold & italic text

Bold text

Italic text

7.2 In document refernces

Equation 1

7.3 Other documents refernce

referenced text[?] test cite [?]

7.4 Acronyms & footnotes

ACRONYM[3]

Footnote[4]

7.5 Hyperlinks / URLs

Exemple of named hyperlink

[3] Acronym text
[4] Footnote text

8 Example lists

8.1 Unordered list

- Item
- Item
- Item

8.2 Ordered list

1. Item
2. Item
3. Item

9 Equation example

$$a^b = c \tag{1}$$

10 Table example

	API Disponible	API Obsoleta	Dificultat	Característiques avançades
Camera	1	21	Senzilla	No
CameraX	21	N/A	Senzilla	Sí[5]
Camera2	21	N/A	Complexa	Sí

Table 1: Comparativa d'APIs de càmera

As we can see in Figure 4, it works

11 Image example

Figure 4: Logo URV

12 Code snippet example

For all minted listings is required to enable *-shell-escape* on the LaTeX executable and have pygments installed

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)
...
```

Code 1: Code example

13 Diagram examples

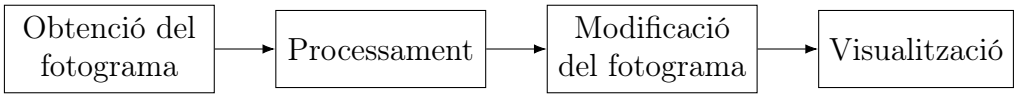

Figure 5: Projecte workflow

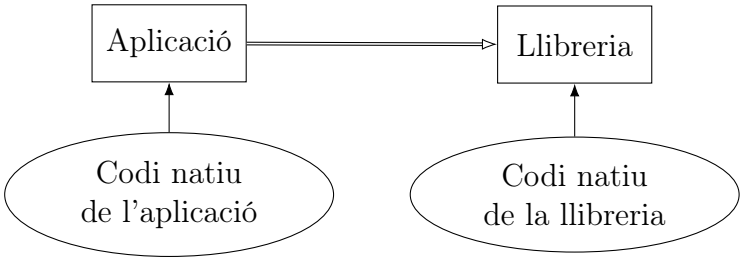

Figure 6: Module dependency

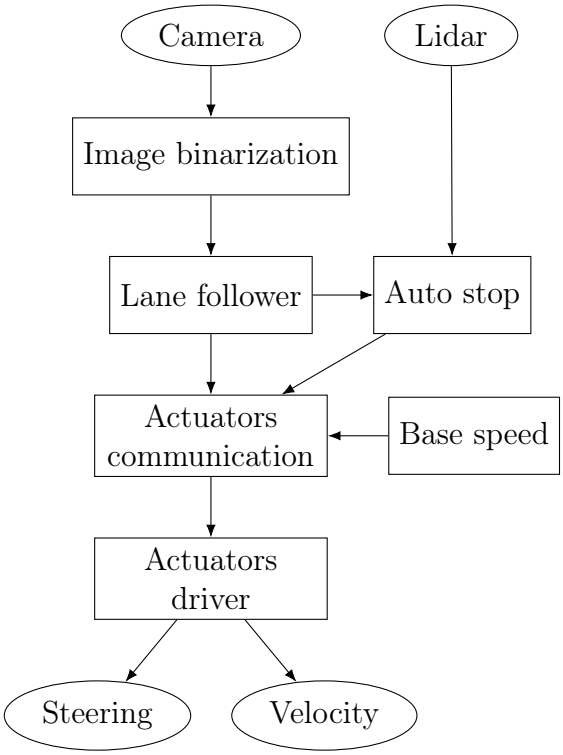

Figure 7: Car nodes layout

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

[3] 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.

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

[5] 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.

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

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

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

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

[10] 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.

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

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

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

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

Appendix A:
Apendix example