UNIVERSITY OF CAMBRIDGE

Joseph Rance

Evaluating attacks on fairness in Federated Learning

Computer Science Tripos - Part II

Robinson College

May 2024

Declaration of originality

Proforma

Candidate number:	**2417C**
Project Title:	**Evaluating attacks on fairness in Federated Learning**
Examination:	**Computer Science Tripos - Part II, 2024**
Word count:	**11,989**[1]
Code line count:	**3,878**
Project originator:	**The Candidate and Dr. Filip Svoboda**
Project supervisor:	**Dr. Filip Svoboda**

Original project aims

This project aims to support the claim that fairness and privacy cannot be simultaneously guaranteed in ML. It provides a new perspective on this problem by testing the robustness of private ML methods against attacks on fairness. The core goal is to construct a general framework on which a series of attacks and defences on private models are implemented. These defences are tested against simple classification, computer vision, and NLP baselines. The extension goals include repeating this analysis for a new defence method that specifically targets attacks on fairness and testing the attacks in a wider range of learning scenarios.

Work completed

This project was successful. I tested each of the selected defences and showed that none of the FL setups, including, surprisingly, the new defence against fairness attacks, were able to consistently yield fair and private training. As part of the framework's structure, I designed a system for simulating attacks on the server side to solve inter-client communication difficulties and yield high GPU utilisation. In addition to my original success criteria, I provided theoretical evidence that these shortcomings translate to a fundamental problem with current FL defences, indicating how future methods could overcome these issues.

Special Difficulties

None

[1]Counted with TeXount

Contents

1. Introduction **1**
- 1.1. Background . . . 1
- 1.2. Contributions . . . 2

2. Preparation **3**
- 2.1. Federated Learning for privacy in ML . . . 3
- 2.2. Fairness in private FL . . . 4
  - 2.2.1. Why fairness is important . . . 4
  - 2.2.2. The problem of fairness in FL . . . 5
- 2.3. Robustness in private FL . . . 5
  - 2.3.1. Attacks on private FL . . . 6
  - 2.3.2. Defences against attacks on private FL . . . 9
- 2.4. Project requirements . . . 11
  - 2.4.1. Success criteria . . . 11
  - 2.4.2. Starting point . . . 12
- 2.5. Software engineering tools and techniques . . . 12
- 2.6. Ethical considerations . . . 13

3. Implementation **14**
- 3.1. High-level design goals . . . 14
- 3.2. Constructing realistic baseline models. . . . 14
  - 3.2.1. Implementation strategy . . . 15
  - 3.2.2. Datasets . . . 16
  - 3.2.3. Baseline 1: Salary prediction . . . 16
  - 3.2.4. Baseline 2: Image classification . . . 17
  - 3.2.5. Baseline 3: Next word prediction . . . 17
  - 3.2.6. Optimising results . . . 18
- 3.3. Creating the attacks . . . 18
  - 3.3.1. General attack structure. . . . 18
  - 3.3.2. The backdoor attack . . . 20
  - 3.3.3. The fairness attack . . . 21
- 3.4. Reproducing the defences . . . 21
- 3.5. Repository overview . . . 21
- 3.6. Theoretical contributions . . . 22
  - 3.6.1. 'Private' gradient estimates can be accurately predicted . . . 23
  - 3.6.2. Attacks cannot be fairly detected . . . 25

4. Evaluation **28**
- 4.1. Attacks . . . 28
- 4.2. Defence: Trimmed Mean . . . 31
- 4.3. Defence: Krum . . . 32
- 4.4. Defence: Weak Differential Privacy . . . 35
- 4.5. Defence: Unfair-update detection *(extension)* . . . 37
- 4.6. Non-functional requirements . . . 38

5. Conclusions **39**
- 5.1. Implications and future work: a new perspective on robust aggregation . . . 39
- 5.2. Criteria completed. . . . 39

5.3. Lessons learnt 40

Bibliography **41**

A. A definition of fairness in FL. **45**

B. Methods for fair FL **46**

C. Defence algorithms. **48**
C.1. Krum 48
C.2. Unfair-update detection 48

D. Detection of fairness attacks by weight magnitude **49**

E. Fairnesss attacks under high data heterogeneity **50**

F. Testing attack recovery by introducing Krum *during* training **52**

G. Notation reference **53**

H. Project proposal **55**

1 Introduction

This dissertation tests how attacks on private Machine Learning methods can affect fairness. Specifically, this is the first exploration of how common, Byzantine-robust Federated Learning (FL) methodologies stack up against a new type of attack which seeks to introduce unfairness into the model. Furthermore, I create an extensible framework to facilitate the same analysis for future defences. In this section, I provide a brief overview of the background and implications of this work.

1.1. Background

In recent years, Machine Learning has become an important piece of infrastructure in our increasingly digital world [Eloundou et al. 2023; Department for Education 2023]. Ensuring the trustworthiness of these systems is therefore an important challenge. During the training of trustworthy ML models, we wish to guarantee, among other things [Liu et al. 2021; Commission, Directorate-General for Communications Networks, and Technology 2019]:

- **Privacy**: End users are not required to turn over their private data during training.
- **Fairness**: The model performs equally well on all categories of data within the test distribution (e.g. even accuracy for each class in the test set).
- **Robustness**: The resulting model reliably completes its task in the presence of Byzantine failures, such as the attacks discussed below.

However, while there exist methods which attempt to promise any two of these attributes (assuming the third is guaranteed) [Blanchard et al. 2017; Tian Li, Beirami, et al. 2021; Xu et al. 2021], it has proven practically difficult to ensure all three simultaneously.

A core framework for maintaining user **privacy** during training is Federated Learning. In FL, instead of sending private training data to a central server, each end user *locally* trains a model on their data, and then the server aggregates together the local models into a single, global model (see section 2.1 for a more in-depth explanation). In this dissertation, I investigate the interaction between **fairness** and **robustness** in the context of FL which provides a foundation to achieve privacy.

I claim that fairness and robustness are mutually exclusive properties in private FL. We have two options:

(a) To defend against attacks on fairness, we introduce a defence method which, as we will see, often introduces unfairness itself.

(b) We do not defend against attacks on fairness, thus leaving the model vulnerable to the unfairness these attacks introduce.

Thus, if there exists an attack that introduces unfairness, we cannot guarantee fairness in a private system.[1] In a recent paper, I showed that it is possible to construct such an attack [Candidate and Svoboda 2023].

[1]Unless we have a defence method that does not introduce unfairness itself.

In other words, imagine a system that can guarantee both fairness and privacy for *trusted* clients (that we assume will not run any attacks). If we use this system for *untrusted* clients (that might run attacks), we cannot continue to guarantee fairness and privacy without a defence against fairness attacks. However, many of these defences compromise fairness themselves.

Our ability to guarantee both fairness and privacy of a model in the presence of untrusted clients remains critical in areas such as healthcare, where patient records must be kept private, and unfair treatment between groups of people could have serious consequences. Therefore, a better understanding of how well common FL defence methods can prevent fairness attacks is important.

1.2. Contributions

In this project, I take a step towards answering the question 'Can private models be made robust against attacks on fairness, without introducing unfairness themselves?'. I present both theoretical (section 3.6.2) and empirical (sections 4.1-4.5) evidence that current defence methodologies either have a realistic chance of introducing unfairness into a dataset themselves or fail to consistently defend against attacks on fairness (justifying **(a)** above). I additionally extend the analysis of my previous work to further justify the effectiveness of fairness attacks in the absence of any defence method (**(b)**; sections 3.6.1 and 4.1).

To allow this analysis to be extended in the future to new defence methods, I construct a modular codebase with an extensible interface that facilitates the addition of new experiments. While there exist many open source implementations of common defences in FL (for example the Flower framework [Beutel et al. 2022]), most implementations of attacks on FL have not been made with extensibility in mind.

High-level project goals

- Produce a modular framework for testing defences against attacks on fairness in Federated Learning (chapter 3)
- Evaluate the effectiveness of the fairness attack in the presence of FL defence methods (chapter 4).
- Investigate the direct effects on the fairness of these defences without any attack (chapter 4).
- **Extension:** Repeat this analysis for a new defence that specifically targets attacks on fairness (section 2.3.2.2)
- **Extension:** Provide theoretical evidence that fairness and privacy are difficult to simultaneously guarantee in private FL (section 3.6.2).
- **Extension:** Establish the effectiveness of fairness attacks in the absence of any defence method for a wider range of training scenarios (option **(b)** above; sections 4.1 and 3.6.1)

In chapter 5, I conclude that, under certain assumptions, current methods for achieving robustness in FL can not reliably yield both fair and private model training. Furthermore, we can expect any new defence based on anomaly detection (a major theme of current methods) will exhibit similar shortcomings, indicating that future work would benefit from approaching the problem from a new perspective. These contributions are original to this dissertation.

2 Preparation

In this chapter, I explain how FL allows us to train models without viewing their private training data (section 2.1). I define fairness, explain why FL can introduce unfairness, and claim that we can, to some extent, simultaneously guarantee fairness *and* privacy in trusted settings (i.e. where we assume no client will run any attack; section 2.2).[1] Finally, I discuss how FL can be attacked, and methods for preventing these attacks, highlighting four algorithms that to implement in chapter 3. Although robustness in FL also remains an open question, these methods can, to some extent, simultaneously achieve robustness *and* privacy in untrusted environments (section 2.3).

However, I claim that each defence can introduce unfairness itself (section 2.3.2), leading to a dilemma for guaranteeing fairness and privacy in untrusted settings (i.e. where clients may run attacks):

- **EITHER:** we use a defence method, thereby introducing unfairness to our model
- **OR:** we do not use any defence, where we remain vulnerable to unfairness from fairness attacks

In both cases, unfairness can be introduced into the training process. At the end of this section, I explain how I plan to test the claim that these defences introduce unfairness themselves, and thus cannot be used to yield a fair and private training process (sections 2.4-2.6).

2.1. Federated Learning for privacy in ML

Consider a model that we wish to train on the private data held on users' local devices. To allow each user's data to remain private, we introduce *Differential Privacy* (DP) by adding noise to their data so its original contents cannot be determined with high confidence, but the model can still learn the overall distribution, given enough data[2].

Applying this technique directly to the dataset usually requires an impractically large amount of noise. In Federated Learning, instead of adding noise to the *data*, each client, i, locally trains a model on its dataset (starting from the global model, G_t) and adds the noise to the model's *parameters*, to yield a new set of parameters, $\mathbf{c}_i$, which are then sent to the server instead of the data (see fig. 2.1).[3] We then aggregate the parameters of each of these local models to produce a new global model, G_{t+1}, for example by taking the weighted mean of each parameter:

$$G_{t+1} = G_t + \sum_{i=0}^{m-1} p_i(\mathbf{c}_i - G_t) = \sum_{i=0}^{m-1} p_i \mathbf{c}_i \tag{2.1}$$

where $\sum_{i=0}^{m-1} p_i = 1$. Here, $\mathbf{c}_i - G_t$ is the *update* associated with client i. A common aggregation function is FedAvg, where p_i is set to be proportional to the size of client i's dataset. Appendix G provides a summary of all prominent symbols and mathematical notation used in this dissertation.

[1] Although the existence of a method that perfectly guarantees both remains an open question.

[2] Privacy is not a solved problem, although I do not discuss the shortcomings of DP (or any other private FL method) in this dissertation.

[3] FL also provides benefits other than for privacy.

In this dissertation, I investigate the tradeoff between fairness and robustness *in the context of private machine learning*. While I focus on Federated Learning, most of the claims made here generally apply to any method of training ML models on private data.

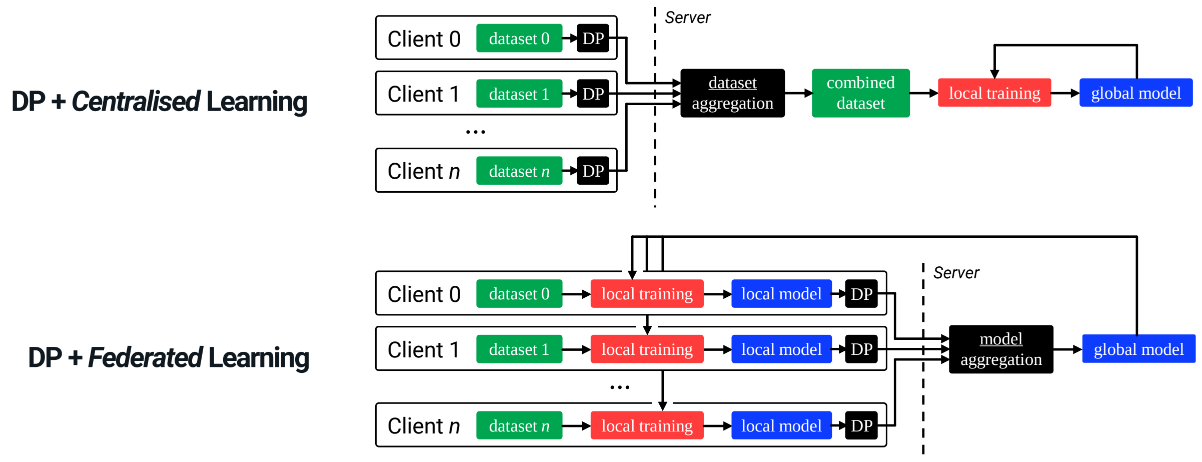

Figure 2.1.: Visualisation of the differences between centralised ML and Federated Learning. While in centralised ML, the training loop is entirely within the server, FL distributes training across the clients, so each iteration of the training loop requires communication in both directions.

2.2. Fairness in private FL

2.2.1. Why fairness is important

Throughout this dissertation, **I define a model to be fair if its performance across all categories of data (e.g. a category could be all data points that contain a specific feature) has low variance**. Appendix A provides a more precise definition for fairness. Fairness is important because unfairness can lead to discrimination against certain groups. Below are listed three instances where unfairness has caused harm in varying Machine Learning settings.

Re-offending rate prediction (Regression). In 2016, a model for predicting reoffending rates was reported to be twice as likely to incorrectly predict black defendants as being higher risk [Larson et al. 2016; Y. Li 2017]. This model was used to inform real sentencing decisions made by judges.

Face recognition (Computer Vision). Buolamwini and Gebru (2018) found that the error in facial recognition technology, when tested on darker-skinned females, was 43 times that of lighter-skinned males [also see NIST (2019)]. The deployment of similar technology for police surveillance has been found to contribute to greater racial disparity in arrests [Johnson et al. 2022].

Large Language Models (NLP). Salewski et al. (2023) demonstrated that LLMs were able to describe a car better when impersonating a man than a woman. As LLMs begin to see increased usage in everyday life, biases like these could reinforce stereotypes and increase the marginalisation of underrepresented groups [Y. Li 2017].

In all three cases, improving the performance on the disadvantaged data populations would be beneficial, even if this results in a drop in overall accuracy.

2.2.2. The problem of fairness in FL

Fairness is difficult to achieve because of the interaction between the following three phenomena:

(1) Slow rate of convergence. While FL can enable privacy guarantees, a single round of training in FL requires communication between the clients and the server, which can be very slow (compare the cycle lengths in fig. 2.1). The rate of convergence is usually improved (but not solved) by local training on multiple batches between communication rounds.

(2) Access to training data. Training models without access to the training data introduces some difficult problems. For example, we cannot change the relative proportions of different categories of data within a private dataset in the same way that data augmentation can be used to do this in the centralised case by 'generating' more data for underrepresented classes [Sharma et al. 2020; Chawla et al. 2002].[4] In section 2.3.1, I discuss how privacy creates a vulnerability used by adversarial attacks on FL.

(3) Heterogeneity. In almost all real-world scenarios, clients that participate in training are not identical. They often exhibit both **data heterogeneity** (their datasets are distributed slightly differently and/or differently sized), and **system heterogeneity** (the performance and uptime of their training hardware vary). The length of a single FL round is usually dominated by the slowest client, so heterogeneous client populations can result in slow round completion times.

Fairness is difficult to achieve. All three of the above issues contribute to making fairness difficult to achieve. Due to data heterogeneity, the distribution of data used to train our model may be different to the distribution that we wish to test on: even if the union of client datasets is the 'correct' distribution,[5] unfairness can still be introduced into the model at training time because models are trained locally for multiple batches. Since we cannot access this data, unlike in the centralised case, we cannot use data augmentation to even out these imbalances.

Improving fairness in FL. Current methods of improving fairness mostly focus on increasing the weight, p_i in eq. (2.1), of clients that report high loss on their local datasets, reasoning that this may increase the impact of categories of data that the model is currently performing poorly on. While these techniques have been shown to have positive effects on fairness, fairness remains an open problem in FL, as using this loss heuristic does not necessarily always guarantee perfect fairness. Critically, if a specific category of data is no longer present during training (as is the case in chapter 4), these methods do not meaningfully improve the model's accuracy on this category of data. Some notable methods are discussed in appendix B.

2.3. Robustness in private FL

In this section, I give an overview of common attacks and defences in Federated Learning, before discussing some shortcomings of these algorithms in achieving robustness and privacy. I also briefly explain how defence methods can introduce unfairness.

[4]Clients also do not communicate high-level attributes of their dataset, such as as the number of data points for each attribute, so we may not even know how the combined training set is distributed.

[5]and all classes are 'equally difficult to learn'

2.3.1. Attacks on private FL

Attacks on centralised (non-FL) models rely on a part of the supply chain becoming compromised [Shumailov et al. 2021; Bober-Irizar et al. 2022; Chen et al. 2017; Rance et al. 2022]. For example, some attacks aim to poison the training dataset. While FL remains vulnerable to such attacks, an attacker can more directly compromise the entire supply chain by posing as a client.

Threat model for attacks on FL

The attacker:

- Has access to the previous global model, G_t, and an estimate for the number of clients.
- Controls a minority subset of clients, $\{i, \ldots, j\}$, so that, at each round, they may submit *arbitrary* parameters, $\{\mathbf{c}_i, \ldots, \mathbf{c}_j\}$, for aggregation.
- Aims to insert some malicious set of parameters, $\mathbf{x}$, into the global model, G_{t+1}.

The defender (FL server):

- May view the model submitted by any client, but does not know which are malicious
- Aims to eliminate the effects of all malicious clients, without significantly reducing the effects of benign clients

Such a threat could reasonably exist in systems where clients are either untrusted (i.e. the attacker joins as a new client), or can be somehow compromised (e.g. by malware).

In a typical attack on FL, (**A**) the attacker has a set of **target parameters**, $\mathbf{x}$, which they wish to insert into the global model. (**B**) The attacker then computes a set of local, **malicious parameters**, $\mathbf{c}_i$, such that, after aggregation with the other clients' models, the resulting global model, G_{t+1}, is approximately equal to $\mathbf{x}$.

A *robust* FL training regime must prevent this kind of malicious insertion of any $\mathbf{x}$ into the model, but previous work has focused on the case where $\mathbf{x}$ contains a *backdoor*. In a fairness attack, $\mathbf{x}$ is instead trained to be unfair.

2.3.1.1. Computing the target parameters (A)

Constructing models with backdoors. A typical backdoor attack aims to introduce new functionality to a model in the presence of some trigger. A simple method of adding this functionality is to update the dataset, D, to include samples $(T(x), F(y))$ for each legitimate sample (x, y) already in D. Here, T constructs the backdoor trigger from input x, for example, by overlaying a small pattern onto the image, and F encodes the backdoor functionality in the presence of the trigger, for example, we could have $F(\cdot) = 0$ [Gu, Dolan-Gavitt, and Garg 2019].

Constructing unfair models. The fairness attack aims to train G_t (and therefore $\mathbf{x}$) to have high accuracy on some target dataset, $D_T \subseteq D$, and low accuracy on the other data, $D_N = D \backslash D_T$. For example, we may want high accuracy on only classes 0 and 1. We therefore compute the parameters for $\mathbf{x}$ by fine-tuning G_t using only data in $\mathcal{D}_T$.

2.3.1.2. Computing the malicious parameters (B)

Once we have computed our target model, $\mathbf{x}$, we need to find some $\mathbf{c}_i$ that results in the global model, G_{t+1}, being approximately equal to $\mathbf{x}$. A naïve method would be to submit $\mathbf{x}$ directly to the aggregator. In practice, these weights are usually unable to significantly change the global model after aggregation with a large number of clean clients. Bagdasaryan, Veit, et al. (2019) propose a more powerful strategy, *model replacement*, that allows the attacker to substitute arbitrary target parameters, $\mathbf{x}$, directly into the global model.

The model replacement attack. The idea behind model replacement is that we can construct $\mathbf{c}_i$, so that, when it is passed into the FedAvg update rule, the value of G_t is replaced with $\mathbf{x}$. We therefore set $\mathbf{c}_i$ to a scaled version of the target parameters, added to a term that removes the original G_t from the update rule: $\mathbf{c}_0 = \frac{n}{n_0}(X - G_t) + G_t$, where n_i is the number of training samples of client i, and $n = \sum_{i=0}^{m-1} n_i$.[6] Now, when we pass this into FedAvg, we get

$$G_{t+1} = G_t + \left[\frac{n_0}{n}(\mathbf{c}_0 - G_t)\right] + \sum_{i=1}^{m-1}\frac{n_i}{n}(\mathbf{c}_i - G_t) \tag{2.2a}$$

$$= G_t + \left[\frac{n_0}{n}\left(\frac{n}{n_0}(\mathbf{x} - G_t) + G_t - G_t\right)\right] + \sum_{i=1}^{m-1}\frac{n_i}{n}(\mathbf{c}_i - G_t) \tag{2.2b}$$

$$= \mathbf{x} + \sum_{i=1}^{m-1}\frac{n_i}{n}(\mathbf{c}_i - G_t) \tag{2.2c}$$

where the previous global model, G_t, has been replaced by our $\mathbf{x}$ in eq. (2.2c). If we assume convergence, as $t \to \infty$, $\mathbf{c}_i - G_t \to 0$ for all $i \neq 0$, we get $G_{t+1} = \mathbf{x}$. Bagdasaryan, Veit, et al. (2019) further show how this attack can be modified to make it more difficult to defend against.

Why model replacement does not work for fairness attacks. While backdoor attacks attempt to leave the main task accuracy unchanged, fairness attacks expect to reduce this significantly, preventing convergence. This is an important difference, because the assumption that updates produced by legitimate clients will converge to 0-length (vector norm) in the model replacement attack will therefore no longer be true.

The solution: an update prediction attack. The model replacement attack requires this convergence assumption because we do not know the updates produced by other clients. In a recent paper, I introduced an attack that solves this problem, making attacks on fairness possible [Candidate and Svoboda 2023].

In chapter 3, I prove that the difference between the mean client update, $\sum_{i=0}^{m}\frac{n_i}{n}(\mathbf{c}_i)$, and some set of weights, $\mathbf{w}$, that have been trained on data that is i.i.d. to the union of the clients' datasets is normally distributed with 0 mean, and variance a decreasing function of the amount of data seen during training, tending to 0 in the limit (**Theorem 1**, section 3.6.1). Thus, for large datasets and batch sizes, an attacker may be able to accurately predict these updates. This forms the basis for the update prediction attack I introduced in Candidate and Svoboda (2023), although the theoretical justification in section 3.6.1 is original to this dissertation.[7]

[6] These definitions are summarised in appendix G for later reference.

[7] The contribution of this dissertation to the formulation of the update prediction attack is twofold: **(1)** I provide new theoretical justification for the attack in section 3.6.1, and **(2)** I generalise the formulation of the attack to a malicious clients (section 3.3).

Instead of subtracting the global model, G_t, as is the case in the model replacement attack, the attacker can now subtract their update prediction, $\mathbf{w}$, thus (approximately) eliminating *all* other terms in the FedAvg update rule. We set $\mathbf{c}_0 = \frac{n_0-n}{n_0}\mathbf{w} + \frac{n}{n_0}\mathbf{x}$, so the FedAvg update becomes

$$
\begin{aligned}
G_{t+1} &= G_t + \frac{n_0}{n}\left(\frac{n_0-n}{n_0}\mathbf{w} + \frac{n}{n_0}\mathbf{x} - G_t\right) + \sum_{i=1}^{m}\frac{n_i}{n}(\mathbf{c}_i - G_t) && \text{(2.3a)}\\
&= G_t + \frac{n_0-n}{n}\mathbf{w} + \mathbf{x} + \sum_{i=1}^{m}\frac{n_i}{n}(\mathbf{c}_i) - \sum_{i=0}^{m}\frac{n_i}{n}(G_t) && \text{(2.3b)}\\
&= \mathbf{x} + \sum_{i=1}^{m}\frac{n_i}{n}(\mathbf{c}_i) - \frac{n-n_0}{n}\mathbf{w} && \text{(2.3c)}\\
&\approx \mathbf{x} && \text{(2.3d)}
\end{aligned}
$$

Here we directly get $G_{t+1} = \mathbf{x}$ without any convergence assumption, thus allowing us to use any set of parameters for $\mathbf{x}$ (see fig. 2.2).[8] The 'catch' is that we require a *sufficiently large dataset and batch size* in order for these predictions to be accurate. This is task-dependent, but I empirically show that this attack works even for small batch sizes in section 4.1.

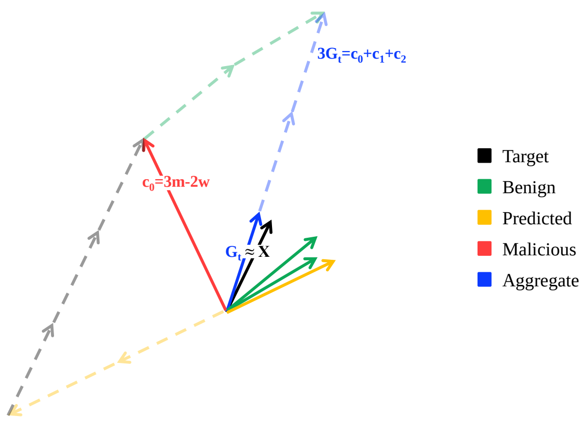

Figure 2.2.: Visual representation of how the attack is constructed. Each arrow represents a single client's parameter vector. In practice, the angles between G_t, c_i for $i > 0$, w, and $\mathbf{x}$ are small, so the length of c_0 is not as extreme as this diagram suggests.

In order to fit in our threat model, **no knowledge of parameters submitted by other clients is required by this attack.** The attacker only requires an approximate estimate for the amount of data, $n - n_0$, contributed by other clients. Such an estimate need not be exact and could be iteratively increased each round until an effective value is found.

2.3.1.3. Why the impact of fairness attacks is important

At the beginning of this chapter (2), I present two scenarios:

- To defend against attacks on fairness, we introduce a defence method which, as we will see, often introduces unfairness itself.
- We do not defend against attacks on fairness, thus leaving the model vulnerable to attacks on fairness.

[8] In this case of a single attacker, the model replacement attack is a special case of the update prediction attack in which we assume all clients submit 0-length updates.

In both cases, unfairness can be introduced into the training process. However, without an attack on fairness, we *could* attempt to guarantee fairness in a private system by using the methods described in appendix B and avoiding the use of any FL defence methods (perhaps there is no advantage to any client to introduce a backdoor). Thus, **the possibility of an attack on fairness implies we cannot guarantee fairness in a private system, even in the presence of a defence method**.[9]

Practical impact. Attacks on fairness do not only have a theoretical impact. When one or more clients benefit from an accuracy imbalance between certain types of data in the global model, these clients are incentivised to perform such an attack.

2.3.2. Defences against attacks on private FL

2.3.2.1. Defences against Byzantine client failures

Anomaly detection: an FL defence philosophy. Almost all attacks on FL follow the general theme of producing out-of-distribution parameters, $\mathbf{c}_i$, to induce out-of-distribution behaviour in the global model.[10] Thus, the proposed solution is quite direct: detect which clients produce these out-of-distribution parameters, and ignore them.[11]

Defence 1: anomaly detection with Krum. Blanchard et al. (2017) introduce one possible method of anomaly detection. The multi-Krum function returns the unweighted average of the models submitted by the n[12] clients with the shortest sum of squared distances to their closest $n - f - 2$ neighbours (see algorithm 1, appendix C).

This function aims to provide some notion of *Byzantine resilience* by discarding parameters that lie far from a cluster of benign models. If the attacker can control f clients, they will not be able to construct a dense cluster of malicious models which can be mistaken for a cluster of benign models because this must contain at least $n - f - 2$ points, which is assumed to be greater than f in our threat model. Specifically, the authors define, and prove for the Krum function, the notion of (α, f)-Byzantine resilience.

Definition 2 ((α, f)-Byzantine resilience)
For $0 \leq \alpha < \pi/2$ and $0 \leq f \leq n$, if $V_i \sim G$, independently, and $\mathbb{E}[G] = g$, F is (α, f)-Byzantine resilient if, for $B_0 \in \mathbb{R}^d, \ldots, B_{f-1} \in \mathbb{R}^d$,

$$F = F(V_0, \ldots, B_0, \ldots, B_{f-1}, \ldots, V_n) \tag{2.4}$$

satisfies $\langle \mathbb{E}[F], g \rangle \geq (1 - \sin(\alpha))||g||^2 > 0$ and, for $r \in [2, 3, 4]$, $\mathbb{E}[||F||^r] \in O(L)$ where L is a linear combination of $\mathbb{E}[||G||^{r_i}]$ and $\sum_{i=0}^{n-2} r_i = r$.

This definition requires that the aggregated value, F, is expected to be within some radius, $r = \|g\| \sin(\alpha)$, of the true, benign update, g. However, there may lie malicious updates within the d-ball defined by g and r, as shown, for example, by Bagdasaryan, Veit, et al. (2019).

[9] Unless we have a defence method that does not introduce unfairness.

[10] If $\mathcal{D}$ is the distribution we wish to train on, some data point s is out-of-distribution if $s \notin \text{Image}(\mathcal{D})$. A set of parameters are in-distribution only if they can be obtained by training on in-distribution data. At the beginning of this section, we assume attacks must produce out-of-distribution parameters.

[11] This does not violate any privacy requirement, because DP guarantees privacy under any transformation: we only analyse the model *after DP has been applied*, which cannot tell us much about the model *before DP was applied* for in-distribution data.

[12] In order to follow the notation used by Blanchard et al. (2017), m and n take different definitions from the rest of this dissertation in this section.

Defence 2: Trimmed mean anomaly detection. In the trimmed mean defence [Yin et al. 2021], we instead eliminate the client parameters with the n highest and lowest norms. The FedAvg function is therefore replaced with

$$\text{trmean}(M)_k = \frac{1}{(1-2\beta)m} \sum_{x \in T(M_{*,k})} x \tag{2.5}$$

where $M \in \mathbb{R}^{m \times d}$ is a matrix with $M_{i,*} = \mathbf{c}_i$, and $T(\mathbf{v})$ returns the set of $\mathbf{v}$'s elements with the $\lfloor \beta m \rfloor$ largest and smallest removed. The authors show that this achieves order-optimal statistical error rates for smooth, non-convex population loss functions.

The problem with anomaly detection. It is practically difficult to achieve high precision (low number of accepted attacks) without low recall (high number of rejected benign clients) when selecting benign clients [see, for example, Rieger et al. 2022]. Mhamdi, Guerraoui, and Rouault (2018) show that these Byzantine-resilient schemes above accept parameters within a margin around the true set of clean client parameters that can have a size in $\Omega(\sqrt{d})$, where d is the number of dimensions in the model, which is usually large. There have therefore been many attacks that take advantage of this loose bound to trick anomaly detection methods into accepting malicious parameters [M. Baruch, G. Baruch, and Goldberg 2019].

Anomaly detection introduces unfairness. As described in section 2.2.2, we can improve fairness by increasing the effect on the global model of parameters in the tail of the update distribution (often using loss as a heuristic for this). However, if our anomaly detection does not have perfect recall, the clients it is most likely to reject are those that submit parameters in the tail of the distribution, therefore introducing unfairness into the model. Because these parameters are entirely removed, this problem cannot be solved by any method of reweighting minority clients discussed in appendix B. Furthermore, in section 3.6.2 I prove that any anomaly detection defence that guarantees robustness *must* also reject some benign clients, and therefore can introduce unfairness. This is a fundamental problem with anomaly detection algorithms, which has so far not been addressed.

Defence 3: Weak Differential Privacy. Given the shortcomings of defences based on anomaly detection, we may wish to construct defences that follow a different philosophy. However, while there are defences that can retain fairness *locally* for each client [Tian Li, S. Hu, et al. 2021], methods for training a fair global model almost always fall back on anomaly detection in some way.[13] For example, Sun et al. (2019) suggest that a weaker version of Differential Privacy (i.e. clipping vector norm and adding noise to each set of parameters) may be an effective defence against some attacks. However, while this does not explicitly remove out-of-distribution updates, its effects disproportionately impact these clients, which induces effects on both fairness [Bagdasaryan and Shmatikov 2019] and robustness through the same mechanisms as the anomaly detection defences.[14] Recent work has shown that the weak DP defence can also be ineffective at preventing attacks in some scenarios [H. Wang et al. 2020].

[13]An alternative method for providing robustness *without* anomaly detection attempts to use a small, verifiably legitimate dataset to 'clean' the final model, post aggregation [Zeng et al. 2022; Mao et al. 2023]. However, our ability to remove backdoors from ML models remains an open question, because SGD cannot guarantee to *remove* functionality in the same way as it can for *adding* it. Additionally, after a successful fairness attack, much of the knowledge of other categories of data will be lost, so fine-tuning an attacked model to a fair dataset is unlikely to recover this functionality.

[14]This does not necessarily mean that differentially private FL cannot be fair in general. Some techniques attempt to provide privacy through DP while avoiding this issue [Jagielski et al. 2019].

2.3.2.2. Defences against Fairness attacks

All three defences described above remain theoretically sound when we change the attacker's objective to target fairness. For example, for the Krum defence, the authors justify the effectiveness of an attack similar to that of the update prediction attack in their first lemma. Similarly, appendix D shows why we might expect the trimmed mean defence to remain effective.

Defence 4: Unfair-update detection - a new defence for fairness attacks. While H. Wang et al. (2020) have shown that verifying that a model does not contain any backdoors is computationally intractable, verifying that a model is fair across a set of predetermined attributes is relatively simple to do with high confidence. This suggests a simpler defence may be to directly measure the fairness impact of each client's model and assume clients that significantly reduce fairness are malicious. This defence is described more specifically by algorithm 2 (appendix C). To minimise the required server-side computation, schemes similar to Credit-Based Client Selection [Khorramfar 2023] could be implemented to allow trusted clients to carry out this validation instead.

2.4. Project requirements

In the core part of this project, I aim to show that common FL defences cannot prevent the fairness attack in a fair manner. I focus this analysis on testing the effects of fairness attacks on models trained with the four defences listed above.[15] As an extension, I generalise these claims by proving that any defence based on anomaly detection cannot be guaranteeably fair. I additionally extend my previous analysis of fairness attacks [Candidate and Svoboda 2023] to show that their formulation is well-founded in a wide range of learning scenarios.

2.4.1. Success criteria

Core success criteria

(a) Build a modular codebase for testing attacks on FL, consisting of implementations of:
   (1) A baseline model with high accuracy on three common FL datasets (section 3.2).
   (2) A model replacement attack and an update prediction attack, achieving results comparable to that of previous papers (section 3.3).
   (3) The existing weak DP, Krum, and trimmed mean defences, achieving results comparable to those of previous papers against the model replacement attack (section 3.4).

(b) Evaluate the effectiveness of the defences on the attack on fairness (sections 4.1-4.5).

(c) Construct a dataset to demonstrate that each successful defence introduces unfairness into the FL training routine (sections 4.3, 4.2, and 4.5).

These criteria align with my project proposal (appendix H), with the addition of requirement **(c)**.[16] As planned, I selected two extensions from my original proposal. I have listed these below, as well as two additional proofs included in chapter 3.[17]

[15] Although I cannot claim that a failure of these defences to prevent attacks on fairness implies this is the case for *all* current defences, every defence I have encountered is, in some way, based on the ideas covered by these methods [for example, Nguyen et al. 2023], so any shortcomings are likely to be present in other defences.

[16] My original proposal aimed to focus this dissertation around the *direct* effects of fairness attacks. However, as I further explored the tradeoff between fairness and robustness, it evolved to focus more on this problem. This motivated the addition of requirement **(c)**, above.

[17] These were not originally intended to be part of my project goals. However, since they make up a significant part of this dissertation, I have included them as distinct criteria.

Extension criteria

(a) Test the unfair-update detection defence described in section 2.3.2.2 (section 4.5).

(b) Provide a proof that fairness and privacy cannot be simultaneously guaranteed using anomaly detection (section 3.6.2).

(c) Test the performance of the update prediction attack against other aggregation functions, such as those described by Reddi et al. (2021) (section 4.1).

(d) Theoretically justify the assumptions of the update prediction attack (section 3.6.1).

To summarise, I build a framework for testing attacks and defences on FL. I use this to implement three baseline models, two attacks, and four defences, and test all 24 combinations of these.

2.4.2. Starting point

Initial codebase. Before this project, I wrote a paper introducing attacks on fairness in Federated Learning. Therefore, I began with a simple implementation of the fairness attack. However, as stated in my initial project proposal, I decided to start this project from scratch.

Knowledge of FL. My previous FL experience was minimal: I had only worked on the previous paper on weekends for three months before the beginning of term, and the majority of this dissertation had been completed before the module on Federated Learning in Lent term.

2.5. Software engineering tools and techniques

Languages and libraries. I implemented this project in PyTorch because **(1)** it has plenty of documentation available online; **(2)** I prefer the dynamic computation graph over static alternatives such as TensorFlow; and **(3)** I already have experience using PyTorch. I chose to use the Flower Federated Learning framework to handle the FL simulation because this is the most common framework used at the Computer Laboratory at the University of Cambridge.

Documentation. I used python docstrings and type hints to annotate every important function, class, or module. To make the project easier to set up, I provided a Docker container which gathers the necessary dependencies, alongside installation information in the codebase's README. I used the PyLint analyser to ensure I kept to the PEP 8 Python style guide.

Testing strategy. This project does not lend itself to a large amount of unit testing because this would result in a significant waste of energy and GPU time in model training. I instead test individual components 'on the fly', to verify the data as the experiments run. I implicitly performed full end-to-end testing of all dataset-attack-defence combinations in my evaluation.

Hardware. I used my personal laptop (Dell G7 15) to implement the codebase, write this dissertation, and run some minor code. However, I used the CamMLSys GPU cluster (mainly Nvidia RTX 2080 GPUs) to run most of the experiments.

Tools. I used the VSCode and Vim text editors with Git for version control, storing my code on GitHub so I could access it from the remote server. To schedule my experiments on shared servers, I used Slurm with a Makefile that automatically generates configurations for each experiment. I used conda to run my code in an isolated development environment.

Development model. I followed an iterative model of software development: I began with an initial fairness attack implementation, and then incrementally planned, implemented, and tested each new feature (attack, defence, or baseline), filling in the main results table as my project progressed. This process minimised risk, as the results gathered from my project grew linearly with the implementation progress, while also working well with the planned structure of my codebase.

Project planning. I used a GitHub project to break down each goal into sprint tasks to actively track the progress of my repository against my original plan (see appendix H). I made minor modifications to this plan, to move some of the implementation work earlier to free up revision time later in the year.

2.6. Ethical considerations

Software licenses. Table 2.1 shows the licenses associated with the libraries used in my project. Because I want my project to be available for others to use and extend, I used an MIT license for my codebase.

Table 2.1.: Licenses for software libraries used.

Library	License
NumPy	BSD-2-Clause
scikit-learn	BSD-3-Clause
imbalanced-learn	MIT
matplotlib	PSF-2.0
Flower	Apache-2.0
PyTorch	BSD-3-Clause
torchvision	BSD-3-Clause
pandas	BSD-2-Clause
🤗 Transformers	Apache-2.0

Datasets. The UCI Adult Census dataset is released under a CC0 license, and the CIFAR-10 dataset is released under an MIT license. The Reddit dataset does not have a license, however, I downloaded it from a reputable source, which is used in many other FL papers.[18] Data in both the Census and Reddit datasets has been anonymised.

Open sourcing malicious code. I intend to make the code for this project open-source. There are ethical considerations for this because the codebase includes implementations of attacks on Federated Learning. However not releasing the code would not prevent a motivated attacker from being able to obtain such implementations. Additionally, open-source codebases provide useful baselines to help improve the robustness of existing systems.

[18]see: `https://fedscale.ai/docs/dataset`, and its inclusion in: `https://leaf.cmu.edu/`

3 Implementation

In this chapter, I discuss the construction of the codebase I use to test each of the four defences against the fairness attack. I will:

1. Construct the realistic FL training scenarios in which a model can be trained fairly (section 3.2)
2. Implement and optimise the fairness and backdoor attacks in each scenario (section 3.3)
3. Implement and optimise each defence to prevent the attacks, without changing the attack configuration (section 3.4)

This tests the best-case scenario for each defence because in practice the defence's configuration would not be optimised for the specific attack setup used. Similarly, I use the simplest version of every attack, rather than including any concealment terms. Therefore, any defence that does not prevent the attack is unlikely to be successful in any other configuration.

3.1. High-level design goals

The aims of my codebase are twofold:

1. To provide a platform to test the effectiveness of the four identified defences at preventing attacks on fairness.
2. To act as an expandable baseline that new defences could be added to in order to compare their performance.

To achieve these aims, I will focus my implementation around the below five, non-functional requirements.

Performance. Because GPU time can be expensive, I optimise my codebase to efficiently use the available GPUs (sections 3.2.6 & 3.3.1).

Configurability. To make it simpler to run many experiments, it is possible to control all of an experiment's hyperparameters through a simple configuration file (section 3.2.1).

Extensibility. The codebase is built in a modular fashion that allows new attacks and defences to be easily added without having to change much/any existing code (sections 3.2-3.4).

Usability. The standard interfaces that are implemented for each attack and defence are clearly documented to make it easier to understand the codebase and add new modules (section 2.5).

Compatibility. I use mainstream libraries (e.g. PyTorch) so that it is simpler to add new functionality (section 2.5).

3.2. Constructing realistic baseline models.

This section describes my implementation for core criterion **(a.1)**, section 2.4.1. To achieve the non-functional requirements in section 3.1, I begin with an outline of how I have split each component of the training pipeline into a configurable, extensible package (section 3.2.1). I then describe the three tasks I train on (section 3.2.2) and how a model is trained for each (sections 3.2.3-3.2.5)

3.2.1. Implementation strategy

Configurability. Each experiment is defined by a YAML configuration file. The configuration of the baseline splits naturally into three independent components: **(1)** the dataset, **(2)** the model, and **(3)** the training setup. The configuration file is therefore structured to reflect this:

```
seed: 0
task:
    dataset:
        name: ...
        ...
    model:
        name: ...
        ...
    training:
        ...
```

The YAML file *should* deterministically control the experiment's results. Unfortunately, in practice, an issue with seeding the Ray library means that this is not quite true.

Extensibility. Each component of the codebase (datasets, models, ...) is defined by an abstract interface so that any new component that implements the correct interface can be frictionlessly inserted into the codebase and selected for testing using the corresponding configuration section.

A full dataset is split into many smaller sets: first, into the train, validation, and test sets, and then each train set into a single set per client, and the validation & test sets into one per attribute that we are interested in the accuracy of. Therefore, the dataset loading function must return an instance of the following `dataclass`:

```
@dataclass
class Datasets:
    name: str
    train_datasets: list[torch.utils.data.Dataset]
    validation_datasets: dict[str, torch.utils.data.Dataset]
    test_datasets: dict[str, torch.utils.data.Dataset]
```

Similarly, a data loader is constructed to efficiently sample each dataset, so another `dataclass` is defined for these:

```
@dataclass
class DataLoaders:
    name: str
    train_datasets: list[torch.utils.data.DataLoader]
    validation_datasets: dict[str, torch.utils.data.DataLoader]
    test_datasets: dict[str, torch.utils.data.DataLoader]
```

PyTorch provides a standard `torch.nn.Module` that each of the implemented models inherits from. This improves compatibility with other code that uses PyTorch (or any ONNX-compatible framework).

Putting it all together. I provide a package containing a collection of objects that follow each interface,[1] and then select the correct dataset and model based on the configuration with

```
dataset = DATASETS[config.task.dataset.name](config.task.dataset)
model = MODELS[config.task.model.name](config.task.model)
```

[1] This setup differs from that of https://github.com/camlsys/fl-project-template, where all components for each task are packaged together.

New components can easily be added by implementing the correct interface, adding an entry to the corresponding collection (i.e. `DATASETS`, `MODELS`, …), and setting the configuration to select this new entry). Given a dataset, model, and a configuration for the training process, the FL simulation closely follows the standard setup for the Flower framework.

3.2.2. Datasets

To ensure my experiments are applicable to real scenarios, I test on the following three datasets, which cover a range of ML fields, reflecting the areas discussed in section 2.2.1:

Salary prediction on the UCI Adult Census dataset (Classification). Classification of whether individuals earn more or less than \$50K per year [Becker and Kohavi 1996]. This dataset has a clear relationship to real societal issues.

Image classification on the CIFAR-10 dataset (Computer Vision). Classification of images into one of 10 classes [Krizhevsky 2009]. This is a standard task, which allows for a direct comparison to previous work.

Next word prediction on the Reddit dataset (NLP). Prediction of the next word in a Reddit comment [Pushshift n.d.]. This dataset represents an NLP task, which, is becoming increasingly relevant in Machine Learning (and especially Federated Learning).

All three datasets have been tested extensively in previous work on robust Federated Learning [Bagdasaryan, Veit, et al. 2019; Bhagoji et al. 2019; H. Wang et al. 2020; Nguyen et al. 2023; McMahan et al. 2023]. For realism, the data must also be distributed between clients in a 'realistic' way, which is discussed in section 4.1 and appendix E.

To train a baseline model on each of these datasets, I must **(1)** define a function to download and preprocess the dataset, returning it as an instance of the `DataLoaders` class, **(2)** construct a model that inherits from `torch.nn.Module`, and **(3)** find a hyperparameter configuration that results in an effective model.

3.2.3. Baseline 1: Salary prediction

Dataset preparation. Each feature was either normalised or One Hot Encoded. In all three baselines, I allocated equal amounts of i.i.d. data to all clients.[2] However, as shown in fig. 3.1, there is a significant amount of bias within the combined distribution.

Model construction. Similarly to previous works [Bhagoji et al. 2019], I constructed the model from three fully connected layers, with the first two using dropout and a ReLU activation, and the last using a sigmoid activation.

Hyperparameter selection. The model trained for 40 rounds, with each of the 10 clients performing 10 iterations over its local dataset on each round. To select hyperparameters, I either manually ran ablation tests, or, in complex cases, employed a grid search over a reasonable range. I set the SGD learning rate to 0.01, reducing it to 0.002 at round 25. With this setup, I was able to achieve 84% accuracy, which is in line with previous work [Bhagoji et al. 2019].

[2]This is a somewhat unrealistic scenario, but, since I am interested in testing whether the defences work *in the best case*, I would not expect this setup to change any results because I expect heterogeneity to make defending the attacks *more* difficult. I have performed experiments to show that heterogeneity does not prevent the attack from functioning in appendix E.

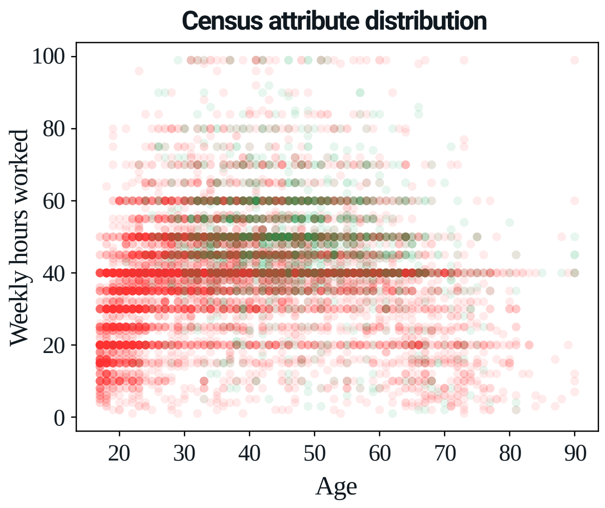

Figure 3.1.: Joint distribution of weekly hours worked and age statistics, where each green point represents a single individual earning more than $50K per year, and each red point represents an individual earning less than or equal to $50K per year.

3.2.4. Baseline 2: Image classification

Dataset preparation. Each colour channel is independently normalised and each sample is (lazily) augmented by randomly applying translations and horizontal flips. This data preparation scheme is the same as that of Zagoruyko and Komodakis (2017). Figure 3.2 shows a sample of the data before and after transformation.

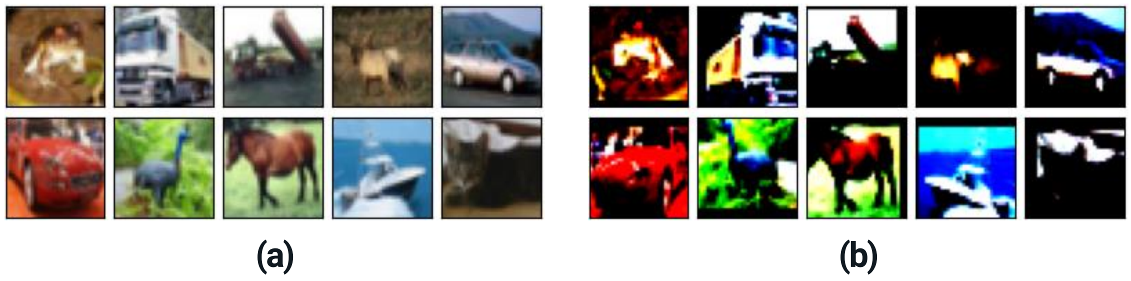

Figure 3.2.: Sample images from the CIFAR-10 dataset. **(a)** shows the default images, while **(b)** shows these images after the preprocessing and augmentation have been applied.

Model construction. Previous work in robust FL uses a ResNet-18 or ResNet-50 for the CIFAR-10 dataset. Although potentially impractical for some FL scenarios (due to its 142MB size), I selected a ResNet-50 to achieve improved performance.

Hyperparameter selection. The local training routine for each client is based on the setup used by Zagoruyko and Komodakis (2017): SGD with Nesterov momentum set to 0.9 and weight decay (L2 regularisation) set to 0.0005. Due to resource constraints, I used a batch size of 32 and a schedule that reduces the client learning rate from 0.1 to 0.0001. This model converges after 120 rounds if each of the 10 clients performs 2 full iterations over its dataset per round.

3.2.5. Baseline 3: Next word prediction

Dataset preparation. I used the `albert-base-v2` tokeniser from Hugging Face to convert the cleaned dataset into a sequence of integers [Lan et al. 2020]. I constructed the dataset from non-overlapping sequences of 62 tokens, where the model must predict the final token in the sequence.

Model construction. An initial embedding layer computes a length-50 vector representation for each of the 30,000 tokens. I then use a 2-layer LSTM, similar to Nguyen et al. (2023) with a hidden state of size 50. A linear decoder layer with its weights tied to that of the initial embedding layer maps the outputs of the LSTM back to tokens.

Hyperparameter selection. Similar to Nguyen et al. (2023), I created 10,000 clients, with 100 participating in each round. The model trains for 100 rounds with each client completing 5 epochs per round. The clients train with SGD and a constant 0.1 learning rate.

3.2.6. Optimising results

Maximising GPU usage. I manually performed a binary search to find the most efficient mini-batch size for the GPUs. Recent results have shown that even large mini-batch sizes can achieve the same error rate as pure SGD in certain scenarios [Goyal et al. 2018], so this selection can be made without much concern for accuracy loss. I also attempted to maximise available GPU memory by delaying data loading into clients until it is needed.

Model evaluation. Flower provides convenient support for federated evaluation, but there is no reason to pay the computational cost of this additional simulation, so I performed all evaluation centrally. I partitioned the training and validation datasets by the attributes that I want to test the fairness of for each dataset (using **Definition 1**, appendix A).

3.3. Creating the attacks

This section describes my implementation for core criterion **(a.2)**, section 2.4.1. To maximise performance, the structure of the attacks is strongly influenced by how the Flower framework allocates resources to clients. I begin this section by describing how attacks are represented based on this (extending the structure described in section 3.2.1).

3.3.1. General attack structure.

Efficient client resource allocation. Consider an experiment in which we have 10 clients, where clients 1-9 are benign, and client 0 performs the more expensive attacking routine. If 1/10th of the resources are naïvely allocated to each client, 9/10ths of the resources will lie idle while client 0 finishes. Therefore, to maximise GPU utilisation, we must provide client 0 more resources.[3]

A standard attack form. In the update prediction attack, we **(1)** compute a benign update as our prediction for what other clients might submit, and **(2)** produce an unfair model to replace the global model with. In the fairness attack setup described below, both of these tasks require almost the same amount of computation as would be required by a benign client. A neat solution is to split the training across two simulated clients, as shown in fig. 3.4. This is helpful because allocating different amounts of resources to each simulated client is difficult and can be inefficient. I refer to clients in the simulation as *simulated* clients, while the clients they represent are *real* clients, so each real, malicious client is represented by two simulated clients.

[3]We could instead run clients sequentially (i.e. let client 0 use all resources, then client 1, and so on), however for large client numbers, this becomes wasteful.

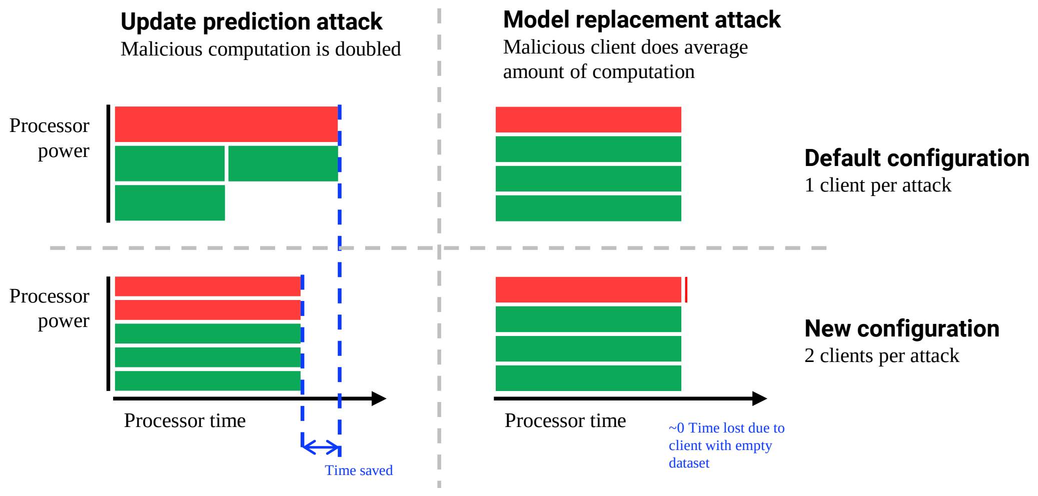

Figure 3.3.: Comparison between training time of the two configurations. Each rectangle represents a simulated client (malicious clients coloured red), and each row within a quadrant represents the work done by a single GPU. For the model replacement attack (right column), running a second client which does no work results in a negligible increase in training time for the new configuration. However, the new configuration is more efficient for the update prediction attack (left column), by reducing idle GPUs by splitting the work of the malicious client across two simulated clients.

The experiment for a fairness attack on 10 real clients with 1 being malicious simulates 11 clients, with simulated client 0 computing the target unfair model. Then, before aggregation, we compute the malicious parameters from simulated clients 0 and 1, set the parameters of simulated client 1 to this value, and delete simulated client 0, leaving a list of 10 real clients. This setup yields high GPU usage while remaining relatively simple and expandable. In the backdoor attack case, the second client is given an empty dataset. We therefore define the attack by a function to produce the unfair dataset, and some method to combine the simulated clients into a real malicious client before aggregation.

```python
@dataclass
class Attack:
    name: str

    # function to generate dataset `a` from a dataset, the config, and
    # the attack index. Important: useful to generate the target model.
    # Not intended to make predictions on client updates
    get_dataset_loader_a: Callable[[torch.utils.data.Dataset, Cfg, int],
                                   torch.utils.data.Dataset]

    # function to generate dataset `b` from data available to clean
    # client, the config, and the attack index. This is the clean data
    # we expect the attack to have *full* access to, e.g. for predicting
    # client updates
    get_dataset_loader_b: Callable[[torch.utils.data.Dataset, Cfg, int],
                                   torch.utils.data.Dataset]

    # update aggregator to generate attacks before aggregation is performed
    # second argument is attack index and fourth argument is `kwargs`
    aggregation_wrapper: Callable[[type[fl.server.strategy.Strategy],
                                   int, Cfg, object],
                                  type[fl.server.strategy.Strategy]]
```

In the example above, client 0 would train with dataset `a` and client 1 would train with dataset `b`. The `format_datasets` function interleaves these datasets for a attacks defined by the configuration.

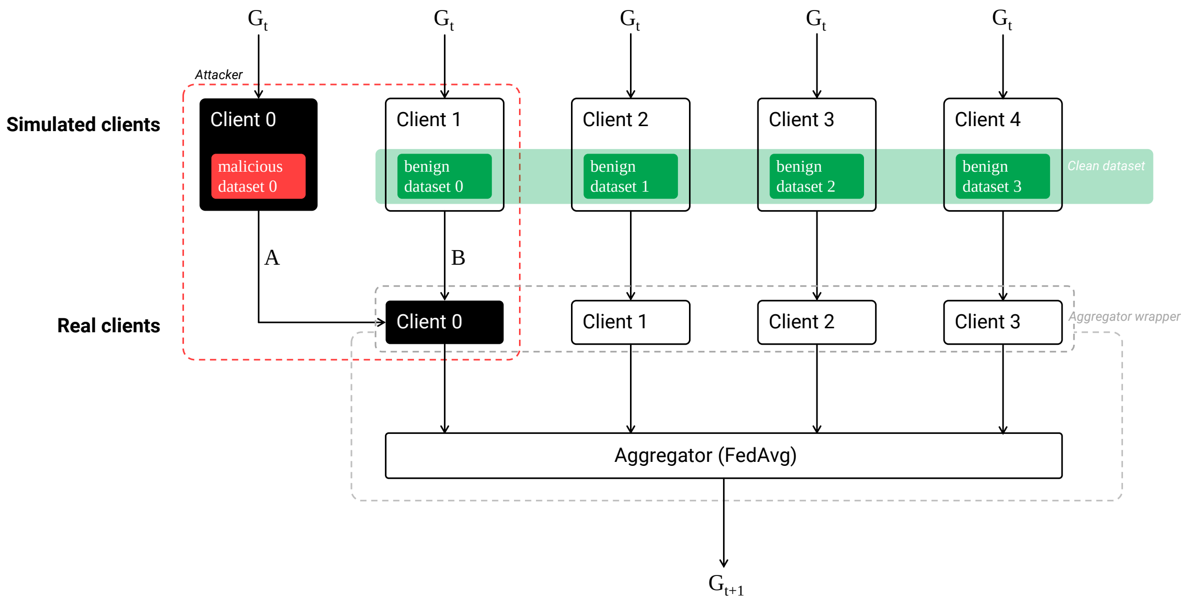

Figure 3.4.: Real client 0 is allocated 2 clients in the simulation to compute the target parameters and update prediction respectively. The aggregator wrapper is then responsible for combining these to perform the attack. This is done before the updates are passed to any other wrappers around the aggregator, such as for the defences.

One issue with this system is that randomly selecting clients to participate in each round leads to attackers rarely having all of their necessary clients available. Therefore, I also added a custom client manager that always selects the attacking clients according to a predefined schedule.

Similarly to the previous section, we can then append a list of attack configurations to the YAML file to select from the attacks package. For example:

```yaml
attacks:
  - name: model_replacement
    start_round: 10
    end_round: 11
    clients: 1
    target_dataset:
        name: backdoor
        size: 1/num_clients
  - name: model_replacement
    start_round: 40
    end_round: 41
    clients: 1
    target_dataset:
        name: backdoor
        size: 1/num_clients
```

3.3.2. The backdoor attack

To implement the backdoor attack, we can apply the following general version of eq. (2.2c) to the general attack structure described in section 3.3.1, extended to handle a attacking clients:

$$\mathbf{c}_i = \frac{1}{a}\left[\frac{n}{an_i}\mathbf{x} + \frac{an_i - n}{an_i}\mathbf{w}\right] \tag{3.1}$$

With this implementation, I achieve results comparable to Bagdasaryan, Veit, et al. (2019) (see section 4.1).

3.3.3. The fairness attack

The general rule for the update prediction attack is similar to that of a model replacement attack. However, our update predictions, $\mathbf{w}^{(i)}$ vary between clients. Therefore, we may want our overall update prediction, $\mathbf{w} = \sum_i p_i \mathbf{w}^{(i)}$ to be weighted by some p_i that we can define to improve prediction accuracy:[4]

$$\mathbf{c}_i = \frac{1}{a}\left[\frac{n}{an_i}\mathbf{x} + \frac{an_i - n}{n_i} p_i \mathbf{w}^{(i)}\right] \tag{3.2}$$

We can extend the general attack structure described in section 3.3.1 by setting dataset `a` to introduce unfairness into the global model instead of a backdoor and dataset to `b` be the same as all other benign datasets. For the CIFAR-10 dataset, I selected only data with classes 0 or 1 for `a`; for the Census dataset, I reduced the accuracy on female records by training on a dataset where all such records were set to earning less than $50K; and for the Reddit dataset, I reduced accuracy following the token '`I`' by creating a dataset in which it is always immediately followed by a '`.`' token.

Although this attack works as expected, one issue I have encountered is that a relatively high learning rate can increase variance in model predictions to cause the predicted gradients to be far enough away from the true values that the model weights become unstable and diverge. When combined with complex models (e.g. the Reddit language model), the experiment is therefore very sensitive to small changes in attack setup. Because the update prediction attack is quite recent, there are also very few examples of working setups.

To implement the fairness attack, I began by allowing the benign processes to communicate their updates to the attacker, before progressively decreasing the amount of information available to the attacker, until I found an attack setup that worked without knowledge of the benign updates.

3.4. Reproducing the defences

To implement core criterion **(a.3)** (section 2.4.1), I first define an abstract interface for a defence. Each defence is represented by a function that maps from a Flower aggregator to a new aggregator that applies the defence before calling the original aggregation functionality. This is very similar to `Attack.aggregation_wrapper`, above.

Next, I implement each defence as described in the corresponding paper [Blanchard et al. 2017; Yin et al. 2021; Sun et al. 2019], which is discussed in section 2.3.2.

Finally, I search the hyperparameter space to find a setup that can prevent the fairness attack (or determine that one does not exist). This is discussed in detail in chapter 4. For the Krum and trimmed mean defences, I was able to find a functioning setup by trial and error, using heuristics to help determine how 'close' a defence is to working. However, I found that the performance of the Weak DP defence was quite limited (see section 4.4).

3.5. Repository overview

Figure 3.5 shows the file structure for my codebase. Each each set of components described above (models, datasets, attacks, and defences) is defined within a separate package, for use by the main scripts (coloured green).

[4]This follows similar reasoning to why we take a weighted average in FedAvg.

```
.
├── configs/
│   ├── templates/                  Template configs for each component
│   │   └── ...
│   ├── default.yaml                Default values for each field
│   └── defence_fairness_testing.yaml  Config for defence_fairness.py
├── scripts/
│   ├── gen_template.sh             Generate a config file from a list of templates
│   └── slurm.sh                    Run with python environment (for slurm)
├── src/
│   ├── datasets/
│   │   ├── __init__.py
│   │   ├── adult.py                Download the 1994 Adult Census dataset
│   │   ├── cifar10.py              Download the CIFAR-10 dataset
│   │   ├── reddit.py               Download the Reddit comments dataset
│   │   ├── format_data.py          Split and organise datasets for the simulation
│   │   ├── typing.py               Define the abstract dataset generator interface
│   │   └── util.py
│   ├── models/
│   │   ├── __init__.py
│   │   ├── fully_connected.py      Fully connected nn for the Adult Census dataset
│   │   ├── resnet_50.py            ResNet-50 for the CIFAR-10 dataset
│   │   └── lstm.py                 2-layer LSTM for the Reddit dataset
│   ├── attacks/
│   │   ├── __init__.py
│   │   ├── backdoor_dataset.py     Modify a dataset to insert a backdoor
│   │   ├── unfair_dataset.py       Modify a dataset to introduce unfairness
│   │   ├── model_replacement.py    Model replacement attack
│   │   ├── update_prediction.py    Update prediction attack
│   │   └── typing.py               Define the abstract attack interface
│   ├── defences/
│   │   ├── __init__.py
│   │   ├── trim_mean.py            Trimmed mean defence
│   │   ├── krum.py                 Krum defence
│   │   ├── diff_priv.py            Weak differential privacy defence
│   │   ├── fair_detect.py          Unfair update detection defence
│   │   └── typing.py               Define the abstract defence interface
│   ├── client.py                   Client object and local training routine
│   ├── server.py                   Custom aggregation functions and client manager
│   ├── evaluation.py               Centralised evaluation function
│   ├── util.py                     Debugging functions
│   ├── defence_fairness.py         Run fairness experiment with synthetic datasets
│   ├── main.py                     Run main defence testing experiment
│   └── graph_gen.py                Generate graphs from main.py checkpoints
├── pyproject.toml
├── Makefile                        Experiment run commands
├── Dockerfile
├── .gitignore
├── LICENSE
└── README.md
```

Figure 3.5.: An overview of my codebase's file structure. Files coloured green are the main python scripts, while the file coloured blue mostly contains code from a previous project.

3.6. Theoretical contributions

The theoretical contributions of this dissertation are twofold: **(1)** I justify the fundamental assumption of the update prediction attack that it is possible to accurately predict the weights of a client update given only its training configuration, and **(2)** I prove that the existence of this (or any other) attack on fairness implies that we cannot guarantee both fairness and robustness in FL model training when using a defence based on anomaly detection. Appendix G provides a summary of prominent symbols and mathematical notation used throughout this section.

3.6.1. 'Private' gradient estimates can be accurately predicted

The update prediction attack assumes we can predict which updates clients may produce by training a local model on representative data. I show that for large batch or dataset sizes, we can expect the variance in model parameters to be small (0 in the limit) for strongly convex functions, assuming training is completed for a sufficient number of epochs between each aggregation round. Then, I discuss how this affects the non-convex case, to conclude that training a local client on data sampled from the clean distribution is an unbiased estimator of the aggregated model with low variance, and therefore has a low expected error.

We begin by considering the following general optimisation problem for client i:

$$\min_{\mathbf{c}_i \in \mathbb{R}^d} f(\mathbf{c}_i) = \min_{\mathbf{c}_i \in \mathbb{R}^d} \frac{1}{n_i} \sum_{j=0}^{n_i - 1} f_j(\mathbf{c}_i) \tag{3.3}$$

where each $f_i \in \mathbb{R}^d \to \mathbb{R}$ is a continuously differentiable function.[5] We want to use mini-batch gradient descent to solve this problem:

$$\begin{align} \mathbf{c}_i^{(k)} &= \mathbf{c}_i^{(k-1)} - \frac{\alpha_k}{b} \sum_{j=0}^{b-1} \nabla f_{s_{k,j}}(\mathbf{c}_i^{(k-1)}) \tag{3.4a} \\ &= \mathbf{c}_i^{(k-1)} - \alpha_k [\nabla f(\mathbf{c}_i^{(k-1)}) + \xi_k] \tag{3.4b} \end{align}$$

where α_k is the learning rate, each $s_{k,j}$ is a uniformly random sample from $\{0, ..., n_i - 1\}$, and ξ_k represents the noise introduced by sampling from the training distribution on round k. This problem description and the following assumptions follow that of Tiejun Li, Xiao, and G. Yang (2023), whose proof of SGD convergence provides a basis for the following proofs. For simplicity, I select a learning rate of $\alpha_k = \alpha_1 k^{-1/2}$, which satisfies the assumptions required by Tiejun Li, Xiao, and G. Yang (2023).

We make the following assumptions in the below proofs.

(A1) Mean and covariance of ξ_k. $\forall \varepsilon > 0$. $\exists$ a symmetric, positive definite matrix, Σ, such that

$$\mathbb{E}[\xi_k | \mathcal{F}_{k-1}] = 0, \qquad \lim_{n \to \infty} P(||\mathbb{E}[\xi_k \xi_k^T | \mathcal{F}_{k-1}] - \Sigma|| \geq \varepsilon) = 0 \tag{3.5}$$

where $\mathcal{F}_k = \sigma(x_0, \xi_1, \xi_2, \ldots, \xi_k)$ is the σ-algebra generated from the random initialisation and noise terms up to round k.

(A2) L-smoothness of f. $\exists L$ such that

$$\forall x, y \in \mathbb{R}^d.\ ||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \tag{3.6}$$

(A3) μ-strong convexity of f. $\exists \mu$ such that

$$\forall x, y \in \mathbb{R}^d.\ f(x) \geq f(y) + \nabla f(y)^T (x - y) + \frac{\mu}{2}||x - y||^2 \tag{3.7}$$

[5]This is a rewritten form of the usual machine learning setup: let $f_j(\mathbf{c}_i) = \mathcal{L}(M(\mathbf{c}_i, x_j), y_j)$, where $\mathcal{L}$ is the loss function, M is the model, $\mathbf{c}_i$ are the weights, and (x_i, y_i) is the ith data point in the dataset.

(A4) Further smoothness condition for f. $\exists p_0, r_0, K_d > 0$ such that, for any $||x - x^*|| < r_0$,

$$||\nabla f(x) - \nabla^2 f(x^*)(x - x^*)|| \leq K_d ||x - x^*||^{1-p_0} \tag{3.8}$$

(A5) Dataset size heterogeneity. If client i has a dataset of size n_i, modelled as a random variable, and $n = \sum_{c=0}^{m-1} n_i$ for m clients, then,

$$\exists h \geq 1 \in \mathbb{R}.\ \forall i \in [0, 1, ..., m-1].\ mn_i \leq nh \tag{3.9}$$

With this definition, if $h = 1$, all client datasets must have the same amount of data, while as $h \to \infty$, the client data distribution constraints disappear.[6]

Lemma 1 (variance for a single client). Under assumptions (A1)-(A4), if $\frac{1}{\alpha_1} < 2\mu$, there exists some matrix, W^*, such that

$$k^{1/4}(\mathbf{c}_i^{(k)} - \mathbf{c}_i^*) \Rightarrow^k N(0, \alpha_1 W^*) \tag{3.10}$$

where $\Rightarrow^k$ denotes convergence in probability, $\mathbf{c}^*$ is the unique minimum of f, $k \in \mathbb{N}$ is large, and $W^*_{k,i,j} \in O\left(\frac{1}{b^2}\right)$.

Proof. This extends the result from Tiejun Li, Xiao, and G. Yang (2023). Under the above assumptions, the authors show that for $b = 1$ we have eq. (3.10) for some matrix W^*, where $AW^* + W^*A^T - d_0 W^* = \Sigma$ and A is independent of all ξ_i.

Now consider the variance of ξ_k as b increases. We can assume that the summed gradients are independent and have finite first 2 moments. Thus, for large b, by the classical CLT, the gradient estimate, $\nabla \hat{f}(\mathbf{c}_i^{(k-1)})$, is unbiased and normally distributed:

$$\nabla \hat{f}(\mathbf{c}_i^{(k-1)}) = \frac{1}{b}\sum_{j=0}^{b-1} \nabla f_{s_{k,j}}(\mathbf{c}_i^{(k-1)}) \sim N\left(\mathbb{E}_{f_j}[\nabla f_j(\mathbf{c}_i^{(k-1)})], \frac{\mathbb{V}_{f_j}[f_j(\mathbf{c}_i^{(k-1)})]}{b}\right) \tag{3.11}$$

This yields a noise term, $\xi_k \sim N\left(0, \mathbb{V}[f_i(\mathbf{c}_i^{(k-1)})]/b\right)$, with variance inversely proportional to batch size.

The maximum element in the covariance matrix for $\text{vec}(\xi_k \xi_k^T)$ (where vec is a function that flattens a matrix into a vector) must be the variance of $(\xi_k)_i^2$ for some i. Since $(\xi_k)_i$ is normally distributed with variance $\frac{c}{b}$ for some constant c (as in eq. (3.11)), we know that each element of this covariance matrix must be bounded by $\mathbb{V}_{\xi_k}[(\xi_k)_i^2] = \frac{2c^2}{b^2}$.

We have established that $AW^* + W^*A^T - d_0 W^* = \Sigma$ and that the elements of the covariance matrix for $\xi_k \xi_k^T$ (and therefore also those of Σ) are in $O\left(\frac{1}{b^2}\right)$, so the elements of W^* must also be in $O\left(\frac{1}{b^2}\right)$. $\square$

Now consider the FedAvg aggregation function to compute the global model, G, from the model $\mathbf{c}_i^{(u)}$ produced by each client i after u batches using the above SGD setup for the current training round:

$$G = \frac{1}{n}\sum_{i=0}^{m-1} n_i \mathbf{c}_i^u \tag{3.12}$$

[6] This constraint is only necessary to show that model variance decreases with the number of clients. By tracing the effects of h through the proof of **Theorem 1**, we can also see that variance increases with h.

Theorem 1 (Variance of FedAvg)**.** Under assumptions (A1)-(A5), if $\frac{1}{\alpha_1} < 2\mu$ for each client, the global model, G, must be normally distributed with covariance matrix M_g such that $M_{g,p,q} \in O\left(1/\sqrt[4]{enm^3b^7}\right)$ for a large number of epochs, e, and batch size, b.

Proof. From **Lemma 1**, we know that each $\mathbf{c}_i^{(e-1)}$ is an independent, normally distributed random variable with covariance matrix M_i, where $M_{i,p,q} \in O\left(1/\sqrt[4]{ub^8}\right) = O\left(1/\sqrt[4]{en_ib^7}\right)$, for large e and b. By applying the FedAvg procedure, we get

$$G \sim N\left(\sum_{i=0}^{m-1} \frac{n_i}{n}\mathbf{c}_i^*, \sum_{i=0}^{m-1} \frac{n_i^2}{n^2} M_i\right) \tag{3.13}$$

Since, by (A5), $\max_i \frac{n_i}{n} \in O\left(\frac{1}{m}\right)$ for all clients, i, then the covariance matrix $M_g = \sum_{i=0}^{m-1} \frac{n_i^2}{n^2} M_i$ must have $M_{g,p,q} \in O\left(1/\sqrt[4]{en_im^4b^7}\right) = O\left(1/\sqrt[4]{enm^3b^7}\right)$. □

Therefore, by Chebyshev's inequality, **the probability of our update prediction error being greater than γ is bounded by $O\left(1/\sqrt[4]{enm^3b^7\gamma^8}\right)$.**

The above proof can similarly be applied to SGD with momentum, using the results from Tiejun Li, Xiao, and G. Yang (2023). It follows that if the attacker simulates the entire FL training process[7] using their own dataset (which is i.i.d. to the union of the benign clients' datasets), the resulting model prediction should be close to the true value so long as the combined dataset size, number of epochs per aggregation round, number of clients and/or local batch sizes are *large*.

We can intuitively see that any claim that variance in all non-convex models tends to 0 under similar requirements must be false because small differences in early batches can push the model to converge to different minima for different, i.i.d. datasets. However, it may be reasonable to assume that for a sufficiently smooth loss function and large enough batch size, this is less likely to happen, and, since the attacker knows the model's initial parameters, the dynamics of training by SGD on a non-convex model are locally similar to the strongly convex case above.

There is also a question of whether practical tasks are sufficiently large for the attack to function well. This is dataset-dependent, however, I found that the fairness attack continues to function even with a batch size set to 1. The above analysis also implies that introducing heterogeneity to the client data distribution does not increase the variance in the aggregated update if clients participate in every round.

3.6.2. Attacks cannot be fairly detected

We can represent a deterministic anomaly detection defence as a predicate, $F : M^m \to \{0,1\}^m$, that returns a vector, $\mathbf{v}$, when presented with the vector of client models, $\mathbf{c}$, with $\mathbf{v}_i = 1$, if the model from client i, should be included in aggregation, and $\mathbf{v}_i = 0$ if it should be discarded. We can characterise such a predicate that is likely to avoid introducing a significant amount of unfairness as having

$$\mathbb{P}(F([C_0, \ldots, C_{m-1}])_{m-1} = 1) \approx 1 \tag{3.14}$$

[7] In my testing, I instead made the update predictions in a centralised manner for improved efficiency. If we have i.i.d. clients with equal dataset sizes, we can use the Lyapunov CLT to show that this is an unbiased estimate of the aggregated model. In the heterogeneous case this is not necessarily true (since FedAvg *is* an unfair aggregation method), although appendix E shows that the update prediction attack can function with high amounts of heterogeneity.

where $\forall i \in [m-2].C_i \sim C$, $[x]$ is the set of natural numbers less than or equal to x, C is the distribution of updates produced by clients which *only train on data within the legitimate distribution*, and $C_{m-1} \sim \text{Uniform}(\text{Image}(C))$[8]. Similarly, a predicate that provides strong robustness would have

$$\max_{\mathbf{c}_{m-1} \in A} [\mathbb{P}(F([C_0, \ldots, C_{m-2}, \mathbf{c}_{m-1}])_{m-1} = 1)] \approx 0 \tag{3.15}$$

where A is the set of updates which would successfully perform an attack on the aggregated model if included in aggregation.[9] If we make the simplifying assumption that the sets A, $\text{Image}(C)$, and the expected value of $\{\mathbf{c}_{m-1} | F([C_0, \ldots, C_{m-2}, \mathbf{c}_{m-1}])_{m-1} = 1\}$ are all closed d-balls, we can arrive at a more practical definition, which preserves the same ordering over both equations:

Definition 3 (r_α-robust and r_β-fair)
We say that an aggregator, F, is r_α**-robust** if and only if r_α is the radius of the largest d-ball that for any set of parameters within the ball, $\mathbf{c}_{m-1}$, there exists some $\mathbf{c}_0, \ldots, \mathbf{c}_{m-2}$ such that $F(\mathbf{c})_{m-1} = 1$ and $\mathbf{c}_{m-1} \in A$.

We say that an aggregator, F, is r_β**-fair** if and only if r_β is the expected radius of the largest d-ball such that all sets of parameters, $\mathbf{c}_{m-1}$, within the ball satisfy $F([C_0, \ldots, C_{m-2}, \mathbf{c}_{m-1}])_{m-1} = 0$ and $\mathbf{c}_{m-1} \in \text{Image}(C)$, where $C_i \sim C$ for $i \in [m-2]$

For example, in the Trimmed Mean case, if we assume the effect of attackers is negligible, the expected lower bound, a, for the magnitude of weight w_i satisfies $\int_{l_i}^{a} (\frac{1}{m} \sum_{j=1}^{m-1} (\mathbb{P}(C_{j,i} = x))\, \mathrm{d}x) = \beta$, while its expected upper bound, b, satisfies $\int_{b}^{u_i} (\frac{1}{m} \sum_{j=1}^{m-1} (\mathbb{P}(C_{j,i} = x))\, \mathrm{d}x) = \beta$, where $\forall j.\ \text{Image}(C_{j,i}) \subseteq [l_i, u_i]$. Then, our Trimmed Mean algorithm is r_β-fair for weight w_i,[10] where $r_\beta = \max(u_i - b, a - l_i)/2$. Thus, we can see r_β decreases with β.

Theorem 2 (Privacy and fairness)**.** In differential private FL training, there does not exist any F that is both 0-robust and 0-fair[11]. Therefore, on any FL training round with submitted parameters $\mathbf{c}$, there is a non-zero probability that either there exists a benign $\mathbf{c}_i$ that has $F(\mathbf{c})_i = 0$ or there exists a malicious $\mathbf{c}_j$ that has $F(\mathbf{c})_j = 1$.

Proof. Proceed by counterexample. Consider a model $M(w, x) = wx$, where $w, x \in \mathbb{R}$ are the model's parameter and input respectively. Now consider $S = (X, X)$, where $X \sim U(0, 1)$. Let $x_1 = [(0.5, 0.5), (0.5, 0.5)]$ be some batch sampled from the distribution of S, and $x_0 = [(0.5, 2), (0.5, -3)]$ be a batch of data that lies partially outside S's image. When we apply gradient descent with MSE loss, in both cases, we get

$$w_1 = w_0 - \alpha \left. \frac{\partial (MSE \circ M)}{\partial w} \right|_{x_0} = w_0 - \frac{\alpha w_0}{2} - \frac{\alpha}{2} \tag{3.16}$$

[8]While the expectation is over the distribution of models, the probability uses a uniform distribution over all models, which yields the notion of fairness in this case. We can interpret this formula as the expected proportion of legitimate models that are accepted by F.

[9]We can interpret this formula as the probability of the most concealed model, $\mathbf{c}_{m-1}$, being accepted by F.

[10]Some minor modification to the definition is required because trimmed mean applies anomaly detection to each parameter individually.

[11]The claim is not that the defences introduce unfairness in *every* scenario, but rather that there exists some where they do. The empirical testing in the next sections provides some evidence that this *possibility* of introducing unfairness translates to a practical issue.

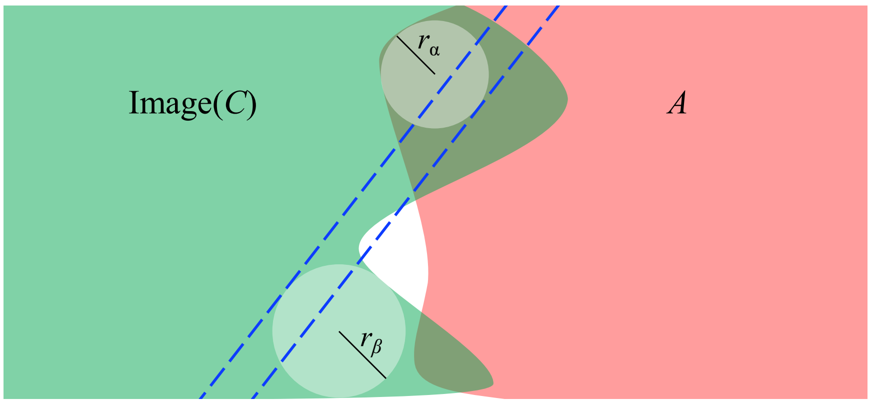

Figure 3.6.: Illustration of **Definition 3**. The green area represents all benign models, while the malicious models are in red. The line on the left represents the boundary of models, $\mathbf{c}_{m-1}$, that are *expected* to be accepted by the anomaly detection, while the other represents the boundary of whether there is *any* set of models submitted by other clients that would result in the model being accepted. We can trivially see that these lines can never cross, so, since there must be some overlap between the red and green areas, there is no boundary where $r_\alpha = 0 = r_\beta$.

Now consider an FL setup, where x_i is the training set of client i. In this case, the weight values submitted by clients 0 and 1 are identical, and thus $F([\dots, M_0]) = F([\dots, M_1])$, however, M_0 is trained using out-of-distribution data, which we define to be in set A, while M_1 is trained on in-distribution data for the distribution of S. No function, F, can decide whether a set of parameters was generated exclusively from data in S because the function from training sets to model weights is not injective for SGD and thus has no left inverse. $\square$

Figure 3.6 provides an intuitive explanation of the proof. However, showing the existence of an overlap between models from different data distributions does not necessarily imply that the models produced by malicious data can also be produced by data from the tail of the benign distribution in practice. Specifically, I have not proven that this issue extends to an arbitrary set A. However, Shumailov et al. (2021) show that backdoors can be produced from entirely clean datasets if the data is randomly (or maliciously) shuffled in a specific way.

4 Evaluation

This chapter presents empirical evidence that the results in section 3.6.2 lead to reduced fairness in the presence of common defence methods in practice.

4.1. Attacks

In this section, I present the results of all three tasks without any defence method. I then individually investigate the performance of the two major components of the fairness attack: **(1)** generation of the unfair target parameters and **(2)** update prediction.

Experimental setup. In each experiment, there is a single malicious client on every round.[1] The Census, CIFAR-10, and Reddit datasets have 10, 10, and 10,000 clients respectively. The Reddit dataset has a client participation rate of 0.01. Each experiment is run three times on 2 Nvidia RTX 2080 GPUs, which provides a good balance between fast training and maximising the usage of available GPUs with parallel experiments. All experiments took a total of around 500 GPU hours to complete.[2]

Baseline and backdoor results. The baseline results (where there is no attack) reported in table 4.1 line up with those of similar previous works [Bagdasaryan, Veit, et al. 2019; Bhagoji et al. 2019; H. Wang et al. 2020; Nguyen et al. 2023; McMahan et al. 2023]. This provides a realistic foundation to test the attacks and defences on.

Table 4.1.: Attack results. 'Overall' indicates the accuracy on a clean test set. 'Backdoor ASR' is the accuracy of the backdoor in the presence of the trigger. 'Fair (inc.)' is the accuracy (%) on only data the fairness attack aims to *increase* the accuracy on, while 'Fair (dec.)' only uses data it aims to *decrease*. 'Fair (chg.)' is $r \times |r| + i \times |i|$, where r and i are the differences in 'Fair (dec.)' and 'Fair (inc.)', respectively, compared to the baseline row. This value is proportional to the increase in the unfairness value due to the attack in eq. (A.1), under certain simplifying assumptions. In a few cases, the signs of i and r are flipped to satisfy these assumptions.

Dataset	Attack	Overall	Backdoor ASR	Fair (inc.)	Fair (dec.)	Fair (chg.)
Census	Baseline	**84.81**	28.92	97.41	51.36	
	Backdoor	84.52	**99.99**	96.98	53.56	-0.0005
	Fairness	81.57	23.25	100.00	0.51	**0.2592**
CIFAR-10	Baseline	**92.70**	10.23	96.05	91.79	
	Backdoor	90.92	**98.89**	95.35	90.38	0.0000
	Fairness	17.90	63.63	89.50	0.00	**0.8383**
Reddit	Baseline	**18.08**	0.00	0.00	18.94	
	Backdoor	0.00	**100.00**	0.00	0.00	-0.0358
	Fairness	4.52	0.00	100.00	0.00	**0.5632**

To verify the defence implementations, I test a model replacement backdoor attack, against which most of the defences should perform well. The backdoor attack is successful if it results in a high Attack Success Rate (ASR) while leaving the overall accuracy close to the baseline value. For the Census and CIFAR-10 datasets without any defence, this is the case.

[1] Attacking on every round is not necessary, but is practically very similar to a single-round attack.

[2] Most of these experiments were run on the CaMLSys cluster over the Christmas weekend to avoid disrupting the work of others with excessive GPU time.

For the Reddit dataset, there is a drop in overall accuracy when the backdoor is inserted. This drop occurs in the target model, $\mathbf{x}$, which implies that more careful training of this target model may be necessary when the client locally computes the target backdoor weights. However, since retention in main task accuracy is not critical for my testing, I didn't extend my original, simple training setup to change this.

Fairness attack results. The fairness attack is successful if there is a significant increase in the 'Fair (chg.)' value after it is applied. Table 4.1 shows that this attack has been successful in all three tasks.

Fairness attack: Testing the target parameters. The accuracy imbalance introduced by the attack comes mostly from damaging the 'Fair (dec.)' performance in the target parameters, $\mathbf{x}$, rather than increasing 'Fair (inc.)'. To provide intuition for why this is the case, I used t-SNE [Maaten and Hinton 2008], PCA, and the activations of a trained InceptionV3 model to project samples from the CIFAR-10 dataset into a 2-dimensional embedding space, colouring the points according to their *predicted* class (fig. 4.1).[3,4]

Figure 4.1 shows that, as unfairness increases, although the accuracy between classes 0 and 1 improves, this is not because the boundary set by the model becomes more refined. Instead, in the 100% unfair plot, accuracy improvements are yielded for these classes by expanding the decision boundary into the previously green regions, thereby correctly classifying the 'borderline' points of the target classes. In the fair plot, some of these borderline points were misclassified as one of the green classes because this model incurred a loss penalty for misclassifying points of classes 2-9.

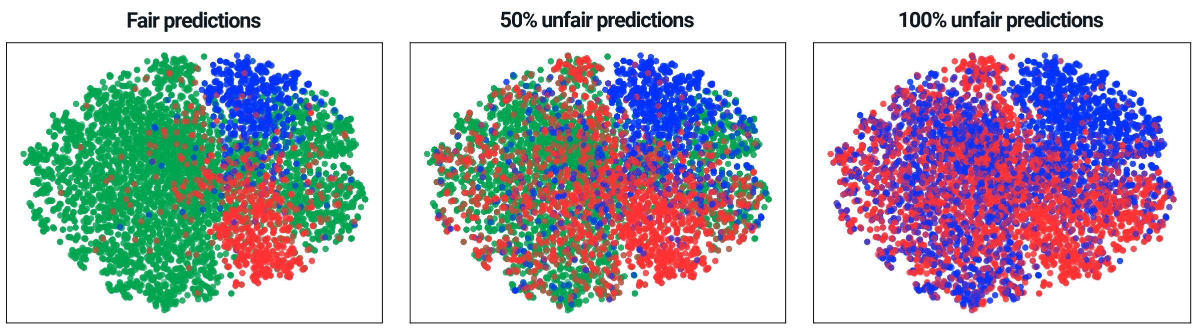

Figure 4.1.: 2D representation of the CIFAR-10 data, where blue-coloured points represent data with class 0, red-coloured points represent class 1, and the other classes are all coloured green.

A limitation of using the model depicted on the right of fig. 4.1 for the target parameters is that the attack can be detected based on the model's low test accuracy. Although the server may not know which client caused the attack, detection that any client has submitted a malicious set of parameters would likely lead to the model not being deployed. Therefore an attacker may want to produce a more subtly unfair model, such as the one model depicted in the middle of fig. 4.1.

[3]The InceptionV3 model only provides embeddings, while the colours are decided by a ResNet-50 model. If the embeddings and predictions were from the same model, the predictions could have had an unrealistic, tight clustering.

[4]The code I used for this was heavily inspired by the implementation at `https://github.com/alexisbcook/keras_transfer_cifar10`

Fairness attack: Testing update prediction accuracy. Given a target model, the second part of the attack relies on accurate update prediction. I directly characterise the update variance to gain additional insight into the prediction accuracy. Figure 4.2 shows **(a)** the average benign model update magnitude and **(b)** angle to the aggregated update. Since the fairness attack prevents the benign clients from converging, the blue update norm distribution (left) shows a slightly higher magnitude and variance. To achieve accurate update predictions, we require graph **(b)** to be tightly distributed with low mean, which remains the case for all three distributions.

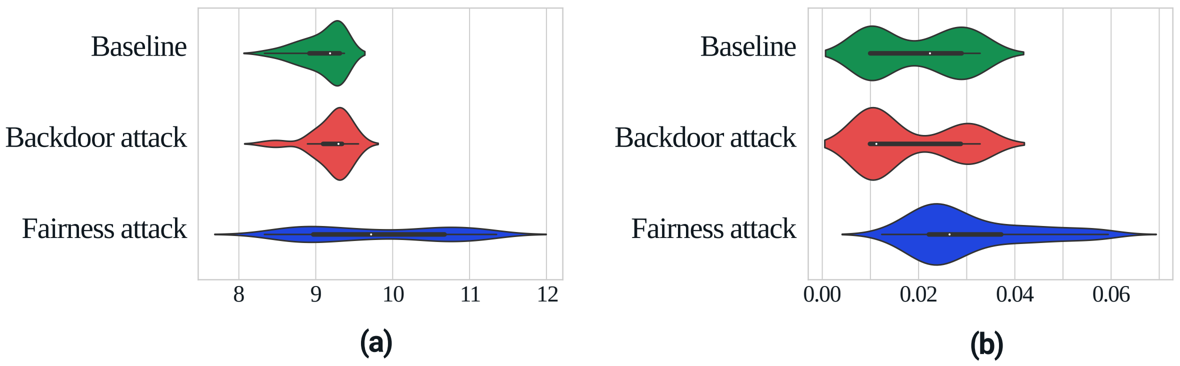

Figure 4.2.: Distributions of clean client updates under each attack. **(a)** shows the update norm and **(b)** shows the angle in radians between the update and the aggregated update. The fairness attack produces a wider update distribution because it prevents convergence.

The reason these distributions are bimodal is unclear. It would make sense if this were due to the scheduler using two different learning rates. However, the distribution shown in fig. 4.3 indicates that, while there is some change in magnitude at round 25 (when the learning rate changes), the average benign update does not change significantly.

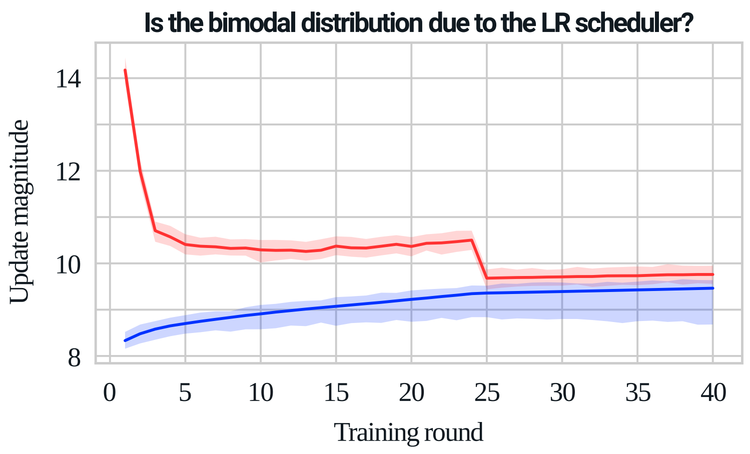

Figure 4.3.: Update magnitude distribution at each round. The red line represents malicious clients while the blue represents benign. This setup used a slightly different batch size compared to fig. 4.2 due to GPU availability constraints.

4.1.0.1. Attack performance on different aggregators *(extension)*

Table 4.2 Shows the performance of the fairness attack on three momentum-based aggregators. I kept all other hyperparameters the same for these experiments. The results are similar to those in table 4.1, which implies the fairness attack functions against these aggregators without the need for modification. The baseline performance of these models is slightly lower than with FedAvg because hyperparameters such as the learning rate schedule have been tuned for the FedAvg setup.

Table 4.2.: Results for different aggregators on the CIFAR-10 dataset

Aggregator	Attack	Overall	Fair (inc.)	Fair (dec.)	Fair (chg.)
FedAdaGrad	Baseline	**84.58**	96.77	53.56	
	Fairness	75.79	93.71	11.19	0.1786
FedYogi	Baseline	**78.72**	99.98	10.17	
	Fairness	80.78	99.61	1.69	0.0072
FedAdam	Baseline	**84.50**	97.52	49.83	
	Fairness	80.14	98.57	6.61	0.1869

Conclusions.

- Both attacks are successful in all three tasks in the absence of any defence methods.
- The update prediction attack can be effectively applied to momentum-based aggregation functions such as FedAdaGrad, FedYogi, or FedAdam.

4.2. Defence: Trimmed Mean

In this section, I test whether the trimmed mean defence can effectively prevent fairness attacks.

Trimmed mean relies on the assumption that malicious clients produce updates with weight values that are out of distribution. In appendix D, I show that we can expect this assumption to hold for the new attack, which is experimentally verified in fig. 4.4, where we can see that the malicious client in the update prediction attack case, **(c)**, generally has a higher update norm than the other clients.

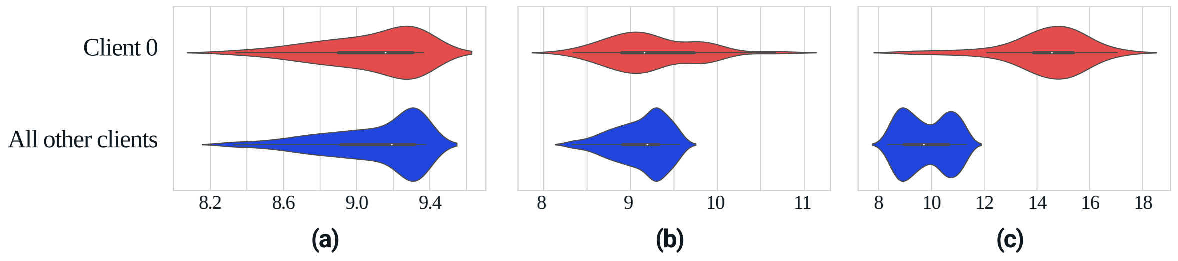

Figure 4.4.: Update magnitude distribution of the malicious client (0) compared to benign clients, for the baseline **(a)**, backdoor attack **(b)**, and fairness attack **(c)**. The fairness attack (**(c)**, red) produces updates that are, on average, larger than the benign updates

The results produced by this defence in table 4.3 generally follow this expectation,[5] with 5/6 attacks being successfully prevented by the defence with little loss of main task accuracy. For example, we can see that the recall score for predicting >$50K for only female data points in the Census dataset remains around 50% under the fairness attack, where it previously dropped close to 0 without any defence. This indicates that my implementation is correct. However, while the defence weakened the fairness attack for CIFAR-10, it is surprising that this attack has been at all successful, given the strong evidence above for the expected effectiveness of this defence, and that no concealment term was used in any of the attacks.

[5]The paper by Yin et al. (2021) instead tested on a different type of attack, so direct comparisons with these results are difficult to make.

Table 4.3.: Results for the trimmed mean defence with $\beta = 0.2$

Dataset	Attack	Overall	Backdoor ASR	Fair (inc.)	Fair (dec.)	Fair (chg.)
Census	Baseline	**84.83**	26.60	97.60	51.02	
	Backdoor	84.84	**37.59**	97.62	51.17	0.0000
	Fairness	84.76	27.26	96.87	48.47	**0.0006**
CIFAR-10	Baseline	**91.98**	10.05	95.35	91.14	
	Backdoor	92.50	**10.18**	95.60	91.73	0.0000
	Fairness	63.11	10.56	71.50	61.01	**0.1477**
Reddit	Baseline	**18.08**	0.00	0.00	18.94	
	Backdoor	18.06	**0.00**	0.00	18.91	0.0000
	Fairness	17.90	0.00	0.00	18.75	**0.0000**

Synthetic dataset tests: Trimmed mean can introduce unfairness. **Theorem 2** gives us two options for any anomaly detection defence: the defence **(1)** has a non-zero false positive rate, or **(2)** the true positive rate is less than 1. Since we have selected $\beta = 0.2$, and therefore remove 2 clients on each round, we know that on every round we are removing at least one benign client's update for each weight. This client is also likely to have data from the tail of the distribution. To show this, I construct a new synthetic dataset for which the defence demonstrates this property more clearly.

The clients are split into two groups: there are 5 clients in group A, which all contain data of the form $(x, y) = ((0, X), X)$, where X is a Bernoulli random variable with $p = 0.5$, and 1 client in group B, which contains data of the form $(x, y) = ((X, 0), X)$. The combined dataset represents the OR logical function. However, a model trained on group A will perform poorly on group B, resulting in larger updates from group B to correct for this, thus forcing the defence to ignore these weight updates and introduce unfairness against group B. When each client was selected for every round, a simple, 2-layer, fully connected network performs at 100% accuracy for all clients, without the defence. However, when the defence is introduced, the accuracy for group A remains at 100%, but the accuracy for group B drops to 50%, indicating that the defence has introduced unfairness.

Conclusions.

- The Trimmed Mean defence can protect some FL models from both attacks.
- There exists at least one dataset for which the trimmed mean defence introduces unfairness into the training process.
- This defence therefore cannot be used to guarantee fairness in the presence of fairness attacks.

4.3. Defence: Krum

In this section, I test whether the Krum defence can effectively prevent fairness attacks.

The Krum defence assumes that malicious updates lie far away from the other updates. To get an idea of how reasonable this assumption is, I used MDS [Kruskal 1964] to project each update vector from the Census dataset onto a 2D plane while attempting to preserve the distance matrix, in fig. 4.5. The difference between the benign (blue) and malicious (red) updates is clear, so we can expect Krum to perform well on this task.

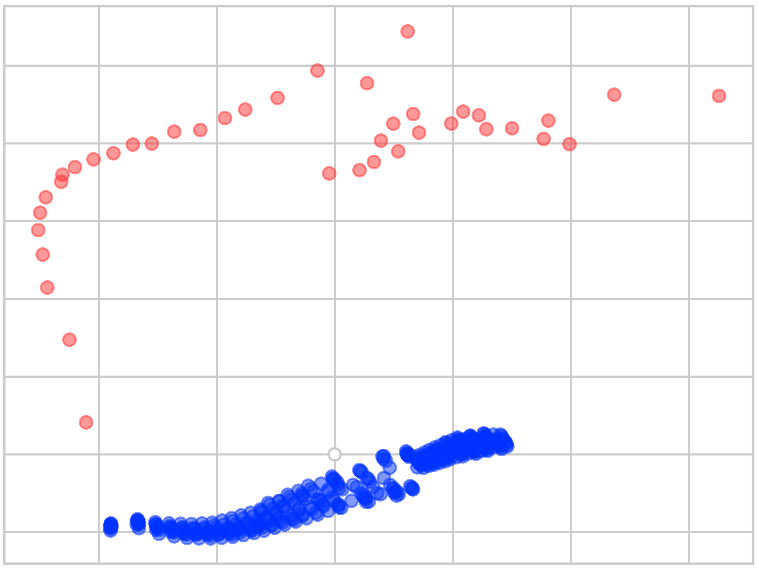

Figure 4.5.: 2D projection of multiple rounds of model updates from the Census task. The red points represent malicious updates, while the blue points represent benign updates. There is a clear separation between the red, malicious points and blue, benign points. The relative distances from the origin are *not* representative in this figure.

Figure 4.6 shows which clients were selected at each training round. This shows that the attacker (client 0) is never selected (the top row contains no blue). Since five clients are removed on each round, there may be a drop in the model's main task performance. I removed five clients because we are more interested in the best-case robustness performance, rather than the best-case accuracy. Additionally, in practice, we would not know the number of attackers.

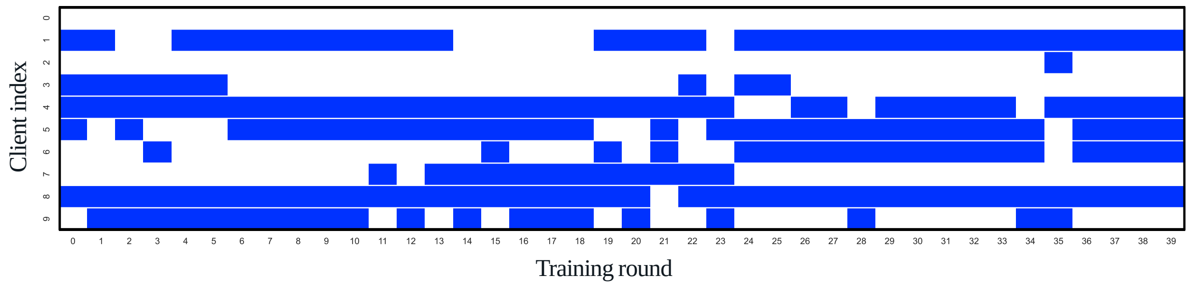

Figure 4.6.: Clients selected at each training round (blue if selected). Client 0 is running the fairness attack and is never selected (the top row is entirely white).

The results in table 4.4 are generally in line with these expectations and with previous work [Blanchard et al. 2017]. However, as with the trimmed mean case, the CIFAR-10 fairness attack proved more difficult to prevent.

How can Krum control the tradeoff between fairness and robustness? We aim to train a model where Krum has high precision (robustness) and high recall (fairness) for selecting benign clients. In the experiments here, by setting the Krum parameter $m = 1$[6], we can obtain a perfect ROC curve. However, we know that Krum can fail to prevent some more complex attacks, in which case the red line in fig. 4.7 (representing the 'friendliness' ranking of the malicious client) would see an improved ranking. In these cases, the m parameter provided by Krum can control the precision-recall tradeoff. Future work could investigate how such a value could be dynamically chosen in an online fashion.

[6]This value is defined in section 3.4 which differs from appendix G

Table 4.4.: Results for the Krum defence with $f = 1$ and $m = 5$

Dataset	Attack	Overall	Backdoor ASR	Fair (inc.)	Fair (dec.)	Fair (chg.)
	Baseline	**84.82**	30.21	97.66	49.66	
Census	Backdoor	84.73	**29.38**	97.47	51.02	-0.0002
	Fairness	84.64	29.05	97.47	51.19	**-0.0002**
	Baseline	**91.59**	10.26	95.40	90.64	
CIFAR-10	Backdoor	91.23	**10.15**	94.75	90.35	0.0000
	Fairness	63.78	11.01	72.60	61.58	**0.0325**
	Baseline	**17.82**	0.00	0.00	18.66	
Reddit	Backdoor	17.80	**0.00**	0.00	18.64	0.0000
	Fairness	17.97	0.00	0.00	18.82	**0.0000**

Appendix F shows further experiments on the effects on fairness of introducing Krum at different times during training.

Synthetic dataset tests: Krum can introduce unfairness. Using the same setup as the trimmed mean defence, with five group A clients and one group B client, Krum falls into the same trap of removing the group B clients when the model is controlled by group A. Thus, we get 50% accuracy for the group B client when Krum is used. A similar analysis for Krum has been done by H. Wang et al. (2020), using the more realistic CIFAR-10 dataset, but with less clear fairness effects on the model. Furthermore, fig. 4.7 shows that, even when the client datasets are large and sampled from i.i.d. distributions, the Krum function is still prone to selecting some clients more frequently than others (compare the average ranking of the blue line against the green line).

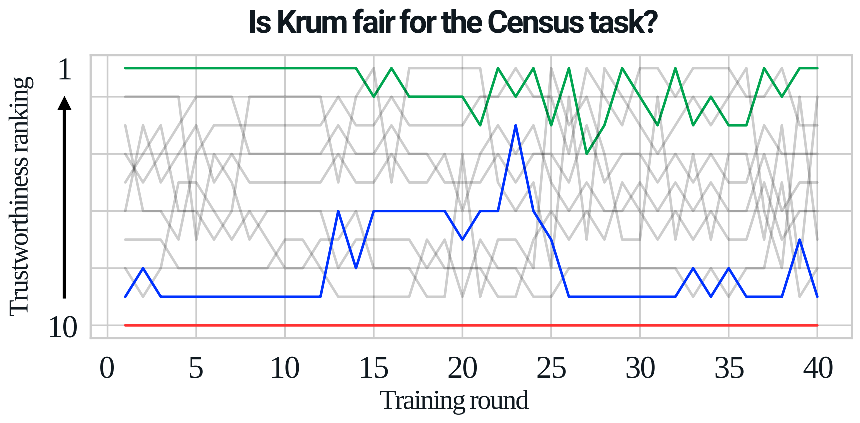

Figure 4.7.: Score rankings assigned by Krum to each client's model at each round with the CIFAR-10 dataset. The red line represents the malicious client, which is consistently ranked the lowest. I have highlighted two other models that show a disparity in their average ranking, despite being sampled from the same distribution.

Conclusions.

- The Krum defence can protect some FL models from backdoor and fairness attacks.
- There exists at least one dataset for which the Krum defence introduces unfairness into the training process.
- This defence therefore can not be used to guarantee fairness in the presence of fairness attacks.

4.4. Defence: Weak Differential Privacy

In this section, I test whether the weak Differential Privacy defence can effectively prevent fairness attacks.

The weak Differential Privacy defence did not produce as strong results as the previous two defences. Figure 4.8 shows a coarse[7] grid search of the overall accuracy after 10 rounds of models trained on CIFAR-10 in the presence of a fairness attack for different clipping and noise values (each box shows the accuracy of a different model). We can see that the model is either unable to learn (darker squares) or only learns the unfair model (lighter squares: these indicate the accuracy of a model that only predicts classes 0 and 1 correctly). In both cases, the accuracy on classes 2-9 remains low.

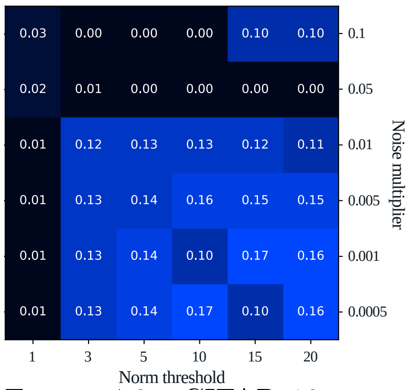

Figure 4.8.: CIFAR-10 accuracy for the DP defence with fairness attack

A similar grid can be obtained for the other tasks: fig. 4.9 shows a similar setup for the Census dataset. We can see that the sets of noise multiplier and norm threshold values that yield high accuracy on both grids are disjoint, so there is no high accuracy setup that is fair for this defence.

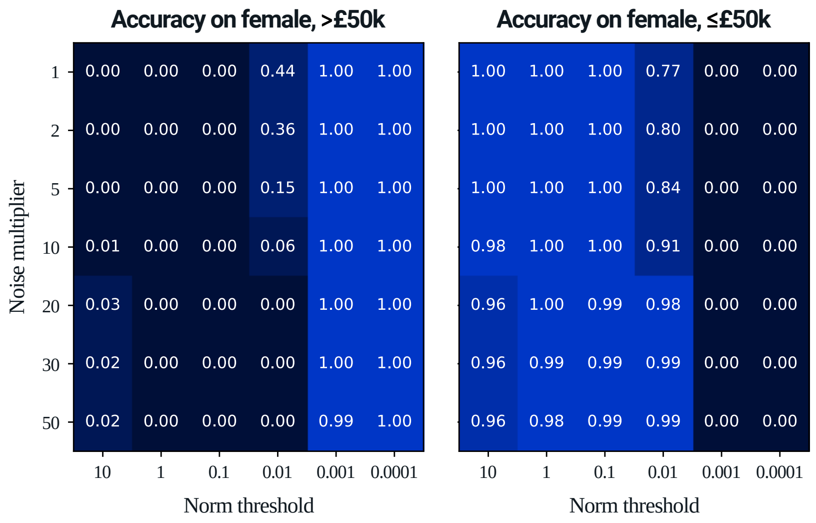

Figure 4.9.: Accuracy on the target data (right) and the untargeted data (left) for the Census dataset. No DP defence setup results in high accuracy for both categories.

In table 4.5, on the Census dataset, the backdoor is prevented, but the accuracy has dropped to 76%, which is the accuracy that would be assigned to a model that always predicts the majority class. As discussed in section 2.3.2.1, the evidence for the effectiveness of this defence is limited to only a few empirical results. Furthermore, H. Wang et al. (2020) also demonstrated a backdoor attack that is not prevented by weak DP. This failure to consistently produce effective results indicates that weak Differential Privacy is unlikely to be an effective defence against fairness attacks. Additionally, recent works have also shown that it can introduce unfairness into a model when used with FL [Bagdasaryan and Shmatikov 2019; Ganev, Oprisanu, and Cristofaro 2022; Farrand et al. 2020].

[7] I also experimentally verified the claims made here on more precise values along the boundary shown in fig. 4.8.

Table 4.5.: Results for the weak DP defence

Dataset	Attack	Overall	Backdoor ASR	Fair (inc.)	Fair (dec.)	Fair (chg.)
Census	Baseline	75.66	0.76	98.28	0.51	
	Backdoor	75.73	**0.72**	98.49	0.34	0.0000
	Fairness	76.38	0.00	100.00	0.00	**0.2604**
CIFAR-10	Baseline	93.60	10.19	95.85	93.04	
	Backdoor	90.36	**99.14**	94.35	89.36	0.0011
	Fairness	17.62	63.63	88.10	0.00	**0.8596**
Reddit	Baseline	08.55	0.00	0.54	8.93	
	Backdoor	0.00	**100.00**	0.00	0.00	-0.0079
	Fairness	04.52	0.00	100.00	0.00	**0.9972**

Update prediction is resistant to norm clipping Both attacks produce updates with a higher magnitude than the average (see fig. 4.4). On the same task, the differential privacy defence applies a norm threshold of 3 to clip each update by. This is less than 25% of the length of almost all malicious updates produced by the fairness attack, so, while the attack produces large updates, a much shorter version can still have a surprisingly large effect.

Figure 4.10 shows the cosine similarity between the *clean* aggregated update and the malicious updates for the Census task. The backdoor attack remains very similar to the aggregated update, but the fairness attack produces updates that point in a different direction. While this may make the fairness attack easier to detect, a larger angle is more 'economical' for a fixed-norm update: if the 10th client has an angle of α to the aggregate of the others, it will cause the direction of the resulting model to change by at least around 0.1 radians for $\alpha = \pi/2$, but as $\alpha \to 0$, it will have an increasingly small effect.[8]

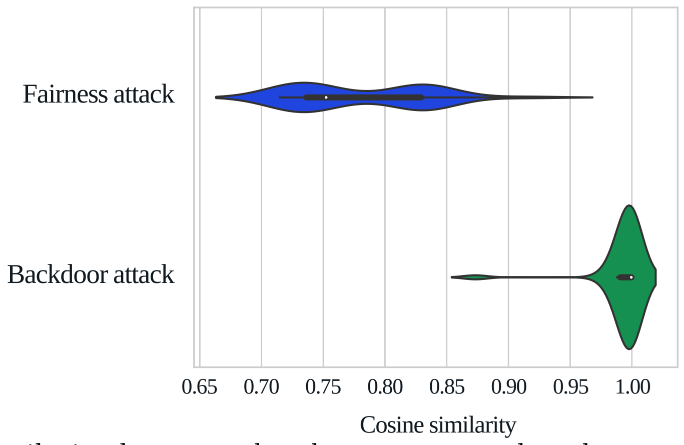

Figure 4.10.: Cosine similarity between the clean aggregated update and the malicious update for the Census task. The backdoor attack is aligned closer to the benign updates than the fairness attack.

Conclusions.

- The weak Differential Privacy defence is neither fair nor robust for many tasks.
- This defence would therefore not be an effective solution to guarantee safety from fairness attacks.

[8]The two attacks have different target models here, so this isn't a totally fair comparison. Instead, it is an illustration of how an attacker may gain more control over a model without increasing the magnitude of their submitted parameters.

4.5. Defence: Unfair-update detection *(extension)*

In this section, I test whether the unfair-update detection defence can effectively prevent fairness attacks.

Table 4.6 shows the results of applying the defence to the CIFAR-10 task. There is a significant accuracy drop, which is likely due to removing so many clients. However, we can see that the drop in fairness is minimal, so unfair-update detection has been an effective defence.

Table 4.6.: Results for the unfair-update detection defence

Attack	Overall	Fair (inc.)	Fair (dec.)	Fair (chg.)
Baseline	**62.39**	70.20	60.44	
Fairness	62.72	70.80	60.70	**0.0000**

Synthetic dataset tests: Unfair-update detection can introduce unfairness. Unfair-update detection is, surprisingly, *not* a fair defence method. The problem arises from the attack's greedy setup: we select the clients that produce the fairest model *on the next round*, not those that result in a fairer *final* model. However, it may be necessary to temporarily reduce fairness to reach a good value. Ironically, this defence is similar to some of the fair aggregation methods discussed in appendix B, which may, therefore, encounter similar issues.

To demonstrate the problem, I construct a dataset made from client groups C and D. Clients in group C contain the data $((X, Y), Z)$, where $Z = X \oplus Y$, $\oplus$ is the XOR operation, and (X, Y) is sampled uniformly from $[(0, 0), (1, 0), (1, 1)]$, and clients from group D contain the data $((X, Y), Z)$, where both X and Y are Bernoulli random variables with $p = 0.5$. To get the maximum fairness, we must include group B, but introducing group B for a single round reduces fairness by producing weights that reduce the accuracy of some instances of group A data. When training on this dataset, a simple, two-layer, fully-connected model achieves 100% accuracy for all clients without any defence. However, when the defence is introduced, the accuracy for group B drops to around 75% on average.

Conclusions.

- Unfairness detection can be an effective defence against fairness attacks.
- Contrary to expectations, unfairness detection *can* introduce unfairness to a model.
- This defence is therefore not an effective choice for guaranteeing fairness in the presence of attacks on fairness.

4.6. Non-functional requirements

At the start of chapter 3, I describe five non-functional requirements to design my codebase around. Due to the successful **configurability** and **extensibility** of my codebase, it was very simple to run many experiments sequentially. Additionally, because my codebase remains **compatible** with many existing implementations, I could quickly iterate through testing different models (e.g. for the CIFAR-10 dataset I tested many ResNet configurations). I also documented my codebase clearly to improve **usability**.

However, while I was partially successful in creating an efficient codebase, the **performance** of my code was reduced by the lack of consistent model checkpoints. During testing, I often had to restart the experiment every time the code needed to be changed, which resulted in additional GPU cost.

5 Conclusions

5.1. Implications and future work: a new perspective on robust aggregation

In section 3.6, I proved that, in the presence of the fairness attack, defences based on anomaly detection cannot guarantee fairness for a private training routine. In chapter 4, I showed that in practice, none of the defences considered can guarantee fairness. Furthermore, to my knowledge, there does not exist any defence that can guarantee the robustness of the global model without using a method based on those described in this dissertation. Thus, assuming such a defence does not currently exist, **in the presence of untrusted clients, we cannot guarantee the fairness of a model trained on private data.**

Future work. To train fair and private (and robust) ML models, we require a guaranteeably fair defence method. I have shown that the current anomaly detection-based defences are fundamentally unable to make such promises. Therefore, future work would benefit from approaching the problem from a new perspective.

For example, in section 4.5, I explained that the unfair-update detection algorithm is not fair because it selects clients *greedily*. We can consider the aggregator's job to be a finite MDP, where the state is the current set of client models, the action is how they are aggregated into a global model, and the reward is the fairness of the final model. The issue we face in selecting the best set of clients is that the optimal client selection depends on future states. Thus an alternative avenue of robustness research may be to investigate how groups of clients can be used to separately train models *to completion*, allowing us to check if the *final* model is fair or not, and then to combine only those which are fair, thereby eliminating clients based only on the final model.

5.2. Criteria completed.

1. **Build a modular codebase for testing attacks on FL.** Chapter 3 describes the construction of a performant codebase for testing attacks on FL. I evaluate the structure of the codebase in section 4.6. The results of section 4.1 show that each baseline model and attack achieve results in line with, and in some cases higher accuracy than, previous work (core criteria **(a.1)** and **(a.2)**). All four of the defences were able to prevent at least one attack,[1] indicating a correct implementation (sections 4.2-4.5; core criterion **(a.3)**). This is the first extensible implementation of the attack on fairness, and an original extension of the update prediction attack to support multiple attackers.
2. **Evaluate the effectiveness of each defence method against the fairness attack.** Chapter 4 describes the evaluation of all four defences and discusses the implication of fairness attacks on the ability of private training routines to guarantee fairness (core criteria **(b)** and **(c)**). For each defence, including unfair-update detection (extension), I conclude that it is either not always effective at preventing the fairness attack, or introduces unfairness itself.

[1]The weak DP defence *was* able to prevent the backdoor attack on the Census dataset, although the large clean accuracy drop means this may not be particularly useful.

3. **Test the attack against the unfairness detection defence and different aggregators.** Section 4.5 evaluates the performance of the new unfairness detection defence against the update prediction attack and analyses its effects on fairness (extension criterion **(a)**). Section 4.1 describes the attack's performance against three momentum-based aggregators (extension criterion **(c)**), finding that it remains effective in these settings.

4. **Theoretically show that fairness and privacy are difficult to simultaneously guarantee.** In section 3.6.1, I provide an original proof to justify the theoretical foundation of the update prediction attack (extension criterion **(d)**). In section 3.6.2, I further show that defences based on anomaly detection cannot guarantee fairness in the presence of this attack (extension criterion **(b)**).

5.3. Lessons learnt

The scale of this project has highlighted some mistakes I made early on. Primarily, I did not emphasise the value of effective experiment tracking enough for a large project like this. I implemented my own system, which has resulted in a zip file of hundreds of experiments on my desktop[2]. A better alternative would have been to use specialised libraries or services such as Hydra and Weights&Biases. More generally, I have learnt that if there is a tool for some specific function, it is almost always better to use the existing tool than to try to implement it myself, despite how simple it may seem.

[2]On the other hand, this system *has* been effective at ensuring all metrics are tracked in a reproducible manner.

Bibliography

Bagdasaryan, Eugene and Vitaly Shmatikov (2019). *Differential Privacy Has Disparate Impact on Model Accuracy.* arXiv: 1905.12101 [cs.LG] (cit. on pp. 10, 35).

Bagdasaryan, Eugene, Andreas Veit, et al. (2019). *How To Backdoor Federated Learning.* arXiv: 1807.00459 [cs.CR] (cit. on pp. 7, 9, 16, 20, 28).

Baruch, Moran, Gilad Baruch, and Yoav Goldberg (2019). *A Little Is Enough: Circumventing Defenses For Distributed Learning.* arXiv: 1902.06156 [cs.LG] (cit. on p. 10).

Becker, Barry and Ronny Kohavi (1996). *Adult.* UCI Machine Learning Repository. URL: https://doi.org/10.24432/C5XW20 (cit. on p. 16).

Beutel, Daniel J. et al. (2022). *Flower: A Friendly Federated Learning Research Framework.* arXiv: 2007.14390 [cs.LG] (cit. on p. 2).

Bhagoji, Arjun Nitin et al. (2019). *Analyzing Federated Learning through an Adversarial Lens.* arXiv: 1811.12470 [cs.LG] (cit. on pp. 16, 28).

Blanchard, Peva et al. (2017). "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent". In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf (cit. on pp. 1, 9, 21, 33).

Bober-Irizar, Mikel et al. (2022). *Architectural Backdoors in Neural Networks.* arXiv: 2206.07840 [cs.LG] (cit. on p. 6).

Buolamwini, Joy and Timnit Gebru (Feb. 2018). "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification". In: *Proceedings of the 1st Conference on Fairness, Accountability and Transparency.* Ed. by Sorelle A. Friedler and Christo Wilson. Vol. 81. Proceedings of Machine Learning Research. PMLR, pp. 77–91. URL: https://proceedings.mlr.press/v81/buolamwini18a.html (cit. on p. 4).

Candidate, The and Filip Svoboda (2023). *Attacks of fairness in Federated Learning.* arXiv: 2311.12715 [cs.LG] (cit. on pp. 1, 7, 11).

Chawla, N. V. et al. (June 2002). "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16, pp. 321–357. ISSN: 1076-9757. DOI: 10.1613/jair.953. URL: http://dx.doi.org/10.1613/jair.953 (cit. on p. 5).

Chen, Xinyun et al. (2017). *Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning.* arXiv: 1712.05526 [cs.CR] (cit. on p. 6).

Cho, Yae Jee, Jianyu Wang, and Gauri Joshi (2020). *Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies.* arXiv: 2010.01243 [cs.LG] (cit. on p. 46).

Commission, European, Content Directorate-General for Communications Networks, and Technology (2019). *Ethics guidelines for trustworthy AI.* Publications Office. DOI: doi/10.2759/346720 (cit. on p. 1).

Department for Education (2023). *The impact of AI on UK jobs and training.* Accessed: 2023-12-30. URL: https://www.gov.uk/government/publications/the-impact-of-ai-on-uk-jobs-and-training (cit. on p. 1).

Du, Wei et al. (2020). *Fairness-aware Agnostic Federated Learning.* arXiv: 2010.05057 [cs.LG] (cit. on p. 47).

Eloundou, Tyna et al. (2023). *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models.* arXiv: 2303.10130 [econ.GN] (cit. on p. 1).

Farrand, Tom et al. (2020). *Neither Private Nor Fair: Impact of Data Imbalance on Utility and Fairness in Differential Privacy.* arXiv: 2009.06389 [cs.LG] (cit. on p. 35).

Ganev, Georgi, Bristena Oprisanu, and Emiliano De Cristofaro (2022). *Robin Hood and Matthew Effects: Differential Privacy Has Disparate Impact on Synthetic Data*. arXiv: 2109.11429 [cs.LG] (cit. on p. 35).

Goyal, Priya et al. (2018). *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. arXiv: 1706.02677 [cs.CV] (cit. on p. 18).

Gu, Tianyu, Brendan Dolan-Gavitt, and Siddharth Garg (2019). *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. arXiv: 1708.06733 [cs.CR] (cit. on p. 6).

Hard, Andrew et al. (2019). *Federated Learning for Mobile Keyboard Prediction*. arXiv: 1811.03604 [cs.CL] (cit. on p. 50).

Hu, Zeou et al. (2023). *Federated Learning Meets Multi-objective Optimization*. arXiv: 2006.11489 [cs.LG] (cit. on p. 47).

Huba, Dzmitry et al. (2022). *Papaya: Practical, Private, and Scalable Federated Learning*. arXiv: 2111.04877 [cs.LG] (cit. on p. 50).

Jagielski, Matthew et al. (2019). *Differentially Private Fair Learning*. arXiv: 1812.02696 [cs.LG] (cit. on p. 10).

Johnson, Thaddeus L. et al. (2022). "Facial recognition systems in policing and racial disparities in arrests". In: *Government Information Quarterly* 39.4, p. 101753. ISSN: 0740-624X. DOI: https://doi.org/10.1016/j.giq.2022.101753. URL: https://www.sciencedirect.com/science/article/pii/S0740624X22000892 (cit. on p. 4).

Khorramfar, Mohammadreza (2023). *Securing Federated Learning Model Aggregation Against Poisoning Attacks via Credit-Based Client Selection*. DOI: 10.36939/ir.202308301626 (cit. on p. 11).

Krizhevsky, Alex (2009). "Learning Multiple Layers of Features from Tiny Images". In: URL: https://api.semanticscholar.org/CorpusID:18268744 (cit. on p. 16).

Kruskal, Joseph B. (1964). "Nonmetric multidimensional scaling: A numerical method". In: *Psychometrika* 29, pp. 115–129. URL: https://api.semanticscholar.org/CorpusID:11709679 (cit. on p. 32).

Lai, Fan et al. (2021). "Efficient Federated Learning via Guided Participant Selection". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (cit. on p. 46).

Lan, Zhenzhong et al. (2020). *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. arXiv: 1909.11942 [cs.CL] (cit. on p. 17).

Larson, Jeff et al. (2016). *How We Analyzed the COMPAS Recidivism Algorithm*. Accessed: 2024-01-01. URL: https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm (cit. on p. 4).

Li, Tian, Ahmad Beirami, et al. (2021). *Tilted Empirical Risk Minimization*. arXiv: 2007.01162 [cs.LG] (cit. on pp. 1, 46).

Li, Tian, Shengyuan Hu, et al. (2021). *Ditto: Fair and Robust Federated Learning Through Personalization*. arXiv: 2012.04221 [cs.LG] (cit. on pp. 10, 45).

Li, Tian, Maziar Sanjabi, et al. (2020). *Fair Resource Allocation in Federated Learning*. arXiv: 1905.10497 [cs.LG] (cit. on p. 46).

Li, Tiejun, Tiannan Xiao, and Guoguo Yang (2023). *Revisiting the central limit theorems for the SGD-type methods*. arXiv: 2207.11755 [math.OC] (cit. on pp. 23–25).

Li, Yubin (2017). *Algorithmic Discrimination in the U.S. Justice System: A Quantitative Assessment of Racial and Gender Bias Encoded in the Data Analytics Model of the Correctional Offender Management Profiling for Alternative Sanctions (COMPAS)*. Accessed: 2024-01-01. URL: http://jhir.library.jhu.edu/handle/1774.2/61818 (cit. on p. 4).

Liu, Haochen et al. (2021). *Trustworthy AI: A Computational Perspective*. arXiv: 2107.06641 [cs.AI] (cit. on p. 1).

Maaten, Laurens van der and Geoffrey Hinton (2008). “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86, pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html (cit. on p. 29).

Mao, Yuzhen et al. (2023). *Last-Layer Fairness Fine-tuning is Simple and Effective for Neural Networks.* arXiv: 2304.03935 [cs.LG] (cit. on p. 10).

McMahan, H. Brendan et al. (2023). *Communication-Efficient Learning of Deep Networks from Decentralized Data.* arXiv: 1602.05629 [cs.LG] (cit. on pp. 16, 28, 46).

Mhamdi, El Mahdi El, Rachid Guerraoui, and Sébastien Rouault (2018). *The Hidden Vulnerability of Distributed Learning in Byzantium.* arXiv: 1802.07927 [stat.ML] (cit. on p. 10).

Mohri, Mehryar, Gary Sivek, and Ananda Theertha Suresh (2019). *Agnostic Federated Learning.* arXiv: 1902.00146 [cs.LG] (cit. on pp. 46, 47).

Mukai, H. (1980). “Algorithms for multicriterion optimization”. In: *IEEE Transactions on Automatic Control* 25.2, pp. 177–186. DOI: 10.1109/TAC.1980.1102298 (cit. on p. 47).

Nguyen, Thien Duc et al. (2023). *FLAME: Taming Backdoors in Federated Learning (Extended Version 1).* arXiv: 2101.02281 [cs.CR] (cit. on pp. 11, 16, 18, 28).

NIST (2019). “Face Recognition Vendor Test Part 3: Demographic Effects”. In: Accessed: 2024-01-01. URL: https://www.govinfo.gov/app/details/GOVPUB-C13-5a5c1d28d99c5a718d50bdbbae85cb (cit. on p. 4).

Pushshift (n.d.). *Reddit Comment Data.* https://files.pushshift.io/reddit/comments/ (cit. on p. 16).

Rance, Joseph et al. (2022). *Augmentation Backdoors.* arXiv: 2209.15139 [cs.LG] (cit. on p. 6).

Reddi, Sashank et al. (2021). *Adaptive Federated Optimization.* arXiv: 2003.00295 [cs.LG] (cit. on p. 12).

Rieger, Phillip et al. (2022). “DeepSight: Mitigating Backdoor Attacks in Federated Learning Through Deep Model Inspection”. In: *Proceedings 2022 Network and Distributed System Security Symposium.* NDSS 2022. Internet Society. DOI: 10.14722/ndss.2022.23156. URL: http://dx.doi.org/10.14722/ndss.2022.23156 (cit. on p. 10).

Salewski, Leonard et al. (2023). *In-Context Impersonation Reveals Large Language Models' Strengths and Biases.* arXiv: 2305.14930 [cs.AI] (cit. on p. 4).

Sharma, Shubham et al. (2020). “Data Augmentation for Discrimination Prevention and Bias Disambiguation”. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society.* AIES '20. New York, NY, USA: Association for Computing Machinery, pp. 358–364. ISBN: 9781450371100. DOI: 10.1145/3375627.3375865. URL: https://doi.org/10.1145/3375627.3375865 (cit. on p. 5).

Shi, Yuxin, Han Yu, and Cyril Leung (2023). “Towards Fairness-Aware Federated Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17. ISSN: 2162-2388. DOI: 10.1109/tnnls.2023.3263594. URL: http://dx.doi.org/10.1109/TNNLS.2023.3263594 (cit. on p. 45).

Shumailov, Ilia et al. (2021). *Manipulating SGD with Data Ordering Attacks.* arXiv: 2104.09667 [cs.LG] (cit. on pp. 6, 27).

Sun, Ziteng et al. (2019). *Can You Really Backdoor Federated Learning?* arXiv: 1911.07963 [cs.LG] (cit. on pp. 10, 21).

Wang, Hongyi et al. (2020). *Attack of the Tails: Yes, You Really Can Backdoor Federated Learning.* arXiv: 2007.05084 [cs.LG] (cit. on pp. 10, 11, 16, 28, 34, 35).

Xu, Han et al. (July 2021). “To be Robust or to be Fair: Towards Fairness in Adversarial Training”. In: *Proceedings of the 38th International Conference on Machine Learning.* Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 11492–11501. URL: https://proceedings.mlr.press/v139/xu21b.html (cit. on p. 1).

Yang, Timothy et al. (2018). *Applied Federated Learning: Improving Google Keyboard Query Suggestions.* arXiv: 1812.02903 [cs.LG] (cit. on p. 50).

Yin, Dong et al. (2021). *Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates*. arXiv: 1803.01498 [cs.LG] (cit. on pp. 10, 21, 31).

Zagoruyko, Sergey and Nikos Komodakis (2017). *Wide Residual Networks*. arXiv: 1605.07146 [cs.CV] (cit. on p. 17).

Zeng, Yi et al. (2022). *Adversarial Unlearning of Backdoors via Implicit Hypergradient*. arXiv: 2110.03735 [cs.LG] (cit. on p. 10).

Zhang, Fengda et al. (2022). "Towards Multi-level Fairness and Robustness on Federated Learning". In: *ICML 2022: Workshop on Spurious Correlations, Invariance and Stability*. URL: https://openreview.net/forum?id=1Gdr2I1c1X5 (cit. on p. 45).

A A definition of fairness in FL.

Unfairness can be introduced into an FL model at most points in the training pipeline, for example, by selecting clients with data that might only represent part of the distribution [Shi, Yu, and Leung 2023]. However, the conflict between privacy, fairness, and robustness is largely irrelevant at most stages, so I will only consider how aggregation affects fairness during the FL training process.

The most fundamental distinction for fairness in Learning is between model *performance* across protected attributes (e.g. whether the model is more accurate for certain minority groups of data) and model *predictions* across protected attributes (e.g. whether the model is more likely to predict a defendant will re-offend for certain minority groups). While prediction bias is usually due to bias in the original dataset, performance bias often arises because the training procedure spends more time optimising for more common categories of data. This dissertation focuses on how we can prevent unfairness from being introduced into a model *during training*, so I will focus on unfairness in performance.

Zhang et al. (2022) separate performance fairness into three categories: **client level** (between different clients), **attribute level** (between data containing different attributes), and **agnostic distribution** (between clients which have participated for a different number of rounds). I will focus on *attribute level fairness*, because this has the clearest practical effects on the trained model. I will use a similar definition to Tian Li, S. Hu, et al. (2021).

Definition 1 (fairness)
For some $\epsilon \in [0, \infty)$, we say that a model with parameters c_i is 'fair' if and only if

$$\mathbb{V}_A \left[\mathbb{E}_{S_A} \left[\mathcal{L}(\mathbf{c}_i, S_A)\right]\right] \leq \epsilon \tag{A.1}$$

where A is a uniformly random sample from the set of all relevant attributes of the data (e.g. class, features, style, ...), and

$$P(S_A = s) = \begin{cases} kP(S = s) \text{ if data point } s \text{ has attribute } A \\ 0 \text{ otherwise} \end{cases}$$

for some k, where S is sampled from the distribution we wish to learn. $\mathcal{L}(\mathbf{c}_i, (x, y))$ represents the loss between y and the output of a model with weights $\mathbf{c}_i$ when given input x.

We can control the degree of acceptable variance with the parameter ϵ. In almost every case in this dissertation, a model labelled as 'fair' would satisfy this definition for any reasonable value of ϵ.

(ref. on p. 1, 4, 18)

B Methods for fair FL

Agnostic Federated Learning. Many simple aggregation functions in FL (FedAvg [McMahan et al. 2023], for example) introduce unfairness by assuming the distribution we want to model and the empirical distribution of each client's dataset are approximately the same. In practice, this is often not the case. Early works on fairness in FL model this by letting the distribution we want to model, D_λ, be a mixture of smaller D_as.

$$\mathcal{D}_{\boldsymbol{\lambda}} = \sum_{s} a\boldsymbol{\lambda}_a D_a \tag{B.1}$$

Here, D_a has a very similar meaning to the distribution S_A is sampled from in **Definition 1**, appendix A. Mohri, Sivek, and Suresh (2019) propose that a better aggregation may therefore be

$$\mathcal{L} = \max_{\boldsymbol{\lambda}\in\Lambda} \mathcal{L}(\mathbf{c}_i, D_{\boldsymbol{\lambda}}) \tag{B.2}$$

Where $\mathcal{L}(\mathbf{c}_i, \mathcal{D}_{\boldsymbol{\lambda}})$ is similar to in **Definition 1** for model parameters $\mathbf{c}_i$. The idea is that if we do not know the true $\boldsymbol{\lambda}$, then we should optimise the worst-case. While this does not directly target the notion of fairness described in **Definition 1**, it intuitively makes sense that models with higher performance variance between subpopulations of the dataset (D_as) would result in higher loss in the worst case.

In practice, we do not have access to any set D_a. However, each client's local data will be a linear combination of the set of D_as, so with a large enough client count, we can achieve approximately the same objective by substituting each D_a with the dataset of a single client.

A more general formula for fair FL. Tian Li, Sanjabi, et al. (2020) proposed q-FFL, in which, instead of weighting each client's loss by a scalar value, we can raise the losses to the power of some $q + 1$, before returning the mean of these values. In general q-FFL and many more recent techniques follow the general form:

$$\mathcal{L}_\Lambda(\mathbf{c}_i) = \sum_{k=1}^{m} f(\mathcal{L}(\mathbf{c}_i, D_i), n_i) \tag{B.3}$$

where n_i is the length of the dataset, D_i, owned by client i, and $f(\ell, n)$ is a differentiable function which is convex and increasing for positive ℓ. For example $f(\ell, \cdot)$ could alternatively be e^ℓ [Tian Li, Beirami, et al. 2021]. These methods all aim to provide clients with higher loss more control over the model, reasoning that such clients likely hold more data containing attributes that the model performs poorly on.

Fairness through sampling. In scenarios where not all clients are selected on every round, biasing the FL training routine to select high-loss clients to participate in training rounds more frequently provides an alternative method to increase their impact on the global model [Lai et al. 2021; Cho, J. Wang, and Joshi 2020]. While this requires potentially unreliable predictions of client loss to decide which clients to select, it has the advantage of preventing any single data point from having a disproportionate impact on the global model within any single round.

Federated Multiple Gradient Descent Algorithm. One drawback of these approaches is that they assume all clients are trustworthy: an attacker could maliciously inflate their loss to gain control over the shared model. Z. Hu et al. (2023) demonstrated that by applying the Multiple Gradient Descent Algorithm [Mukai 1980] to FL, each client's loss can be simultaneously minimised according to the partial order

$$\mathbf{u} \leq \mathbf{v} \iff \forall i \in \{0, \ldots, n\}.\mathbf{u}_i \leq \mathbf{v}_i \tag{B.4}$$

The authors claim this implicitly protects fairness by discouraging the aggregator from improving one client's loss at the cost of another's. Z. Hu et al. (2023) also suggest that normalising the gradient produced by each client may further improve the algorithm's robustness to adversarial attacks.

Loss functions with fairness constraints. While the above methods focus on how clients' models can be more fairly aggregated, Du et al. (2020) suggest a different perspective in which the clients are themselves incentivised to learn fair models by adding a fairness term to their loss function. In the context of **Definition 1**, this could be the inter-attribute variance in accuracy. Similarly to Mohri, Sivek, and Suresh (2019), the authors claim that simply adding this term alone may not be sufficient due to the distribution shift between the data used by the clients and the distribution we are trying to model. To solve this, they suggest reweighting each data point by some function, $\theta_{\boldsymbol{\alpha}}$, subject to

$$\frac{1}{n}\sum_{k=1}^{p}\sum_{i=1}^{n_k}\theta_{\boldsymbol{\alpha}}(\mathbf{x}_i^k) = 1 \tag{B.5}$$

where $\mathbf{x}_i^k$ is the ith data point on the kth client.[1] For example, we can parameterise $\theta_{\boldsymbol{\alpha}}$ as a sum of basis functions, K:

$$\theta_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{m=1}^{M}\boldsymbol{\alpha}_m K_m(\mathbf{x}) \tag{B.6}$$

By allowing the server to iteratively select the values for $\boldsymbol{\alpha}$ which maximise the overall loss, we can use a similar minimax setup to Mohri, Sivek, and Suresh (2019) to compute $\theta_{\boldsymbol{\alpha}}$ without direct knowledge of either distribution.

(ref. on p. 5, 8, 37)

[1]This notation differs from appendix G for consistency with the paper.

C Defence algorithms.

C.1. Krum

Algorithm 1 Krum

```
Input: number of clients, $n$; number to select each round, $m$; max Byzantine clients, $f$
Output: global model at round $T$, $G_T$
1: Initialise global model $G_0$
2: Initialise matrices distances $\in \mathbb{R}^{n \times n}$ and KM_distances $\in \mathbb{R}^n$
3: $c \leftarrow n - f - 2$        ▷ we consider the closest $c$ models to each client
4: for each training round, $t$, in $[1, T]$ do
5:     for each client, $i$, in $[1, n]$ do
6:         $r_i \leftarrow$ get_client_update($G_{t-1}$)        ▷ server sends client $i$ $G_{t-1}$ and gets response $r_i$
7:     end for
8:     for each client, $i$, in $[0, \ldots, n-1]$ do
9:         for each client, $j$, in $[0, \ldots, n-1]$ do
10:            distances$_{i,j} \leftarrow$ distances$_{j,i} \leftarrow ||r_i - r_j||^2$
11:        end for
12:    end for
13:    for each client, $i$, in $[0, \ldots, n-1]$ do
14:        closest_distances $\leftarrow$ $c$ smallest values in distances$_i$
15:        KM_distances$_i \leftarrow \sum_{j=0}^{c-1}$ closest_distances$_j$
16:    end for
17:    selected_clients $\leftarrow$ indexes of $m$ smallest values in KM_distances
18:    $G_t \leftarrow \frac{1}{m} \sum_{i=0}^{m-1} r_{\text{selected\_clients}_i}$
19: end for
```

(ref. on p. 9)

C.2. Unfair-update detection

Algorithm 2 Unfair-update detection

```
Input: number of malicious clients, $a$; vector of datasets to compute fairness across, $\mathbf{d}$
Output: global model at round $t$, $G_t$
1: Initialise global model $G_0$
2: for each training round, $t$, in $[1, T]$ do
3:     for each client, $i$, in $[1, m]$ do
4:         $r_i \leftarrow$ get_client_update($G_{t-1}$)        ▷ server sends client $i$ $G_{t-1}$ and gets response $r_i$
5:     end for
6:     for each combination of $m - a$ clients, $c$ do        ▷ loop is expensive for large $a$ and $|\mathbf{d}|$
7:         $G'_t \leftarrow$ aggregate($\{r_i | i \in c\}$)
8:         $f \leftarrow$ compute_fairness($G'_t, \mathbf{d}$)        ▷ Using Definition 1
9:         if $f$ is the most fair so far then
10:            $G_t \leftarrow G'_t$
11:        end if
12:    end for
13: end for
```

(ref. on p. 11)

D Detection of fairness attacks by weight magnitude

We can practically see that the trimmed mean methodology is likely to work against the new update prediction attack by determining its expected update vector norm, which correlates with weight magnitude. Consider an attack with 1 malicious client out of n. If the angle between the target model and the global model (for model replacement) or the predicted update (for update prediction) is α, and both the target and global models have norm x, we can expect to produce a malicious set of parameters with norm

$$x\sqrt{2n(n-1)(1-\cos(\alpha)} \tag{D.1}$$

for both attacks. If $n = 10$, $x = 9.25$, and $\alpha \in [0, \pi/4]$ (based on fig. 4.2), we will get a norm in $[0.0, 67.2]$, which can be far out of the distribution shown in fig. 4.2.

In practice, a determined attacker could reduce the update magnitude, by using more malicious clients, or clipping the updates to the expected mean norm (which would likely retain much of the attack's previous ability, as shown in section 4.4). Nevertheless, since I do not employ any such methodologies to hide the attacks, we can expect the trimmed mean defence to remain effective against the fairness attack.

(ref. on p. 11, 31)

E Fairnesss attacks under high data heterogeneity

Data heterogeneity Real-world tasks exhibit many types of heterogeneity, most commonly through varying data composition, dataset size, device availability, and hardware performance between clients [T. Yang et al. 2018; Hard et al. 2019; Huba et al. 2022]. In the case of attacks on fairness, heterogeneity in data composition across clients is likely to have the biggest effect on the attack.

Testing with heterogeneity I have not directly introduced heterogeneity into any of the experiments performed in the main body of this dissertation. Therefore, to show that the fairness attack remains effective under non-i.i.d. client datasets, I have gathered the results below which show that this remains the case.

Introducing data heterogeneity. I split the CIFAR-10 dataset using a log-normal distribution[1]. I tested with the class distributions parameterised by $\mu = 0$ and $\sigma \in \{0, 1, 2\}$. Figure E.1 shows how the distribution of data between clients with different values of σ. I tested the fairness attack with the same parameters as in section 4.1.

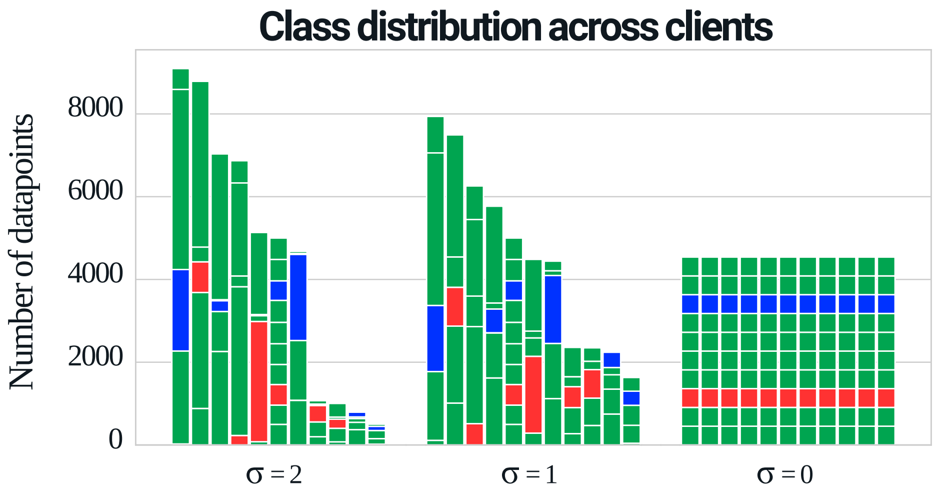

Figure E.1.: Data distribution for different σ. Each vertical bar represents one client's data, where the length of a single section represents the number of data points of the corresponding class. For readability, I have coloured all but two classes green. In all three cases, at least one client has a perfectly even distribution of data, which is the client that represents our attacker.

Results. Figure E.2 shows that although heterogeneity has some effect on the performance of the fairness attack, the attack remains effective. Additionally, adding heterogeneity would make it more difficult to detect the attack.

[1]using a function provided in the CaMLSys template

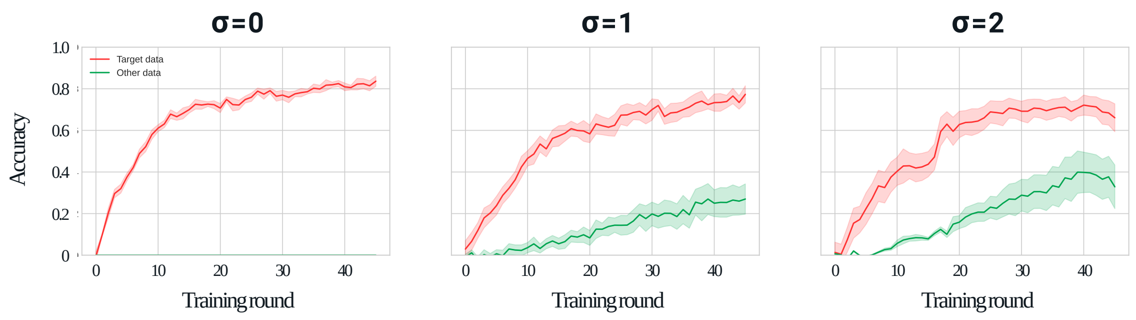

Figure E.2.: Training curves under the fairness attack with different heterogeneity values. As heterogeneity increases (higher σ), the attack's strength decreases.

(ref. on p. 16, 16, 23)

F Testing attack recovery by introducing Krum *during* training

Here, I investigate the dynamic between adding attacks and defences at different times during training. Specifically, I am interested in:

- Does the model retain some residual knowledge of non-target classes that allows it to rapidly re-learn a fair representation after an attack is removed?
- Does the Krum defence reduce the recovery speed of a model after an attack finishes by removing the large updates produced due to the unfair model?

Figure F.1 shows how the accuracy on the set of data that is disadvantaged by the fairness attack changes over time in different scenarios. Accuracy increases faster after an attack is removed compared to training the initial model (compare accuracy increase at round 0 to at round 30 in the upper middle plot).

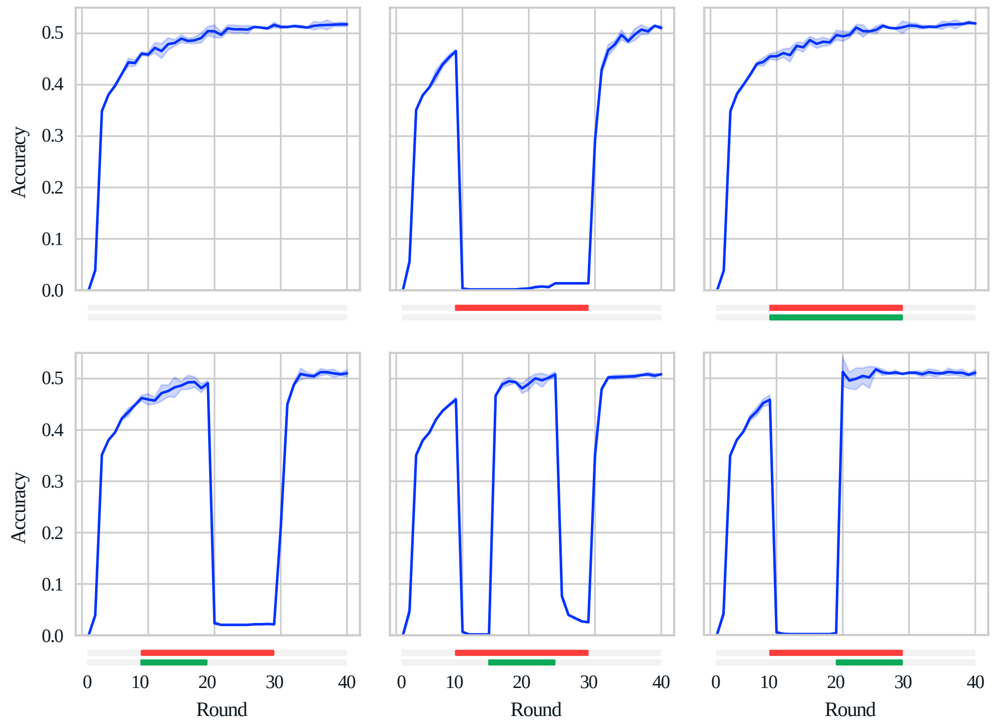

Figure F.1.: Accuracy curves where the fairness attack and Krum defence are added at different points in training

Furthermore, including the defence counter-intuitively *increases* rate of recovery (now compare accuracy increase at round 0 to at round 20 in the lower right plot), which may be due to lower variance in client updates when the model removes those that lie furthest from the centre of the distribution. This implies that it is preferable to continue training an attacked model using Krum over restarting the training process entirely.

(ref. on p. 32)

G Notation reference

Federated Learning

G_t	The parameters of the global model at time t as a vector (upper-case used for consistency with previous work)
$\mathbf{c}_i$	The vector of client i's parameters at the current training round
C_i	The random variable representing the parameters submitted by client i
C	The distribution of updates that could be produced by clean clients
m	The total number of clients in the current training round (redefined in section 2.3.2.1)
p_i	The weighting applied to parameters of client i during aggregation ($\frac{n_i}{n}$ for FedAvg)
n_i	The number of training samples owned by client i
n	The total amount of data owned by all clients (redefined in section 2.3.2.1)
$\mathcal{D}$	The clean training data distribution
$\mathcal{D}_i$	The data distribution for client i
s	A tuple (x, y) sampled from the clean training distribution
D	The union of clean client datasets, sampled from $\mathcal{D}$
D_i	The dataset owned by client i
d	The number of parameters in the model being trained
h	The heterogeneity factor of the participating clients

Local model training

u	The number of batches each local model is trained for
e	The number of epochs each local model is trained for
k	The current epoch
α_k	The local learning rate at epoch k
$\mathbf{c}_i^{(k)}$	The estimate of client i's parameters at epoch k
b	The mini-batch size
$s_{k,j}$	The jth uniformly random sample of a data point index at epoch k
ξ_k	The noise introduced by sampling from $\mathcal{D}_i$ on round k

Loss functions

$\mathcal{L}(\mathbf{c}_i, s)$	The loss between y and the output of a model with weights $\mathbf{c}_i$ when given input x, where $(x, y) = s$
$\mathcal{L}(\mathbf{c}_i, \mathbf{l})$	The sum of $\mathcal{L}(\mathbf{c}_i, s)$ for each $s \in \mathbf{l}$
μ	The loss function's convexity term
L	The loss function's smoothness term
$\mathbf{c}_i^*$	The unique minimum of the loss function for client i

Fairness

ϵ	The performance variance threshold for a model to be defined as "fair"
A	A uniformly random sample from the set of all relevant attributes of the test set (e.g. class, feature, style, ...). The difference to A below should be clear from context
S_A	A sample from the legitimate training set that contains attribute A
D_A	The set of all data points in dataset D that contain attribute A

Attacks on FL

A	The set of updates that could successfully perform an attack
a	The number of malicious clients
$\mathbf{x}$	A vector of parameters that the attacker wishes to substitute into G_{t+1}
T	The function to modify a data point to contain the backdoor trigger
F	The function to modify a data point to follow the backdoor functionality
D_T	A dataset containing only data the fairness attack aims to increase the accuracy of
D_N	A dataset containing only data the fairness attack aims to decrease the accuracy of
$\mathbf{w}$	The attacker's prediction of the aggregated update without the malicious parameters
$\mathbf{w}^{(i)}$	The ith attacker's prediction of the aggregated update without the malicious parameters
F	A function representing a deterministic anomaly detection defence as a predicate

Operations

$\lfloor x \rfloor$	The largest integer less than or equal to x ($x \geq 0$)
$[x]$	The set of natural numbers less than or equal to x
$[x_0, \ldots, x_{n-1}]$	A vector with x_i at element i
$\mathbf{v}_i$	The value at index i of vector $\mathbf{v}$
$M_{i,j}$	The value at row i and column j of matrix M.
$M_{i,*}$	A vector representing row i of matrix M
$\nabla f(x)$	The gradient of f at point x
$\nabla \hat{f}(x)$	An unbiased, normally distributed estimate of $\nabla f(x)$

Operations (probability)

$P(f(X))$	The probability predicate f is true for random variable X
$\text{Image}(X)$	The set of possible values for random variable X
$\mathbb{E}_X$	The expectation of $f(X)$ over random variable X
$\mathbb{V}_X$	The variance of $f(X)$ over random variable X
$N(\mu, \sigma^2)$	A normal distribution with mean μ and variance σ^2
$X_k \Rightarrow^k X$	X_k converges in probability to X, where $k \to \infty$

H Project proposal

Project proposal: Evaluating attacks on fairness in Federated Learning

Candidate 2417C

Introduction

Federated Learning (FL) is a machine learning paradigm that allows models to be trained on private data by distributing training across user devices. In FL, each user locally trains a copy of the model on its own private data, and then sends the resulting weights to a server. The server then aggregates the weights to produce a central model that has learnt from the data of all clients [McMahan et al. 2016].

Consequently, by using FL, we can train a model without requiring each user to send their private data to the server. FL has therefore been used in industries such as health and telecommunications in order to keep sensitive user data private. However, the reduced transparency of private training data has also opened the door for a variety of training time backdoor attacks. This is because it is difficult for the server to verify updates produced by clients as it cannot access the training data used to produce them. Backdoor attacks, where the client produces updates which cause the model to have new functionality in the presence of some trigger, have been widely studied, along with defences against them [Bagdasaryan et al. 2020; Wang et al. 2020; Fang et al. 2020; Bhagoji et al. 2019].

However, attacks on machine learning do not necessarily need to insert a backdoor. What if the attacker instead wants to modify the existing behaviour to provide them with some advantage over other users? In this project, I will investigate attacks which target *fairness* in FL. Here, instead of introducing new functionality, we attempt to force the model to have higher accuracy on data that has a certain attribute. In short, this is a denial of service attack on data that does not contain this specific attribute. Through employing a fairness attack, a malicious user could gain an unfair advantage over other users by sabotaging the shared model's accuracy on only certain types of data. Preventing this kind of attack would be critical in areas such as healthcare, where unfairness in treatment between groups of people could have serious consequences.

Description of the project

In a previous paper, I have introduced an attack on fairness in FL [Candidate and Svoboda 2023]. However, this attack has so far only been tested on a simple dataset with no defence. It has not yet been fully established how susceptible FL currently is to attacks on fairness, and how well it can be defended. In this project, I will implement three existing backdoor defences to test their effectiveness on the fairness attack when applied to a variety of common datasets. My dissertation will attempt to answer the question: "to what extent can current defences for backdoor attacks be used to protect federated learning models from attacks on fairness?"

I will first implement the fairness attack against models trained on the CIFAR10, Reddit, and 1994 adult census datasets [Krizhevsky 2009; Völske et al. 2017; Becker and Kohavi 1996]. Then, I will test the performance of the attacks against three backdoor defences:

- Weak differential privacy: using update clipping and Gaussian noise to reduce the effect of malicious updates [Sun et al. 2019].
- Robust aggregation: using the geometric mean [sic] to reduce the effect of out-of-distribution updates [Yin et al. 2021].
- Krum: using outlier detection to detect and suppress updates that may be malicious [Blanchard et al. 2017].

These three defences were selected to provide a wide coverage of common techniques to defend against adversaries in FL. They are intended for backdoor attacks, however fairness attacks have very similar characteristics: the malicious clients produce updates that are distributed differently to the clean clients. Therefore, it is reasonable to believe that these defences will be at least partially successful in defending against attacks on fairness.

One of the major risks with this project is that the defences are not guaranteed to work against the fairness attack. If this is the case, my project can still be a success, as this is the question I am trying to answer. However, I would need to verify that I have correctly implemented the defences. Therefore to mitigate this risk, I will also implement a simple backdoor attack to verify each defence against.

I expect the main implementation challenge of my project to be a combination of constructing effective baseline models for each of the three datasets I have chosen and implementing the three defences above.

As a potential extension, I will investigate some other possible methods of attacking fairness in FL. Below are three ideas for new attacks.

- Test the existing fairness attack on aggregation functions other than FedAvg. For example, FedAdam [Reddi et al. 2021], or TERM [Li et al. 2021].
- Existing "fair" aggregators such as TERM attempt improve [sic] fairness by boosting the influence of outliers on the aggregated model. This introduces a vulnerability by allowing us to increase the influence of a single client by producing unusual updates. An attack could therefore be constructed by splitting updates into components that will later combine back into the original update, while individually being boosted due to being detected as outliers. This is an interesting attack because it highlights a contradiction with defending against fairness attacks: to improve fairness you must boost outliers, but to defend against attacks you must suppress outliers.

- The Krum defence makes some assumptions about the distribution of model updates. We may be able to trick the defence by producing updates that lie within, but on the edge of, clusters of clean updates, so that they are not detected as outliers, but still apply the attack.

I would also be interested in investigating alternative defences to these as an extension. For example, it might be possible to prevent this attack specifically by assigning a random random weighting to the updates of client to make inverting the aggregation function more difficult. Alternatively, we might be able to detect clients that are attacking fairness by considering the change in variance of validation accuracy between classes when the update from the client is passed to the aggregator compared to when it is suppressed.

Completing all of these extensions would be quite difficult, so I plan on selecting two of these to complete once I have finished the core part of my project.

Success criteria

Core criteria:

- Construct a baseline FL model that has high accuracy on each of the three datasets
- Implement a simple backdoor attack on each model
- Implement the fairness attack on each model and achieve similar results to the paper
- Implement each of the three defences against each of the three models and achieve the expected results when tested against the backdoor attack
- Test the effectiveness of the defences against the fairness attack

Possible extension criteria:

- Implement the attack against another aggregation function than FedAvg
- Implement a new attack against the TERM aggregator
- Implement a new attack that can bypass the Krum defence
- Implement a defence that assigns a random weighting to each client
- Implement a defence that detects updates which reduce fairness

Evaluation

I will first need to evaluate the baseline models' accuracies to ensure the attacks are made on a realistic setup. I will perform centralised evaluation by providing an evaluation function to Flwr's built in FedAvg strategy.

To evaluate the attacks and defences, I will record the mean and standard deviation of the model's accuracy on data containing the target attributes and data not containing these attributes. The attack will be successful if the data containing the target attributes have a higher mean or lower standard deviation. A hypothesis test could be used to decide if the difference is statistically significant in cases where the two values are relatively close.

In addition to this testing, I will further investigate how the defences interact with the attack when introduced at different stages. For example, if the model has trained fairly before the attack started, when the defence is added will it be able to recover from the attack faster than if the attack had been implemented from the beginning?

I will ensure my testing is reproducible by keeping track of all hyperparameters and random seeds for each run in a single YAML file that can be stored alongside the metrics.

Starting point

For the paper on the fairness attack I have implemented the attack against a model trained on CIFAR10. The main implementation difficulty of my project will be around implementing the FL models for other datasets and the defences rather than the attack itself. However, the original codebase has been created without extensibility in mind, so I will also reimplement the original attack in a more expandable and efficient way.

The original implementation of the attack simulates running the malicious code locally on the malicious client. However, this is inefficient because each client is allocated the same hardware, so we want to keep the load evenly distributed between clients to maximise our GPU utilisation. In new implementation [sic], the aggregation function would simulate the malicious client directly. This will move the malicious computation out of the client's training function, and will be more expandable because we will be able to easily change the attack by substituting the aggregation function.

The development of my project will be in python, using the Flwr and PyTorch libraries for simulating the federated learning environment. I will use Slurm to schedule jobs on the CamMLSys servers (see Resource Requirements). I already have some understanding of both libraries and Slurm.

Timetable

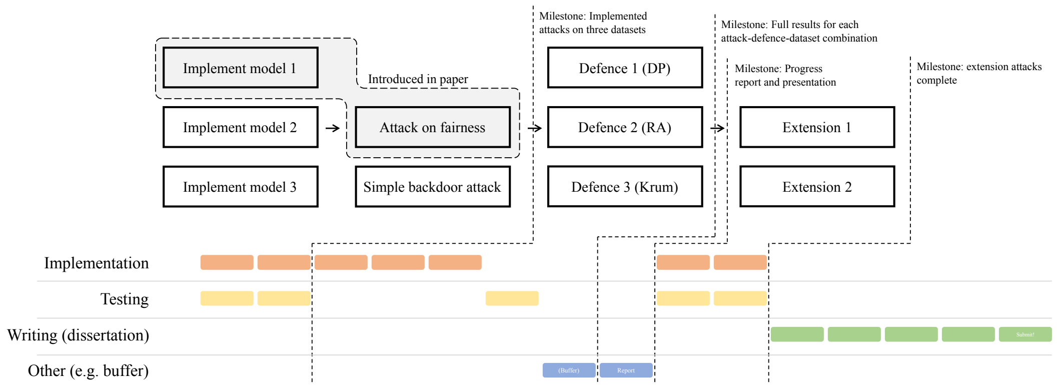

Figure H.1.: A chart showing the timetable, where each column of coloured blocks represents a two-week period.

Michaelmas Term

1. 16/10/23 - 29/10/23
   - Read up on current implementations of FL models for the datasets I will be training on
   - Read further on backdoor attacks in Federated Learning
   - Implement baseline model for two of the three datasets

   Milestone: Implemented a FL classifier for two datasets
2. 30/10/23 - 12/11/23
   - Implement baseline model for the remaining dataset
   - Implement a simple backdoor attack
   - Implement the fairness attack

   Milestone: Implemented attacks on three datasets
3. 13/11/23 - 26/11/23
   - Implement the first backdoor defence

   Milestone: Working implementation of the first defence
4. 27/11/23 - 10/12/23
   - Implement the second backdoor defence

   Milestone: Working implementation of the second defence

Winter Vacation

1. 11/12/23 - 24/12/23
   - Implement the third backdoor defence

   Milestone: Working implementation of the third defence
2. 25/12/23 - 07/01/24
   - Implement any remaining testing infrastructure required to collect full results
   - Perform the testing on each combination of attack, defence, and dataset.

   Milestone: Full results for each attack-defence-dataset combination
3. 08/01/24 - 21/01/24
   - Buffer period

Lent Term

1. 22/01/24 - 04/02/24
   - Prepare progress report and presentation

   Milestone: Progress report and presentation
2. 05/02/24 - 18/02/24
   - Implement the fairness attack on a different aggregator than FedAvg
   - Repeat the testing on this new attack

Milestone: Extension 1 complete

3. 19/02/24 - 03/03/24
   - Implement one of the other proposed extensions
   - Repeat the testing for this third fairness attack

   Milestone: Extension 2 complete

4. 04/03/24 - 17/03/24
   - Begin dissertation: write first two chapters

   Milestone: Introduction & preparation sent for review

Easter Vacation

1. 18/03/24 - 31/03/24
   - Write-up implementation: complete chapter 3 of dissertation

   Milestone: Implementation write-up sent for review

2. 01/04/24 - 14/04/24
   - Finish dissertation: write chapters 4 and 5

   Milestone: Full draft of dissertation sent for review

3. 15/04/24 - 28/04/24
   - Review feedback
   - Make changes to dissertation based on feedback

   Milestone: Final draft ready

Easter Term

1. 29/04/24 - 10/05/24
   - Make final changes to dissertation
   - Submit dissertation

   Milestone: Final dissertation for submission

Resource requirements

I will use my personal laptop (Dell G7 15) as my main device for writing both the codebase and dissertation. I will use the GPUs from the CamMLSys group for model training, which I already have permission to use. I will use GitHub to back up my codebase and Overleaf for writing my dissertation. I will also back up my written work to Google Drive. As mentioned above, I will use the python libraries Flwr and PyTorch. I will require a copy of each of the three datasets I have mentioned, which I have already downloaded.

References

Bagdasaryan, Eugene et al. (2020). “How To Backdoor Federated Learning”. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, pp. 2938–2948. URL: https://proceedings.mlr.press/v108/bagdasaryan20a.html (cit. on p. 55).

Becker, Barry and Ronny Kohavi (1996). *Adult*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20 (cit. on p. 56).

Bhagoji, Arjun Nitin et al. (Sept. 2019). “Analyzing Federated Learning through an Adversarial Lens”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 634–643. URL: https://proceedings.mlr.press/v97/bhagoji19a.html (cit. on p. 55).

Blanchard, Peva et al. (2017). “Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf (cit. on p. 56).

Candidate, The and Filip Svoboda (2023). “Modifying backdoor attacks to compromise fairness in federated learning”. In review (cit. on p. 55).

Fang, Minghong et al. (2020). “Local Model Poisoning Attacks to Byzantine-Robust Federated Learning”. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. SEC’20. USA: USENIX Association. ISBN: 978-1-939133-17-5 (cit. on p. 55).

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images*. Tech. rep. (cit. on p. 56).

Li, Tian et al. (2021). *Tilted Empirical Risk Minimization*. arXiv: 2007.01162 [cs.LG] (cit. on p. 56).

McMahan, H. B. et al. (2016). “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *International Conference on Artificial Intelligence and Statistics*. URL: https://api.semanticscholar.org/CorpusID:14955348 (cit. on p. 55).

Reddi, Sashank et al. (2021). *Adaptive Federated Optimization*. arXiv: 2003.00295 [cs.LG] (cit. on p. 56).

Sun, Ziteng et al. (2019). *Can You Really Backdoor Federated Learning?* arXiv: 1911.07963 [cs.LG] (cit. on p. 56).

Völske, Michael et al. (Sept. 2017). “TL;DR: Mining Reddit to Learn Automatic Summarization”. In: *Proceedings of the Workshop on New Frontiers in Summarization*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 59–63. DOI: 10.18653/v1/W17-4508. URL: https://www.aclweb.org/anthology/W17-4508 (cit. on p. 56).

Wang, Hongyi et al. (2020). “Attack of the Tails: Yes, You Really Can Backdoor Federated Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 16070–16084. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/b8ffa41d4e492f0fad2f13e29e1762eb-Paper.pdf (cit. on p. 55).

Yin, Dong et al. (2021). *Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates*. arXiv: 1803.01498 [cs.LG] (cit. on p. 56).