University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Robustness in Federated Learning

Daniel Demeter
Zurich, Switzerland
Student ID: 19-756-451

Supervisor: Dr. Alberto Huertas Celdran, Muriel Franco
Date of Submission: August 24, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Beim neuartigen Paradigma des föderierten Lernens können kollektive Geräte gemeinsam maschinelles Lernen über sensible Daten betreiben. Die Kollaborateuren - Kunden - müssen niemals ihre privaten Datensätze teilen. Basierend auf der Verteilung von Daten unter den Kunden, kann föderiertes Lernen in eine von drei Szenarien kategorisiert werden: horizontales föderiertes Lernen, vertikales föderiertes Lernen und föderiertes Transferlernen. Jeder dieser Szenarien besizt ein einzigartiges Set von Algorithmen, sowie einzigartige Ziele für feindliche Angriffe.

Sie können darauf abzielen, den Trainingsprozess zu stören, das gelernte Modell durch eine Hintertür zu verfälschen, zu verhindern, dass ein Modell überhaupt gelernt wird, oder sogar vertrauliche Informationen von Teilnehmern abzuleiten. Die Robustheit eines Systems bedeutet dessen Widerstandsfähigkeit gegen derartige Angriffe. Die Experimente in dieser Arbeit bewerten die Robustheit eines Systems anhand der Metriken Genauigkeit, Konfusionsmatrizen und relative Wichtigkeit.

Vorhandene Arbeit fokussiert diese Szenarien individuell, durch den Vergleich von Robustheit mit der "baseline" des jeweiligen Szenarios. Die Forschung liefert zwei wichtige Beiträge: *(a)* eine visuelle Taxonomie über die Angriffe und die Gegenmaßnahmen der Szenarien und *(b)* eine föderale Lernsimulation, die die Robustheit gegen die zwei häufigsten Form von Angriffen testet: verfälschte Daten (data poisoning) und zunehmende Verfälschungsangriffe (gradient poisoning attacks). Die Simulation verwendet für beide Szenarien denselben Datensatz, eine angemessene Verteilung, dieselbe Modellarchitektur und dieselbe Anzahl von Teilnehmern.

In dieser Arbeit werden zwei Modellarchitekturen für neuronale Netze für die Aufgabe des Lernens aus dem MNIST-Datensatz untersucht. Die Experimente zeigen, dass wenn Gegner nur 0.5% der zugängliche Daten vergiften, sind sowohl die peer-to-peer horizontale, als auch die SplitNN-basierte vertikale Implementationen anfällig für erfolgreiche Hintertürangriffe. Wenn der Angreifer das System mit einem Gradientenvergiftungsangriff angreift, bei dem die angewandten Gradienten zunächst mit einem nicht positiven Wert multipliziert werden, erwies sich die horizontale Implementierung als robuster. Während das vertikale System selbst bei einem Gradientenmultiplikator von -1 nicht in der Lage war, ein Modell zu erlernen, wurde die horizontale Implementierung nur bei einem Gradientenmultiplikator von -10 in ähnlicher Weise gebremst.

In the novel paradigm of federated learning, collectives of devices are able to collaboratively machine learn over sensitive data. The collaborators - clients - never have to share their private datasets. Depending on the distribution of the data amongst the clients, federated learning can be classified into one of three scenarios: horizontal federated learning, vertical federated learning, and federated transfer learning. Each of these scenarios comes with a unique set of algorithms, as well as unique targets for adversarial attacks.

Adversarial attacks in federated learning can have a wide variety of goals. They may aim to disrupt the training process, corrupt the learned model with a backdoor, prevent a model being learned at all, or even infer confidential information from participants. The robustness of a system is its resiliency to such attacks. The experiments in this work evaluate the robustness of a system through the lens of the metrics of accuracy, confusion matrices, and relative importance.

Current works focus on these scenarios individually, comparing robustness against each scenario's respective baseline. This work offers two main contributions: *(a)* a visual taxonomy of the attacks and countermeasures of these scenarios and *(b)* a federated learning simulation that tests its robustness against the two most common types of attacks: data- and gradient poisoning attacks. The simulation uses the same dataset, distributed appropriately, the same model architecture, and the same number of participants for both scenarios' implementations.

This work investigates two neural network model architectures for the task of learning from the MNIST dataset. The experiments detailed in this work show that when adversaries poison even as little as 0.5% of the data samples available to them, both the peer-to-peer horizontal and the SplitNN-based vertical implementations are prone to an entirely successful backdoor attack. When the adversary attacks the system with a gradient poisoning attack in which the applied gradients are first multiplied by some nonpositive value, the horizontal implementation proved more robust. While the vertical system was prevented from learning a model even with a gradient multiplier of -1, the only experiment in which the horizontal implementation was thwarted similarly was with a gradient multiplier of -10.

Contents

Chapter 1

Introduction

1.1 Motivation

The deluge in generated data has provided deep learning and machine learning applications with the massive amount of data they require to operate. However, with users becoming more aware of how their data is used and data privacy finding its way into the mainstream debate, a new approach to machine learning is required to appease these concerns. Introduced by Google in 2016 [9], federated learning emerged as a possible solution. The paradigm enables participants to collaboratively learn a shared machine learning model using their individual datasets and, most importantly, does so without having to share said dataset with one another. The paradigm provides solutions for different distributions of data. In *horizontal* federated learning the clients share the same feature space but do not share samples, whereas in *vertical* federated learning the clients do not share features, but do share samples. When neither feature space nor samples are shared, the scenario is termed federated transfer learning. Using decentralized datasets, federated learning is able to achieve comparable levels of accuracy, as classical machine learning algorithms can on the combined dataset [4]. Furthermore, because the datasets never leave the participants' possession, the logistical problem of aggregating them in a data center is completely eliminated.

Federated learning is not without its drawbacks. Due to the decentralized nature of its training phase, it exposes new attack surfaces to adversaries [2]. Furthermore, it presents a trade-off between the accuracy of the global model and the privacy of the participants' sensitive data. Because the paradigm aims to expose as little information about the individual participants' data as possible, recognizing the presence of inaccurate data samples is difficult, whether their intent is malicious or not. Because of the novelty of the paradigm, the extent of the damage that adversaries can do in a federated setting is unquantified. While the existence of different categories of attacks is well known, a direct comparison between their efficiency in different settings is unexplored.

The robustness of a system is its resiliency to such adversarial attacks. This work examines implementations' robustness through the metrics of accuracy, confusion matrices, and relative importance. Other current works investigate the scenarios discretely, providing no basis of comparison for implementations of different scenarios.

1.2 Description of Work

With the goal of improving the previous limitations, the main contributions of this work are two-fold. First, this thesis is a survey of the different adversarial attacks that a federated learning system could face. A taxonomy is created which compares the efficacy of different attacks and defenses in the different types of FL settings. Aggregating the findings of the related works, the taxonomy provides a comprehensive view of the interaction of these algorithms, their countermeasures, and the different scenarios of federated learning.

The second main contribution of this work is both a horizontal and a vertical federated learning system implementation. Both scenarios will share a dataset suitable for federated learning, which will be segmented in a way appropriate to the scenario. These systems are attacked with common adversarial attacks in a series of experiments to test and compare the resiliency of the systems to adversaries in the role of participants. The most common adversarial attacks were chosen for these tests: data poisoning and gradient poisoning attacks [8]. The systems are evaluated with multiple metrics to gain a holistic understanding of the effects of each. The results are compared with not just the corresponding scenario's baseline results, but also with the other scenario's results.

The work investigates two neural network model architectures to learn from the MNIST dataset. The system is attacked with a data poisoning attack that implements a backdoor and a gradient poisoning attack to deteriorate the model performance. Even by marking only 0.5% of the training data samples available to them, adversaries are able to implement a successful backdoor in both scenarios' implementations. The horizontal scenario proved more robust to the gradient poisoning attack, when holding the gradient multiplier constant across scenarios.

1.3 Outline

The organization of this paper is as follows. First, some background is offered on federated learning in Chapter 2. Then, related works are discussed in Chapter 3. In Section 4.1, the design of this work's experiments are detailed. Section 4.2 focuses on the concrete implementation of these design principles. The results of the numerous experiments are evaluated in Chapter 5. Chapter 6 discusses these results. Finally, in Chapter 7, the work is summarized and possible future works are outlined.

Chapter 2

Background

Federated learning is a broad category of algorithms that can be classified in many ways. The most common approach is to categorize based on how the dataset is partitioned.

The scenario in which participants' datasets share the same feature space, but do not share samples is referred to as Horizontal Federated Learning (HFL). This category can further be broken up into Horizontal to Businesses (H2B) and Horizontal to Consumers (H2C) [8].

In H2C, the number of participants is large (possibly millions), whereas in H2B, the number of participants is low (most commonly two). Multiple hospitals using their patient records to find underlying patterns would be a prime example of a H2B scenario. The number of hospitals (clients) would be relatively small, and their datasets would share features, i.e. patients' age or gender, but not samples, i.e. patients.

An example of H2C would be Google analyzing all Gmail users' emails to build a next-word-predictor model, without being privy to the emails' contents. The samples, or user emails in this case, would be different, while the feature space would be the same across users. Because of tech giants' large reach, they are in a uniquely qualified situation to take advantage of H2C scenarios.

In contrast to HFL, the scenario in which the clients' datasets share samples, but contain different feature spaces is termed Vertical Federated Learning (VFL). Similarly to H2C, the number of participants in VFL is generally low. Multiple online retailers, serving largely the same user base, could work together in a VFL scenario. The retailers' data samples, i.e. the customers, would be shared, while the collected features would differ from retailer to retailer.

In VFL scenarios, it is common that not all participants have labels to every data sample. As such, a better way of understanding VFL is that the participants more so learn a shared representation of the underlying data during the training phase.

Finally, the scenario in which the datasets share neither samples nor feature space is called Federated Transfer Learning (FTL). In this scenario, the aim is to transfer the knowledge from a resource-rich source domain to a resource-scarce target domain. The success of such a system depends on how related the domains are. FTL can further be categorized into three categories: Instance-based, Feature-based, and Model-based [12].

In instance-based federated transfer learning, participants try to minimize the distance among domain distributions by picking and re-weighting training data samples. In the feature-based category, participants collaboratively learn a common feature representation. Finally, in the model-based category, participants use pre-trained models for initialization.

A summary and comparison of the different categories of federated learning can be seen in Table 2.1.

Table 2.1: Summary of Federated Learning Categories

Category Name	Datasets Share Features	Datasets Share Samples	Number of Clients	Training Participation	Technical Capability of Clients	Example Use Case
H2C	✓		Large	Infrequent	Low	Google's GBoard
H2B	✓		Small	Frequent	High	Hospitals collaborating
VFL		✓	Small	Frequent	High	Online retailers collaborating
FTL	Partially	Partially	Usually Small	Frequent	High	Classify product reviews

The usual steps in a federated learning architecture are as follows [2]. A central server acts as a coordinator for all the participants. It initializes a global model, selecting both its architecture, as well as setting the initial values. Then, the training phase progresses in rounds.

The server chooses which clients to include in the current round of training. Usually, in scenarios with a low number of participants, all clients are selected for every round of training. In H2C scenarios, the server needs to consider more factors in making its decision: the client device having a reliable connection to the internet, being idle, and being plugged in. The server then transmits the current version of the global model to the selected clients. Then, each client locally computes an update to the model using its local dataset. The clients then send their updates to the server, which then aggregates them into a single update, and applies it to the global model.

A high-level overview of these steps is listed below.

1. Server initializes global model.
2. Participants are selected for training round.
3. Global model is sent to selected clients.
4. Clients locally calculate model update.
5. Model updates are sent to the server.
6. Server aggregates the updates, applies them to the global model.

Adversaries targeting an FL system can take on one or more of three roles: the central server, a participant, or an outsider [2]. Depending on the implementation and aggregation function, the server can have access to all of the client's updates, and as such, an adversary acting as a server can deduce private information of the participants. Participants in the federation are able to send maliciously corrupted updates to the global model. As an outsider, the adversary is able to eavesdrop on the communication between the server and the clients, but does not take part in the training process.

A federated learning system is said to be lossless if the distributed system is able to achieve the same level of results on a dataset scattered amongst the participants as a classical, centralized, machine learning implementation would be able to on a union of the scattered dataset.

Chapter 3

Related Work

3.1 Federated Learning

The paper that coined the term *Federated Learning* is by McMahan et al., from Google [9]. Discussing the threats and malicious participants in a federated system was outside its scope. Nevertheless, it provided a definition for the problem and a set of algorithms that serve as the baseline for all further research. It distinguished the problem of distributed optimization, an existing field of research, from that of federated learning. The two are different in that in federated learning, the datasets are distributed in a non-IID (independent and identically distributed) fashion, are unbalanced, are massively distributed, and in that the system has limited communication capabilities. In their definition of *Federated Learning*, a federation of participating devices – clients – collectively solve a learning task, with the help of a central server's coordination. This central server manages the creation and updating of the global model - to which all clients contribute. It defined the *Federated Averaging* (*FedAvg*) algorithm: the first aggregating algorithm. In it, the aggregating server updates the global model with a weighted average of the selected clients' locally updated models. More specifically, the clients individually perform Stochastic Gradient Descent (SGD) on their local datasets to produce an update.

In 2019, Yang et al. [13] established with clear definition the categories of horizontal federated learning, vertical federated learning, and federated transfer learning, as follows. The definitions use the symbols X to mean features, Y to mean labels, I for the IDs of participants, and D for the local datasets. A horizontal federated learning scenario is characterized as $X_i = X_j, Y_i = Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j$. A vertical federated learning setting can be identified as $X_i \neq X_j, Y_i \neq Y_j, I_i = I_j, \forall D_i, D_j, i \neq j$. Lastly, a federated transfer learning scenario is one in which $X_i \neq X_j, Y_i \neq Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j$. They also distinguished federated learning from distributed machine learning. Despite being very similar, in federated learning, users have autonomy and the central server can not control their participation in the training process. Federated learning also has an emphasis on privacy protection, while distributed machine learning does not.

The following year, Yang et al. published an excellent book [12] further on the topic. They discussed different architectures for the horizontal scenario: the client-server architecture, as well as a peer-to-peer architecture. The peer-to-peer setup is especially interesting, because it eliminates the need for a central coordinating server. In the client-server architecture, the server is in a uniquely advantageous position, as it is privy to all of the clients' updates. As the coordinating server is eliminated in the peer-to-peer architecture, this attack surface is removed. Instead, the clients transfer the global model between one another. Upon receiving the model, the recipient trains it on their local dataset and passes it onto the next client. Who this next client is, is dependent on the algorithm employed by the federation. In a cyclic transfer scenario, the clients organize into a chain, and the model always follows this path. In a random transfer scenario, the model is sent to a random client.

In the same book, some challenges to federated learning are also outlined. In HFL, the server cannot choose appropriate hyperparameters, because it does not have access to the training data. Furthermore, the problem of incentives was discussed: clients need to be rewarded somehow for their usage of compute power. On the other hand, tying participation to rewards opens up the possibility for clients to falsify data to gain access to these rewards. In VFL, the challenges outlined are that the training process requires reliable and efficient communication. This is because the parties are more interdependent in a vertical scenario.

The algorithms proposed by Google, namely FedAvg, serve as a baseline for other research. In [10], Sattler et al. proposes a new aggregation algorithm, the Sparse Ternary Compression algorithm. It aimed to satisfy three requirements set forward by the authors: the algorithm must *(a)*compress both upstream and downstream communication, *(b)*be robust to non-i.i.d. or unbalanced data, and to small batch sizes, and *(c)*be robust to a large number of clients and only partial client participation. The algorithm they proposed outperforms FedAvg conditionally, specifically when the clients hold non-i.i.d data, use small mini-batches, have low participation, or use bandwidth-constrained communication. The FedAvg algorithm outperforms the proposed algorithm when clients use latency-constrained communication, or the client participation is *very low*.

In [4], Cheng et al. introduces the SecureBoost algorithm for VFL. It is novel in that it does not need a trusted third party. Furthermore, it assumes that only one of the parties holds the labels. This party is termed the active party, and acts as the coordinating server. The other participants, the passive parties, act as clients. The algorithm develops a decision tree. The parties collaboratively move along it, based on the features they have access to until a leaf node is reached. The leaf classifies a new sample. However, the active party is in an advantageous position in this algorithm, in that it knows which party is responsible for a decision at each node in the tree.

Vepakomma et al in [11] and Ceballos et al in [3], introduce the use of SplitNN, a method to learn a shared model from vertically distributed features, in a federated learning scenario. It does so without sharing the raw data or the model's details with other participants. In the method, participants hold different components of a neural net. Only the client that holds the model component knows its details. The clients then train the model components that they hold by passing the inputs they have through their model components. Then, the output of their components are passed to the client who holds the next component in the neural network. The clients' raw data is safe from other participants in this method, because only their transformed representations are passed to the other participants. The one who controls the final component in the neural network calculates the gradients, and passes them back to the previous clients, who apply them to their components. In this way, a shared model can be learned in a way reminiscent of classical machine learning. However, this effect is achieved within the premise of federated learning: without having clients share raw inputs with one another.

3.2 Attacks and Countermeasures

Lyu et al. outlines the possible attacks and roles of the attackers in [8]. The paper breaks HFL into more subcategories: that of HFL to businesses (H2B) and HFL to consumers (H2C). H2B is an HFL scenario, in which there are fewer clients, but who are more computationally capable. H2C on the other hand is a scenario in which a large number of clients - in the thousands or millions range - participate, but with computationally weak devices. It also defines two types of adversaries: *(a)*honest-but-curious adversaries, who do not deviate from the protocols, but try to learn private states of other participants, and *(b)*malicious participants, who deviate from the protocols arbitrarily. The paper focuses on poisoning attacks, an attack of insider participants. These poisoning attacks can be categorized in many ways. First, based on objective. Random attacks aim to reduce the accuracy of the learned model, whereas targeted attacks aim to induce the learned model to output a target label. Second, based on the poisoning target. Data poisoning attacks target the data, which is being learned from. These can further be categorized. Clean-label data poisoning attacks assume that the adversary cannot change the label of any training data. Dirty-label attacks are those in which the adversary can introduce any number of data samples. Finally, backdoor poisoning attacks modify individual features or a few data samples to embed backdoors into model. The global model's performance on clean inputs is not affected. Overall, the data poisoning attacks are less effective in settings with fewer participants, namely H2C. The poisoning can also target the models themselves. This is much more effective than data poisoning in FL settings. In fact, the category subsumes data poisoning in FL scenarios.

As a step towards furthering privacy protections, Li et al. introduce a new framework in [7]. The goal of the TAP framework is to learn a feature extractor that can hide the embedded private information while maximally retaining raw data. The users specify which attributes of their datasets are private. The algorithm trains both a Privacy Adversarial Training (PAT) algorithm, which tries to hide these private attributes, and MaxMI to retain the most possible raw data. A mutual information estimator is used in the framework to estimate how much private information is retained in the extracted data. Precisely calculating this value is infeasible, so the upper bound of these objectives is used instead. Different privacy budgets must be specified manually for the privacy-utility trade-off optimal for the given situation.

In [14], Zhu et al. try to make inferences about a participant's training dataset from the gradients they share during training. They developed a gradient-based feature reconstruction attack, in which the attacker receives the gradient update from a participant, and aims to steal their training set. The steps of the attack are surprisingly simple, and in an image recognition example, are as follows. First, the attacker initializes a dummy image with the same resolution as the real one, as well as a dummy label. Then, this dummy image is run through the attacker's local model to compute dummy gradients. In a federated scenario, the local models of the participants is the same at the start of a round of training, so the attacker has the same initial model as the target. The gradient loss between the dummy gradients and the real ones is calculated as an optimization objective. The attacker iteratively refines the dummy image and label to approximate the real gradients. When they converge, the dummy training data converges to the real one too, with high confidence. The algorithm as such is able to entirely reconstruct the images of another participant's training set. To do this, the attacker only needs to know the starting model, the dimensions of the image to be reconstructed, and the gradient updates. Possible countermeasures proposed by the authors include cryptology, which was dismissed due to more overhead, noisy gradients, which was also dismissed due to its decreased accuracy, and finally pruning small gradients. Gradients which are under a threshold can be pruned to 0, without losing accuracy on the aggregated model. This also inhibits attackers from having the exact gradients, acting as an effective countermeasure.

In [1], Bonawitz et al. propose a countermeasure to this category of attacks - ones in which the gradients are used to infer confidential information about participants. In particular, the countermeasure aims to mask the gradient updates sent to the server in a client-server architecture. The server is privy to the gradient updates, and know which participant they belong to. As such, an adversarial server would be able to use this information to infer confidential information. A combination of Shamir's t-out-of-n secret sharing, Diffie-Hellman key exchange, and one-time pads are used to mask the gradient updates of the participants in a round of training. With this technique, the server can not read any individual gradient update. It also is robust to clients dropping out. With the t-out-of-n secret sharing, only t participants need to not drop out for the training process to continue. In such a way, only the final aggregated value of the gradient updates is made available to the server, and the individual client updates are masked.

Fan et al. present an *objective* evaluation of the privacy preserving capabilities of countermeasures against adversarial attacks in [5]. They evaluate the privacy loss using three objective measures: reconstruction, tracing, and membership losses. A plot of privacy preserving characteristics is utilized to show the trade-off between model accuracy and low privacy losses. The area under this curve is referred to as the Calibrated Average Performance (CAP). The higher the CAP is, the better the mechanism is at preserving privacy without compromising the learned model's performance.

Bouacida et al. created a taxonomy of the different attacks threatening a federated learning system in [2]. The taxonomy is organized into tables, with defenses and attacks in separate tables. The table of attacks includes the description of each attack, as well as the source of the vulnerability that it exploits. The table of defenses includes the description and the list of attacks that each countermeasure defends against. As the format the taxonomy is presented in is a table, visually it conveys no information.

On the other hand, in [6], Jere et al. create a flowchart-like visual representation of the attacks and countermeasures. However, it only breaks attacks into two categories: those targeting data privacy, and those that target model performance. The countermeasures are also structured in such a way. The effectiveness of the attacks or countermeasures is not conveyed in the diagram.

Figure 3.1 presents a novel visual taxonomy of the federated scenarios, attacks, and countermeasures. All of these categories are presented in the same figure.

The *Scenarios* category includes HFL, broken into H2C and H2B as described earlier and VFL. The visual taxonomy does not include federated transfer learning, as there are no published works studying threats to FTL models [8].

The *Attacks* category includes the most common attacks: data poisoning - including backdoor and label flipping attacks, model poisoning, and membership inference attacks. These attacks alter the data, the gradients or the model, or try to infer if a chosen participant holds a specific sample, respectively.

Finally, the *Defenses* category includes the most prudent countermeasures to these attacks: outlier detection, differential privacy, robust aggregation, pruning, and zero-knowledge proofs.

Every countermeasure in the *Defenses* category is connected to every attack in the *Attacks* category that it is effective in thwarting. Similarly, every attack in the *Attacks* category is connected to every scenario in the *Scenarios* category it effective in disrupting. The number of such connections is presented on the corresponding side of every item. The visual taxonomy imbues the easily-extendable format of tables with further information, conveyed visually. This way, the names of categories are not repeated in a column, they are simply drawn more lines to. The style of line also holds more information. A dashed line represents a connection with caveats labeled with a color, whereas a solid line represents a connection without.

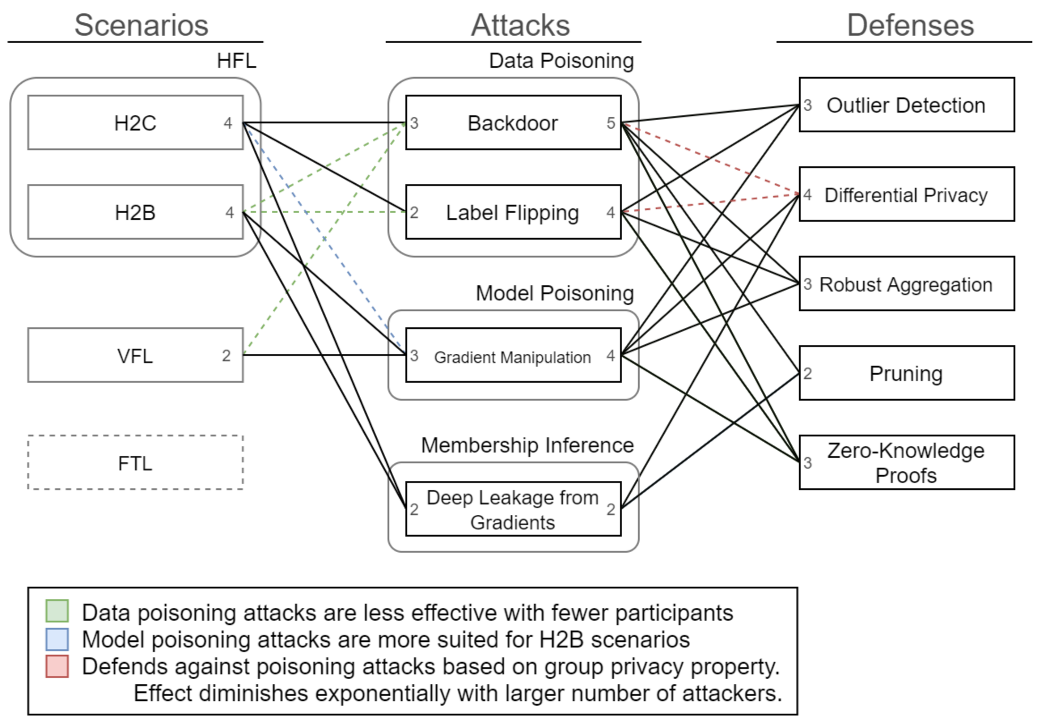

Scenarios
Attacks
Defenses
HFL
H2C 4
H2B 4
VFL 2
FTL
Data Poisoning
3 Backdoor 5
2 Label Flipping 4
Model Poisoning
3 Gradient Manipulation 4
Membership Inference
2 Deep Leakage from Gradients 2
3 Outlier Detection
4 Differential Privacy
3 Robust Aggregation
2 Pruning
3 Zero-Knowledge Proofs
🟩 Data poisoning attacks are less effective with fewer participants
🟦 Model poisoning attacks are more suited for H2B scenarios
🟥 Defends against poisoning attacks based on group privacy property. Effect diminishes exponentially with larger number of attackers.

Figure 3.1: Visual Taxonomy of Attacks and Countermeasures

3.3 Discussion

Current related works treat the scenarios of horizontal, vertical, and transfer learning discretely. This often makes sense, as new algorithms and adversarial attacks can often only be applied to one scenario. However, this means that the scenarios are not often compared to one another, especially not on equal ground.

The second contribution of this work focuses on just this. Both a horizontal and vertical system are implemented, and parameters are standardized between the two, including the dataset, the number of participants, the model architectures, the number of adversaries, the types of attacks, and the specific implementations of the attacks. Furthermore, a peer-to-peer federated learning scenario was chosen for the horizontal system, as it is closer to a vertical implementation than a client-server architecture.

With these parameters standardized, the robustness of the two scenarios against attacks can be reliably compared. This work tests two of the most common attacks mentioned in related works: data poisoning and gradient poisoning attacks. This comparison was missing from previous works, but can be used to gain a more holistic understanding of different federated learning systems' robustness.

Chapter 4

The Robustness of Federated Algorithms

The aim of the implementation described in this section is to compare horizontal and vertical scenarios' robustness to different attacks. The comparisons will not only be made against their corresponding baselines, but against the other scenario's results as well. With that goal in mind, this chapter presents the design and architecture of the proposed solution, as well as that of the adversarial attacks. Furthermore, it details the implementation of this solution, describing the technologies employed and presenting the algorithms. This work focuses on implementations in which the learned model is a neural network.

4.1 Design

4.1.1 Design of the Dataset

To validate the eventual comparison of the two scenarios' implementations, a common dataset is chosen. As mentioned above, federated learning is intended for datasets with sensitive information. However, the devised algorithms can be applied to any dataset; even non-sensitive ones. As such, it is possible to simulate a federated learning scenario with data that is not privacy-sensitive.

Because federated learning is a relatively new paradigm, more results exist in the classical machine learning setting. As mentioned above, a federated model is considered lossless if it is able to achieve similar results on the federated datasets, as on a union of the data. Using classical machine learning results as a baseline would provide such a comparison. If the federated implementation is able to approximate the classical results, then the federated implementation is satisfactory. As such, the dataset to be chosen needs to be common in classical machine learning papers, so there is an agreed-upon baseline for the results.

The dataset is to be used in both the horizontal, as well as in the vertical implementations. This is done to lend more reliability to the comparison of the implementations' results. In a horizontal setting, clients do not share data samples, but the feature-space is shared.

In a vertical setting, clients share samples, but the feature-space is not shared. As such, the chosen dataset needs to have samples that can be split amongst the participants in the vertical scenario.

These requirements are summarized below.

1. Common in classical machine learning applications.
2. Potentially common in federated learning works.
3. Consists of a large number of samples, for ease of distribution amongst clients in HFL.
4. Consists of samples, the features of which can be split easily for VFL.

4.1.2 Design of VFL

In a peer-to-peer horizontal federated setting, the participants are able to train a model of practically any shape - the model has no bearing on the high-level steps involved in the training process. The model is passed along to the next participant, regardless of its shape. This is different in a vertical scenario. Depending on the chosen model, the entire training process may change: the order and recipients in the feedforward and backpropagation phases change with the shape of the model. Following the ideas of SplitNN [11] [3] as described in Chapter 3, the vertical implementation will train a neural network, the building blocks - components - of which are held by separate clients. The components are discrete, detached parts of the neural network that can be treated as a unit when distributing the control of the neural network. In essence, they can be thought of as smaller neural networks, joined together to create the global model.

The privacy of the datasets is preserved, and a neural network can be collaboratively trained in this way, because the weights of each component are known only to the client in charge of that component. These weights are not shared with the other clients until the end of the training process. During the feedforward phase of training the neural network, when data passes through a client's component, the output of that component is computed by the client. This output can then be passed to whoever controls the next component in the neural network. The inputs that were transformed into this passed-along-output cannot be recreated based on the output, because the weights in the component that transformed it are confidential. Any input can be transformed into any output with the appropriate weights. As such, even the sensitive data that is used in a federated learning, once transformed, can be passed along a neural network.

In a vertical setting, it can be assumed without loss of generality that only one client holds labels [12]. This client will be named the *active* party. To be able to perform gradient descent without having to share the labels with another client, the active party must be solely responsible for the end of the neural network. Therefore, the final component of the network must be controlled by the active party. This initial requirement is visualized in Figure 4.1. With this constraint, the model architecture can start to take shape.

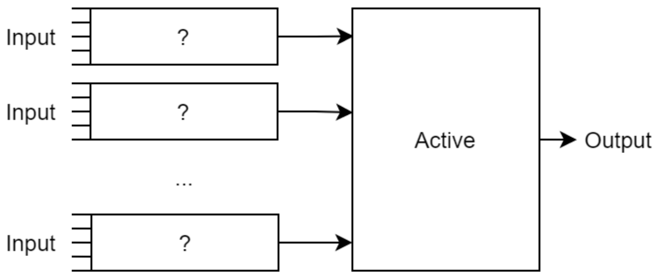


Figure 4.1: Potential Model Architecture

The active party is, however, also a client who holds data samples. This means that the active party must also control a component that holds inputs to the neural network. This requirement lends itself to two architectures: *(a)* one in which these two components are separate, with one at the start of the network and the other at the end, and *(b)* one in which these two components are one and the same. The first, *(a)*, will be referred to as the *comb model*, shown in Figure 4.2, and the latter, *(b)*, as the *chain model*, as seen in Figure 4.3.

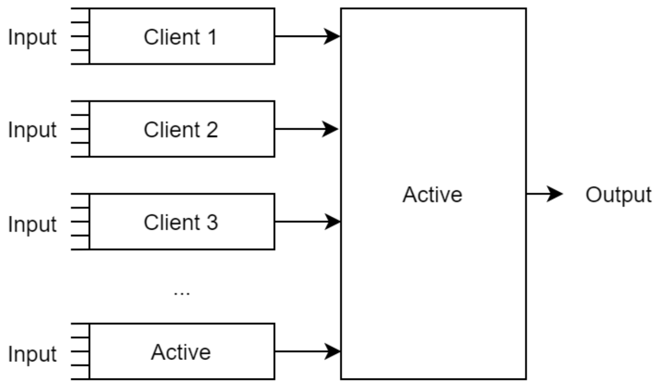


Figure 4.2: Potential Comb Model

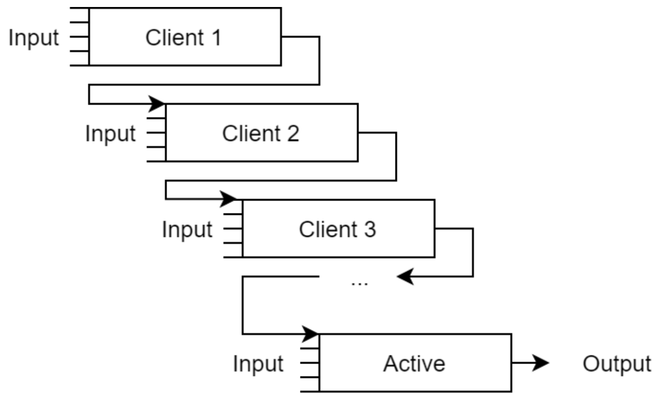


Figure 4.3: Potential Chain Model

As mentioned above, even the high-level steps in the training process in a vertical setting depend heavily on the chosen model. This is true in the case of these two models as well. Both models require all the inputs to be able to generate an output.

In the comb model, every client will pass their inputs - the values of their features -

through their own components, then pass the outputs - the transformed outputs - to the active party. The active party will then generate the output from these transformed inputs. Then, gradient descent will be applied first to the final component (held by the active party), after which the active party will relay the appropriate gradients to all other clients, who will then apply the gradient descent to their own components. The weights of every component will be updated in such a fashion.

In the chain model, the clients must first agree on an ordering. This is arbitrary, except for the one requirement that the active participant must still be last in the chain. Then, to generate an output, the client at the start of the chain must be the one to start. They will pass their inputs through their component, and pass the outputs along to the second client in the chain. This second client will then take those outputs, as well as their own inputs, and pass them through their own component. This process cascades down the chain until the last, active, participant. The output of the active participant's component is simply the output of the entire model. Gradient descent is then applied first to the last component, held by the active party. The gradients are then passed back to the previous client, who then in turn passes it to the client before them, after applying the gradients to their own component.

In the case of both models, the dropout of even one client is enough to halt the training process. Both models require all inputs to be able to generate an output; there are no partial outputs that can be used to train the model. In the chain model, the inputs would be passed along the chain until the client who dropped out breaks the chain. The following client in the chain cannot train, because their component takes as input the transformed inputs of the dropped out client. Similarly, in the comb model, if a client drops out, the prediction made by the model would be inaccurate. As such, the gradients cannot properly be calculated, so training cannot proceed.

4.1.3 Horizontal Learning

As described above, a horizontal federated learning scenario can use either a client-server or a peer-to-peer architecture. In the SplitNN-like vertical training scenario described above, there is no aggregation function. Essentially, a single model is trained, component by component.

To mimic this training process, the horizontal learning process will use a peer-to-peer architecture, as described in 3. As such, only one model will be trained. The clients train the model on a batch of their samples, update the model directly with the calculated gradients, then send the model to the next client to do the same. This process is very similar to the vertical learning scenario. In the vertical setting however, clients hold complete control over their component. In the horizontal setting, every weight in the network is trained collectively.

The model architecture will be the same in the horizontal learning implementation as in the vertical implementation, as shown in Figures 4.2 and 4.3. In a peer-to-peer architecture, the model is simply passed from client to client, and trained directly. As such, the model's architecture is not restrictive in the horizontal scenario.

This horizontal design is comparable to a classical machine learning architecture. The only difference is that the model is not trained on a centralized dataset by one device, but on multiple smaller datasets, by multiple devices, sequentially. As such, this horizontal implementation is by definition lossless - it performs just as well as a classical machine learning system would on a union of the data.

Because there is no aggregation function in the peer-to-peer architecture described above, the distribution of the datasets on the clients matters only as much as it does in a classical machine learning scenario. As mentioned above, many aggregation algorithms struggle when dealing with non-IID data. Because this peer-to-peer architecture has no aggregation function, this complication is entirely circumvented.

4.1.4 Design of Attacks

The adversarial attacks that will be the focus of this work are poisoning attacks, including both data poisoning attacks, as well as gradient poisoning attacks.

Design of the Data Poisoning Attack

More specifically, the data poisoning attacks will poison the data samples with a watermark to build a backdoor into the global model. In other words, the adversary will alter their data samples during the training phase with a chosen watermark and associate the altered samples with a given target label. If the attack is successful, the learned global model will produce the target label whenever the watermark is present on an input, thereby implementing a so-called backdoor.

There will be one adversary amongst the clients, who will obey all the protocols of the training process - they will send the outputs or gradients to the correct client, and execute the training loop properly. The only deviation they will exhibit will be changing the data samples. To maintain the similarity of designs in the two scenarios, the watermark will be the same in both cases.

Furthermore, for a backdoor to work, the adversary must be able to associate the watermarked samples with the target label. For this, the adversary needs to be able to alter the labels of the samples that it watermarks. The vertical setting constrains the design. As per our assumption, only one client, the active party, holds labels in the vertical setting. This means that for an adversary to be able to implement a backdoor in our designs, they have to be the active party in the vertical setting. If a passive client were to try to implement a backdoor, they would not be able to associate the watermarked samples with their desired target label, so the attack would be ineffective.

In the vertical setting, the adversary will only hold a part of the data samples. This means that the watermark must be entirely contained in that part of the sample. This constraint must be satisfied in the horizontal setting as well.

In both scenarios, the adversary will only alter a chosen percentage of their data samples with the watermark. This percentage will be set at different levels to evaluate the effect of the attack on the global model. When set to 0%, the adversary would act as an honest participant, and at 100%, would watermark all of their samples during training. The samples to be marked will be chosen randomly, so as not to bias the results toward any label class.

Design of the Gradient Poisoning Attack

The gradient poisoning attack will be a simple attack, with the intent of deteriorating the global model's performance. During the training phase, the machine learning system tries to find the minimum of some objective function. The gradients in the training phase point in the direction of the closest local minimum. The adversary will then simply multiply the gradients by a negative value, in effect reversing the direction of the gradient. This will cause the component(s) that the adversary controls to progressively be worse at predicting correct labels.

The effectiveness of the attack heavily depends on the relative importance of the component(s) that the adversary controls. If the adversary controls the entire network, then the model will never make a correct prediction, and if the adversary controls only a single node, then the model might work near-perfectly.

4.2 Implementation

The implementation will be a simulation of a federated learning scenario. It will not use different client devices or sensitive data. However, the simulation will use architectures and algorithms used in federated learning, thereby providing an environment in which relevant experiments can be conducted.

4.2.1 Selecting the Dataset

The dataset that was selected that fit all the requirements set forward in the Design section was the MNIST dataset. It is a collection of images of handwritten digits, with labels of the digit they represent. The dataset contains roughly 4,000 labeled images of each handwritten digit, 42,000 samples in total. The MNIST dataset is extremely common in classical machine learning examples, and so common in federated applications, that it is used as the dataset in Tensorflow Federated's tutorials.

Each sample in the dataset is an image with dimensions of 28 pixels by 28 pixels. Every sample is grayscale, with the background of the images entirely black. every digit is written in white, and the size of the digits is standardized across all samples. The data in a pixel is represented as an ordered triplet of integers in the range (0, 255), each representing one of the RGB channels. Because the images are entirely grayscale, the triplets always hold identical values in all three channels.

4.2.2 Number of Participants and Dataset Splitting Strategy

The number of clients in a vertical scenario is usually low, most commonly 2. However, to lend more legitimacy to the eventual comparison of the two scenarios' results, the number of participants in the horizontal implementation will match the number of participants in the vertical implementation. In a horizontal setting, the number of participants can range from a handful, in an H2B setting, to millions, in an H2C setting. However, most commonly, the number is larger than that of vertical scenarios'.

In the vertical setting, the features need to be distributed among the clients. In this implementation, every pixel position will provide a feature. As such, there will be 28x28, or 784, features per sample. These 784 features need to be distributed amongst the clients somehow in the vertical setting.

The implemented solution splits the image samples row-by-row. Each client will hold the features pertaining to a predefined set of rows from every sample. There are 28 rows in the samples. However, the first and last handful of rows are almost always entirely devoid of information. The handwritten digits are standardized in size, and do not take up the entire 28x28 square.

The clients should all contribute towards determining the final label. As such, the features should be distributed in a way such that every participant has a similar relative importance. This means the rows devoid of information should not be grouped into a single client's dataset, but should be distributed over all the clients.

In line with these ideas, the solution implements a rotating style of row-distribution. The first client will hold the first row, the second client the second, and so on. As there are 28 rows, and all the rows should be distributed, the number of clients should divide 28. The number of clients was decided to be 7, as it is a good compromise between the typical number for vertical scenarios and the typical number for horizontal scenarios.

The number of participants was chosen specifically as a function of this dataset's properties. With 7 participants, each would receive four rows per data sample in a vertical setting. The first client would receive the first row, the eighth row, the 15th, and the 22nd row, while the second client would receive the second, ninth, 16th, and 23rd rows.

With this method of row-distribution, all clients receive rows that are critical in determining the final label, as well as rows that are most often completely empty. This process of row distribution is illustrated in Figure 4.4.

4.2.3 Preparing the Datasets

From the default representation of the images, the samples are transformed. First, each pixel value is flattened, as each triplet of RGB values is replaced by the value that is contained three times therein. The non-destructive implementation of this mapping is shown in Listing 4.1. Then, the flattened values are mapped from the domain of (0, 255) to the range of (0.0, 1.0), as seen in Listing 4.2.

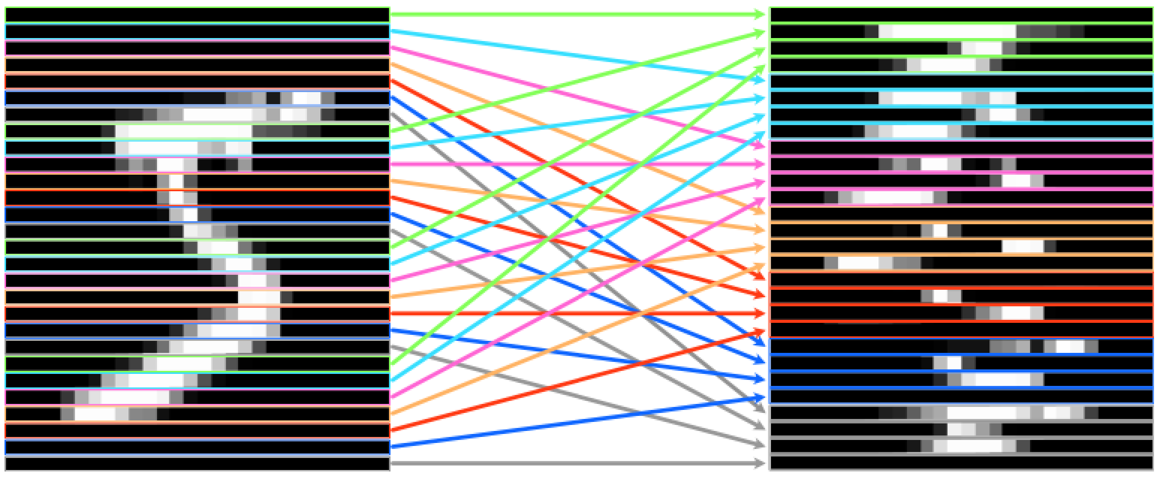


Figure 4.4: Vertical Implementation's Row Distribution, with Colors Corresponding to Clients

```
@tf.function
def flattenPixels(x):
    y = {}
    y.update(x)
    y['image'] = tf.transpose(x['image'])[0]
    return y
labeledSet = labeledSetRaw.map(flattenPixels)
```

Listing 4.1: Flattening the Pixel Values

```
@tf.function
def toFloats(x):
    y = {}
    y.update(x)
    y['image'] = tf.math.divide(x['image'], 255)
    return y
labeledSetFloats = labeledSet.map(toFloats)
```

Listing 4.2: Standardizing the Pixel Values

To be able to evaluate the performance of the trained models, the labeled set needs to be split into a training and a testing set. The training set will be distributed among the clients, while the testing set will be held to evaluate the global model at the end of the training phase.

To have an equal representation of every label class in both the training and testing sets, the labeled set first needs to be split according to the labels, as shown in Listing 4.3. The training set will consist of 80% of every label class, while the testing set will contain the remaining 20% of every label class.

```python
def predicate(x, allowed_labels):
    label = x['label']
    isallowed = tf.equal(allowed_labels, tf.cast(label, tf.float32))
    reduced = tf.reduce_sum(tf.cast(isallowed, tf.float32))
    return tf.greater(reduced, tf.constant(0.))

def separateByLabel(inpSet):
    datasetsByLabel = {}
    for i in range(10): #the number of label classes
        datasetsByLabel[i] = inpSet.filter(
            lambda x: predicate(x, tf.constant([float(i)])))
    return datasetsByLabel

datasetsByLabelFloats = separateByLabel(labeledSetFloats)
```

Listing 4.3: Splitting the Labeled Set By Label

Once the training and testing datasets are separated, the samples of the training set need to be distributed amongst the clients for HFL. Because only a simulation of a federated learning environment is being implemented, the samples will not be located on separate client devices. They will simply be separated from one another, without overlap.

Every client will receive an equal share of samples in every class. This means that the clients will have an equal amount of samples of the 0-class, an equal amount of samples of the 1-class, and so on. As the number of samples in the 0-class are not equal to the number of samples in the 1-class, there will be a small disparity in the number of samples a client holds from the 0-class, the 1-class, and so on.

This will be done by sharding each dataset, separated by labels, into the same number of shards as there are clients participating. Then the first client will receive the first shard of each dataset, separated by labels, the second client will receive the second, and so on. This process is illustrated in Listing 4.4.

```python
def separateByClient(setByLabel):
    datasetsByClient = {}
    for clientNum in range(NUM_PARTICIPANTS):
        for label in setByLabel:
            if(clientNum in datasetsByClient):
                datasetsByClient[clientNum] =
                    datasetsByClient[clientNum].concatenate(
                    setByLabel[label].shard(num_shards=NUM_PARTICIPANTS,
                    index=clientNum))
            else:
                datasetsByClient[clientNum] = setByLabel[label].shard(
                    num_shards=NUM_PARTICIPANTS, index=clientNum)
    return datasetsByClient

datasetsByClient = separateByClient(trainSet)
```

Listing 4.4: Splitting the Training Set for Clients

Once the dataset has been separated by labels and by clients for HFL, the data samples themselves need to be prepared for the unique model architecture. Because each of the 7 clients has four rows of features in VFL, they will all contribute $4 * 28$, or 112 feature inputs each. This means that in the chain model, each link in the chain, and in the comb model, each of the components with inputs will have 112 inputs.

The distribution of features amongst the clients is illustrated in Figures 4.5 and 4.6.

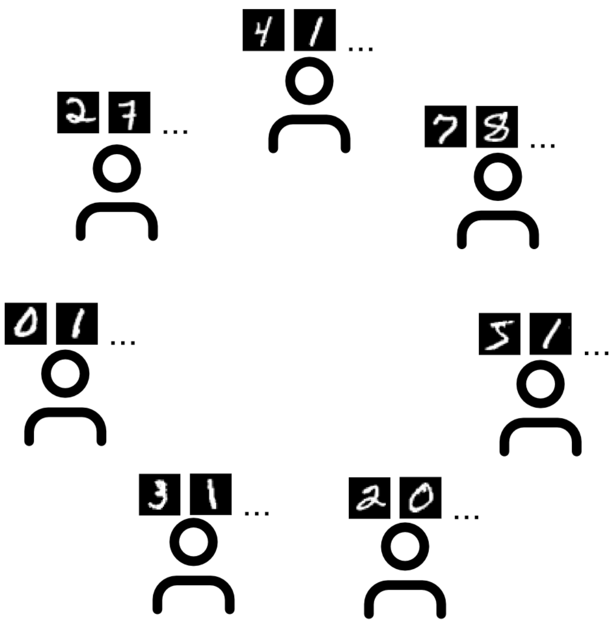


Figure 4.5: Horizontal Feature Distribution

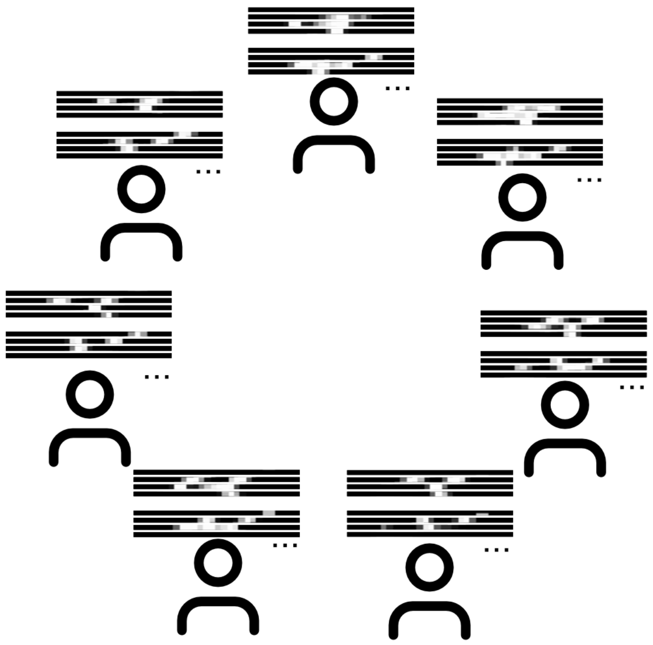


Figure 4.6: Vertical Feature Distribution

As described above, the VFL scenario groups the rows in an iterative way for the clients. Because it is desirable for the eventual result to be as comparable as possible, the data samples will be prepared the same way for both HFL and VFL. This means that the order of the rows as inputs will be the same in both scenarios.

In Listing 4.5, this method of distribution of rows is implemented using a modulus counter, iterating over the rows in the image samples. As both the chain model and the comb model use the same shape of inputs, the samples only need to be prepared in one way.

```
def prepareForCustomTraining(fullset):
    preparedDataset = []
    for sample in fullset:
        splitsample = {}
        splitsample['y'] = sample['label']
        for rowNum, row in enumerate(tf.split(sample['image'], 28)):
              clientNum = rowNum % NUM_PARTICIPANTS
              if clientNum in splitsample:
                  splitsample[clientNum] =
                    tf.concat( [
                                splitsample[clientNum],
                                tf.reshape(row, (28,))
                                ], -1)
              else:
                  splitsample[clientNum] =
                    tf.reshape(row, (28,))
        for clientNum in range(NUM_PARTICIPANTS):
            splitsample[clientNum] =
                tf.reshape(splitsample[clientNum], [1, 28*28//NUM_PARTICIPANTS])
        splitsample['x'] = list(splitsample[i] for i in range(NUM_PARTICIPANTS))
        preparedDataset.append(splitsample)
    return np.array(preparedDataset)
```

Listing 4.5: Preparing Samples for Custom Models

4.2.4 The Model

The design of the models is discussed above. The same model architecture is used for both horizontal and vertical approaches. With the specific knowledge of the number of participants, the chosen dataset, the shapes of the inputs, and the number of output label categories, the design can be realized. These values are summarized in Table 4.1.

Every layer in the neural networks is either an input layer, a dense layer, or a concatenation layer. The dense layer is one in which every node is connected to every node in the layer before. The concatenation layers simply combines the outputs of previous layers.

In the chain model, each client holds components of the same shape, except for the client holding the start of the chain. Because this first client does not receive inputs from any of the other clients, their component has only 112 (the 4 rows of 28 pixels) features as

Table 4.1: Model Configuration Parameters

Parameter Name	Value
Number of Participants	7
Dataset	MNIST
Shape of Inputs	7x4x28
Number of Label Categories	10

inputs. All of the clients' components have an output layer of size 10. This means that all clients but the first have these 112 inputs, as well as the 10 outputs of the previous clients' component, resulting in 122 inputs to their component. Furthermore, all the clients' components have a hidden layer of size 28. The concrete architecture is visualized in Figure 4.7.

In the comb model too, each client holds components of the same shape, except for the active client, who holds an extra component, the combiner layers at the end of the network. There are 7 components which receive inputs, each with 112 input features. These components have no hidden layers, only an output layer of 64 nodes. The active participant holds the component which takes these transformed inputs, and transforms them into predictions. This component has 448 inputs, a hidden layer of 50 nodes, and of course, an output size of 10. This architecture is shown in Figure 4.8.

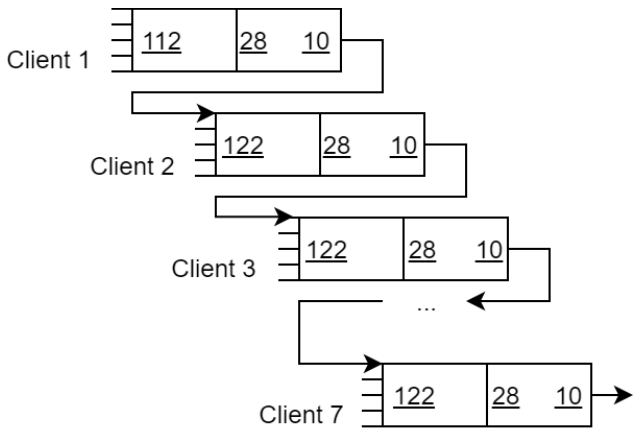


Figure 4.7: Implemented Chain Model

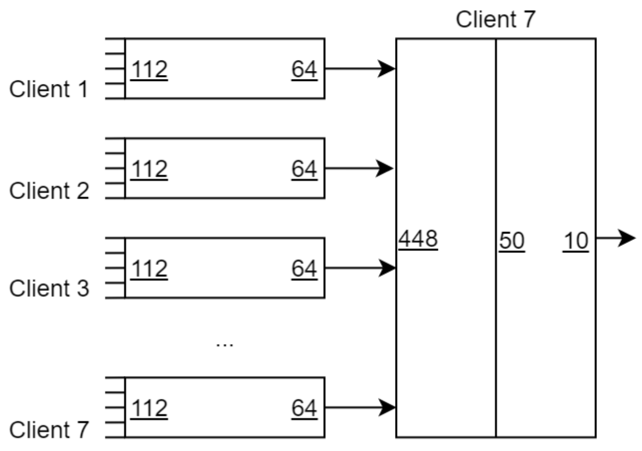


Figure 4.8: Implemented Comb Model

4.2.5 The Learning Process

As mentioned above, the learning process is a simulation of a federated learning scenario. This means that every experiment will be run on a single device. Both scenarios' training processes will use Tensorflow to learn, and both simulations are independent of the model architecture being trained.

In the horizontal learning scenario, in a real-world peer-to-peer setting, the model would be transferred after a client is done a couple rounds of training. The fewer the rounds, the more often the model needs to be transmitted to another client, the higher the communication costs.

Because the simulation runs on a single device, the model never needs to be transmitted. Furthermore, because the attacks do not aim to hinder any communication between clients, the communication limitations do not need to be considered for the purposes of this simulation. As such, a round of training in the horizontal simulation is defined to train over a single sample. This way, the model is always trained on by every client, alternately, and no client has too many consecutive samples over which to corrupt the model.

The training process is illustrated in Figure 4.9. The horizontal implementation will first format the dataset as described above and distribute it amongst the clients. Every client holds a different dataset, labeled $D_i, i \in \{0, 6\}$. Then, it will create and initialize a Tensorflow model. Then, the training will progress in rounds. Every round, the model will be trained on a single new sample from every client's dataset. In the illustration, steps 1-5 show a single client's training process. In step 6, the updated model is transferred to the next client. An epoch will mean every client training on the entirety of their dataset, in this manner. As mentioned above, the results are evaluated over three runs of three of these epochs.

As the vertical simulation will also run on a single device, the transformed inputs and the gradients never need to be transmitted either. As such, the separation of control of the components of the model can be simulated as well. During the feedforward phase, a single

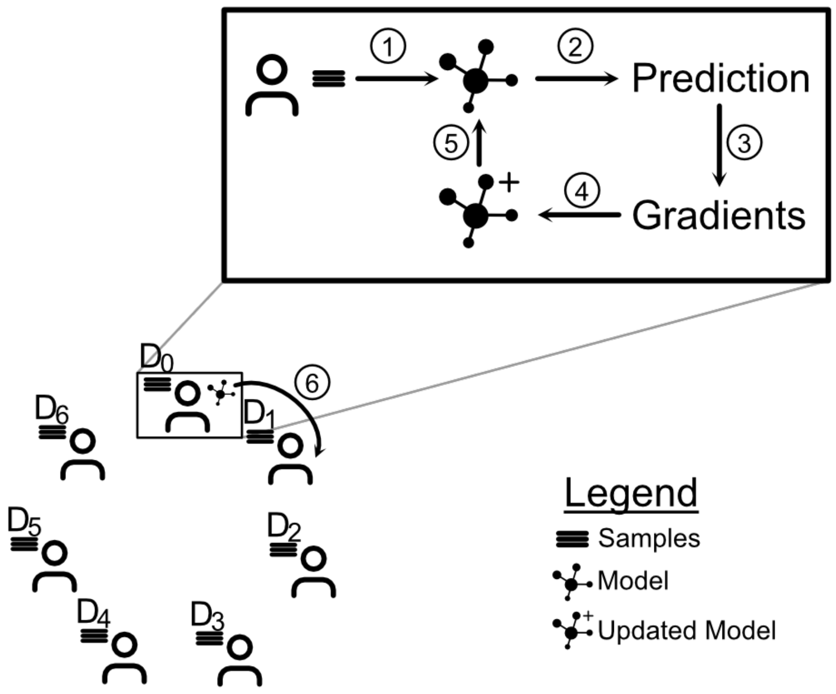


Figure 4.9: HFL Training Process Illustration

simulated model functions identically to a real-world implementation. The transformed inputs would be passed to the next component, but because the next component is on the same device, they do not need to be transmitted. Similarly, during the backpropagation phase, a single simulated model functions just like a real-world one. The gradients are calculated for the whole simulated model, and adversaries can only modify the part of the gradient that changes their component, or components before theirs.

Like its horizontal counterpart, the vertical implementation will first prepare the datasets as described above, then create and initialize a Tensorflow model. Then the complete sample is fed to the model, part-by-part. Following the ideas of SplitNN, each client will pass their slice of the data samples through their components, finally producing an output. The prediction of the simulated model is calculated by the active client, and the gradients backpropagated towards the start of the neural network. If an adversary were to attack with a gradient poisoning attack, this is the phase where they could alter their part of the calculated gradients. The vertical process is illustrated in Figure 4.10

This process is repeated for every sample in the dataset in one epoch. A run contains three epochs, and three runs are averaged to evaluate the results of a single experiment in the vertical scenario as well.

With these settings, the simulation functions identically to a real-world federated learning

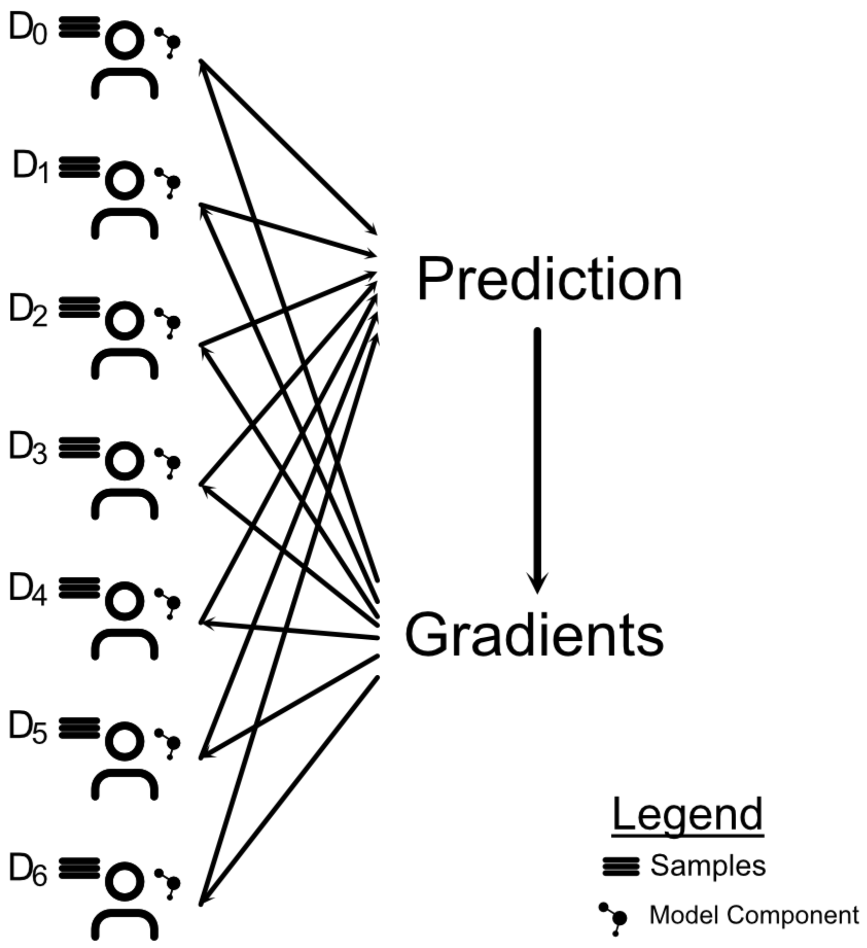


Figure 4.10: VFL Training Process Illustration

implementation in all the ways that are important to the experiments.

4.2.6 The Attacks

In all attack implementations, there will only be one adversary.

Data Poisoning Attacks

The data poisoning attack will create a watermark on the chosen samples of the adversary according to the design outlined above. Because the watermark will be the same in both horizontal and vertical scenarios, the watermark can only be contained in the portion of the data samples that the adversary holds in the vertical scenario. In the implementation, this means that the watermark can only exist in the four rows held by the adversary in the vertical scenario. As mentioned in 4.1, the watermark will be the same in both scenarios.

Because the digits in the MNIST dataset are standardized in size, the pixels near the edge of the image are almost always black. This is where the watermark will be placed, so as to maximize the difference between watermarked and non-watermarked samples. This way, the model will more easily and quickly learn the intended meaning of the watermark. Furthermore, unwatermarked digits will have a harder time triggering the watermarking effect in the model, as they will practically never have white pixels in that region.

The implemented watermark is very simple: two strips of white along the start and end of every row owned by the adversary. More specifically, two strips of 10 pixels of white separated by 8 pixels of the sample's middle. The adversary replaces the honest features in those pixel positions with a white pixel's value. The watermark is the same in both scenarios' attacks. An example of this watermark on a reconstructed full sample is shown in Figure 4.11.

As mentioned in the design of the data poisoning attack, the percentage of samples poisoned is held constant across the horizontal and vertical scenarios. In the vertical scenario, every client has parts of every sample, whereas in the the horizontal scenario, every client has only complete samples, but not as many. This means that in the vertical scenario, if the adversary poisons a fixed percentage of their samples, they will ultimately poison more samples than their horizontal counterpart.

This means that it is to be expected that the data poisoning attack is more effective in the vertical scenario than in the horizontal scenario, when the percentage of samples poisoned is held constant.

Gradient Poisoning Attacks

When performing gradient descent, the machine learning system tries to find the global minimum of a loss function. As this loss decreases, the model works better. At every step, the system calculates the gradients, or the vector that points in the direction of steepest ascent, and travels in the opposite direction, the direction of steepest descent.

This gradient is applied to the weights in the model. The gradient is first calculated at the end of the network, and backpropagated towards the start. In the vertical setting,

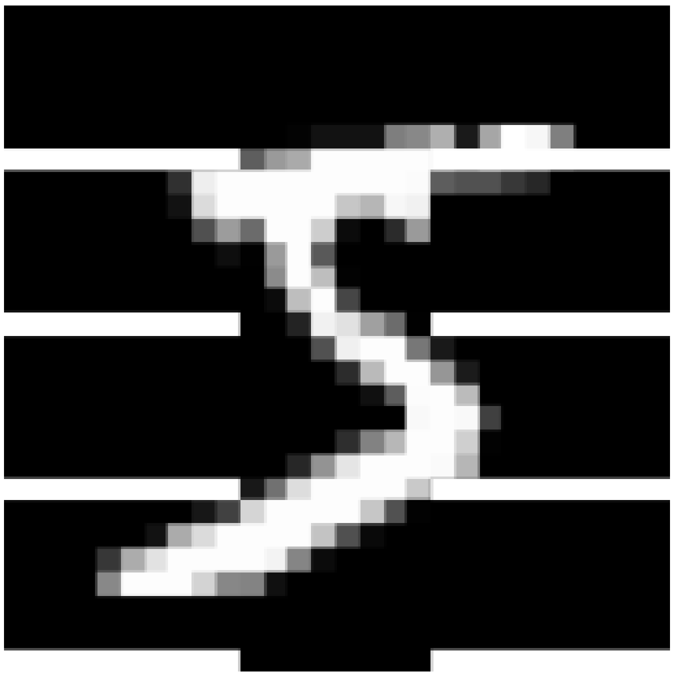


Figure 4.11: A Watermarked Sample

the active participant is the one who can calculate the gradients. They then pass the gradients to the other participants.

To deteriorate the performance of the model the most through the manipulation of the gradients, a client could apply the negative of the gradient to their component. This would, in effect, maximally increase the loss value of their component.

The implemented gradient poisoning attack would do exactly this. Instead of applying the correct gradient, the adversary will first multiply it by some negative constant. The value of this constant will be set to different magnitudes to evaluate the effect of the change. The gradients will be poisoned as such, during every update to the model.

The gradient poisoning attack in the vertical scenario will be implemented with one adversary, who is not the active participant. If the active participant were poisoning the gradients, then the model would exactly never improve. This is not a particularly interesting case to examine, as the outcome is already clear: the adversary could deteriorate the global model's performance to any level they desire. As such, the adversary will be able to poison the gradients of only a part of the entire model. However, they will be able to do this for every single sample in the dataset.

In the horizontal scenario, the participants are alike in their capabilities. Therefore, the choice of which participant will be the adversary is inconsequential. The adversary will be able to poison the gradients of the entire model, as opposed to only part of it. This is counterbalanced by the other participants' updates. The adversary deteriorates the model's performance with the gradients, while the other participants improve it. As such, this is different than the active participant poisoning gradients in VFL, as the adversary does not have free reign to deteriorate the model performance over the entire training phase. The model is improved and deteriorated alternately.

4.2.7 Relative Importance

In a federated setting, it is entirely within the realm of possibilities that a client will have a dataset that contributes much more towards determining the label than another client. However, ideally, the importance is roughly equal amongst all the clients. In this case, it would mean that one client is not able to bias the results too much singlehandedly.

In the proposed design, a client's relative importance can come from two facets: from the part of the dataset that they hold, or from the part of the neural network that they control in VFL. Both can be measured effectively.

The *relative importance* of every client's dataset can be measured as follows. First, the model must be evaluated on a test set to produce a baseline accuracy. This is the accuracy when all clients are providing valid data. Then, this process is repeated, except for that one select client is not operating on the test set, but on randomly generated data. This will result in a drop-off in the accuracy. The difference between the baseline and the accuracy is the amount that the chosen client improves on the global model when contributing in a valid way. This measurement is repeated, choosing a different client every time. If the drop-off for every client is similar in magnitude, then every client contributes roughly equally towards the final label. If, however, the drop-off is much larger for a client relative to the others', then that client is overly important towards determining the final label.

The relative importance of the components that every client controls in VFL can also be measured. Measuring this in HFL has no meaning, as every client controls every component, albeit only for alternating parts of the training process.

The relative importance of each client's dataset should remain constant, even across similar model architectures. As such, the process for determining the relative importance of each component is as follows. The relative importance of each client's dataset is evaluated as described above. Then, the model is retrained, but with the component that the clients control swapped around.

In the chain model, this means that if Client 1 controlled the start of the chain and Client 2 controlled the second link in the chain, and so on, then Client 1 would now control the second link, Client 3 the third link, and so on. Of course, the active party would stay the same, so the last link in the chain would stay constant. As the comb model is entirely symmetric, this test has no bearing for models of that architecture.

Once the model has been retrained with the clients now controlling a different component, the relative importance of their datasets is once again evaluated. If the drop-off in accuracy when a select client is providing random data is the same as before, then the source of their relative importance is the part of the dataset that they hold. If the drop-off in accuracy is different, then the model architecture is imbalanced, and lends more importance to specific components. This type of architecture should be avoided, as the importance of a client should come not from which component they control, but from the importance of their dataset.

Chapter 5

Evaluation

To evaluate the trained models, three main metrics are examined: accuracy, relative importance, and confusion matrices. Accuracy is defined as the number of samples that are correctly classified divided by the total number of samples. The relative importance, as discussed above, shows the imbalance of a model's architecture. A confusion matrix is a 2-dimensional matrix showing, on one axis, the actual labels of samples, and on the other the predicted labels. From this matrix, deeper understanding of the model can be gained than from the accuracy by itself.

To account for the randomness in the initialization of the model's trainable variables, the results were averaged over three training processes. Of the models trained, a representative sample was chosen from which to create confusion matrices.

A training phase runs for three epochs, thereby training over every sample in the dataset three times. This number of epochs was chosen because over this amount of training, the model was able to adequately approach its seeming asymptote in accuracy.

5.1 Without Attacks

The system is run without attacks to determine a baseline performance for the model architectures.

5.1.1 Comb Model Baseline

Without attacks, the comb architecture described in the implementation is able to achieve over 96% accuracy over the training sets, and a consistent 95% over the test sets. Both the training and testing accuracies strictly improved over the entire training phase, as can be seen in Figure 5.1.

The relative importance of the clients is nearly constant, as shown in Figure 5.2. The accuracy dropoff when a client is providing random information is constant, regardless

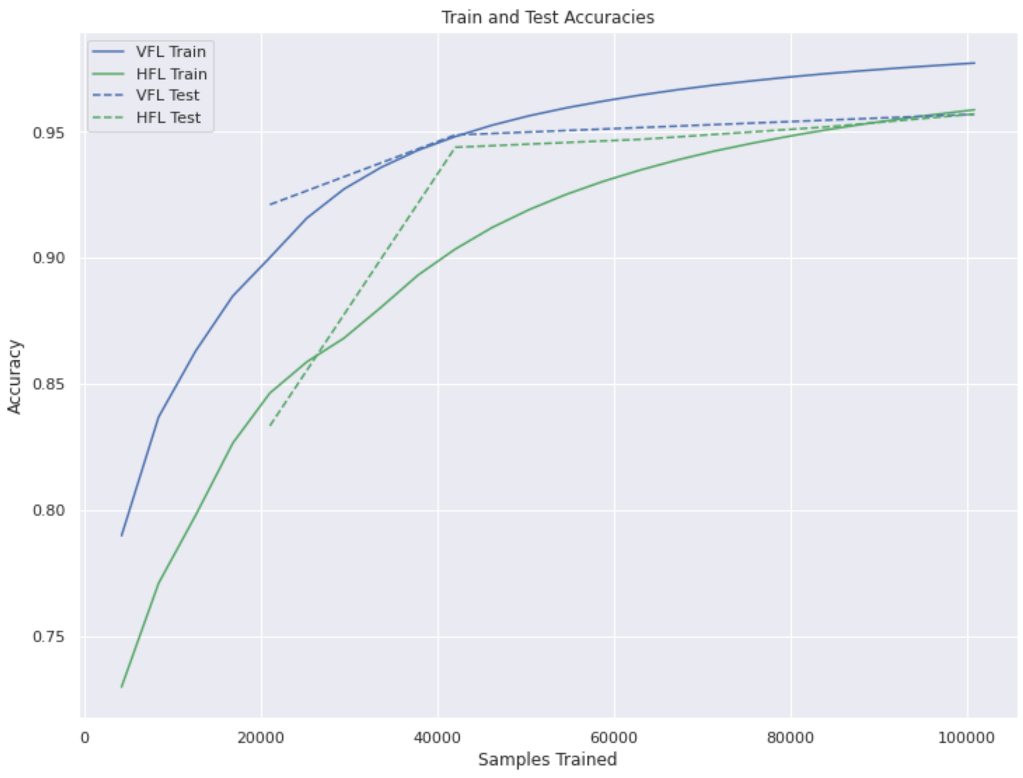


Figure 5.1: Train and Test Accuracies Without Attacks

of which client it is. This result shows that every client contributes equally towards determining the predicted label.

As can be seen in Figure 5.3, both horizontal and vertical models overwhelmingly classify the samples correctly. A few misclassifications can be observed in every row, and the misclassifications not restricted to any single class of labels. The scale of the colors is $0 - 50$ so that even a small number of misclassifications can be clearly seen.

The model is also evaluated on samples that have been watermarked. When the model is not attacked with a data poisoning attack during training, like in this experiment, this serves to test the robustness of the model to noisy samples. The less the deviation of the accuracy is from the baseline, the more resilient the trained model is to imperfect, noisy samples.

When the test samples are watermarked, the performance of the model deteriorates, as can be seen in Figures 5.3c and 5.3d, under Figure 5.3. Noise is introduced to nearly every label's classification. However, the model does not classify the watermarked samples with the watermark target. This is to be expected, as the watermarking data poisoning attack was not present during the baseline training process. As such, the watermarked test samples are just perceived as noisy regular samples by the model.

In conclusion, the comb model performs with high accuracy across all labels, is resilient against watermarked - noisy - samples, and distributes the importance of clients evenly.

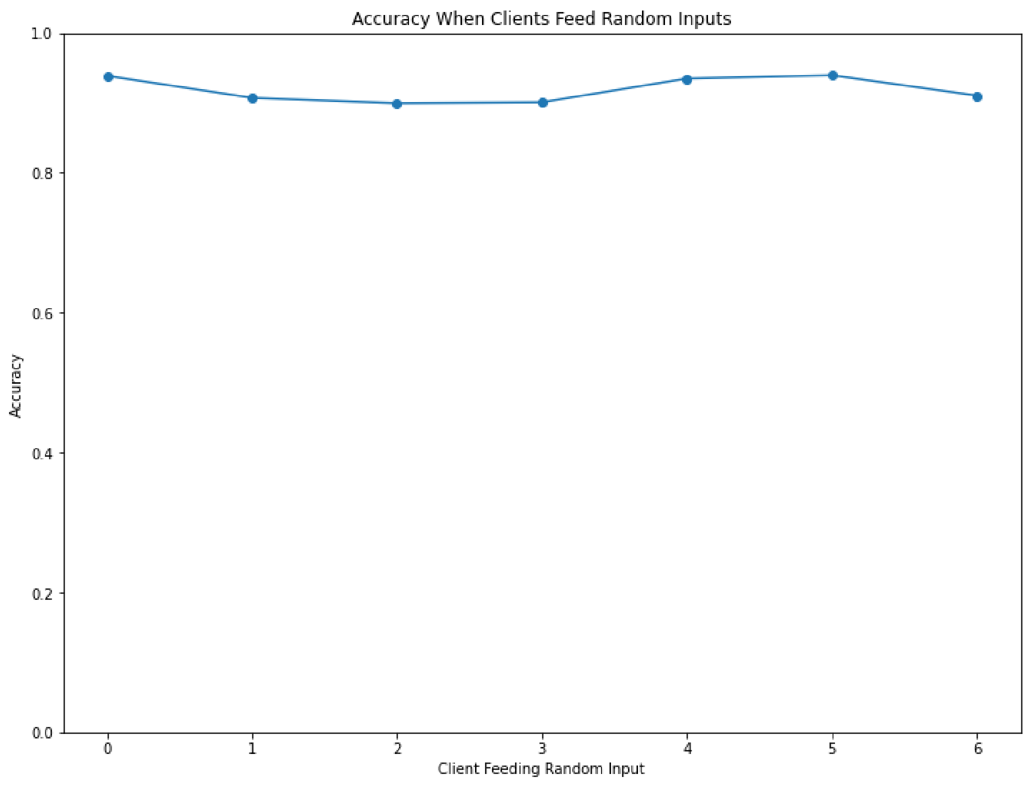
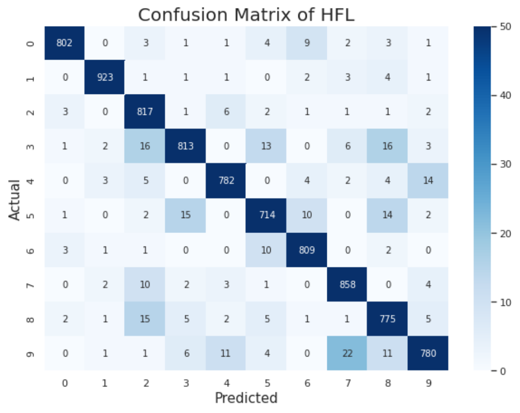
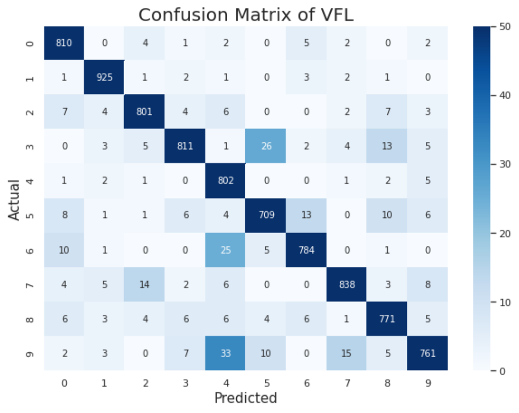


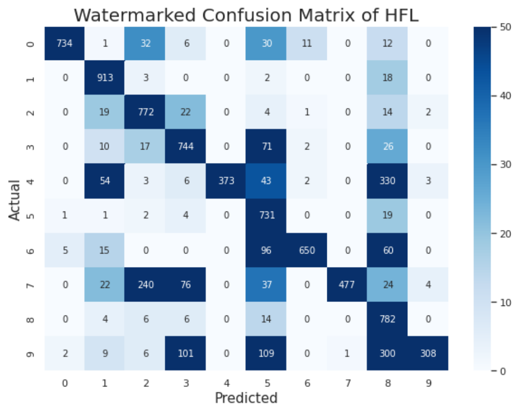
Figure 5.2: Relative Importance of Clients in Comb Model VFL



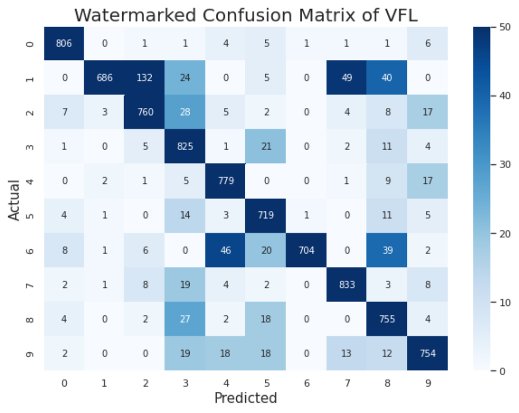
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.3: Confusion Matrices Without Attacks

5.1.2 Chain Model Baseline

The chain model described above is able to achieve a training accuracy of 95% in HFL and 90% in VFL, with the difference most likely attributable to initialization differences. The test accuracies for both scenarios reached 87%, but were not consistently improving over the training phase. These values can be seen in Figure 5.4. Both of these values are significantly lower than the comb model's evaluated accuracies. To get a complete understanding of the chain model's performance, though, the other metrics needs to be evaluated too.

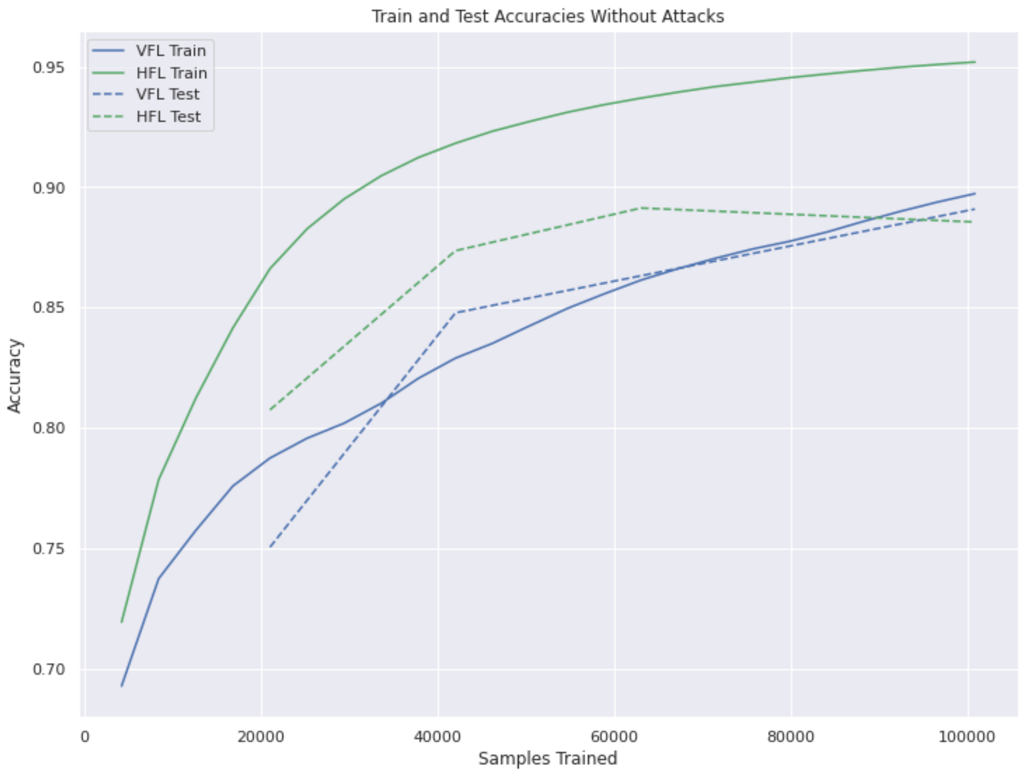
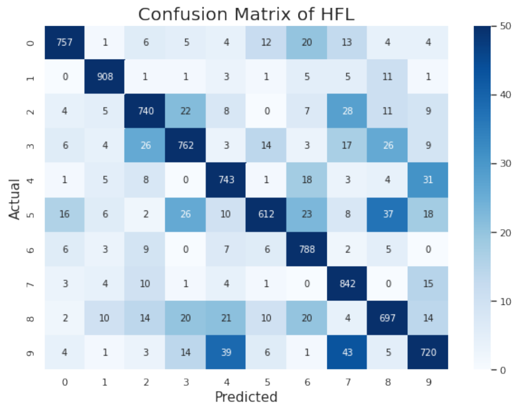


Figure 5.4: Train and Test Accuracies Without Attacks

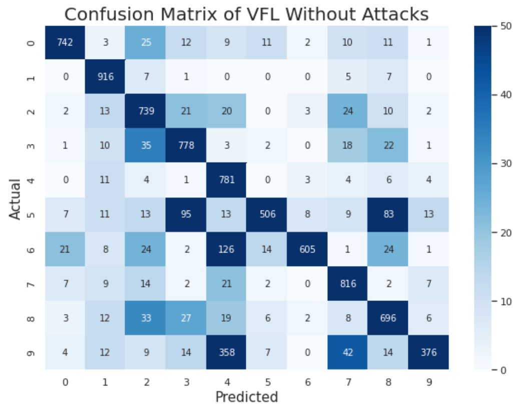
The confusion matrices in Figure 5.5 reveal more about the shortcomings of the chain model. The model misclassifies more samples in every label category than the comb model, as can be seen in Figure 5.5a and 5.5b.

However, the shortcomings of the model are on full display when the test samples are watermarked. The watermarking process here, too, only serve to test the resiliency of the model to noisy samples. The little amount of noise that the watermarks introduce to the samples is already enough to completely break the chain models. The accuracy over watermarked samples drops down to 29% in HFL and 23% in VFL, with both models classifying nearly all samples as either 2's or 3's.

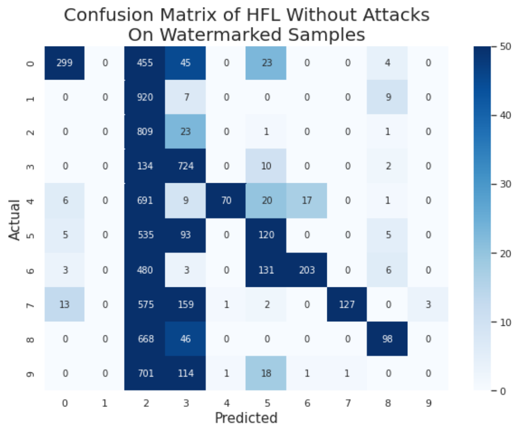
Finally, the relative importance analysis shows another reason why the chain model is inadequate. As can be seen in Figure 5.6, the accuracy significantly decreases as clients towards the end of the chain, especially the last client, feed random inputs. This means



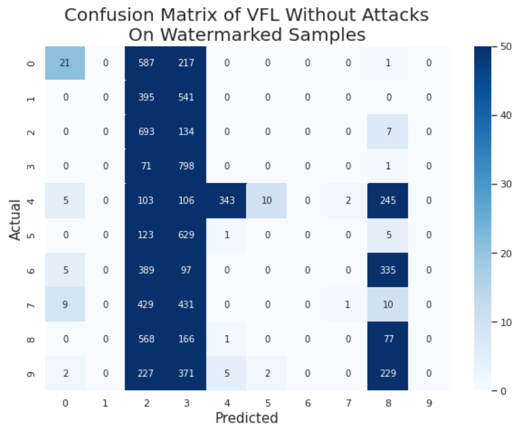
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.5: Confusion Matrices Without Attacks

that the inputs of the last client, the active party, account for the more than 50% dropoff in accuracy.

This relative importance discrepancy must come from the chain model architecture, and not from the way the data is distributed. In Section 5.1.1, the distribution of data samples and rows within samples was identical to this experiment's. The comb model's baseline relative importance was nearly constant across all clients, so the distribution of the data could not have caused the chain model's discrepancy. The only conclusion that can be drawn is that the chain model's architecture is not suited for this task.

All of the evaluated metrics show that the chain model is an inadequate model architecture for the task at hand. The accuracy with which it classifies unaltered new samples is lower than the comb model's. Noisy samples break the model completely, resulting in a dropoff of over 60% over all label categories. Finally, the last client in the chain is orders of magnitude more important to the correctness of the predicted label than the other clients. This relative importance discrepancy comes from the architecture, not the data. Combined, these insights show that the chain model's architecture is inadequate, and as such, the chain model will not be investigated in later experiments. All following experiments will use the comb model architecture.

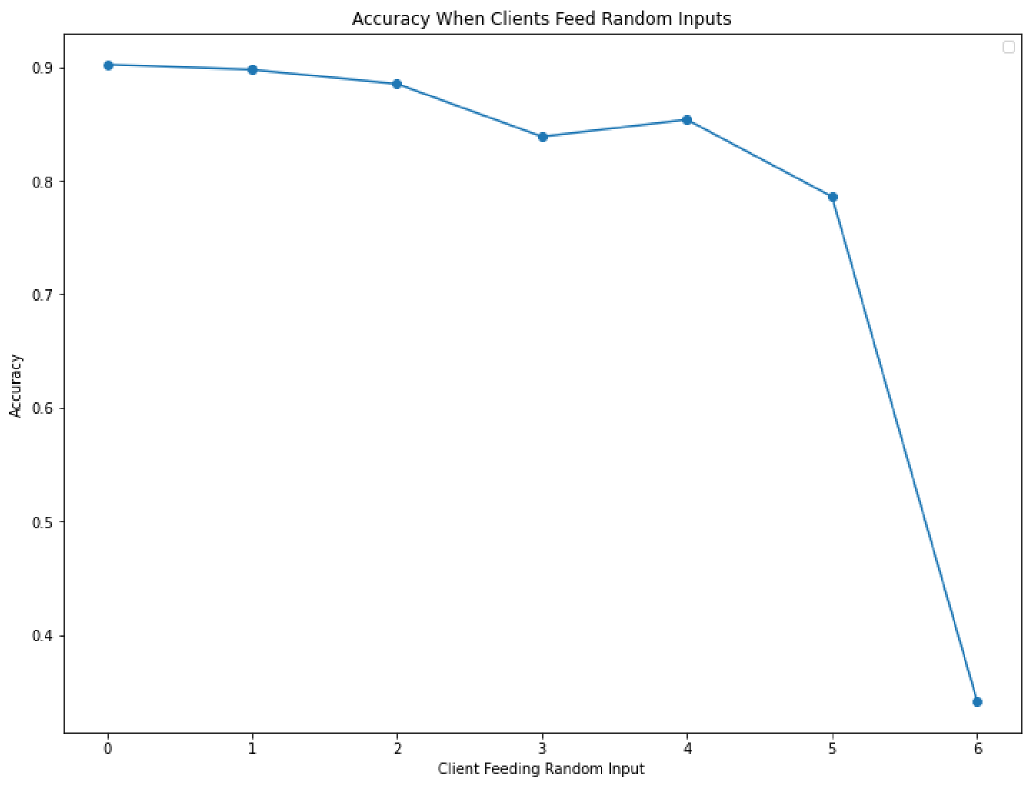


Figure 5.6: Relative Importance of Clients in Chain Model VFL

5.2 Data Poisoning Attack

As discussed in Section 4.1 and Section 4.2, in the data poisoning attacks experiments, the adversary will poison a selected percentage of the samples available to them during training. The percentages examined are 25%, 10%, 1%, and 0.5%.

5.2.1 25% Poisoned

With 25% of their samples watermarked by an adversary during training, the global model shows no signs of deterioration. Both in HFL and VFL, the model was able to achieve nearly 99% accuracy over the training set, and over 95% accuracy over the test set. Both accuracies strictly increase over the training phase, as can be seen in Figure 5.7

In the confusion matrices over the watermark-free test set, nothing seems amiss. All label classes are classified properly, with only a small handful of misclassifications. These results are almost identical to the confusion matrices of the unattacked experiment. Comparing the results in Figure 5.8a and 5.8b in Figure 5.8 to those in 5.3, it is impossible to tell that the former has any data poisoned at all.

However, in the confusion matrices over watermarked samples, the effect is striking. Nearly 100% of the samples are classified as the watermark target. The adversary was completely successful in implementing a watermark backdoor.

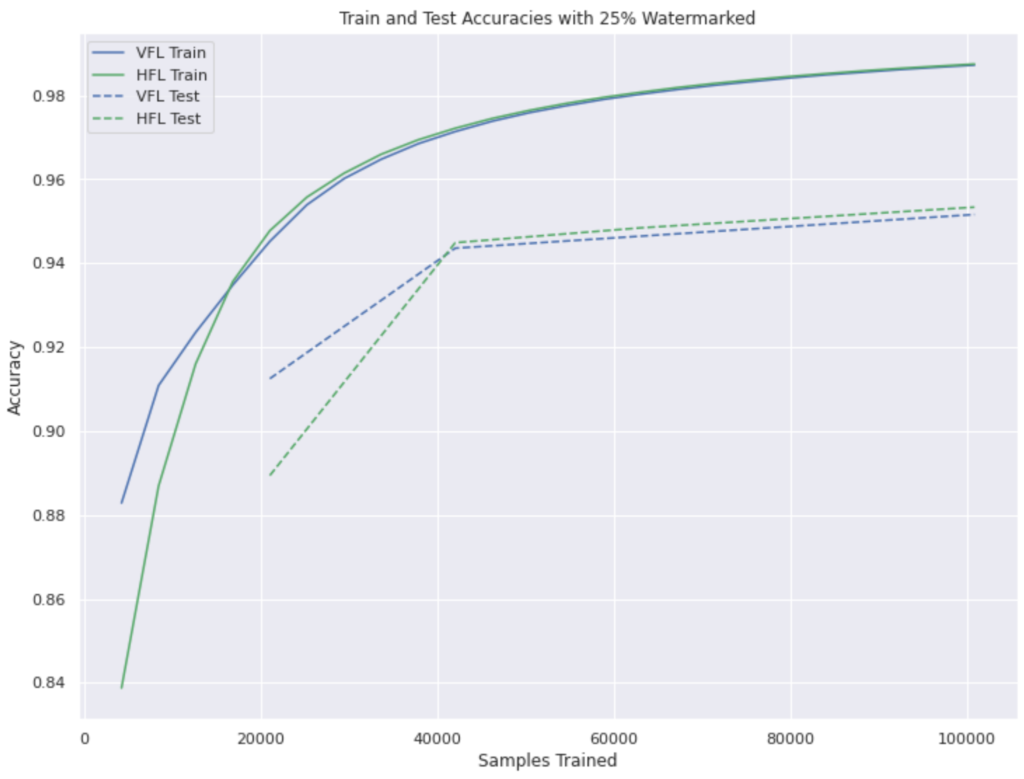


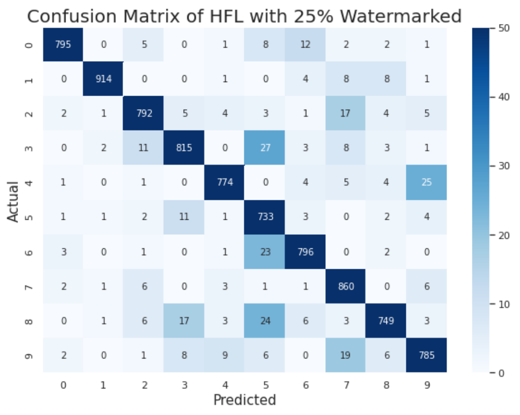
Figure 5.7: Train and Test Accuracies With 25% Watermarked

The only one of the evaluated metrics that betrays the watermarking is the relative importance. As can be seen in Figure 5.9, the dropoff in accuracy is constant for all clients, except the adversary. With the adversary providing random input, the accuracy drops from 95% to 15%. Given knowledge that the data is distributed evenly with regards to importance (provided by the baseline experiment), from this metric alone, it is clear that the adversary is not acting in an honest way.

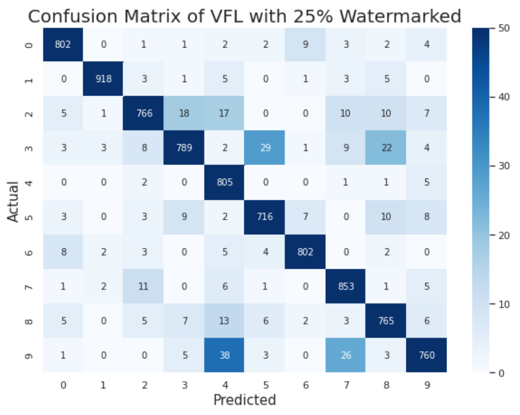
The reason for this discrepancy in relative importance is as follows. Whenever the adversary introduces a watermark to the sample, the model learns to output the watermark target. This means that the adversary can single-handedly control the label of the model. Therefore, the adversary is more important towards determining the final label than the other clients.

There is no discrepancy in the efficiency of the watermarking between HFL and VFL, despite the adversary in VFL having watermarked seven times as many samples. With 25% of the samples watermarked, adversaries in both scenarios were able to achieve 100% watermark accuracy in the global model.

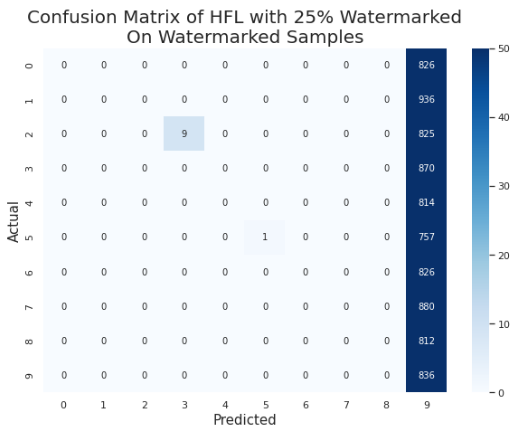
In conclusion, with 25% of the samples watermarked, neither the accuracy, nor the confusion matrices over unmarked test samples revealed the presence of any data poisoning attack. The evaluated relative importance metric showed that the adversary was significantly more important than the other clients, despite the model and the datasets being balanced in importance. Finally, the efficiency of the attack was the same across both scenarios, despite the adversary in the vertical setting watermarking seven times as many samples than their horizontal counterpart.



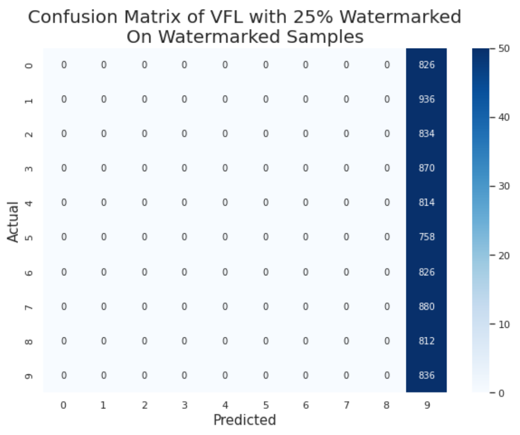
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.8: Confusion Matrices With 25% Watermarked

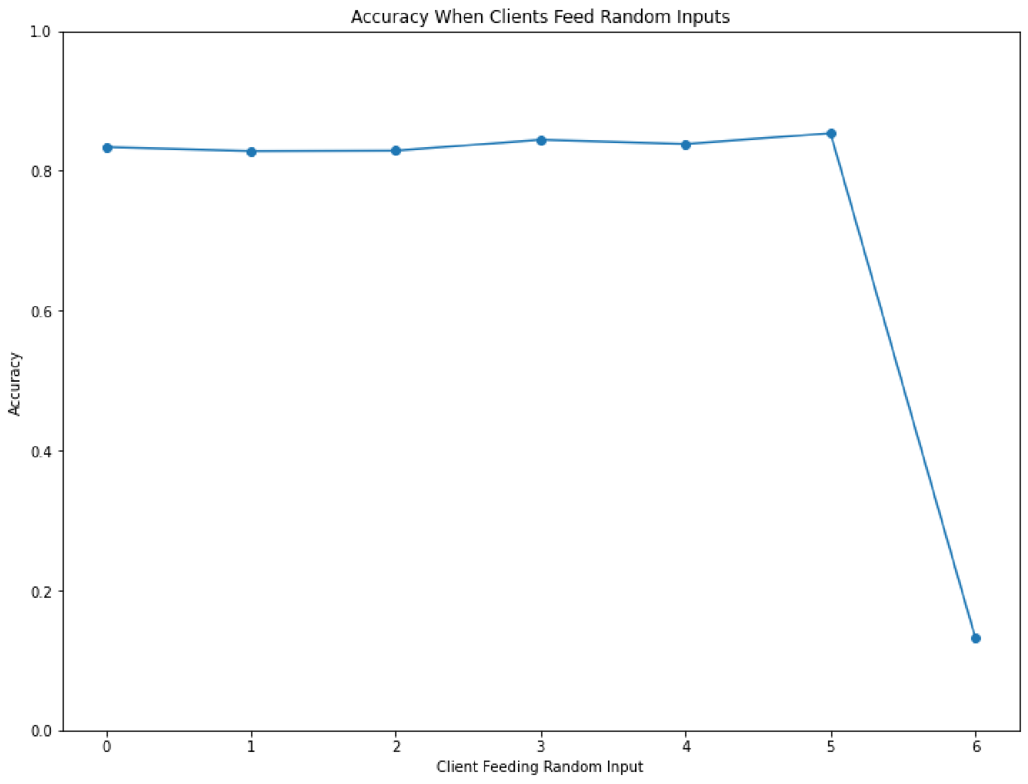


Figure 5.9: Relative Importance of Clients in VFL With 25% Watermarked

5.2.2 10% Poisoned

Because the adversarial attack was entirely effective in both HFL and VFL when 25% of the samples were poisoned, in the next experiment a lower percentage was examined, namely 10%.

As can be seen in Figure 5.10, the global model's accuracy still shows no signs of deterioration. The training accuracy surpassed 98% and the test accuracy 95% in both settings. Both accuracies strictly improved over the training phase.

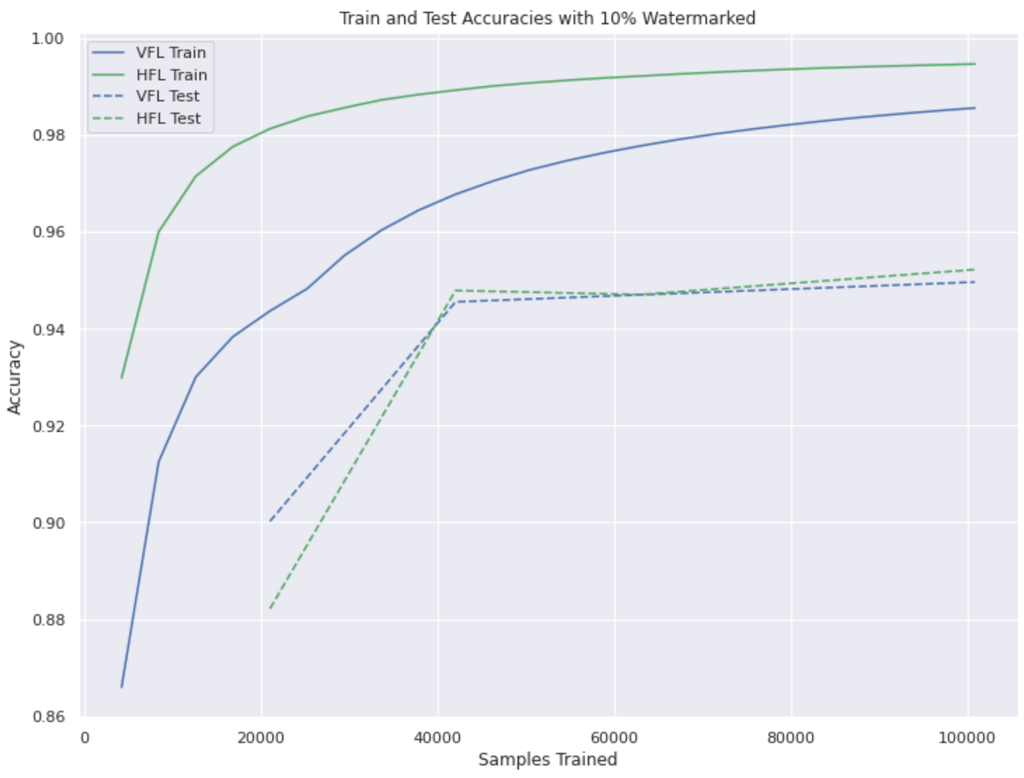
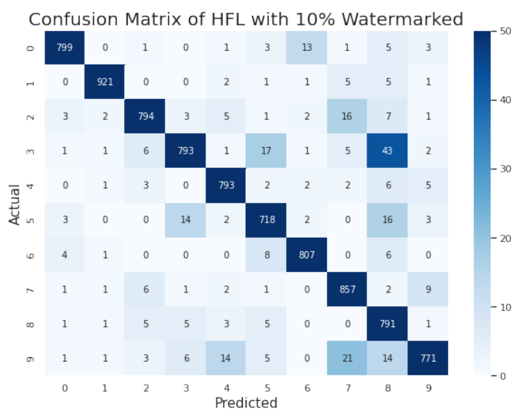


Figure 5.10: Train and Test Accuracies With 10% Watermarked

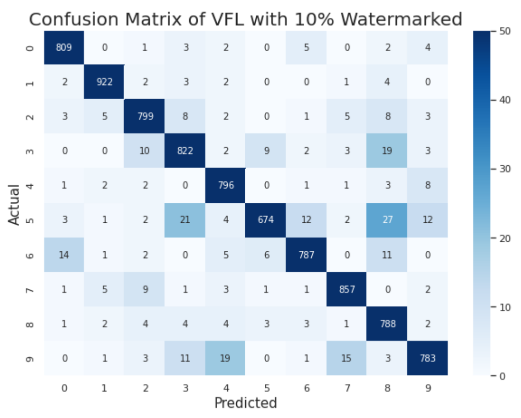
Even with only 10% of the samples watermarked, the confusion matrices over unmarked seem unaltered. Over the watermarked samples, though, the model once again performs in accordance with the attack: nearly 100% of the samples are classified as the watermark target. This stark difference can be seen in Figure 5.11.

As in the experiment in 5.2.1, the relative importances are constant, except for the adversary, who once again is significantly more important than the other clients, as seen in Figure 5.12.

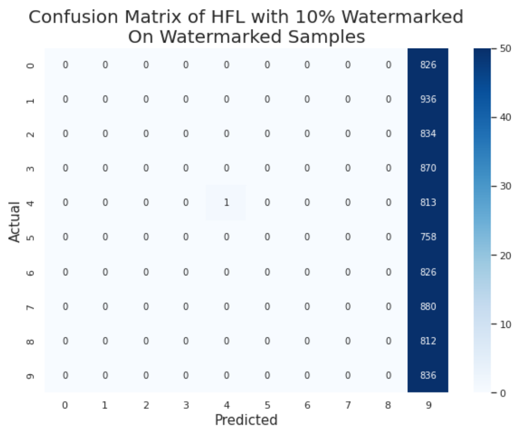
The adversary was entirely effective by watermarking only 10% of the samples. There was no difference in the efficiency between the attack's efficiency in the horizontal and the vertical scenarios. There seems to be no added benefit to marking 25% of samples over marking only 10%.



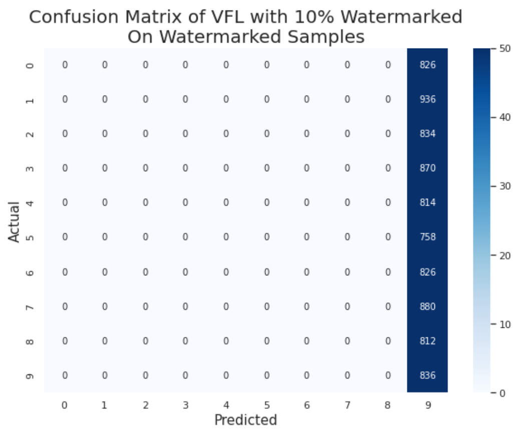
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.11: Confusion Matrices With 10% Watermarked

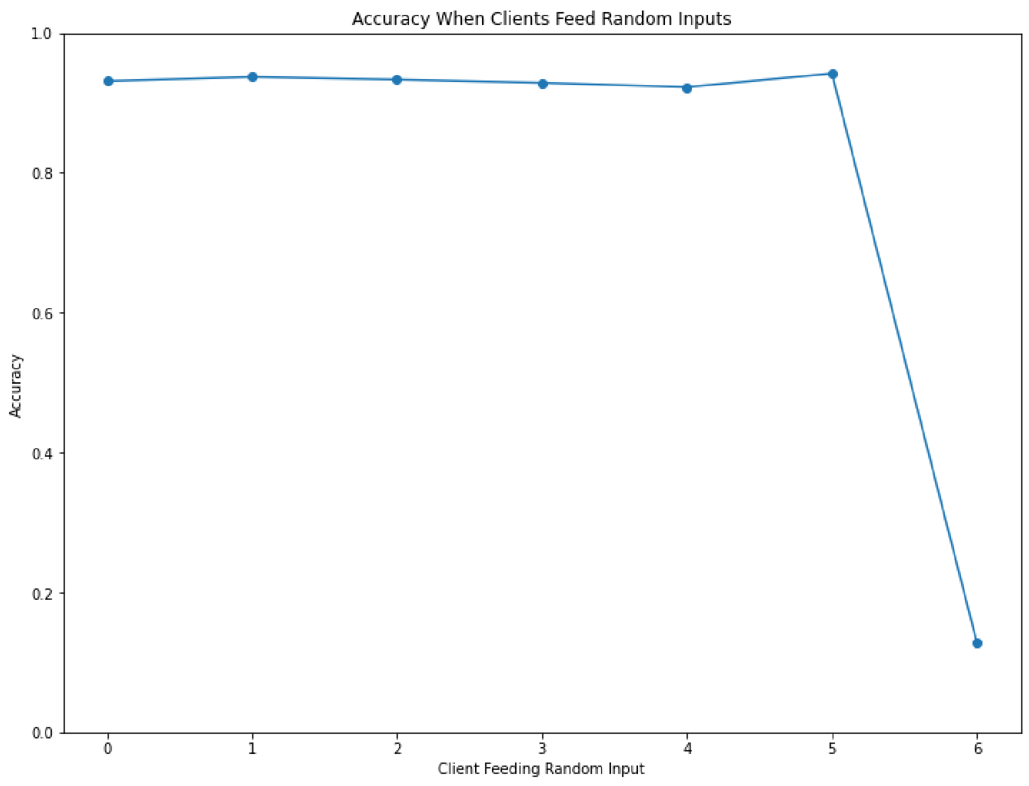


Figure 5.12: Relative Importance of Clients in VFL With 10% Watermarked

5.2.3 1% Poisoned

Even with only 10% of the samples poisoned, the adversarial attack was entirely effective, so the next experiment entailed poisoning only 1% of the adversary's samples.

The trained model's accuracy seems unaltered, as shown in Figure 5.13. The accuracy over the training set increased to 98%, while the testing accuracy still surpassed 95%. These are the same values the model reached in previous experiments, so the watermarking seems to have no bearing on the accuracy metrics.

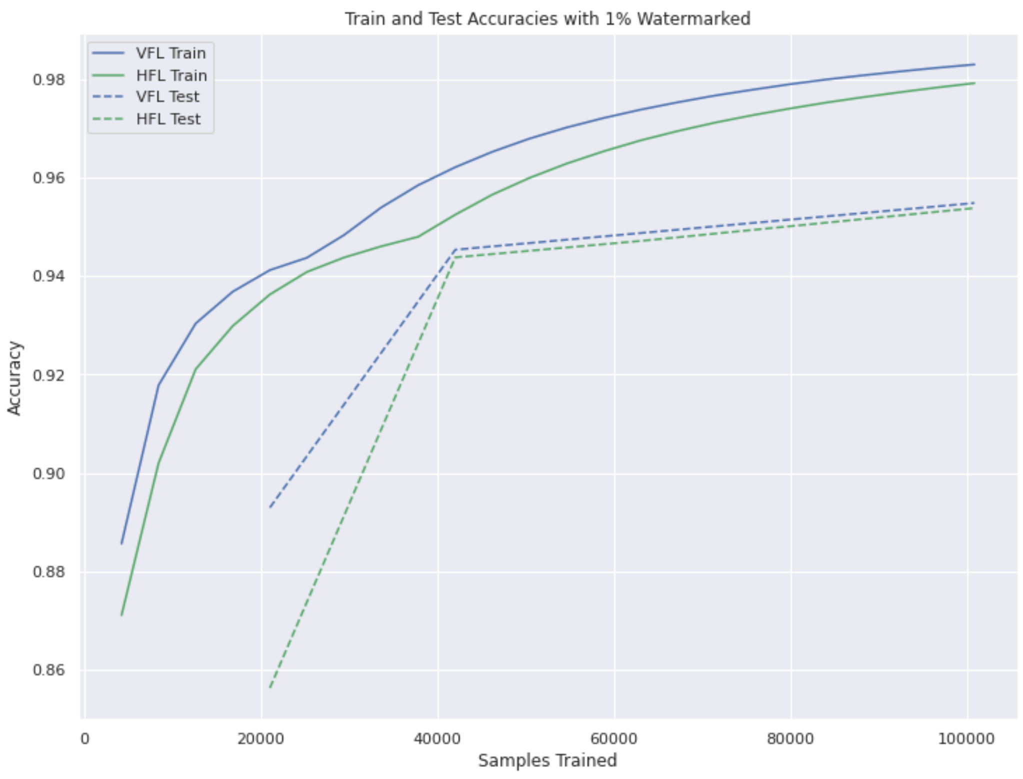
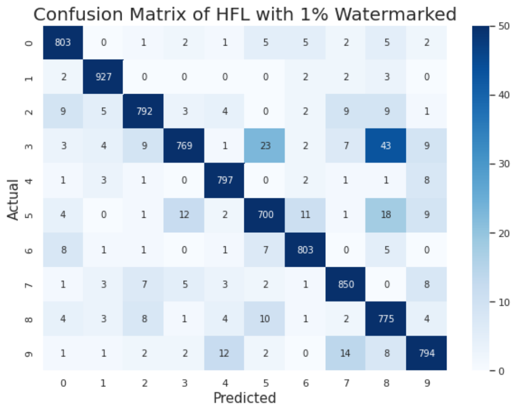


Figure 5.13: Train and Test Accuracies With 1% Watermarked

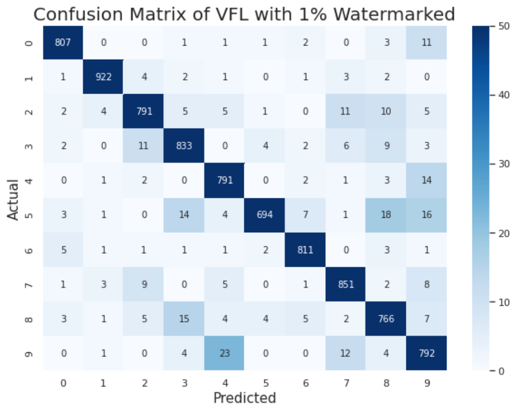
As expected, the confusion matrices over unmarked samples is identical to the earlier experiments'. As can be seen in 5.14, the confusion matrices over watermarked samples reveal nearly 100% watermark efficiency. However, the attack in the vertical scenario is exactly 100% efficient, while the one in the horizontal scenario has a handful of samples that were not classified as the watermark target. We can interpret this as the result of the vertical attack watermarking seven times as more samples.

The relative importances in Figure 5.15 show the same amount of discrepancy as earlier experiments, despite the watermarking percentage's decline. The pattern of the relative importances is the same as earlier, with the honest participants' relative importance nearly equal, and the resultant accuracy just slightly below the model's baseline accuracy.

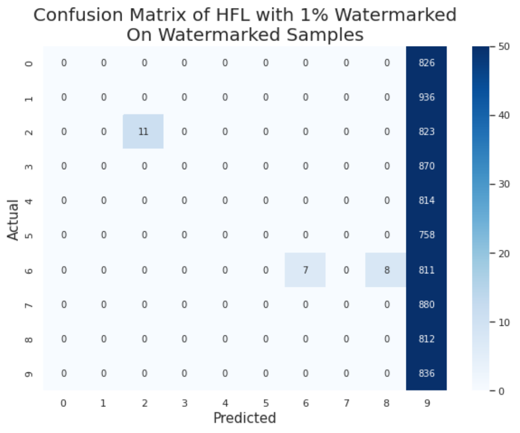
Once again, the adversarial attack was completely effective, despite marking only 1% of the samples. The vertical scenario's attack was a touch more efficient than the horizontal scenario's, however both were overwhelmingly successful across all labels.



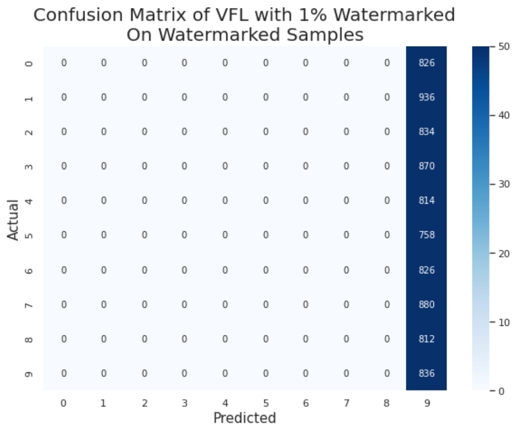
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.14: Confusion Matrices With 1% Watermarked

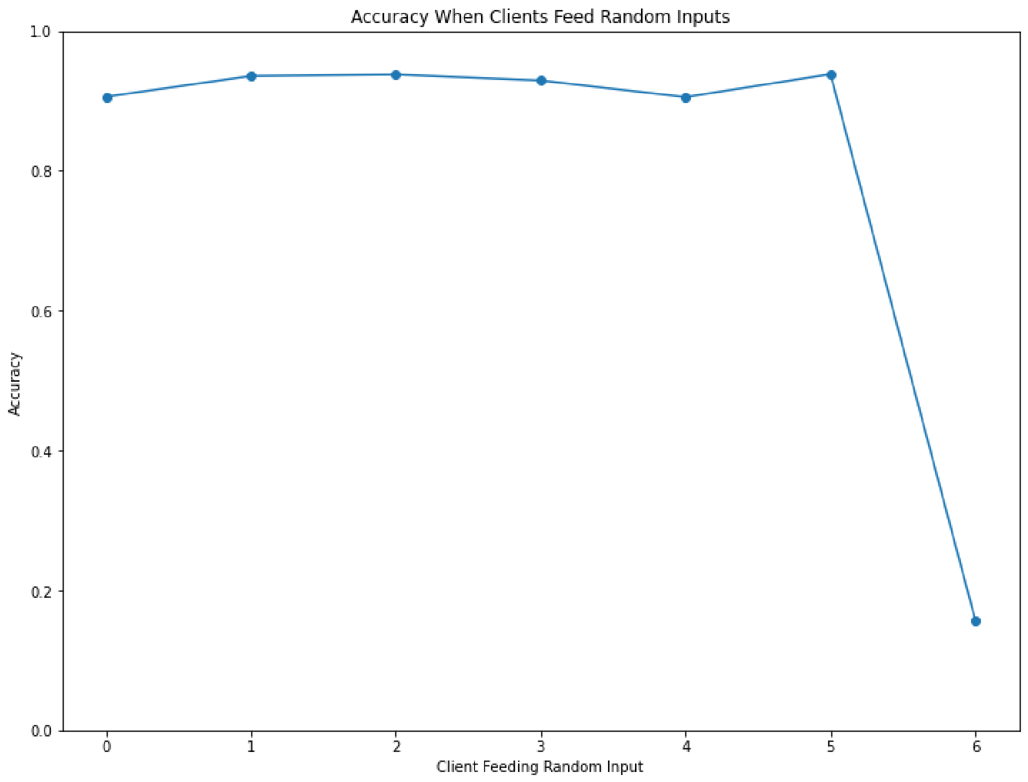


Figure 5.15: Relative Importance of Clients in VFL With 1% Watermarked

5.2.4 0.5% Poisoned

The final experiment in this line is to mark only 0.5% of the samples. In the horizontal scenario, this would mean marking just 20 samples every epoch, out of a dataset of over 32000. In the vertical scenario, seven times as many, over 140, would be marked per epoch.

As expected, as per the previous experiments' results, both training and test accuracies match the baseline, as Figure 5.16 shows. There is a slight dip of 0.5% in the horizontal testing accuracy in its final evaluation period, but there is nothing to suggest that this is the result of the adversarial attack.

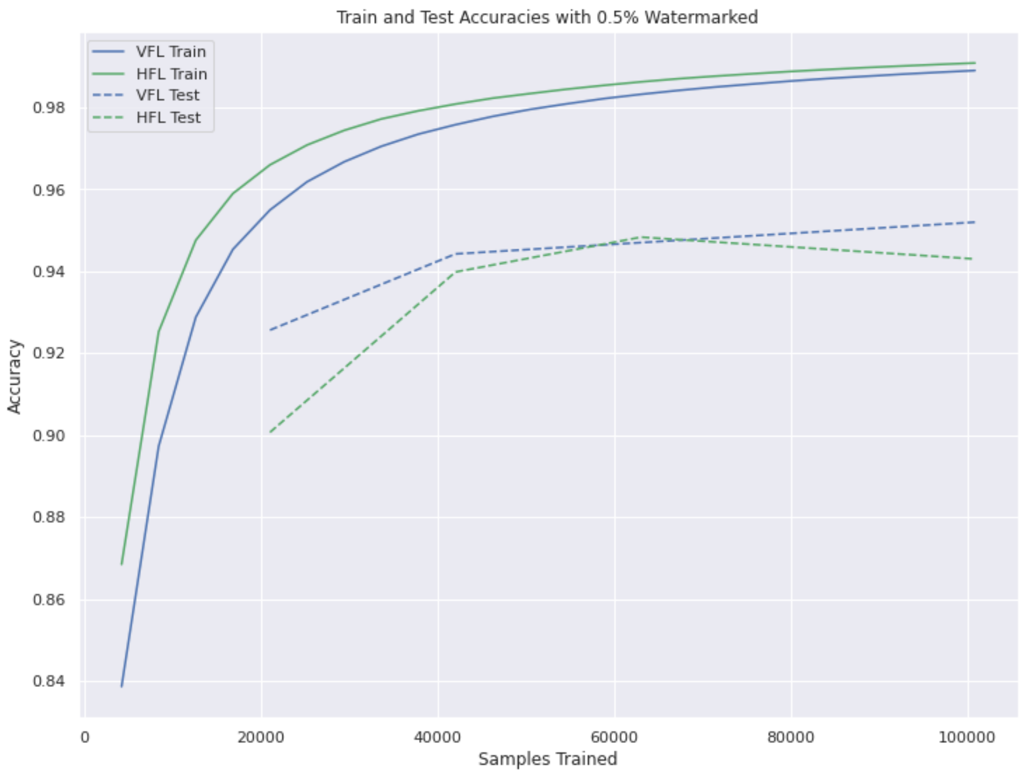


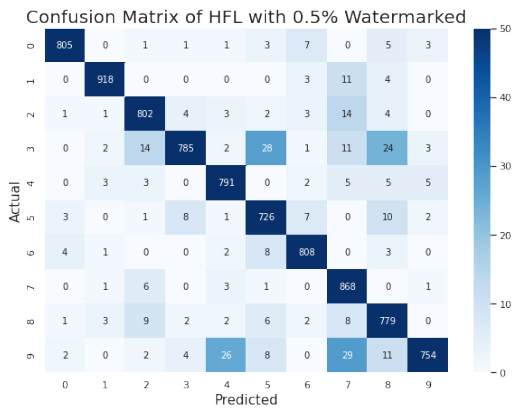
Figure 5.16: Train and Test Accuracies With 0.5% Watermarked

At this percentage of marked samples, we finally see some differences in the confusion matrices, Figure 5.17. Those evaluated over unmarked samples are identical to the previous experiments'. The confusion matrices over watermarked test samples, though, show new results.

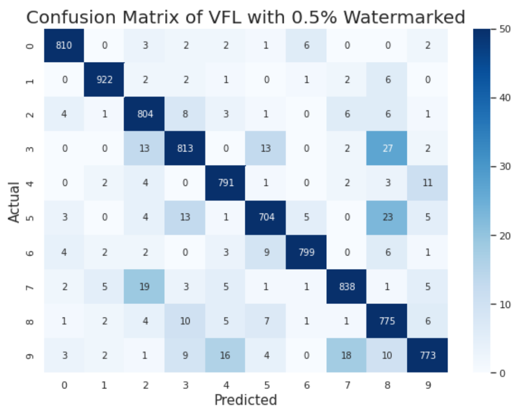
As seen in Figure 5.17c, the attack is only partially effective in the horizontal setting. For over half of the label classes, the watermarking is significantly less effective. However, no class is entirely unaffected by the attack.

The attack in the vertical scenario is also showing signs of being less effective. The attack is still overwhelmingly effective in this setting, but where in earlier experiments the vertical attack was absolutely effective, now there are a handful of cases where the attack did not produce the desired label.

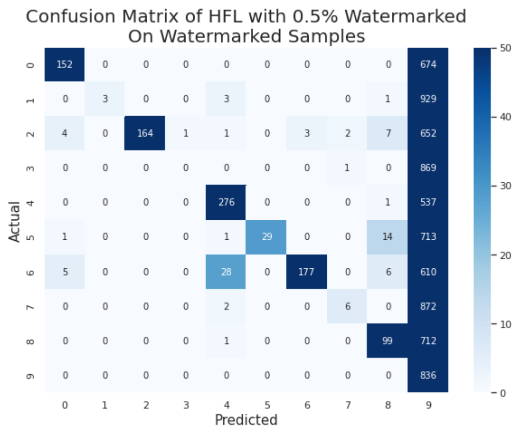
In comparing the two scenarios' attacks' effectiveness, it is clear that the vertical scenario's is significantly more effective than the horizontal attack. This is the first experiment in which the two attacks showed significant differences in their effectiveness.



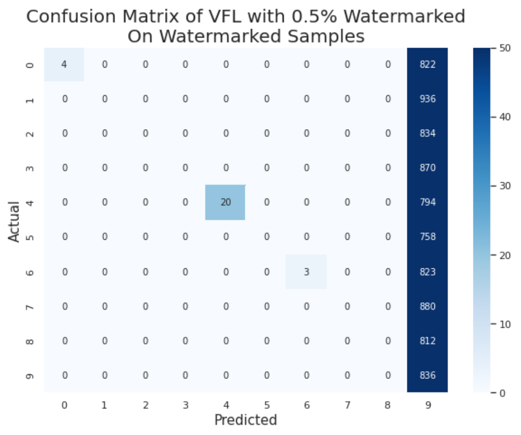
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.17: Confusion Matrices With 0.5% Watermarked

The relative importance graph, Figure 5.18, is similar to those in the earlier experiments. The relative importance of the adversary is closer to the honest participants' in this experiment, though. This is attributable to the attack being less effective, and in effect, the model relying less on the adversary's input to produce the final label.

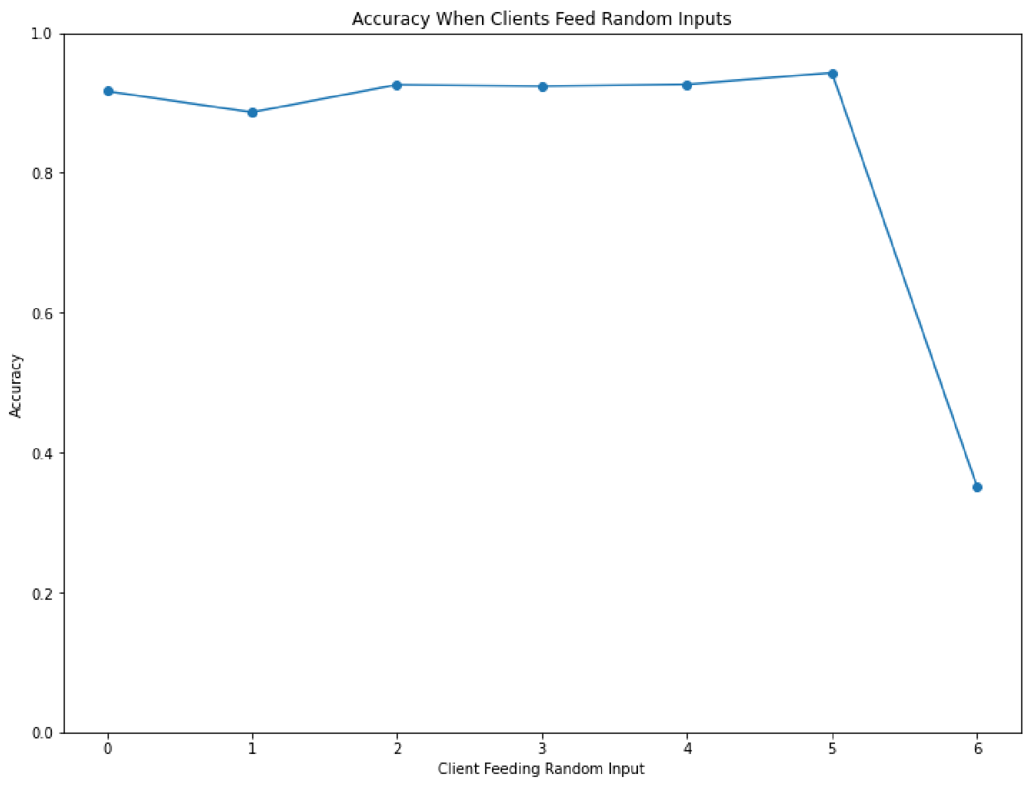


Figure 5.18: Relative Importance of Clients in VFL With 0.5% Watermarked

5.3 Gradient Poisoning Attack

As described in the implementation, the gradient poisoning attack will multiply selected gradients with some value to deteriorate the global model's performance. The values chosen for the experiments are -1, -10, and 0.

The same metrics are used to evaluate these attacks as the data poisoning attacks.

5.3.1 Gradient Multiplier of -1

With a gradient multiplier of -1, the attacked component(s) takes a step in the exact opposite direction, but of equal magnitude.

In the horizontal setting, there are seven participants, six of whom are taking steps in the direction of steepest descent, while one is taking steps in the opposite direction. As a rough calculation, we can expect the adversary to cancel out the contributions of an honest participant, resulting in a model being trained, in effect, by five honest participants. As such, we can expect the horizontal scenario to still achieve good results.

In the vertical scenario, one component will apply updates exclusively in the direction of steepest ascent. Earlier, in the baseline results, we saw that even when a client is feeding random information as input, the model was able to achieve a high accuracy. As such, we know it is possible that a model functions well without the contributions of a participant, input-wise, but this experiment is different in that the component under the adversary's control is intentionally malicious.

Figure 5.19 shows the training and test accuracies of this experiment. As expected, the horizontal training and test accuracies are both high, above 90%, but don't reach the accuracy levels recorded in the baseline experiment. In the vertical scenario, however, the training accuracy starts out at a high value, and then drops down under 30%. The test accuracy is consistently low, but also decreases over the training phase. As there are 10 label classes, the test accuracy is almost as bad as guessing the label completely randomly.

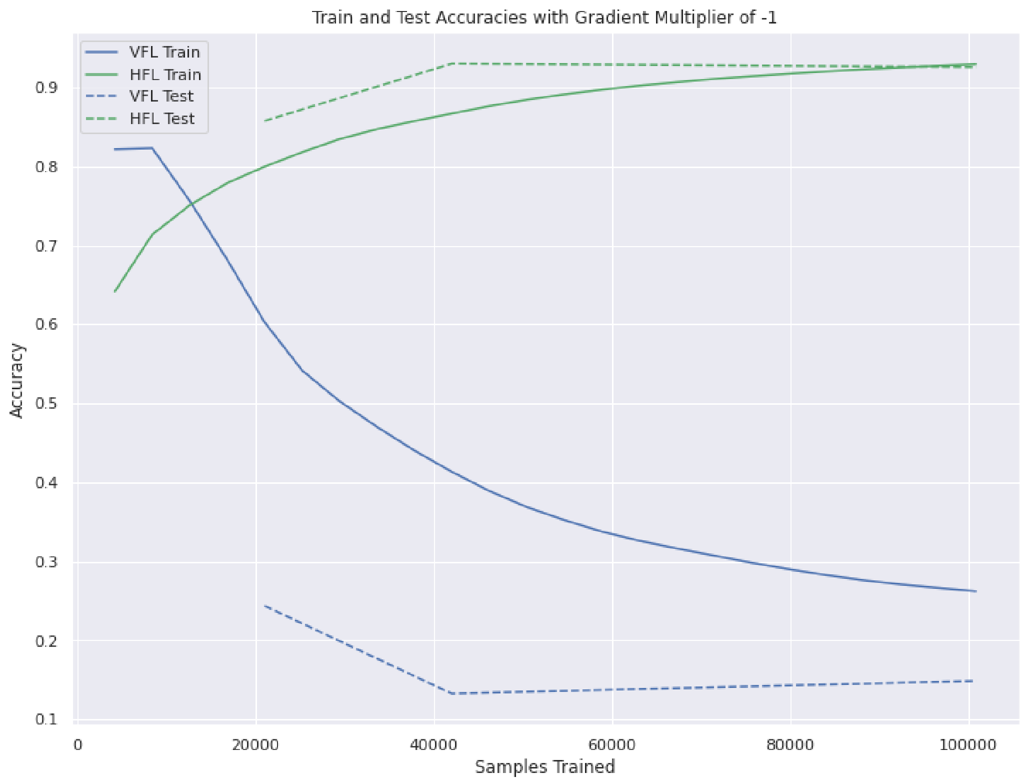


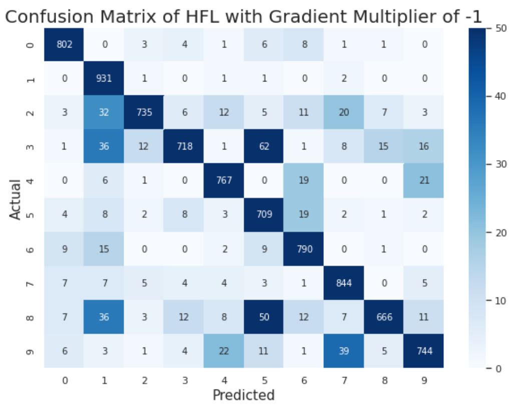
Figure 5.19: Train and Test Accuracies With a Gradient Multiplier of -1

The confusion matrices tell a similar story in Figure 5.20. The horizontal scenario's confusion matrices look similar to the baseline ones, just less accurate overall. The watermarked samples introduce more noise to the labeling, but the model still functions reasonably well.

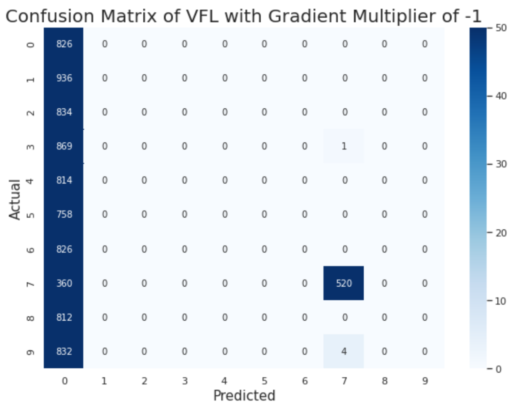
The watermarked confusion matrices here again only serve to test the resiliency of the trained model to noisy samples, as the employed attack was not poison data, but the gradients.

In the vertical scenario's confusion matrices, we can see that nearly all samples are classified as the default output of the model. Only one label class was labeled correctly, and even that happened less than 60% of the time. The confusion matrix looks identical to this over the watermarked samples. As the model couldn't even correctly classify the valid digits, it is to be expected that the noisy, marked, samples will result in a similar output.

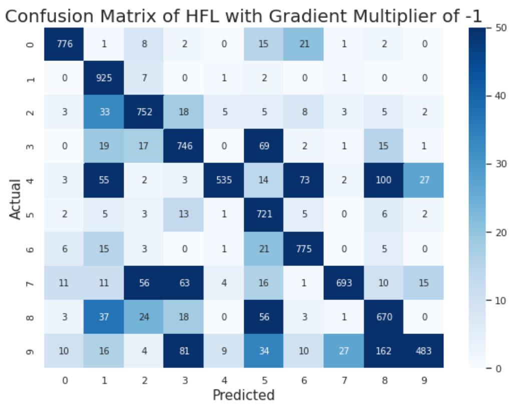
Because the adversary prevented a performant model being learned at all, it doesn't much matter whether a participant is providing random inputs or not, the model will still



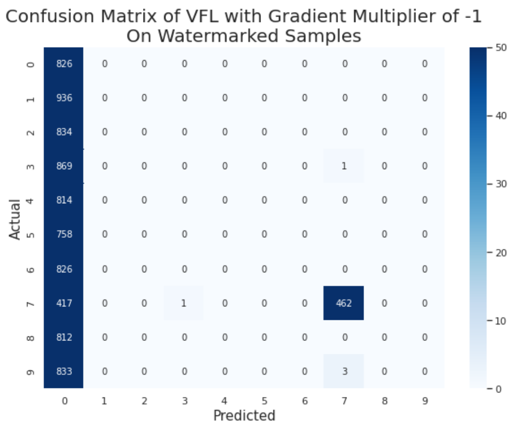
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.20: Confusion Matrices With a Gradient Multiplier of -1

perform very poorly. As such, the relative importance graph has no valleys as can be seen in Figure 5.21; all participants are equally important towards determining the final label, despite the adversary's component being the only one trained to be malicious.

In summary, the attack was successful to different degrees in the two scenarios. In the horizontal setting, the attack slowed down the progress of the training, but did not prevent a model from being learned. In the vertical scenario however, the attack deteriorated the model's performance to an almost random level. Where the relative importance graph clearly revealed the adversary in the data poisoning attacks, it was constant across all participants in this experiment. As such, we can say that the attack was completely successful in the vertical scenario, and partially successful in the horizontal.

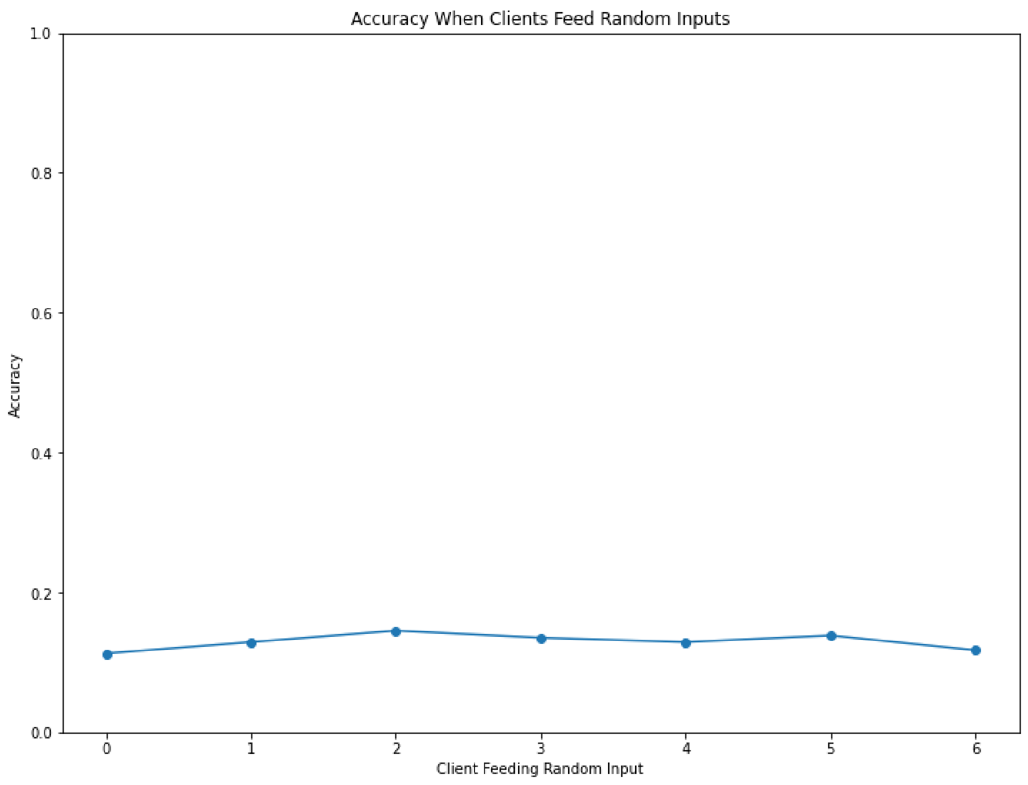


Figure 5.21: Relative Importance of Clients in VFL With Gradient Multiplier of -1

5.3.2 Gradient Multiplier of -10

By increasing the gradient multiplier to -10, we can expect the attack to be more effective. Not only does the attacked component take a step in the exact opposite direction as it's meant to, but the step it takes is also 10 times that in magnitude.

Using the same rough calculation as in the previous experiment, we can expect the horizontal attack to be successful as well. There are six participants who each take a step in the correct direction, and there is one adversary who takes 10 steps in the opposite direction. As such, we can even expect the horizontal scenario not to be able to learn a model at all.

In the vertical scenario, even a multiplier of -1 was enough to deteriorate the model to an almost-random level, so with a multiplier of -10, we can expect the attack to be even more successful.

Figure 5.22 confirms the hypothesis. Both horizontal and vertical training accuracies decline over the training phase. The test accuracies are constantly at 10%, which is equivalent to guessing randomly. In both cases, the attack is completely successful in deteriorating the global model performance to the point where it's no better than random.

The confusion matrices are similar to the vertical scenario's ones in the previous experiment, as shown in Figure 5.23. In all cases, the model simply outputs the default value, regardless of what the input is.

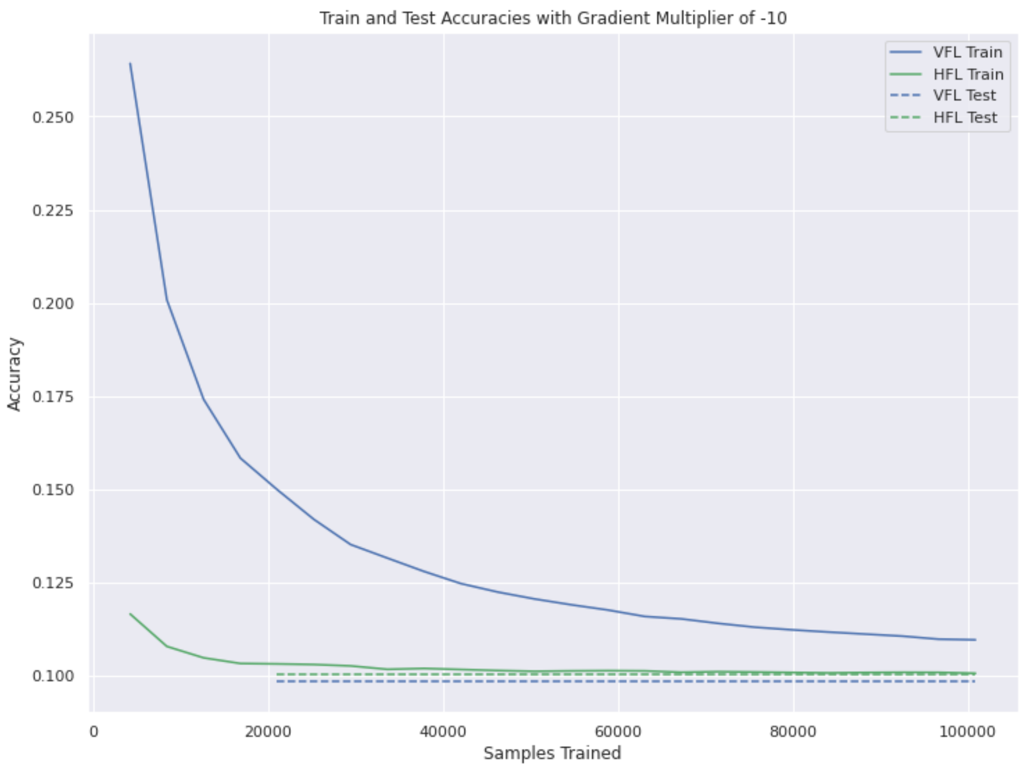
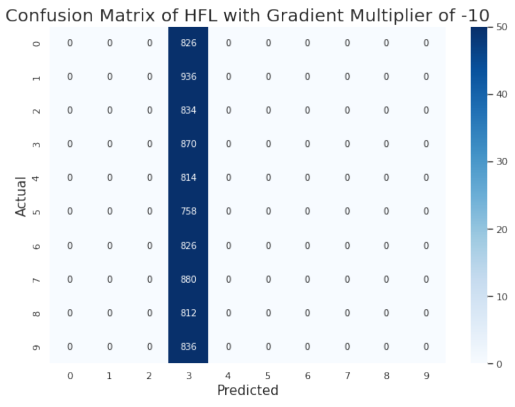


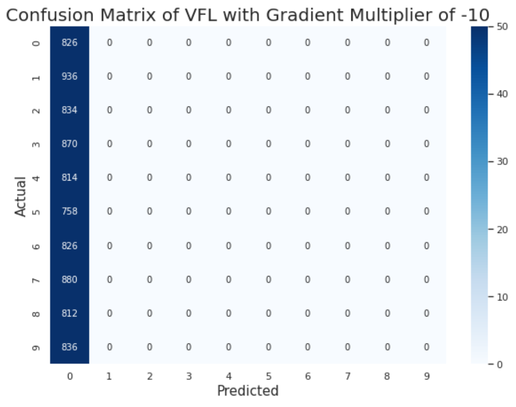
Figure 5.22: Train and Test Accuracies With a Gradient Multiplier of -10

The relative importance graph, Figure 5.24, is nearly identical to the previous experiment's; all clients are equally important. This metric, once again, does not betray the presence of an adversary, as it did in the data poisoning attacks.

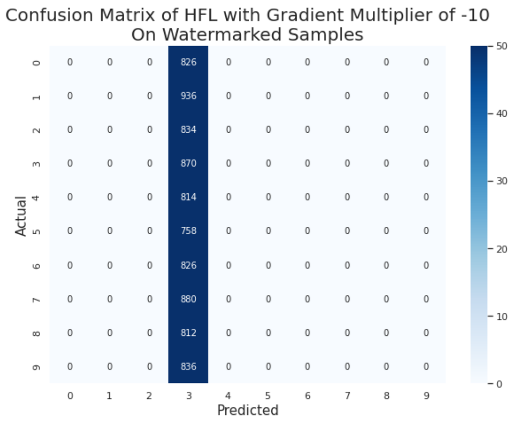
To summarize, the attacks were completely successful in both horizontal and vertical scenarios. In both cases, the global model was deteriorated to being no more accurate than random. Furthermore, the metric that betrayed the data poisoning adversary reveals nothing about the presence of an adversary when the gradient poisoning attack is employed.



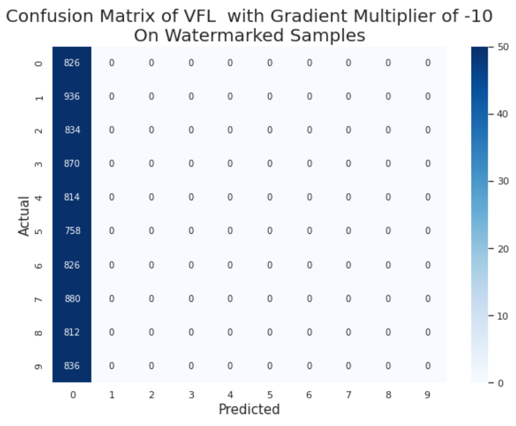
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.23: Confusion Matrices With a Gradient Multiplier of -10

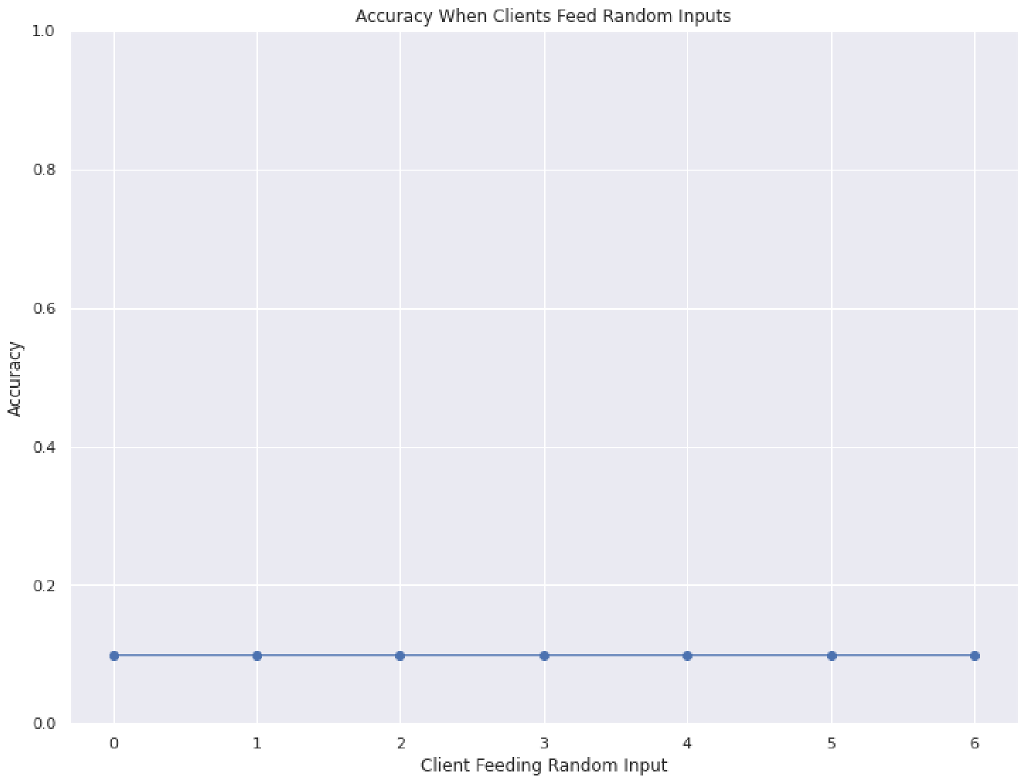


Figure 5.24: Relative Importance of Clients in VFL With Gradient Multiplier of -10

5.3.3 Gradient Multiplier of 0

With a gradient multiplier of 0, the targeted component(s) will simply not change their weights during training. This means that in the horizontal scenario, this attack is equivalent to having one fewer participant in the system. Therefore, we can still expect a competent model to be learned. In the vertical scenario, the adversary's component will simply maintain its initialized weights for the entirety of the training phase. Therefore, the outcome of the experiment depends heavily on the initialization in this case.

In Figure 5.25, the accuracies reflect exactly this. In both scenarios, a model was learned, but they are not as accurate as the baseline models. In the horizontal scenario, the training accuracy reached 98% and the test 95%. The vertical training accuracy surpassed 92% and the test accuracy also hit 95%. While the test accuracies are comparable, the training accuracies are not as high as in the baseline.

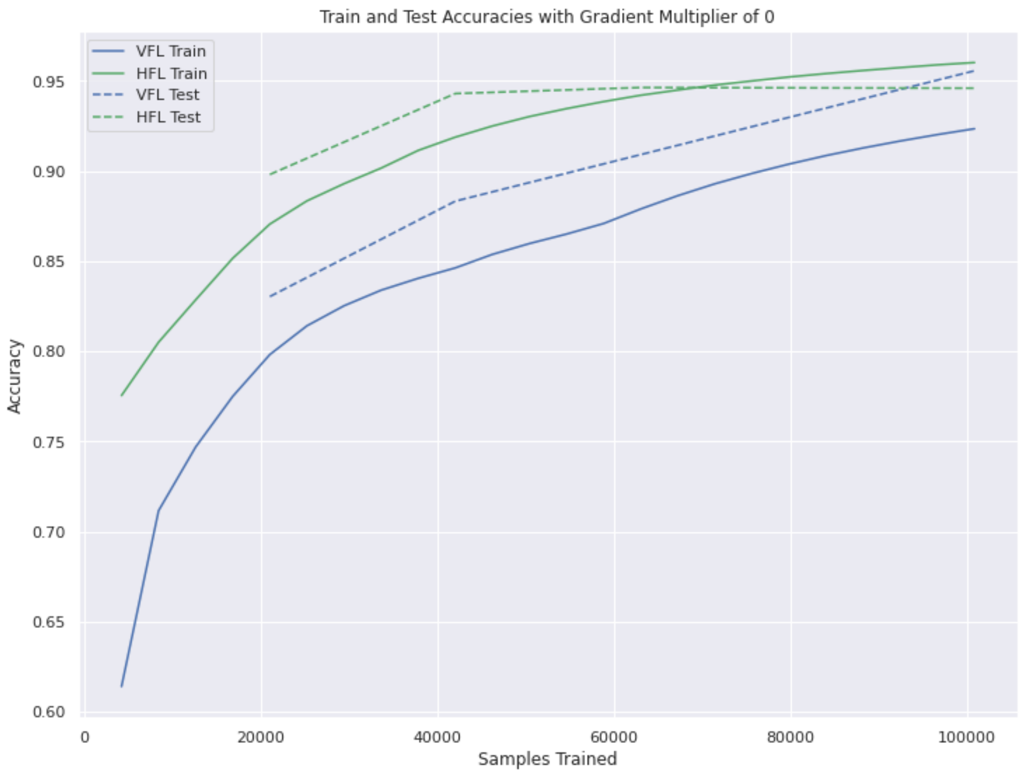
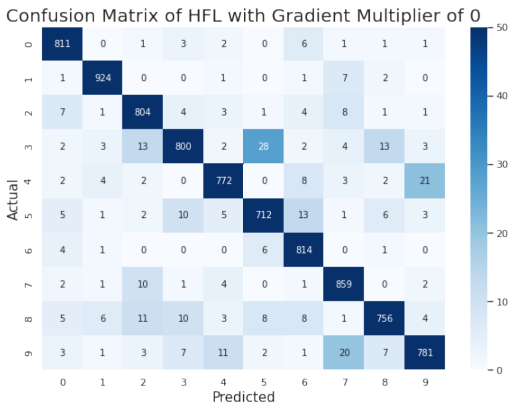


Figure 5.25: Train and Test Accuracies With a Gradient Multiplier of 0

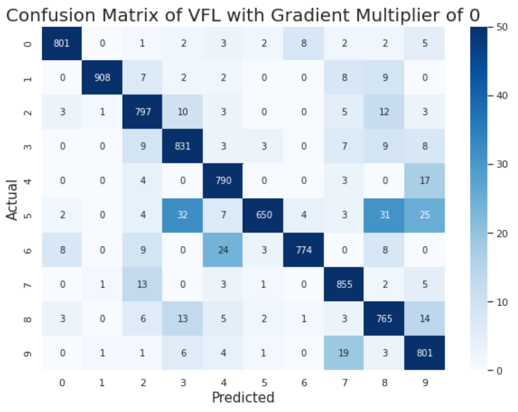
The confusion matrices, as seen in Figure 5.26, are very similar to the baseline's. They classify all labels overwhelmingly correctly with only a handful of misclassifications in each label class. The watermarked samples just seem to introduce noise, but the model functions relatively well despite it, as can be seen in Figures 5.26c and 5.26d.

Despite a capable model being learned, the relative importance graph, Figure 5.27, still does not betray the adversary. All clients' relative importances are constant, and the resultant accuracies are all close to the evaluated test accuracy.

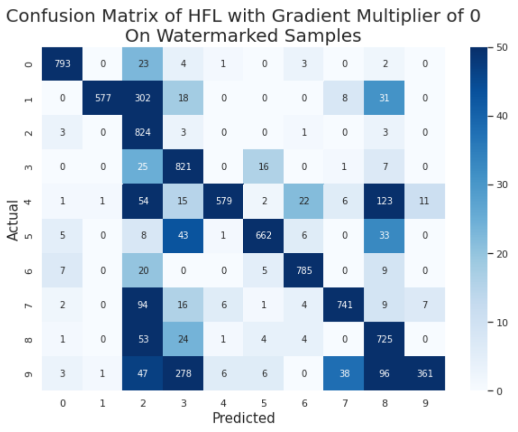
All in all, with a gradient multiplier of 0, the attack is not successful in either scenario. A capable model is learned in both cases, reaching test accuracies comparable to the baseline.



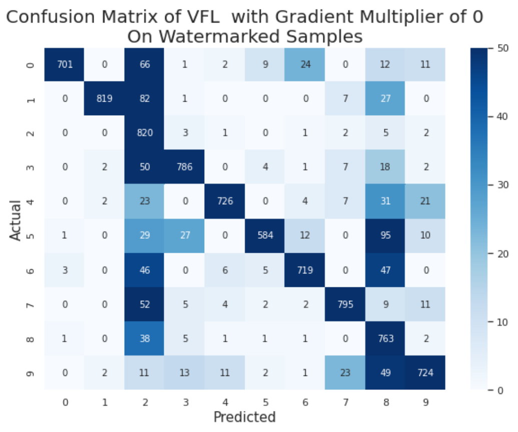
(a) HFL Tested on Unmarked Samples



(b) VFL Tested on Unmarked Samples



(c) HFL Tested on Watermarked Samples



(d) VFL Tested on Watermarked Samples

Figure 5.26: Confusion Matrices With a Gradient Multiplier of 0

The adversary is not revealed by the relative importance graph, and the adversary is as important as the other clients in determining the final label.

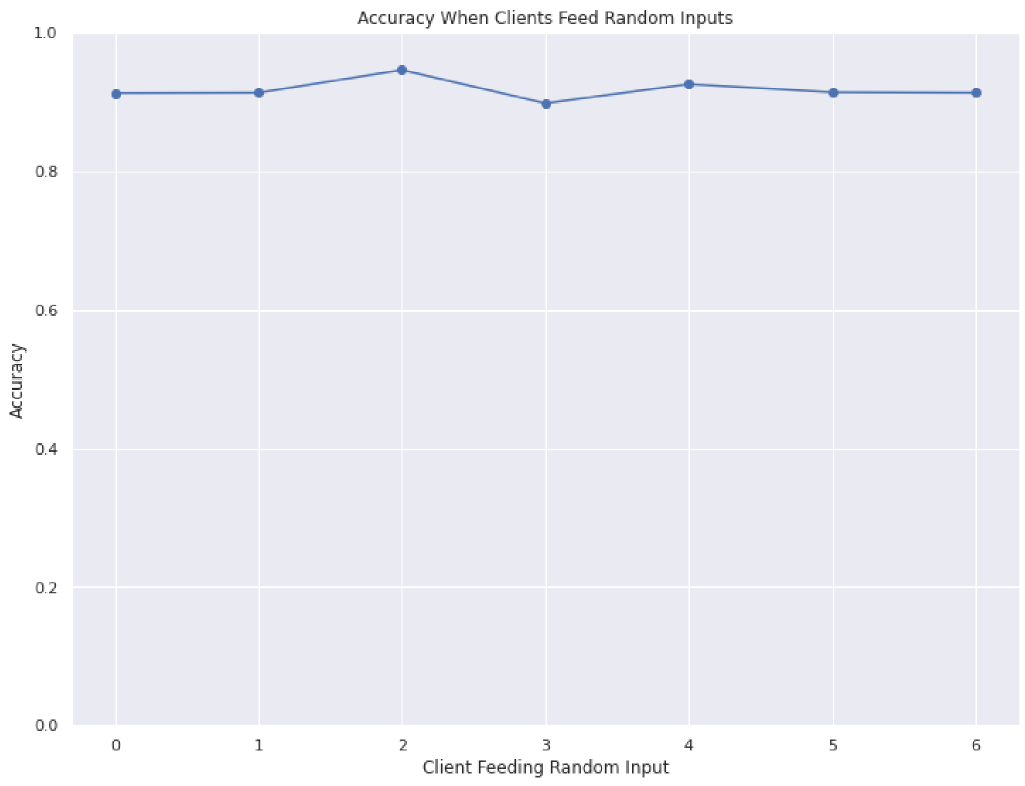


Figure 5.27: Relative Importance of Clients in VFL With Gradient Multiplier of 0

Chapter 6

Discussion

Only the comb model was sufficiently stable, even without attacks, to evaluate further of the two model architectures proposed. The chain model placed significantly higher stock in the inputs of the clients towards the end of the chain, despite the datasets not warranting such design. It was also only able to perform significantly worse than the comb model, according to all the evaluated metrics. As such, all the following evaluations focused solely on the comb model.

The comb model was able to achieve a high baseline accuracy of 98% over the training set, and 95% over the test set, in both the horizontal- and vertical settings. Once this baseline was established, the implementations were attacked with two common types of attacks: a data poisoning attack and a gradient poisoning attack.

The data poisoning attacks poisoned a fixed percentage of the samples available to the adversary. Defining the attack parameters in this way gave the adversary in the vertical setting an advantage, as they hold seven times as many (parts of) samples, as the adversary in the horizontal setting. However, the attack was extremely successful in both horizontal and vertical settings, able to achieve nearly 100% watermarking accuracy even with a very low number of samples marked during training. The difference in the efficiency of the attack arose only when 0.5% of the samples were watermarked, at which the attack in the vertical setting performed significantly better. At all higher percentages, the attacks in the two scenarios behaved identically.

The data poisoning attack was so successful, because the chosen watermark was so distinct. In all of the authentic samples, the pixels that would be watermarked were entirely black, devoid of information. As such, the model was able to very easily differentiate a watermarked sample from one without, thereby only needing a small number of samples to learn the backdoor behavior.

The data poisoning adversary was revealed in every experiment by the relative importance graph. Because the adversary was able to singlehandedly control the output of the model through watermarking, they became more important than the other clients. Because a reliable, unattacked baseline was evaluated first, this discrepancy was a clear giveaway as to the presence of such an attack. However, without such reliable baseline metrics, it would be impossible to know, from a single trained model, if the discrepancy in relative importances was from the inherent differences in the data, or from a malicious data poisoning attack.

Neither the accuracy, nor the confusion matrices revealed the presence of the data poisoning adversary in any of the experiments. Both metrics stayed near-identical to their baseline counterpart.

As such, the data poisoning attacks were extremely successful, and without an adequate and authentic baseline, their presence could absolutely go unnoticed.

The gradient poisoning attacks were even more successful than the data poisoning attacks. In the horizontal setting, the adversary applied poisoned gradients to the entire model in every one of their updates, while in the vertical setting, the adversary applied them only to the component of the model that they held. However, in the vertical setting, the adversary had seven times as many updates to poison as their horizontal counterpart.

With a gradient multiplier of 0, both adversaries succeeded only in delaying a learned model. With a value of -10, both were completely successful: both horizontal and vertical systems failed to learn a capable model. The differences in the two settings were revealed only with a gradient multiplier of -1. Here, the horizontal system was able to learn a model, albeit delayed when compared to the baseline, whereas the vertical system was deteriorated to a near-random level.

Despite being able to poison only their component in the network, the gradient poisoning adversary was able to avoid detection in the relative importance graph. As such, the gradient poisoning attacks, too, were entirely successful. Even with a reliable baseline, the adversaries were able to completely obstruct a model being learned, avoiding detection all the while.

Chapter 7

Summary and Future Work

This work presented an easily extendable visual taxonomy of the adversarial attacks and countermeasures in a federated learning environment. The taxonomy communicates the impact of each item with the number of lines permeating from it. In this format, the protection against a type of attack, the vulnerability of an FL scenario, and the versatility of a countermeasure is all represented clearly.

Then, both a peer-to-peer horizontal and SplitNN-based vertical simulation was implemented to learn from the MNIST dataset. Two different model architectures were evaluated to provide a baseline, one of which was adequate to continue investigating.

The system was then attacked with two types of attacks, data-poisoning and gradient-poisoning attacks. Both of these categories of attacks were evaluated with different parameters that controlled the efficiency of the attack. The results in the two scenarios were compared to the other, as well as to their corresponding baselines. The attacks were overwhelmingly successful in their respective objectives: implementing a backdoor or deteriorating the model performance.

The simulation took place in a controlled environment. The adversaries' capabilities were limited in that they could only attack the system with the attacks defined in the experiments. An avenue for future work could be to explore systems in which the adversaries are equipped with multiple attacks at once and can choose to employ any at any time.

The experiments did not include any active countermeasures. The investigated attacks were both extremely successful, but both have countermeasures that could limit their efficiency. Future work could focus on extending the system presented in this work with countermeasures and continuing the comparisons in the way laid out in this work.

In much the same way as relative importance was in this work, new metrics could be defined to evaluate the performance of a system. More concretely, the gradient poisoning adversary was not revealed by any metrics in this work, but the data poisoning adversary was clearly revealed by their relative importance. Future work could create new metrics that would reveal a gradient poisoning adversary, as well any adversary utilizing any other attack.

Bibliography

[1] Keith Bonawitz. Practical secure aggregation for privacy-preserving machine learning. *ACM*, 10 2017.

[2] Nader Bouacida. Vulnerabilities in federated learning. 04 2021.

[3] Iker Ceballos. Splitnn-driven vertical partitioning. 07 2020.

[4] Kewei Cheng. Secureboost: A lossless federated learning framework. 01 2019.

[5] Lixin Fan. Rethinking privacy preserving deep learning: How to evaluate and thwart privacy attacks. 06 2020.

[6] Malhar Jere. A taxonomy of attacks on federated learning. *IEEE*, 03 2021.

[7] Ang Li. Task-agnostic privacy-preserving representation learning via federated learning. 2020.

[8] Lingjuan Lyu. Threats to federated learning: A survey. 03 2020.

[9] Brendan McMahan. Communication-efficient learning of deep networks from decentralized data. 02 2016.

[10] Felix Sattler. Robust and communication-efficient federated learning from non-i.i.d. data. 03 2020.

[11] Praneeth Vepakomma. Split learning for health: Distributed deep learning without sharing raw patient data. 08 2018.

[12] Qiang Yang. *Federated Learning*. 2019.

[13] Qiang Yang. Federated machine learning: Concept and applications. *ACM*, 01 2019.

[14] Ligeng Zhu. Deep leakage from gradients. 01 2019.

Abbreviations

FL	Federated Learning
HFL	Horizontal Federated Learning
H2B	Horizontal to Businesses
H2C	Horizontal to Consumers
VFL	Vertical Federated Learning
FTL	Federated Transfer Learning

List of Figures

List of Tables