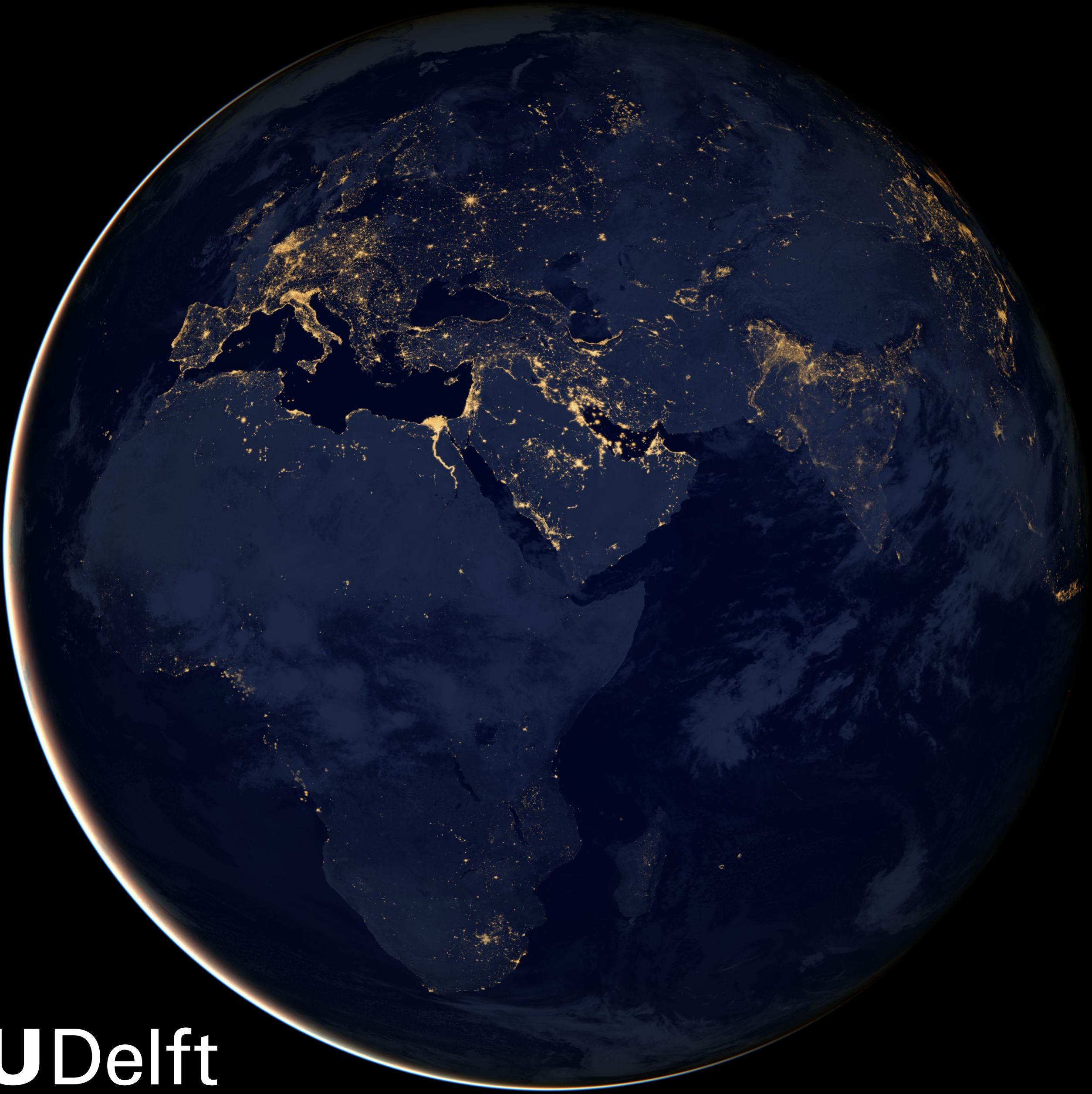
FLVoogd: Robust And Privacy Preserving Federated Learning

IN5000: Final Project

Yuhang Tian



Delft University of Technology

TU Delft

FLVoogd: Robust And Privacy Preserving Federated Learning

by

Yuhang Tian

to obtain the degree of **Master of Science**

in **Embedded Systems**

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

at the Delft University of Technology,

to be defended publicly on Thursday August 25, 2022 at 14:00 PM.

Student number:	5219728	
Project duration:	December, 2021 – August, 2022	
Thesis committee:	Dr. Kaitai Liang	(responsible supervisor)
	Rui Wang	(co-supervisor)
	Prof. Ir. Sicco Verwer	(the chair)
	Prof. Marco Zuniga	

This thesis is confidential and cannot be made public until October 1st, 2022.

Preface

Federated Learning is a private-by-design collaborative learning framework where data holders upload locally trained models to the server for aggregation. Unlike the centralized learning setting, where a server collects substantial users' data to build a commonly used model, data never leaves from local devices under the federated learning framework. Therefore, Federated Learning claims that privacy is preserved.

However, many later works have shown that the original federated learning paradigm with naive aggregation rule cannot provide a robust model to clients under attack, even can still disclose personal information to the server and adversaries.

With more investigators dedicate to this research, many recent works propose more secure aggregation rules and privacy-preserving frameworks. Nevertheless, most designs still have two common disadvantages. The first one is some secure aggregation rules are not adaptive. They can successfully defend one attacking scheme but fails in another. Besides, some defense methods involve too many unintuitive hyperparameters to tune. The second is the designing or the computational cost of secure-and-private aggregation is high for many frameworks. Many secure aggregation rules are incompatible with popular secure-computation protocols leading to extra designs, computations, or communications.

In this work, we propose FLVoogd, an updated federated learning framework in which servers and clients collaboratively eliminate Byzantine attacks while preserving privacy. In particular, servers use automatic Density-based Spatial Clustering of Applications with Noise combined with Secure Multi-party Computation to cluster the benign majority without acquiring sensitive personal information. Meanwhile, clients build dual models and perform test-based distance controlling to adjust their local models toward the global one to achieve personalizing. Our framework is so automatic and adaptive that servers or clients don't need to tune the parameters during the training. In addition, our framework leverages only simple Secure Multi-party Computation operations, including multiplications, additions, and comparison, where costly operations, like division and square root, are not required.

Evaluations are carried out on some conventional datasets from the image classification field. Results show that FLVoogd can effectively reject malicious uploads in most scenarios without adversely influencing the predicting accuracy; meanwhile, it avoids data leakage from the server-side. Of course, there are some limitations, the most of which are the data distribution cannot be highly non-i.i.d. between the peers, and the reconstruction attack for inferring the class representatives cannot be entirely eradicated. However, we believe that FLVoogd can inspire other academics with its efficient and effective architecture and be a critical aspect towards the ultimate robust and confidential Federated Learning.

In hindsight, the entire thesis process was enjoyable. Dr. Kaitai gave me enough freedom to design and implement my ideas. Even though they were sometimes incorrect and I might be trapped in a dilemma, Rui would always help me out in time. We consistently held weekly meetings where I could learn a lot and be inspired. Both were reliable and provided supportable feedback on time, so I was well mentored throughout the whole project. Therefore, I sincerely appreciate their efforts in this unforgettable journey. A particular thanks to Rui, who provided not only excellent comments regularly but also assisted much in polishing the final work and framing it in such a manner that it would appeal to the conference reviewers. I would also like to thank Prof. Sicco and Prof. Marco for participating on my thesis committee and assessing my work.

Yuhang Tian
Delft, The Netherlands
August 2022

Contents

Preface i

List of Figures iv

List of Tables vi

Nomenclature vii

1 Introduction 1
- 1.1 Problem Statement 1
- 1.2 Thesis Object 2
- 1.3 Thesis Outline 2

2 Background Knowledge 3
- 2.1 Federated Learning 3
- 2.2 Differential Privacy 4
- 2.3 Density-based Clustering 6
  - 2.3.1 DBSCAN 7
  - 2.3.2 HDBSCAN 7
- 2.4 Secure-two-party Computation 8
  - 2.4.1 CrypTen 9
  - 2.4.2 SyMPC 10

3 Problem Setup 13
- 3.1 Inference Attack 13
  - 3.1.1 Leaking surface 13
  - 3.1.2 Inference strategy 13
- 3.2 Poisoning Attack 14
  - 3.2.1 Classification of attacks 14
  - 3.2.2 Attacking schemes 15

4 FLVoogd 18
- 4.1 Encrypted FLVoogd Server 18
  - 4.1.1 Assumption 18
  - 4.1.2 Pipeline and algorithm 19
  - 4.1.3 Evaluation 21
- 4.2 Unencrypted FLVoogd Server 22
  - 4.2.1 Assumption 22
  - 4.2.2 Pipeline and algorithm 23
  - 4.2.3 Evaluation 24
- 4.3 FLVoogd Client 24
  - 4.3.1 Assumption 24
  - 4.3.2 Pipeline and algorithm 24
  - 4.3.3 Evaluation 25

5 Experiment 27
- 5.1 Experimental Setup 27
- 5.2 Experimental Results 28
  - 5.2.1 Fending off byzantine attacks 28
  - 5.2.2 Adding and tracking DP noise 30
  - 5.2.3 Non-IID interference 31

6 Related Work **34**
6.1 Poisoning Defense 34
6.2 Privacy Preserving 35
6.3 Other 36

7 Conclusion **37**

Conference Version **38**

References **60**

List of Figures

2.1 Comparison between centralized and federated learning . . . 3
- 2.1a Centralized learning . . . 3
- 2.1b Federated learning . . . 3

2.2 A toy-example of HDBSCAN . . . 8
- 2.2a 2-dimension samples . . . 8
- 2.2b Linkage tree . . . 8
- 2.2c Condensed tree . . . 8

2.3 An example of FSS for comparison . . . 10
- 2.3a deviation due to “less than” . . . 10
- 2.3b deviation due to “greater than” . . . 10

3.1 Inference attacks . . . 14
- 3.1a Trusted boundaries in FL . . . 14
- 3.1b GAN reconstruction . . . 14

3.2 Some examples of the trigger-backdoor attack . . . 17
- 3.2a Digits with top-left rectangle . . . 17
- 3.2b Cars in green . . . 17
- 3.2c Cars with stripes . . . 17
- 3.2d Cars in front of stripe wall . . . 17

3.3 Samples of the edge-case . . . 17
- 3.3a Edge cases for MNIST . . . 17
- 3.3b Edge cases for CIFAR-10 . . . 17

4.1 Encrypted FLVoogd server framework . . . 18

4.2 Unencrypted FLVoogd server framework . . . 23

4.3 FLVoogd client framework . . . 24

5.1 Byzantine attacks on MNIST . . . 29
- 5.1a A1: PMR=19/40 . . . 29
- 5.1b A2: PMR=19/40 . . . 29
- 5.1c A3: PMR=19/40 . . . 29
- 5.1d A4: PMR=19/40, PDR=1.0 . . . 29
- 5.1e A5: PMR=19/40, PDR=0.2 . . . 29
- 5.1f A6: PMR=19/40, PDR=0.33 . . . 29

5.2 Byzantine attacks on CIFAR-10 . . . 30
- 5.2a A1: PMR=19/40 . . . 30
- 5.2b A2: PMR=19/40 . . . 30
- 5.2c A3: PMR=19/40 . . . 30
- 5.2d A4: PMR=19/40, PDR=1.0 . . . 30
- 5.2e A5: PMR=19/40, PDR=0.33 . . . 30
- 5.2f A6: PMR=19/40, PDR=0.33 . . . 30

5.3 Byzantine attacks on EMNIST . . . 31
- 5.3a A1: PMR=19/40 . . . 31
- 5.3b A2: PMR=19/40 . . . 31
- 5.3c A3: PMR=19/40 . . . 31
- 5.3d A4: PMR=19/40, PDR=1.0 . . . 31
- 5.3e A5: PMR=19/40, PDR=0.2 . . . 31
- 5.3f A6: PMR=19/40, PDR=0.33 . . . 31

5.4 DP’s effect on MNIST . . . 32

5.4a $\sigma = 1$, $q = 20/100$ 32
5.4b $\sigma = 2$, $q = 40/100$ 32
5.4c $\sigma = 3$, $q = 60/100$ 32
5.4d $\sigma = 4$, $q = 80/100$ 32
5.4e $\sigma = 5$, $q = 100/100$ 32
5.4f ϵ recording 32
5.5 Non-IID effect on MNIST 32
5.5a A1: PMR=9/20 32
5.5b A2: PMR=9/20 32
5.5c A3: PMR=9/20 32
5.5d A4: PMR=9/20, PDR=1.0 32
5.5e A5: PMR=9/20, PDR=0.2 32
5.5f A6: PMR=9/20 32
5.6 Pretraining and a lower PDR help the defense 33
5.6a $Deg_{nIID} = 0.3$: lowering PDR to 6/20=30% 33
5.6b $Deg_{nIID} = 0.3$: pretraining the model to accuracy around 25% 33
5.6c $Deg_{nIID} = 0.5$: pretraining the model to accuracy around 25% 33

List of Tables

2.1 CrypTen operations 9

3.1 Summary of poisoning attacks 15

4.1 SMPC operations in encrypted FLVoogd 21

5.1 $2\times$CNN architectures 27
5.2 Attack settings 28

List of Algorithms

1 FL with the naive aggregation rule 4
2 FSS Comparison for $x \leq \alpha$ 12
3 Encrypted FLVoogd - server 22
4 Auto DBSCAN 23
5 FLVoogd - client (1) 26
6 FLVoogd - client (2) 26

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
BA	Backdoor Accuracy
CNNs	Convolutional Neural Networks
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DP	Differential Privacy
DPF	Distributed Point Function
FTL	Federated Transfer Learning
FL	Federated Learning
FSS	Function Secret Sharing
GANs	Generative Adversary Networks
GDPR	General Data Protection Regulation
HDBSCAN	Hierarchical Density Based Spatial Clustering of Applications with Noise
MA	Matin Task Accuracy
ML	Machine Learning
NLP	Natural Language Processing
PDR	Poisoned Data Rate
PIR	Private Information Retrieval
PMR	Poisoned Model Rate
PRZS	Pseudo Random Zero Sharing
RDP	Rényi Differential Privacy
S2PC(STPC)	Secure 2(Two) Party Computation
SMPC	Secure Multi Party Computation
TSS	Total Sum of Square
TTP	Trusted Third Party

Symbols

Symbol	Definition
acc_{ref}	predicting accuracy of the global model based on the local validation set
acc_{local}	predicting accuracy of local models based on the local validation set
acc_{thres}	threshold accuracy of local models learning from the global
b	clipping indicator $\{0, 1\}$
b_{avg}	averaged of clipping indicators
b_{noise}	compensating DP noise for b_{avg}
$\mathcal{B}$	local mini-batch
c	clipping boundary
$\mathcal{C}$	a set of clients
$\mathcal{D}_{train}$	local training set
$\mathcal{D}_{eval}$	local validation set
e	e^{th} local training epoch
E	total number of local training epochs
F	objective function of neural networks
g	gradients of neural networks
m	number of remaining clients after filtering
M_{cos}	cosine distance matrix
M_{tss}	total-sum-of-square distance matrix
M_{adjcos}	adjusted-cosine distance matrix
n	number of selected clients per round
N	total number of clients
q	ratio of the round chosen clients
t	t^{th} FL training round
T	total number of FL training rounds
v	last-layer model's update of weights
w	model's update of weights
w_{global}	average of individual updates
$\tilde{w}_{global}$	w_{global} added by DP noise
w_{noise}	compensating DP noise for $\tilde{w}_{global}$
W_{global}	global model weights
W_{local}	local model weights
δ	failure probability of differential privacy
ϵ	upper bound of max-divergence for differential privacy
η_c	learning rate of adaptive clipping
η_{ditto}	learning rate of adaptive Ditto
η_{global}	learning rate of the global model
η_{local}	learning rate of local models
γ	expected ratio of clients who should clip updates
$\hat{\gamma}$	ratio of clients who did not clip updates
λ_{ditto}	coefficient to control the distance between the global and local model
λ_{max}	upper boundary of λ_{ditto}
λ_{min}	lower boundary of λ_{ditto}
σ	DP noise multiplier/strength of model aggregation
σ_b	DP noise multiplier/strength of averaged indicator
$\overline{*}$	unit vector of $*$
$\lvert * \rvert$	length of $*$
$\lvert\lvert * \rvert\rvert$	$l2$-norm of $*$
$*^{(t)}$	t^{th} round of $*$
$\nabla *$	partial derivative of $*$

1 Introduction

Machine Learning (ML) is the main subset of Artificial Intelligence (AI) techniques. To improve the performance of ML, data extraction is usually required before the training. However, programmers do not always know which features should be selected unless they have verified them on an established model [12]. In order to overcome this challenge, computer scientists invent Artificial Neural Networks (ANNs), which usually do not necessarily need feature extraction; still, its main weakness is the requirement of an adequate and large number of training data. Thanks to the trend of Big Data, tremendous data collection is relatively undemanding nowadays [114]. However, the traditional training mechanism adopts the centralized learning setting where a server collects the data from its clients/users to build a common model for predictions. Consequently, this mechanism cannot sufficiently protect the privacy of users' data from the server-side and even leaks personal information to other unauthorized third parties. For instance, users' data collected by Facebook were revealed and used for political advertising by another party without consent [112]. Another paradigm was that a standard anonymous dataset of Netflix could be completely de-anonymized [79] which showed that a centralized training framework, even with careful anonymity, could hardly guarantee personal privacy. Furthermore, with the growing privacy concerns from customers and rigorous policies promulgated and executed by European Union's General Data Protection Regulation (GDPR) [118], people tend to be unwilling to share their private data. These factors form a phenomenon so-called *data isolation*. To tackle this awkward problem, Konečnỳ et al, in 2015 [54] and 2016 [55], introduced a new collaborative learning framework latterly generally named as Federated Learning (FL). In the past half-decade, FL had been widely researched and applied in many fields [60] such as image recognition [42], natural language processing (NLP) [20, 69], financial system [119], medical care [44] and far more.

1.1. Problem Statement

Rather than gathering the raw personal data by a server for specific training purposes, FL requires the model parameters that clients train independently based on their own data and devices. In FL paradigm frameworks, such as FedAvg [70], the server iteratively aggregates local models trained by individuals and sends the global one back to clients for their local updates to achieve collaborative training. Thus, no actual data is sent to the server. However, this cutting-edge framework still faces two main challenges - privacy and security. Firstly, the original FL setting cannot get rid of the disclosure of clients' information and even enlarge the attacking surface. The model provided by a client can be regarded as a high-level generalization of that person's data. Taking Convolutional Neural Networks (CNNs) as an example, convolutional layers and pooling layers constitute a network for feature extraction [82], and weights in the last fully connected layer are directly relevant to the frequency of classes in the personal dataset [91]. Not only can the server be an adversary to infer the information from the models sent by clients in this scenario, but also every participant can perform the inference attack [72] to the global model constructed by each individual. In this situation, some research employs Secure-multi-party Computation (SMPC) to encrypt the uploads, such as [85]. However, the design or operations cost is high, especially when dealing with the division and the square root. Secondly, since participants train their local models without constraints, the model training can betray the regulation. If the server adopts the naive aggregation rule, the robustness of the global model cannot be guaranteed because adversaries can substitute the data with the poisoned one [109] or even directly change the uploads into meaningless random numbers such as zero-mean Gaussian noise [116]. Some research, like [61, 91], builds some practical defensive frameworks to eliminate the attacking consequence. However, those frameworks include too unintuitive hyper-parameters to tune for different situations. Besides those two significant challenges, some other aspects currently have also been considered by researchers, such as communication efficiency [2, 63], fairness [122], non-IID [125] data, etc., but they are not comprehensively included in our research. It should be motioned that non-IID data would be used for our experiments, but mainly for testing and presenting the robustness of our secure aggregation rule, not for the convergence analysis.

1.2. Thesis Object

To improve the efficiency and save expensive operations, we develop an updated framework that combines SMPC [53, 87], Density-based Spatial Clustering of Applications with Noise (DBSCAN) [31], differential privacy [111], and personalized local model [61] to eliminate the malicious uploaded parameters without revealing any sensitive information and guarantee (ϵ, δ)-DP after the aggregation. Compared with the past research, our framework 1) filters the abnormal uploads without knowing their sensitive information; 2) performs the training process adaptively, requiring no parameter tuning; 3) uses SMPC operations efficiently supported by most protocols. We leverage conventional image classification datasets to evaluate the framework. The results show that the filter can reject the byzantine attacks under most situations without degrading the model performance. The trade-off between predicting accuracy and DP strength can be customized for different scenarios.

1.3. Thesis Outline

The thesis is organized as the following:

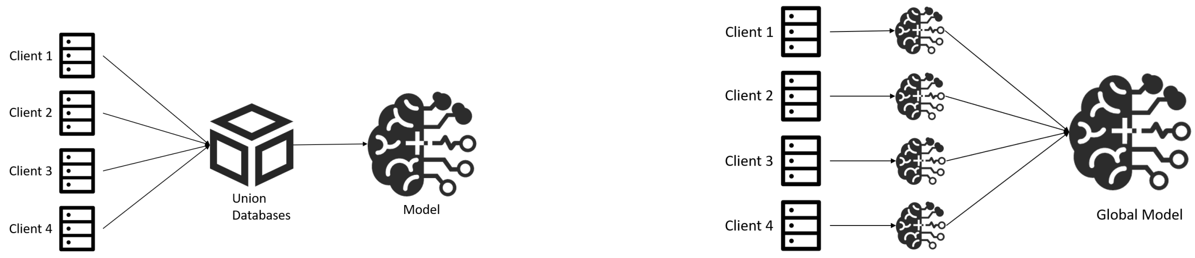
- **Chapter 2** will detail the background knowledge to FL, especially FedAvg, which will be the base stone where all the defensive schemes will latterly be added. In addition, it will sequentially demonstrate the background knowledge of Differential Privacy, (H)DBSCAN, and Secure-two-party Computation, which will be exploited for security and privacy guarantees.
- **Chapter 3** will present the adversary strategies which malicious clients/servers use to perform the inference and poisoning attack, respectively. Furthermore, it will exhibit that those attacks are realistic and powerful to threaten the current FL frameworks.
- **Chapter 4** will provide our FL framework and the design ideas behind it. It will show that our design effectively and efficiently resists both malicious and inference attacks.
- **Chapter 5** will show the setup of our experiments, including the assumptions based on different scenarios and settings of different hyper-parameters. Subsequently, the outcomes of the investigation will be visualized and analyzed.
- **Chapter 6** will mainly provide the literature study of other designs based on robustness and privacy-preserving FL. After that, some other interesting fields relative to FL will also be mentioned.
- **Chapter 7** will summarize the whole project and offer suggestions and future research directions.

2 Background Knowledge

This chapter will provide the fundamentals to assist readers in understanding our Federated Learning framework. In terms of different branches of Federated Learning, we will choose the horizontal one as our research object. Other two are vertical and transfer federated learning. Then, two significant challenges on this stem inevitably occur - 1) how can we safeguard the horizontal Federated learning in case of being poisoned, and 2) how can we preserve the personal information from the model leakage? We will study three state-of-art techniques - Differential Privacy, Density-based Clustering, and Secure Multi-party Computation, and try to build defensive mechanisms based on these theorems to tackle the above issues. The following will sequentially provide the background for Federated Learning and the fundamentals of defense strategies.

2.1. Federated Learning

Federated Learning (FL) differs from conventional training in which clients are required to upload their factual data to the server. Figure 2.1 visualizes this main difference. In a centralized setting (figure 2.1a), clients are required to upload their data to a server that will receive all uploads to obtain a larger joint dataset and train a model relying on this centralized dataset. In contrast, the FL server (figure 2.1b) will only collect independently trained models from clients; thereby, it can never access factual data that does not depart from local devices at all.



(a) Centralized learning **(b)** Federated learning

Figure 2.1: Comparison between centralized and federated learning

FL can be mainly divided into three categories - horizontal FL, vertical FL, and Federated Transfer Learning (FTL), and more details can be found in [118]. We think the latter two frameworks are more applicable for B2B scenarios. In contrast, as the former is more suitable for B2C scenarios and caters to our interest, we will select the horizontal FL framework for our study. We extend the definition of FL from [118] for horizontal FL as: N data holders hold datasets $\{D_1, ..., D_N\}$, and traditionally they unionize them $D = D_1 \cup ... \cup D_N$ to train a centralized model $\mathcal{M}_{central}$ to achieve accuracy $Acc_{central}$, whereas now they train their own models based on D_is and upload Up_is to a server who unionized the uploads based on a specific operation $Up_1 \circ ... \circ Up_N$ to obtain a model $\mathcal{M}_{FL}$ with accuracy Acc_{FL}, and two accuracy follow eq.(2.1) where δ is a non-negative real number so-called δ-accuracy loss.

$$|Acc_{central} - Acc_{FL}| < \delta \tag{2.1}$$

The distribution of datasets can be either IID or Non-IID. The uploads can be gradients, weights [70], weights' differences [61] or signs [67, 115]. The operation $\circ$ can be average, weighted-average [50] or other criteria such as Krum function [13].

There are two typical model-averaging frameworks - FedSGD and FedAvg [70]. FedSGD is early developed from SGD [40] and latterly studied by [70] where it is compared with FedAvg. The main difference is that each client uploads its local gradients for every mini-batch iteration in FedSGD, but each client uploads its weights/weight-differences after several mini-batch iterations in FedAvg. According to the experiments in [70], the basic structure of FedAvg will be exploited in our research since it is more customizable and requires fewer communication rounds which also represents that the privacy loss can be reduced. Algorithm 1 shows the fundamental structure of our framework without any security or privacy-preserving relevant modules. There are two differences compared to FedAvg. Firstly, it uses average instead of the weighted average for the security concern. Secondly, clients upload the differences in parameters rather than the raw parameters. According to the structure, it is not a secure aggregation because clients can use any data for their training and freely modify their uploads. In addition, since individual uploads are purely plain-text from the server-side and the global model update is simply the average of uploads for receivers in this communication round, it is not privacy-preserving.

Algorithm 1 FL with the naive aggregation rule

```
Server:
 1: Initialization: $w_0$
 2: for round $t := 1, 2, ..., T$ do
 3:     subset$_t$ ← subsample(clientSet, $sampling\_ratio = q$) and $|$subset$_t| = K$
 4:     for client with index $k \in$ subset$_t$ do
 5:         $w^k_{diff} \leftarrow$ Client$_k(w_t)$
 6:     $w_{t+1} \leftarrow \beta w_t + (1-\beta)\frac{1}{K}\sum_{t=1}^{K} w^k_{diff}$        ▷ global model update

Client:                                                  ▷ clients locally and parallel do
 7: Update: $w \leftarrow w_t$                                ▷ local model update
 8: for local epoch $epoch := 1, 2, ..., E$ do
 9:     batches ← dataLoader($D_k, batch\_size = B$)
10:     for batch $b \in$ batches do
11:         $w \leftarrow w - \eta \nabla L(w; b)$
12: return $w^k_{diff} \leftarrow w - w_t$
```

2.2. Differential Privacy

Either the server or participants can analyze the uploaded parameters to infer the membership, such as the information on whether a specific client has contributed to the model. This inference is detrimental in specific scenarios, such as the patient's information in a hospital. Differential Privacy (DP) as a privacy-preserving technique has already been studied and applied in centralized-training DL [1] where DP can bound and quantify the privacy loss. Normally, DP introduces randomness to the response [29] which we want to protect so as to produce indistinguishable outcomes. FL can also incorporate this mechanism. In particular, whether a specific client participates in the training is hidden after the DP process - the accuracy is not significantly changed - and this participant cannot be deduced with any arbitrary side information [5].

$(\epsilon, \delta) - DP$ also called approximate differential privacy is formally defined as eq.(2.2) where M is a randomized mechanism mapping domain D to range R: $D \to R$, and $d, d' \in D$ are two neighboring databases: $||d - d'||_1 \leq 1$. It quantifies that with the probability $1 - \delta$, the maximum privacy loss

$\ln \frac{\Pr[M(d)\in S]}{\Pr[M(d')\in S]}$ is no larger than ϵ. The privacy loss here uses Max Divergence which can also be regarded as the maximum difference/distance between the outputs. In contrast, it has the probability δ that the mechanism fails to produce 'close' enough outcomes.

$$\Pr[M(d) \in S] \leq \exp(\epsilon) \Pr[M(d') \in S] + \delta \tag{2.2}$$

Gaussian noise is commonly used as a randomized mechanism for the centralized training setting [5], as the l_2 based sensitivity measurement [29] and the composition theorems of the Gaussian mechanism [1] provide less privacy loss for the scenario where high dimensional outputs are iteratively queried. As known, the dimension of clients' uploads is usually high in FL, e.g., 269,722 parameters for ResNet-20 [46], and the number of training epochs is always hundreds or even thousands [91] to approach the convergence. Therefore, the Gaussian noise mechanism will be embraced in our framework. The formal definition of the Gaussian mechanism is shown as eq.(2.3). The noise adding can be viewed as a sacrifice to the accuracy to achieve indistinguishable results; thus, considering the amount of added noise is a trade-off between privacy and accuracy.

$$M(d) = f(d) + \mathcal{N}(0, \sigma^2 \cdot \Delta_2^2(f)) \tag{2.3}$$

The standard deviation of the added Gaussian noise is $\sigma\Delta_2(f)$ where $\Delta_2 = \max_{||d-d'||_1=1} ||f(d)-f(d')||_2$ is the maximum l_2-sensitivity. If $\sigma \geq \frac{\sqrt{2\ln(1.25/\delta)}}{\epsilon}$ and $\epsilon \in (0,1)$, $M(d)$ is $(\epsilon,\delta)-DP$ by theorem 3.22 [29]. However, due to the multiple rounds of training, ϵ is hard to be limited in $(0,1)$; as a result, ϵ is normally abused and extended to values larger than 1 in the FL setting [3, 5, 38, 80]. A larger ϵ will influence the composition theorem, because the higher-order term cannot be ignored when ϵ grows large which will be detailed later.

As mentioned, the FL training process usually takes hundreds or thousands of rounds to converge or achieve acceptable accuracy. Intuitively, the adversary can obtain more information through quires during multiple rounds to infer whether a client exists in training. In fact, ϵ will correspondingly grow larger if someone can query multiple times. There are many quantifying methods for this privacy accountant [71] with multi-time queries. Since the DP's output can be regarded as a random variable whose upper bound is ϵ, the simplest accountant method is naively summing the upper bound of T rounds $(\epsilon,\delta) - DP$: $\epsilon' = \epsilon_1 + \epsilon_2 + ... + \epsilon_T$. If ϵ is set identical for each round as FL normally does, $\epsilon' = T\epsilon$. However, this method is obviously not rigorous because the upper bound of the sum of random variables should be much smaller than direct adding up. If the range and expectation of random variables are considered, a much tighter bound can be found. In [30], Dwork et al studied the upper bound of the conditional expectation and used *Azuma's Inequality* to derive a tighter bound and this composition is so-called advanced composition shown in eq.(2.4).

$$\epsilon' = \sqrt{2T\ln(1/\delta^*)}\epsilon + T\epsilon(e^\epsilon - 1) \qquad\qquad \delta' = T\delta + \delta^* \tag{2.4}$$

If ϵ is small, the second term of ϵ' approximates zero which can be ignored, thereafter the T rounds $(\epsilon,\delta)-DP$ can be approximate to $(\epsilon\cdot\sqrt{2T\ln(1/\delta)}, T\delta)-DP$. As a consequence, advanced composition brings the linear increase respective to T down to $\sqrt{T}$. In addition, Abadi et al, noticing the advanced composition theorem only includes the first moment - expectation, proposed Moments accountant in 2016 which further takes higher-order moments - up to 32^{nd} - into consideration [1]. It provides a rational tighter bound especially for DL. If using moments accountant for T rounds $(\epsilon,\delta) - DP$, it becomes $(O(q\epsilon\sqrt{T}),\delta)-DP$ where q is the subsampling rate and will be given more details in the next paragraph. Another commonly used accountant method is Rényi Differential Privacy (RDP) provided by Mironov who replaced the Max divergence for measuring privacy loss by Rényi divergence shown in eq.(2.5) and then received $(\alpha,\epsilon) - RDP$ where α is the order [74]. When $\alpha \to \infty$, Rényi divergence is exactly Max divergence, thus the original privacy loss measurement is a special case for the RDP accountant.

$$D_\alpha(P||Q) = \frac{1}{\alpha - 1} \log E_{x\backsim Q}(\frac{P(x)}{Q(x)})^\alpha \tag{2.5}$$

In his research, Mironov provided a transformation between $(\alpha, \epsilon) - RDP$ and $(\epsilon, \delta) - DP$ as eq.(2.6) shows. It can be noticed from the equation that ϵ can save $\frac{\ln 1/\delta}{\alpha - 1}$ if a smaller α can be found. In other words, the privacy budget ϵ can be further shrunk by RDP.

$$(\alpha, \epsilon) - RDP \backsim (\epsilon + \frac{\ln(1/\delta)}{\alpha - 1}, \delta) - DP \tag{2.6}$$

Similarly, RDP was researched and applied in DL with subsampling by Wang et al. in 2019 [111]. Our research will adopt these two prevalent methods, Moments accountant and RDP, for tracing the privacy budget ϵ. No matter which one of the methods is chosen, it will not change the strength of DP. For instance, we set a fixed δ to 10^{-6} and the standard deviation of Gaussian noise $\sigma \cdot \Delta_2(f)$ to $1 \cdot \Delta_2(f)$, after several training rounds, $\epsilon = 20$ if using simple composition whereas $\epsilon = 12$ if using advanced composition. It does not represent advanced composition provides a stronger DP guarantee because the amount of the added noise is identical for both accountant settings. They only differ in how they “view” the privacy loss, so “the world is the same world but viewers with different angles”. The main reason we chose those two accountant methods is that they are prevalent and exhaustively researched, but some other methods provide even lower bound, such as Bayesian DP [104].

So far, we only provide the definition of DP and some methods for tracing DP but do not provide any methods to enhance DP or improve its usability. The following paragraphs will detail subsampling that was mentioned in previous paragraphs. Besides, it will introduce one crucial property of DP, post-processing resistance. FL always involves a larger number of clients, but due to the limitations of connection and communication, e.g., bandwidth and dropping-off [96], in each iteration, the server will ordinarily only choose a part of clients for the communication so-called random sub-sampling [38]. Meanwhile, this random subsampling strategy also enhances the DP process [86]. If the subsampling ratio is $q < 1$, then $(\epsilon, \delta) - DP$ becomes $(O(q\epsilon), q\delta) - DP$ shown in eq.(2.7) [111]. Thanks to this sub-sampling, whether a specific client participates in this training round is probabilistic, and whether an adversary participates in this training round who tries to receive the averaged model, is also probabilistic. Accordingly, even without DP Gaussian noise, subsampling itself introduces the randomness to amplify the DP process. Thus, less noise can be added if the server takes advantage of random sub-sampling.

$$\epsilon' = \log(1 + q \cdot (e^{\epsilon} - 1)) \qquad \delta' = q \cdot \delta \tag{2.7}$$

In terms of post-processing, it guarantees the $(\epsilon, \delta) - DP$ noisy outcome cannot be enervated. The formal definition is given by proposition 2.1 in [29]: If M is a $(\epsilon, \delta) - DP$ randomized algorithm and f is an arbitrary mapping, $M \circ f$ is still $(\epsilon, \delta) - DP$. Therefore, using side information for post-processing to improve the usability of the result without increasing the privacy loss is possible. For instance, based on our prior knowledge, we can assume that the quired results should be non-zero when counting the number of people, so we can regard the output as zero when it outputs a negative number because of the added noise [83].

In conclusion, Moments accountant[1] and RDP[2] will be selected for tracking the privacy loss; meanwhile, random sub-sampling and post-processing will be exploited by our research to improve usability.

2.3. Density-based Clustering

To separate benign and malicious clients from uploads, algorithms like unsupervised ML are suitable, such as clustering. However, unlike conventional clustering problems, the number of clusters/centroids in FL is unknown in advance. There are two types of clusters - the benign cluster and the malicious cluster. Nonetheless, because the malicious uploads can be capriciously far away from the benign cluster, algorithms such as K-mean [117] where the number of clusters is a hyper-parameter is impractical. To overcome this barrier, density-based clustering algorithms such as mean-shift clustering [26], density-based spatial clustering of applications with noise (DBSCAN) [31] and density-based clustering based on hierarchical density estimates (HDBSCAN) [18] are embraced for classifying benign and

[1] https://github.com/SAP-samples/machine-learning-diff-private-federated-learning

[2] https://github.com/yuxiangw/autodp

malicious uploads. After taking efficiency and stability into consideration [84, 85, 91], HDBSCAN will be used where S2PC is not applicable, and DBSCAN will be used in the case where S2PC is enabled. Details of DBSCAN and HDBSCAN will be given in the following paragraphs.

2.3.1. DBSCAN

DBSCAN is initially designed for clustering and distinguishing the noise from the high dimensional database depending on the variance of density [31]. A non-negligible quantity of samples should be in the cluster if a cluster is formed. In contradiction, the cluster can hardly be formed in areas where samples are located sparsely. These "depopulated zones" can be used as gaps to separate the different clusters with different classes and to sift out noisy samples. We will consistently follow some of the concepts and symbols used in [31]. $N_{Eps}(p)$ represents neighbors of a point p within a range with radius Eps (Eps is a preset hyper-parameter). A point p is a $corepoint$, if $|N_{Eps}| \geq MinPts$ ($MinPts$ is a preset hyper-parameter). In addition, a $corepoint$ is the centroid of a cluster, so in other words, a cluster is only formed when its centroid is a $corepoint$. A point p is a $borderpoint$, if its neighbours contain at least one $corepoint$. It should be noted that a point can be a $borderpoint$ for different clusters, but it will be only assigned to a unique cluster eventually, and it depends on which cluster it assigns the point to first. If a point is neither a $corepoint$ nor $borderpoint$, it will be classified as noise. The whole clustering process can be divided into five main steps:

Step 1. **Initialization**: Label all points as unsigned and initialize an empty set.

Step 2. **Core Point Discovering**: Retrieve every point from the database and verify if it is a $corepoint$ or not. If true, it will be labeled as a $corepoint$. A temporary cluster is formed and added into set.

Step 3. **Border Point Discovering**: Retrieve the rest unsigned points and verify if it is a $borderpoint$ or not. If true, it will be labeled as a $borderpoint$.

Step 4. **Cluster Merging**: Retrieve every temporary cluster from set and repeatedly verify every point in this temporary cluster except the original centroid if the point is $corepoint$ or not. If true, the temporary cluster will be linked with another temporary cluster by this $corepoint$, and points from the newly added cluster will be appended to the verification process.

Step 5. **Noise Discovering**: Label the unsigned points as noise.

According to the process, there are mainly two hyper-parameters that need to be determined initially - Eps and $MinPts$. Both together define how conservative the clustering will be. To be specific, the user needs to define how dense a region is to be regarded as a cluster. There are many later researches for these two parameters tuning [51, 58, 101]. Nevertheless, these two parameters are not easily and appropriately set if the foreknowledge of data distribution is lacking [68].

2.3.2. HDBSCAN

HDBSCAN is built on DBSCAN and additionally uses a hierarchical structure to automate the extraction of clusters. It builds a linkage tree and then auto-estimates the density of each cluster to determine whether two clusters merge or not. Referencing [68], the whole process can be divided into five steps:

Step 1. **Transform the space**: Rather than directly measuring the distance between two samples, it introduces *core distance* $d_k(a, b)$ (eq.(2.8)) to measure the distance and names it *mutual reachability distance* to distinguish it from the true distance $d(a, b)$ between two samples. $core_k(a)$ is the distance between point a and its k^{th} nearest neighbors. The transformation is for amplifying the difference between dense region and sparse region so as to build a robust single linkage tree. For instance, if a point p locates in a dense area where it can find k nearest neighbors easily, even these k neighbors are pushed further ($d_k = core_k(p) \geq d(p, q),\ q \in kNeighbors$), the distances between p and its neighbors are still comparably close; in contrast, if a point p locates in a desolate area where it searches a broad region to find k nearest neighbors, because these k neighbors are pushed further, p looks more "lonely". In HDBSCAN library, k is called $min\_samples$.

$$d_k(a, b) = \max\{core_k(a), core_k(b), d(a, b)\} \tag{2.8}$$

Step 2. **Build the minimum spanning tree**: *Prim's algorithm* - the shortest edge is selected every time - is used to find and connect edges for building the minimum spanning tree.

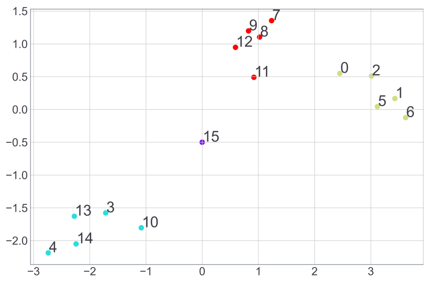
Step 3. **Build the cluster hierarchy**: The single linkage - the distance between two groups, which is measured by the closest two points respectively in each group - is used to build the hierarchy.

Step 4. **Condense the cluster tree**: It is the core step of HDBSCAN because the clusters are picked out during the condensation. It requires a hyper-parameter $min\_cluster\_size$ as a threshold to define a cluster, and the minimum threshold is 2. It travels the hierarchy from the top to down and disconnects the point while the distance is decreasing. Accordingly, the process can also be viewed as the “points falling out of a cluster”. A point falls out merely because of two reasons: 1) the distance is too low to preserve the point so that it actively leaves the cluster; 2) the cluster is split into two smaller clusters so that it passively leaves the old cluster and enters a new one. The reciprocal of distance $\lambda_p = \frac{1}{distance}$ when the point falls out from a cluster will be recorded so that a point may have multiple λ_p corresponding to different clusters. When a cluster is split into two new clusters, the reciprocal of current distance is λ_{death} for the death cluster and λ_{birth} for the born clusters, so every cluster has a pair of λ_{birth} and λ_{death} and the parent cluster's λ_{death} is the λ_{birth} for its children clusters. The updated tree will only reveal the information for the second reason; thus, it is the reason why it is called a condensed cluster tree.

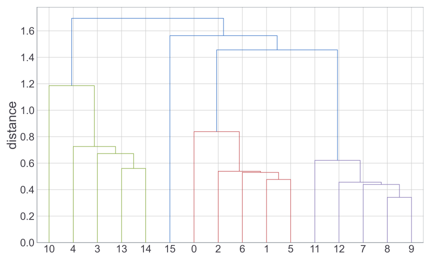
Step 5. **Extract the clusters**: It initially sets all clusters at leaf nodes to be selected ones. Then, it travels from the bottom to the top and compares the stability defined as eq.(2.9) between children nodes and parent nodes. The stability of the parent node will be replaced by the sum of children nodes' if the stability of children nodes is larger; otherwise, the parent node will be selected as a cluster, and all its descendants will not be selected. In addition, points that do not belong to any cluster will be labeled as noise.

$$stability = \sum_{p \in cluster} (\lambda_p - \lambda_{birth}) \tag{2.9}$$

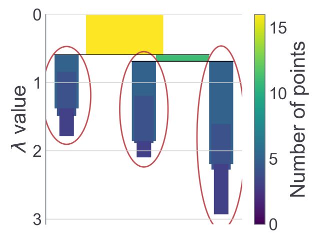
Correspondingly, $min\_samples$ and $min\_cluster\_size$ are two primary hyper-parameter for HDBSCAN. $min\_samples$ influences the space transformation and a larger value can enlarge the density variance. However, if the original distance space has already obviously revealed the relations of samples, $min\_samples$ can be set to 1 [84]. $min\_cluster\_size$ influences the depth of a condensed tree, so the smaller it is, the more conservative the clustering will be. A toy example is shown in figure 2.2 in which there are three clusters with one additional noise sample. Figure 2.2b and figure 2.2c show the hierarchy and condensed hierarchy respectively. ($min\_samples = 2$ and $min\_cluster\_size = 2$)



(a) 2-dimension samples



(b) Linkage tree



(c) Condensed tree

Figure 2.2: A toy-example of HDBSCAN

2.4. Secure-two-party Computation

As mentioned, uploading weights instead of the raw data to the server is not privacy-preserving. As is shown in [72, 124], model parameters can disclose some of the information about individual data. For example, adversaries can use Generative Adversary Networks (GANs) [49] to reconstruct the class representatives from the aggregated parameters. This powerful reconstruction is more harmful if it happens on the server-side because the server can steal the class representatives from each individual uploading. To avoid revealing the uploads to the server, Secure Multi-party Computation can be used for private aggregation, and the result will only be revealed eventually. Following the structures in [84, 85, 91], we will use Secure 2-party Computation (S2PC), a sub-domain of SMPC, to guarantee that the individual upload will not be plain-text to the server. Under the S2PC setting, each client will not directly send the model parameters to the server but separate the upload into two parts and share one with the server for aggregation and another with the external server. As both servers hold merely one piece of the secret, the secret cannot be known if they do not collude. Based on the secret sharing scheme, each server can do arithmetic operations relying on its own share and through some communication. To achieve this target, two libraries CrypTen[3] [53] and SyMPC[4] [87] derived from PySyft [127] are used for the experiment. Both of them use secret sharing but with different protocols to achieve S2PC. CrypTen is currently designed only for semi-honest parties, while SyMPC can tolerate minor malicious parties.

2.4.1. CrypTen

CrypTen integrating with Pytorch API [90] is developed by researchers from Facebook, and SMPC of CrypTen is based on pseudorandom zero sharing (PRZS) [23] with trusted third party (TTP) [53], where PRZS generates $|P|$ randomnesses which sum is 0 and they will mask the secret x into $|P|$ shares, so only the sum of all shares reveal the secret while TTP comes up with Beaver triples [8] for multi-party computation such as the multiplication of shares. In [53], researchers claimed that they would remove the reliance of TTP using the homomorphism of Paillier [88] or the oblivious transfer [10] based on Mascot [52] for the future plan. The library is originally designed for the whole neural network process, and it supports operations more than FL needs. This work will only present the operations that will be used in our FL setting, and other details can be obtained from their original paper. Table 2.1 gives the partial operations of CrypTen.

Table 2.1: CrypTen operations

Operation	Description
Arithmetic sharing	The secret value $x \in \mathbb{Z}/Q\mathbb{Z}$ where $\mathbb{Z}/Q\mathbb{Z}$ is a Q elements ring is shared with $p \in P$ parties by PRZS, so each party holds a share $[x] = \{[x]_p\}_{p \in P}$ and x can be reconstructed by $x \leftarrow \sum_{p \in P}[x]_p \mod Q$.
Boolean sharing	Boolean sharing is a special case for arithmetic sharing, where Q now is 2. Each party holds a share $\langle x \rangle = \{\langle x \rangle_p\}_{p \in P}$ and x can be reconstructed by $x \leftarrow \oplus_{p \in P}\langle x \rangle_p$.
A2B	Each party $p \in P$ locally converts its arithmetic share $[x]_p$ of x to binary format y_p ($y_p \equiv [x]_p$). Then, each party uses Boolean sharing to share $\{\langle y_p \rangle_q\}_{q \in P}$ with $\lvert P \rvert$ parties. Thereafter, each party $q \in P$ receives all the Boolean shares from other and locally computes its own share by $\langle x \rangle_q \leftarrow \sum_{p \in P}\langle y_p \rangle_q$.
Addition	For calculating $[z] = [x] + [y]$, each party $p \in P$ locally computes $[z]_p \leftarrow [x]_p + [y]_p$.
Multiplication	Beaver triples [8] $([a]_p, [b]_p, [c]_p)_{p \in P}\ s.t.\ c = a \cdot b$ generated by a TTP are sent to each party. Then, each party locally computes $[\epsilon] = [x] - [a]$ and $[\delta] = [y] - [b]$. After that, all parties collaboratively reconstruct $\epsilon \leftarrow \sum_{p \in P}[x]_p - [a]_p$ and $\delta \leftarrow \sum_{p \in P}[y]_p - [b]_p$ and two values are then public. Since a and b work as random masks, x and y will not be revealed. The result of multiplication can be reconstructed by $z \leftarrow \sum_{p \in P}([c]_p + \epsilon[b]_p + [a]_p\delta) + \epsilon\delta$.
Square	Square is a special case for multiplication where Beaver pairs $([a]_p, [b]_p)_{p \in P}\ s.t.\ b = a^2$ are used. Similarly to multiplication, each party locally computes $[\epsilon] = [x] - [a]$, and then collaboratively reconstructs and publishes $\epsilon \leftarrow \sum_{p \in P}[x]_p - [a]_p$. The result of square can be reconstructed by $[x^2] \leftarrow \sum_{p \in P}([b]_p + 2\epsilon[a]_p) + \epsilon^2$.
Comparison	To compare $[x]$ and $[y]$, their difference $[z] = [x] - [y]$ is computed. Then, parties use *A2B* to convert $[z]$ to $\langle z \rangle$ and only extract the most significant bit of $\langle z \rangle$ by logical right shifting. Finally, the most significant bit is collaboratively reconstructed by all parties.

[3]https://github.com/facebookresearch/CrypTen
[4]https://github.com/OpenMined/SyMPC

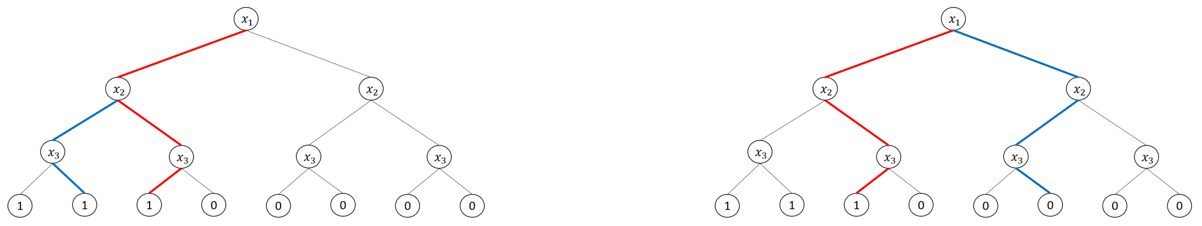
2.4.2. SyMPC

SyMPC mainly supports two different protocols, FSS [92] + SPDZ [24, 25] where SPDZ acts as an extension to support matrix operations, and Falcon [108] + ABY3 [75] where ABY3 provides the comparison for the missing part of Falcon. FSS cannot tolerate malicious parties, but Falcon can because Falcon uses replicated secret sharing that each party holds $N-1$ shares. The demonstration of SyMPC will be divided into two parts respective to two different protocols.

FSS: The original designing purpose of Function Secret Sharing (FSS) is mainly for 2-server *Private Information Retrieval* (PIR) where the inquirer only obtains the corresponding result without being aware of other possible results while the query is not revealed to servers. Inspired by Distributed Point Functions (DPF) from [39], Boyle et al. extended the original sharing framework into a more general one in 2014 [15] and further shrunk the computational cost in 2016 [16]. In the later work, Ryffel et al. referenced this idea and applied FSS to neural networks training and federated learning [92] in 2021. A point function is namely a function: $f_{x,y}(x) = y$ and $f_{x',y}(x') = 0$ for all $x' \neq x$. In other words, $f_{x,y}$ is only allowed to output a value y at a specific position x. This function can be shared by sharing its truth table, but the complexity of keys will be $O(2^{\sqrt{n}\lambda})$ which is exponentially increased with respect to the output space $\{0,1\}^n$. If recursive DPF is applied, the complexity will be lowered to $O(n^{1.58}\lambda)$ [39]. If an optimized tree-based structure is used, it will be further decreased to $O(n\lambda)$ [16]. As FSS is a kind of additive secret sharing scheme, it also complies with basic conventions of additive secret sharing. Therefore, in terms of adding, each party can perform it locally, while multiplication can be completed by Beaver triples [8]. The main uniqueness of FSS is its comparison.

Taking 2-party secret sharing as an example, FSS scheme contains two processes $(KeyGen, Eval)$ - $(k_0, k_1) \leftarrow KeyGen(\{0,1\}^\lambda, f)$ and $[f]_i(x) \leftarrow Eval(i, k_i, x)$ where λ, i, f and x are the length of a seed, the index of each party, the decryption of a function and the input respectively - such that $Pr[Eval(0, k_0, x) + Eval(1, k_1, x) = f(x)] = 1$. The completed proofs of correctness and security can be found in [15]. The process of comparison is shown in algorithm 2 [92] that is the improved version of [17]. α is a private number that waits to be compared, x is the public input number held by two evaluators, j represents the id of evaluators, $s_j^{(i)}$ is the random seed for the i^{th} layer, $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1} \times \{0,1\}^{n+1} \times \{0,1\}^{n+1}$ is a pseudo random generator which takes a λ length input and outputs $2(\lambda + n + 2)$ length output, $cw^{(i)}$ is the secret for the i^{th} layer, $CW^{(i)}$ is the correction word that maintains the randomness and 1-share in the *special path*, and 0-share for other paths.

At the beginning, since two evaluators own different initial $t^{(1)}$ ($t_0^{(1)} \oplus t_1^{(1)} = 1$), one will choose to add $CW^{(1)}$ and another one will not. In this case, due to line 6, if they are on the *special path* ($\alpha[1] = x[1]$), then their next round seeds $s^{(2)}$ and indicators $t^{(2)}$ has following properties: $s_0^{(2)} \oplus s_1^{(2)} = s_0^L \oplus s_1^L$ ($s_0^L \oplus s_1^L$ is a λ length random number) and $t_0^{(2)} \oplus t_1^{(2)} = 1$; otherwise ($\alpha[1] \neq x[1]$), $s_0^{(2)} \oplus s_1^{(2)} = 0$ and $t_0^{(2)} \oplus t_1^{(2)} = 0$, which means that both will continue to use and generate same seeds s^i for the rest of path. In other words, they will maintain the randomness share if both are on the *special path* or they will maintain the zero share if both deviate from that path.



(a) deviation due to “less than” **(b)** deviation due to “greater than”

Figure 2.3: An example of FSS for comparison

For better understanding, we show an example in figure 2.3, where secret α is 010_2 and red bold path is the *special path*. Leaves are labeled with the final status, but they are not visible to either of the evaluators in reality. The following part tests the correctness of the comparison for different scenarios and affirms that the combined shares $T \leftarrow [T]_0 \oplus [T]_1$ output 1 only when $x \leq \alpha$ otherwise 0.

- **Equality**: If the input x is exactly 010_2, it will follow the *special path* till the end. During the evaluation and before reaching the leaf, $x[i] = \alpha[i] = 0$ or $x[i] = \alpha[i] = 1$ for every nodes. If $x[i] = \alpha[i]$, we will have $s_0^{(i+1)} \oplus s_1^{(i+1)} = \{0,1\}^{\lambda}_{random}$, $t_0^{(i+1)} \oplus t_1^{(i+1)} = 1$, $\sigma_0^{(i+1)} \oplus \sigma_1^{(i+1)} = 0^{\lambda}$ and $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 0$, then the layer output will be $out_0^{(i)} \oplus out_1^{(i)} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = 0$. When both reach the leaf, we have $s_0^{(n+1)} \oplus s_1^{(n+1)} = \{0,1\}^{\lambda}_{random}$ and $t_0^{(n+1)} \oplus t_1^{(n+1)} = 1$, then the leaf output will be $out_0^{(n+1)} \oplus out_1^{(n+1)} = (t_0^{(n+1)} - t_1^{(n+1)}) \cdot CW_{leaf}^{(n+1)} + (s_0^{(n+1)} - s_1^{(n+1)}) = (1 - 2t_1^{(n+1)}) \cdot (-1)^{t_1^{(n+1)}} \cdot (1 - s_0^{(n+1)} + s_1^{(n+1)}) + (s_0^{(n+1)} - s_1^{(n+1)}) = 0$ no matter $t_1^{(n+1)} = 0$ or 1. As a result, $[T]_0 \oplus [T]_1 = \sum_i out_0^{(i)} \oplus out_1^{(i)} \mod 2^n = 1$. Therefore, the equality test holds.
- **Less than**: If the input x is less than 010_2, e.g. 001_2 in figure 2.3a, the tracking path at a node i will turn left from the *special path* shown as the blue bold path. This situation occurs when $x[i] = 0$ and $\alpha[i] = 1$. If so, we have $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 1$, then the node output will be $out_0^{(i+1)} \oplus out_1^{(i+1)} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = (1 - 2\tau_1^{(i+1)}) \cdot (-1)^{\tau_1^{(i+1)}} \cdot (\alpha[i] - \sigma_0^{(i+1)} + \sigma_1^{(i+1)}) + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = \alpha[i] = 1$ no matter $\tau_1^{(n+1)} = 0$ or 1. In addition, since both leave the *special path*, they will use identical seeds for generators such that the zero share is maintained for the following nodes also the final leaf ($out_0^{(j)} \oplus out_1^{(j)} = 0, i < j \leq n+1$). As a result, $[T]_0 \oplus [T]_1 = \sum_i out_0^{(i)} \oplus out_1^{(i)} \mod 2^n = 1$. Therefore, the less than holds.
- **Greater than**: If the input x is greater than 010_2, e.g. 101_2 in figure 2.3b, the tracking path at a node i will deviate to the right blue bold path. This situation happens when $x[i] = 1$ and $\alpha[i] = 0$. If so, we have $\tau_0^{(i+1)} \oplus \tau_1^{(i+1)} = 1$, then the node output will be $out_0^{(i+1)} \oplus out_1^{(i+1)} = (\tau_0^{(i+1)} - \tau_1^{(i+1)}) \cdot CW_{leaf}^{(i)} + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = (1 - 2\tau_1^{(i+1)}) \cdot (-1)^{\tau_1^{(i+1)}} \cdot (\alpha[i] - \sigma_0^{(i+1)} + \sigma_1^{(i+1)}) + (\sigma_0^{(i+1)} - \sigma_1^{(i+1)}) = \alpha[i] = 0$ no matter $\tau_1^{(n+1)} = 0$ or 1. In addition, since both leave the *special path*, they will use identical seeds for generators such that the zero share is maintained for the following nodes also the final leaf ($out_0^{(j)} \oplus out_1^{(j)} = 0, i < j \leq n+1$). As a result, $[T]_0 \oplus [T]_1 = \sum_i out_0^{(i)} \oplus out_1^{(i)} \mod 2^n = 0$. Therefore, the greater than holds.

In terms of security, since neither evaluators can realize the deviation from the *special path* and the output information can only be revealed from the querying side who holds the secret α; thus, the whole process reveals no information respective to α. As both evaluators should obey the rule to choose left or right based on $x[i]$, the whole process cannot tolerate traitors; thus, FSS is only for semi-honest settings, not for malicious ones. There is a latest and simpler version for the comparison demonstrated in [14], but since the library has not cathed up yet, we will keep using the current version.

Falcon: Falcon consists of techniques from SecureNN [107] and ABY3 [75] under the assumption of honest majority (e.g. two-out-of-three) replicated secret sharing that guarantees the *security with abort* in the malicious setting [37]. Since parties hold replicated secret shares, both addition and multiplication can be done locally in the semi-honest setting. For instance, if there are three parties P_i for $i = 1, 2, 3$ who individually hold share $([x]_i, [x]_{i+1})$ and $([y]_i, [y]_{i+1})$ $(\sum_i [x]_i = x, \sum_i [y]_i = y)$, the addition can be locally computed by $([z]_i \leftarrow [x]_i + [y]_i, [z]_{i+1} \leftarrow [x]_{i+1} + [y]_{i+1})$ $(\sum_i [z]_i = x + y)$ whereas the multiplication can be locally computed by $[z]_i \leftarrow [x]_i[y]_i + [x]_{i+1}[y]_i + [x]_i[y]_{i+1}$ $(\sum_i [z]_i = x \cdot y)$. In terms of multiplication, so as to maintain the replicated secret shares, P_i will mask $[z]_i$ by $[\alpha]_i$ and send $[z]_i \leftarrow [z]_i + [\alpha]_i$ to P_{i+1} where $\{[\alpha]_i\}_{i=1,2,3}$ are correlated randomness generated by pairwise keys such that $\sum_i [\alpha]_i = 0$. In the offline phase, P_i and P_{i+1} share an identical key k_i, then in the online phase, each party P_i can locally generate mask $[\alpha]_i \leftarrow F_{k_i}(counter) - F_{k_{i-1}}(counter)$.

Under the malicious setting, there is one malicious adversary among three parties who can arbitrarily betray the protocol to deviate the correct result. Thus, parties will check the validity of the result before accepting it. To achieve this, each party P_i sends share $[z]_i$ to P_{i+1} and $[z]_{i+1}$ to P_{i-1}. They achieve a consensus and reconstruct from shares only if they receive an identical value of shares from other two parties otherwise they will abort the reconstruction. Differing from addition, multiplication includes a semi-honest broadcasting of shares in order to maintain the replicated secret sharing. Therefore, only the correctness of received shares is guaranteed, parties latterly can perform the above verification.

Algorithm 2 FSS Comparison for $x \leq \alpha$

```
KeyGen:
1: Initialization: $\alpha \leftarrow \mathbb{Z}_{2^n}$, sample random $s_j^{(1)} \leftarrow \{0,1\}^\lambda$ and set $t_j^{(1)} \leftarrow j$, for $j = 0, 1$
2: for i=1...n do
3:     $((s_j^L||t_j^L, s_j^R||t_j^R),\ (\sigma_j^L||\tau_j^L, \sigma_j^R||\tau_j^R)) \leftarrow G(s_j^{(i)})$, for $j = 0, 1$
4:     if $\alpha[i]$ then $cw^{(i)} \leftarrow ((0^\lambda||0, s_0^L \oplus s_1^L||1),\ (\sigma_0^R \oplus \sigma_1^R||1, 0^\lambda||0))$    ▷ true secret for right path
5:     else $cw^{(i)} \leftarrow ((s_0^R \oplus s_1^R||1, 0^\lambda||0),\ (0^\lambda||0, \sigma_0^L \oplus \sigma_1^L||1))$    ▷ true secret for left path
6:     $CW^{(i)} \leftarrow cw^{(i)} \oplus G(s_0^{(i)}) \oplus G(s_1^{(i)})$    ▷ correction word
7:     $state_j \leftarrow G(s_j^{(i)}) \oplus (t_j^{(i)} \cdot CW^{(i)}) = ((state_{j,0}, state_{j,1}), (state'_{j,0}, state'_{j,1}))$, for $j = 0, 1$
8:     parse $s_j^{(i+1)}||t_j^{(i+1)} = state_{j,\alpha[i]}$ and $\sigma_j^{(i+1)}||\tau_j^{(i+1)} = state'_{j,1-\alpha[i]}$, for $j = 0, 1$
9:     $CW_{leaf}^{(i)} \leftarrow (-1)^{\tau_1^{(i+1)}} \cdot (\alpha[i] - \sigma_0^{(i+1)} + \sigma_1^{(i+1)}) \mod 2^n$
10: $CW_{leaf}^{(n+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot (1 - s_0^{(n+1)} + s_1^{(n+1)}) \mod 2^n$
11: return $k_j \leftarrow [\alpha]_j||s_j^{(1)}||\{CW^{(i)}; i = 1...n\}||\{CW_{leaf}^{(i)}; i = 1...n+1\}$, for $j = 0, 1$

Eval:    ▷ each party locally evaluates
12: Inputs: $(j, k_j, x)$ where $j$ refers the id of the evaluator
13: parse $k_j = [\alpha]_j||s^{(1)}||\{CW^{(i)}; i = 1...n\}||\{CW_{leaf}^{(i)}; i = 1...n+1\}$ and set $t^{(1)} \leftarrow j$
14: for i=1...n do
15:     $state \leftarrow G(s^{(i)}) \oplus (t^{(i)} \cdot CW^{(i)}) = ((state_0, state_1), (state'_0, state'_1))$
16:     parse $s^{(i+1)}||t^{(i+1)} = state_{x[i]}$ and $\sigma^{(i+1)}||\tau^{(i+1)} = state'_{x[i]}$
17:     $out^{(i)} \leftarrow (-1)^j \cdot (\tau^{(i+1)} \cdot CW_{leaf}^{(i)} + \sigma^{(i+1)}) \mod 2^n$  ▷ $out^{(i)} = 1$ only when first time $<$ happens
18: $out^{(n+1)} \leftarrow (-1)^j \cdot (t^{(n+1)} \cdot CW_{leaf}^{(n+1)} + s^{(n+1)}) \mod 2^n$    ▷ $out^{(n+1)} = 1$ only when equality holds
19: return $[T]_j \leftarrow \sum_i out^{(i)} \mod 2^n$
```

Similar to other secret sharing schemes, it will exploit Beaver triples [8] to generate replicated secret shares to avoid using semi-honest broadcasting. The correctness of multiplication relies on Beaver triples [8] which parties can adopt cut-and-choose [21] during the offline phase to generate adequate of. When using cut-and-choose paradigm, it requires each party locally performs random permutation and random verification which are detailed by protocol 3.2 $F_{triples}$ in [37]. Supposed that parties apply $F_{triples}$ to obtain a valid Beaver triple $([a], [b], [c])\ s.t.\ c = ab$ in advance, each party P_i locally computes $([\rho]_i \leftarrow [x]_i - [a]_i, [\rho]_{i+1} \leftarrow [x]_{i+1} - [a]_{i+1})$ and $([\sigma]_i \leftarrow [y]_i - [b]_i, [\sigma]_{i+1} \leftarrow [y]_{i+1} - [b]_{i+1})$. Then, any two of three collaboratively reveal ρ and σ and all parties involve to check the reconstructed ρ and σ are identical for any two of them. Since a and b are randomness, x and y will not be disclosed. After that, each party P_i locally computes its first replicated secret share $[z]_i \leftarrow [c]_i + \rho[b]_i + \sigma[a]_i + \max(0, 2-i) \cdot \rho\sigma$ and the second one $[z]_{i+1} \leftarrow [c]_{i+1} + \rho[b]_{i+1} + \sigma[a]_{i+1} + \max(0, 2-i) \cdot \rho\sigma$ where $\max(0, 2-i)$ can also be regarded as and replaced by Kronecker delta $\delta_{ij_{|j=1}}$. In this case, they can do analogous correctness verification as they do for the addition: P_i sends share $[z]_i$ to P_{i+1} and $[z]_{i+1}$ to P_{i-1}. It can be tested that $\sum_i [z]_i = \sum_i ([c]_i + \rho[b]_i + \sigma[a]_i) + \rho\sigma = x \cdot y$.

To privately compare a secret number x and a public number r, parties should be shared a random bit $\beta \in Z_2$ and a random secret integer $m \in Z_p^*$ beforehand. Firstly, all parties collaboratively compute and reveal every bit $u[i] \leftarrow (-1)^\beta (x[i] - r[i]) = (1 - 2\beta) \cdot (x[i] - r[i])$ for $i = 1, ..., l$ (supposed l-bit length). Bit $u[i]$ discloses no information about $x[i]$ or $r[i]$ because of the random masking bit β. Then, each party P_j locally computes $\langle w[i] \rangle_j \leftarrow \langle x[i] \rangle_j + \delta_{j1} r[i] - 2r[i] \langle x[i] \rangle_j$. After that they use the results to jointly disclose $c[i] \leftarrow u[i] + 1 + \sum_{k=i+1}^{l} w[k]$ where $\langle u[i] \rangle_j$ acts like a masking bit for party P_j. The next step before the final is to compute $[d] \leftarrow [m] \cdot \prod_{i=1}^{l} c[i] \mod p$. Finally, each party obtains shares of $\beta' \oplus \beta \in Z_2 := (x \geq r)$ as return where $\beta' = 1$ if $d \neq 0$ and 0 otherwise. The general designing purpose can be found in section 3.3 algorithm 1 in [108].

3 Problem Setup

Any security of a system is assessed in terms of the adversarial aims and capabilities that it is meant to counter against the threat model [89]. This chapter will taxonomize the definition and scope of threat models in the FL system. To determine where and how an adversary would seek to undermine the system by different attacks, it will determine the threat surface of systems built on FL. Depending on adversaries' goals, the attacking strategies can be divided into two main categories - inference attack and poisoning attack. The former aims to infer the personal information from the individual's upload or the aggregated model, while the latter tries to mitigate the accuracy of the model or mislead the prediction result [50].

3.1. Inference Attack

FL seeks to preserve users' privacy by requesting them to provide local training model parameters rather than their actual data. However, the parameters are the high summary of personal data; thus, the gradients have the potential to expose private training data [126]. Leakage from parameters backfires the primitive design principle of FL. This section will indicate the leaking surface of FL and the strategies to compromise privacy in FL.

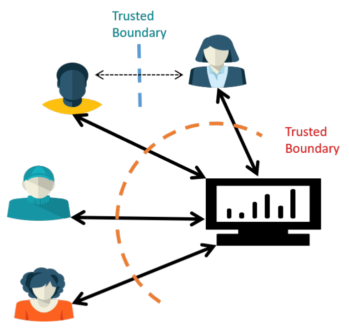
3.1.1. Leaking surface

There are two kinds of *trust boundaries* in FL, as shown in figure 3.1a, where a trusted boundary is between the client and the server while another is between two different clients. On the one hand, the server can observe the individual upload before aggregation, as clients directly upload plain-text parameters to the server. The server can use reverse engineering, e.g., model inversion [34], to infer the training samples. On the other hand, every participant can receive the merged model after aggregation. The participant can deduce the sum of other participants' models by subtracting his/her model from the aggregated one. Compared with the server, inference from the client is limited, as the client only obtains the aggregated model. However, as a participant, the adversary can participate in the FL training to trigger specific clients to release more information and reconstruct the class representative by GAN [47]. In figure 3.1b, the adversary who does not have class "0", "3", and "4", but can learn from others and forge the class representatives by joining in the training within 50 rounds.

3.1.2. Inference strategy

The inference attacks are diverse. Based on the target of an adversary, there are mainly four different types of inference [64, 76].

I1. **Class representatives inference**: The adversary uses Generative Adversarial Networks (GAN) [49] to reconstruct the class representatives that the adversary does not have. The global model of FL is a natural discriminative model. The adversary builds a generative model locally and feeds the global model with the forgery of samples. Consequently, the generative model can be well trained during the training of FL [47]. However, as the discriminative model incorporates many individual models, the adversary can only obtain every class representative that dominates this class, not the actual training samples of this class.



(a) Trusted boundaries in FL

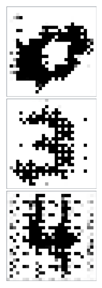

(b) GAN reconstruction

Figure 3.1: Inference attacks

I2. **Membership inference**: In FL, the privacy should be guaranteed on the client level [38], so we slightly modify the original data level inference to the client level: Given a particular dataset of a client, membership inference is to decide whether this particular client participates in the training. The inference can be performed in either passive or active way [98, 105]. In the passive setting, the adversary is only an observer of the network who tries to use statistical methods to extract the information from the model but cannot modify the global model. In contrast, under the active setting, the adversary can upload the malicious parameters to the server to allure the FL model to share more information regarding a particular client or a particular group of clients [81].

I3. **Properties Inference**: In [6, 35, 72], adversaries' goal is to infer attributes that characterize a particular class. Taking CIFAR-10 as an example, given a trained global model in which one of the classes is "bird", the adversary's goal is to infer what a bird looks like, e.g., the colorful feather. Another example is given by [72]: Infer what Bob looks like, e.g., whether Bob wears glasses, from a face recognition model when one of the classes is Bob. Similar to the membership inference, the properties inference can also be passive or active. Especially under an active setting, properties inference can proactively modify the objective function of the local model to force the global model unintentionally leak side information about other clients [72].

I4. **Training inputs and labels inference**: This inference is powerful but computationally costly. The honest-but-curious server can carry out this inference to reconstruct the pixel-wise accurate original images and token-wise matching original texts [64] if the given deep learning model is not encrypted. The adversaries use the DLG algorithm [126] by minimizing the gradients' distance to reduce the differences between the dummy input and the actual input and the differences between the dummy label and the actual label.

3.2. Poisoning Attack

The attack model in this paper follows previous works [19, 33, 50, 84, 85]. Specifically, an adversary can pretend to be a legal client before the start of training or hijack a genuine client during the online phase, but they cannot compromise the server. Adversaries, a.k.a model attackers in this case, can collude during the training phase, but they have no idea about other benign clients.

3.2.1. Classification of attacks

This section will compare different settings and restrictions on the byzantine client. Furthermore, it will detail the adversaries' capability and the knowledge that adversaries master.

C1. **Client-side attacks and server-side attacks**: An attack is defined as a client-side attack if the adversary compromises an authentic client or pretends to be an authentic client to poison the model. In contradiction, an attack is a server-side attack if the adversary can ultimately compromise the server to modify the aggregation rule or the aggregated model. This paper assumes that the server cannot be compromised, so adversaries can merely attack the model from the client-side.

C2. **Full knowledge attacks and partial knowledge attacks**: In the full knowledge scenario, the adversary has access to every worker device where the local training dataset and model are known.

However, the adversary only has access to training datasets and models belonging to compromised workers in a partial knowledge situation. Since full knowledge assumption is impractical under decentralized setting [36, 96], we will merely consider that adversaries have no information of other uncompromised clients' behavior.

C3. **Colluded attacks and non-colluded attacks**: Depending on whether the adversaries can communicate with each other, the attacks can be colluded or non-colluded. The attack can be more powerful and latent if adversaries collude because they can elaborate their datasets or uploads to make some of them escape the server's inspection. In this paper, no restriction is added on byzantine workers, and they can free contact each other.

C4. **Model update poisoning attacks and data poisoning attacks**: Through arbitrarily manipulating compromised clients' updates or directly sending elaborated updates to the server, attackers try to corrupt the aggregated model, which so-calls model update poisoning. In terms of data poisoning attacks, adversaries are not allowed to corrupt the updates; instead, they can only manipulate the local data to output updates that they expected. Both attacking types are considered in this paper.

C5. **Targeted attacks and untargeted attacks**: Targeted attacks corrupt the model such that the model predicts attacker-desired outcomes for attacker-chosen samples. In contrast, untargeted attacks corrupt the aggregated model such that the overall accuracy and usability of the model are declined. This separation is mainly for data poisoning attacks, and both will be considered in this research.

3.2.2. Attacking schemes

This section will provide the state-of-art attacking schemes used to test the robustness of FLVoogd's defense [19, 61, 65, 84, 85, 91, 115]. Their summary is shown in table 3.1.

Table 3.1: Summary of poisoning attacks

Attack	C1	C2	C3	C4	C5
A1	client-side	✗	✗	update	untargeted
A2	client-side	partial	colluded	update	untargeted
A3	client-side	partial	colluded	update	untargeted
A4	client-side	partial	✗	data	(un)targeted
A5	client-side	partial	✗	data	targeted
A6	client-side	partial	✗	data	targeted

A1. **Random upload**: As its name suggests, the adversary substitutes the factual update with a random noise chosen from $X \sim \mathcal{N}(0,1)$. Consequently, if there is no defense, the average of parameters can arbitrarily deviate from $w_{avg} = \frac{1}{n}\sum_{i=1}^{n} w_i$ to $w_{dev} = \frac{1}{m}\sum_{i=1}^{m} w_i + \frac{1}{n-m}\sum_{i=m+1}^{n}\mathcal{N}(0,1)$, where m is the number of honest updates and $(n-m)$ is the number of malicious updates. It is equivalent to adding Gaussian noise $\mathcal{N}(0, n-m)$ to $\frac{1}{m}\sum_{i=1}^{m} w_i$ if each adversary draws the noise independently. Due to the high dimensional randomness, adversaries can at least postpone the convergence or even destroy the model. In addition, the distribution of malicious random uploads is sparse and unpredictable because of the randomly diverse directions; thus, they are not easily filtered at the beginning of the FL training or after the model converges.

A2. **Krum attack**: Krum attack is designed initially to crack the Krum aggregation rule shown in eq.(3.1) proposed in [13]. In a nutshell, Krum selects one vector from a set of n vectors that is the most comparable to the rest. Even if the chosen vector is given by a compromised client, the impact is limited in this situation. If there are $f < \frac{n-2}{2}$ compromised vectors, Krum guarantees the model convergence. The proof relevant to the convergence can be found in [13].

$$\begin{gathered}\text{for all } i = 1, ...n: \text{ compute } score(i) = \sum_{j\in(n-f-2)NearestNeighbours} ||w_i - w_j||^2 \\ \text{s.t. } KR(w_1, ..., w_n) = w_{i^*}, \text{ where } score(i^*) \leq score(i)\end{gathered} \tag{3.1}$$

We assume that the first f workers are compromised without loosing the generality, and w_i denotes the true update without being compromised and w'_i is the compromised one. When adversaries try to invalidate the Krum aggregation rule, they can collude to elaborate a set of vectors to make $KR(w'_1, ..., w'_f, ..., w_n)$ output w'_1 such that w'_1 mostly inversely differs from the true selected one without being attacked [33]. Therefore, the problem can be converted to optimize eq.(3.2), where s is the indicator vectors $\{-1, 1\}^*$. When $s_j = 1$ (or $s_j = -1$), it represents the truth j^{th} parameter of w^{cur} increases (or decreases) compared to the previous round w^{prev}.

$$\begin{aligned} \max_{w'_i, .., w'_f} \; & s^T(w - w'), \\ \text{subject to } & w = KR(w_1, ..., w_f, ..., w_n), \\ & w' = KR(w'_1, ..., w'_f, ..., w_n). \end{aligned} \tag{3.2}$$

To solve the optimization, in each round, adversaries collaboratively do the following steps.

Step 1. f adversaries receive the global update w^{prev} from the server.
Step 2. Before attacking, they estimate the current model by factual vectors as $\tilde{w}^{cur} \leftarrow \frac{1}{f}\sum_{i=1}^{f} w_i$.
Step 3. They estimate the indicator vector by $\tilde{s} \leftarrow \text{sign}(\tilde{w}^{cur} - w^{prev})$.
Step 4. They use binary search [113] to seek λ in eq.(3.3).
Step 5. After λ is found, $(f-1)$ vectors $\{w'_2, ..., w'_f\}$ are randomly chosen whose distance is at most ϵ to w'_1. This can be achieved by adding uniformly random noise on w'_1.
Step 6. Then, f workers replace $\{w_1, ...w_f\}$ by $\{w'_1, ..., w'_f\}$ and send them to the server.

$$\begin{aligned} \max_{\lambda} \; & \lambda, \\ \text{subject to } & w'_1 = KR(w'_1, w_1, ..., w_f), \\ & w'_1 = w^{prev} - \lambda\tilde{s}. \end{aligned} \tag{3.3}$$

However, in Step 4, adversaries may fail to find a large enough λ. In this case, they add another w'_i, e.g., $w'_2 = w^{prev} - \lambda\tilde{s}$, until λ is solved and exceeds the desired threshold (i.e., 10^{-5} [33]).

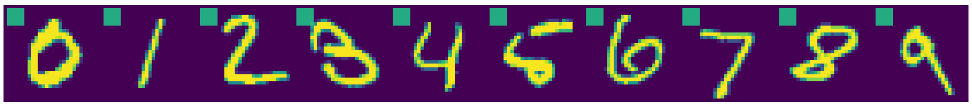
A3. **Trimmed-mean attack**: Trimmed-mean sorts n updates for each j^{th} parameter $sort(w_{1j}, ..., w_{nj})$, eliminates the highest and smallest β amount from the sorted list, and averages the remaining $(n - 2\beta)$ parameters as the global model's j^{th} parameter. It is originally proposed in [121], where authors have proved the convergence when $f \leq \beta \leq \frac{n}{2}$. To enervate this aggregation rule, adversaries do the following steps:

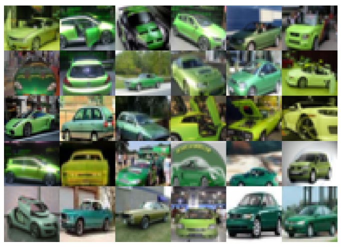
Step 1. f adversaries receive the global update w^{prev} from the server.
Step 2. Before attacking, they estimate the current model by factual vectors as $\tilde{w}^{cur} \leftarrow \frac{1}{f}\sum_{i=1}^{f} w_i$.
Step 3. They estimate the indicator vector by $\tilde{s} \leftarrow \text{sign}(\tilde{w}^{cur} - w^{prev})$.
Step 4. They estimate the mean $\tilde{\mu}_j$ and the standard deviation $\tilde{\sigma}_j$ for each j^{th} parameter.
Step 5. Each adversary independently chooses the j^{th} parameter based on eq.(3.4) as the compromised j^{th} parameter [33].
Step 6. Then, f workers replace $\{w_1, ...w_f\}$ by $\{w'_1, ..., w'_f\}$ and send them to the server.

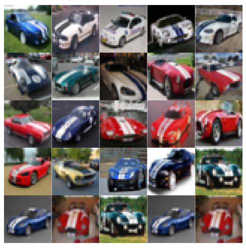
$$w'_{ij} = \begin{cases} x \sim \mathcal{N}(\tilde{\mu}_j + 3\tilde{\sigma}_j, \tilde{\mu}_j + 4\tilde{\sigma}_j) & s_j = -1 \\ x \sim \mathcal{N}(\tilde{\mu}_j - 4\tilde{\sigma}_j, \tilde{\mu}_j - 3\tilde{\sigma}_j) & s_j = +1 \end{cases} \tag{3.4}$$

A4. **Label flipping**: Each adversary flips the label of a sample from l to $L - l - 1$, where l is the truth label of the sample and L is the total number of classes [78]. For instance, adversaries label digits “0” to “9” and “9” as “0” to label-flip the MNIST [59] data.

A5. **Backdoor triggering**: This kind of attack is also known as trojan attacks [41]. The adversary inserts a specific pattern into training samples or uses the existing one in samples to render the corresponding testing samples with that pattern to be classified as the desired class. This pattern functions as a trigger. After the global model learns this pattern, it will be triggered and output the misled prediction. If the adversary uses the existing pattern in the sample, this backdoor attack is a semantic backdoor attack [91]. Figure 3.2 shows some examples of how a pattern can be used as a trigger. A 5×5 pixels rectangle is inserted into the image and used as a trigger to mislead the learning process in figure 3.2a [85, 91]. Furthermore, some naturally existing patterns in images can be found as semantic triggers. For instance, figures 3.2b 3.2c 3.2d show that images of cars contain some natural features that can be used as triggers [7].


(a) Digits with top-left rectangle


(b) Cars in green


(c) Cars with stripes

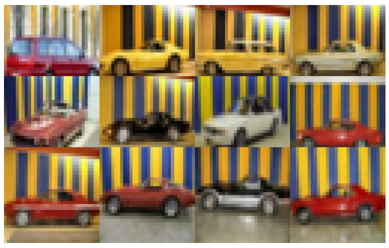
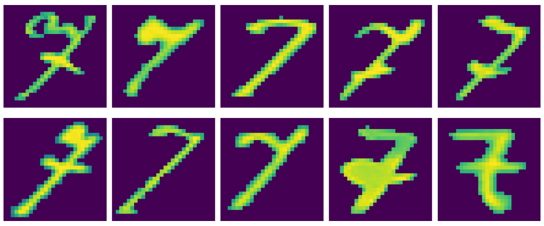

(d) Cars in front of stripe wall

Figure 3.2: Some examples of the trigger-backdoor attack

A6. **Edge-case attack**: Under the edge-case attack setting, adversaries aim to attack the heavy-tail of the prediction [109]. They try to find or manufacture samples that the model predicts correctly but with a comparably low confidence value; then, they label those samples with a label they want. The intuition behind it is that the model cannot assure the correctness of predictions even if the result is correct, as the predicting score is not such high, so it can be easily misled by the attacker who feeds those edge-case samples with wrong labels. We will use the samples provided by [109] and follow the experimental setup of [91, 109] to carry out our experiment. Figure 3.3 shows some samples of edge-case attack for (E)MNIST and CIFAR-10, respectively, where edge cases in figure 3.3a are sourced from Ardis [57] and others in figure 3.3b are provided by FedML [45]. Those "7"s will be labeled as "1"s, and blue airplanes will be labeled as "truck"s, to delude the server model.


(a) Edge cases for MNIST

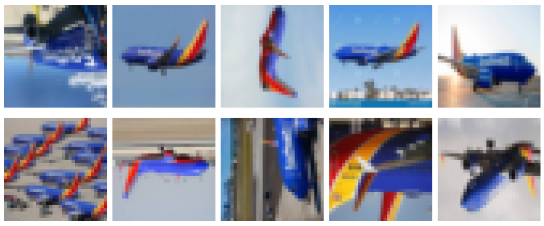

(b) Edge cases for CIFAR-10

Figure 3.3: Samples of the edge-case

4 FLVoogd

This chapter will present our FL framework - FLVoogd, which has two versions for the server side. One version does not encrypt the uploaded parameters, which means that the server can monitor every individual upload. Thus, a trusted server is required, but the filtering and aggregation can be efficient. Another version encrypts uploads, so uploads are invisible to the server. The encryption guarantees the process can run on an honest-but-curious server but retards the processing of filtering and aggregation. Two versions have different application domains. In terms of the client side, there is only one version. Their technical details will be exhibited in the following sections respectively, and their background knowledge has been demonstrated in chapter 2.

4.1. Encrypted FLVoogd Server

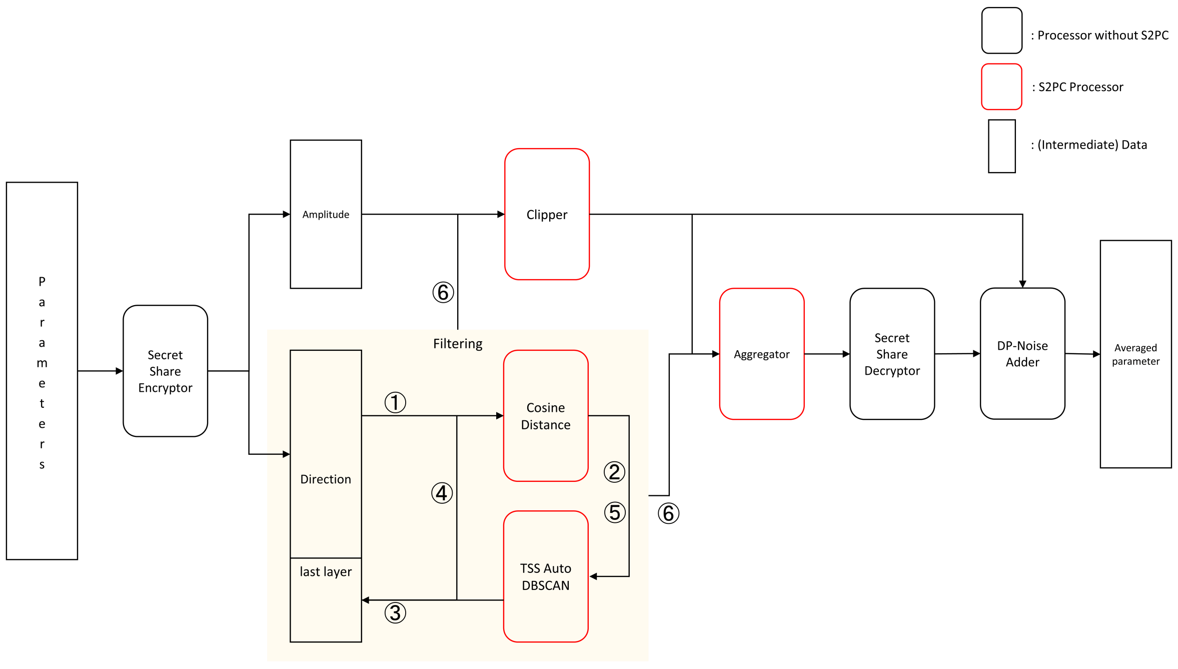


Figure 4.1: Encrypted FLVoogd server framework

4.1.1. Assumption

This section will give basic assumptions about the server and clients. The server acts as an aggregator that receives uploads from clients, averages those uploads, and sends them back to each client. In addition, the server needs to guarantee the correctness of the merged model, so it tries its best to sift and merely aggregate benign uploads. Clients train their local models based on their datasets and upload the parameters if they are chosen in this round. After receiving the averaged parameters from the server, they can update their local model accordingly. However, not all of them are required to obey the regulations. They can be malicious to degrade the performance of the global model.

We primarily consider possible server-side and client-side risks. There are two types of servers in our setting. Firstly, servers can be honest-but-curious who infer the actual data or relevant information from uploads while heeding the regulation. Secondly, if FLVoodg runs under SyMPC-Falcon [108], servers can be malicious (minority) who betray the secure aggregation rule and perform incorrect computations. In terms of participants, in each round, less than half of them can be malicious and perform byzantine attacks to deteriorate the performance of the global model. In addition, any client can be curious about information from others and performs client-level inference attacks [38], inferring whether a particular client participates in the training, given a specific dataset of that client.

4.1.2. Pipeline and algorithm

The overview of our framework, FLVoogd with SMPC, has been shown in figure 4.1. The symbols used in this section are presented in table 2 for reader's reference. Initially, each client computes the l2-norm of its uploaded parameters w_i as the amplitude $||w||_i$ and unitizes the parameters to $\overline{w}_i$ as the direction. Their relations satisfy eq. (4.1). The unitizing can simplify the later computations, e.g. cosine distance where the division is avoided, and convert the high dimensional parameters to floating numbers between -1 to 1 for the fixed-point encoding. Similarly, the client performs the same thing for the parameters from the last layer, and obtains the unitized last layer parameters $\overline{v}_i$. The reason for extracting the last layer is that parameters in the final layer reveal more explicit information relevant to the dataset's distribution [91] which can be used for distinguishing backdoor uploads. Then, the client secretly shares the direction $\overline{w}_i$, the amplitude $||w||_i$, and last-layer direction $\overline{v}_i$ with two parties, the server for aggregation and external server. If the SMPC protocol uses Falcon, one more server is added and receives the third share from the client. Falcon can detect if either of them betrays the regulation.

$$||w||_i = \sqrt{\sum_j w_{ij}^2} \qquad\qquad \overline{w}_i = \frac{w_i}{||w||_i} \tag{4.1}$$

Once servers receive the uploads from the selected clients, they can perform secure aggregation. It is supposed that the number of received uploads is n and the clipping boundary for the current round is $c^{(t)}$. The server firstly carries out the filtering process, the block with a light yellow background in figure 4.1, and the procedure can be divided into six steps. ①: In the first step, the server uses the directional vector $\overline{w}$ to compute the cosine distance matrix M_{cos} by eq. (4.2), where $M_{cosij} = cos - dist(i,j)$ for $i \neq j$ and $M_{cosij} = 0$ for $i = j$, i or j denotes the client's index, and u denotes the parameter's index. Since the vectors are unit vectors, the cosine distance between two vectors can be simplified into a dot production. The computation is collaboratively completed by two/three servers, involving the addition and multiplication among secret shares. ②: In the second step, the server feeds the Total-Sum-of-Square (TSS) based DBSCAN with the distance matrix from the former step. DBSCAN calculates the TSS by eq. (4.3) for each pair of rows to obtain a new distance matrix M_{tss}. M_{cos} provides the directional similarity, while M_{tss} enlarges the variance of counter-directions and narrows the variance of identical directions such that the filter can capture the difference more easily. There are two hyper-parameters for DBSCAN, Eps and $MinPts$ as mentioned in Chapter 2. As honest-majority is the basic assumption, $MinPts$ is set to $\lfloor n/2 \rfloor + 1$ and Eps is the average of the lower-bound median ($\lfloor n/2 \rfloor$) in each row of the distance matrix $M_{tss} \in \mathbb{R}^{n \times n}$. If the index of each row starts from 0 and ends at $n-1$, the median locates at $\lfloor n/2 \rfloor$ no matter whether n is even or odd, which is different from the standard median for an even number. This setting for Eps and $MinPts$ guarantees the DBSCAN can automatically adjust its radius accordingly through the whole process without manual involvement. The selection makes full use of the honest-majority assumption. This step includes the addition, multiplication, and comparison of shares.

$$cos - dist(i,j) = \frac{\sum_u \overline{w}_{i,u}\overline{w}_{j,u}}{\sqrt{\sum_u \overline{w}_{i,u}^2}\sqrt{\sum_u \overline{w}_{j,u}^2}} = \sum_u \overline{w}_{i,u}\overline{w}_{j,u} = \overline{w}_i \cdot \overline{w}_j \tag{4.2}$$

$$tss - dist(i,j) = \sum_u (M_{cosi,u} - M_{cosj,u})^2 \tag{4.3}$$

③: In the third step, DBSCAN filters out the noise and minority group and returns indices of the majority group. The server selects the corresponding clients' parameters from the last layer according to the indices. ④: For the next step, similar to step 1, the server again computes the cosine distance matrix but only uses the last-layer parameters. Then, the server obtains cosine distance matrix M'_{cos}. ⑤: The fifth step is identical to the second step. ⑥: In the final stage, DBSCAN outputs the indices that the server will consider benign.

After knowing which clients are considered as benign ones, the server clips their amplitudes before performs the aggregation by $||w||_i = \min(||w||_i, c^{(t)})$. The clipping controls the step size. Although the server sieves potential malicious uploads based on the direction, some cunning clients may still use the correct direction but amplify the norm of parameters multiple times to make the actual learning rate larger than expected. The learning process will become fluctuated, so the server needs to perform clipping on all filtered clients. The weight difference after clipping is equivalent to eq. (4.4).

$$w_i := w_i \cdot \min(1, \frac{c^{(t)}}{||w||_i}) \tag{4.4}$$

In addition, during the clipping, the clipper records the ratio of clients not being clipped as $\hat{\gamma}$. The expected clipping ratio is set as γ. The next round clipping boundary $c^{(t+1)}$ is updated by eq. (4.5) where η_c is the learning rate of the clipper. If the actual non-clipping number of clients is larger than expected, the clipping boundary will decrease to cut more clients in the next round; otherwise, it will increase to be looser. The exponential base guarantees that any adjustment is a positive number. Therefore, the clipping boundary learning process becomes adaptive and can accompany the clipping procedure (only need count), which is computational efficiency under SMPC. SMPC's operation in this step requires comparing a share with a public number.

$$c^{(t+1)} = c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma} - \gamma)) \tag{4.5}$$

The aggregation rule is simply averaging benign clients' uploads by eq. (4.6). It is supposed that the number of clients after filtering is $m(\leq n)$. After obtaining the merged model w_{global}, servers collaboratively reveal and announce the plain text of w_{global}. The aggregation contains the addition and multiplication of shares and the multiplication of shares and public numbers.

$$w_{global} = \frac{1}{m}\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)}) \tag{4.6}$$

The server eventually knows the global parameter till finishing aggregation, and local parameters are already merged into one; thus, the server has no idea of individual local updates. Before sending the global update back to clients, the server adds Gaussian noise to provide a differential privacy guarantee. The mean is 0, and the standard deviation is the maximum $l2$-sensitivity multiplied by a coefficient σ that represents the strength of DP. Thanks to the clipping, all uploads are bounded into a sphere whose radius is exactly the clipping boundary $c^{(t)}$. Therefore, the noise is added following eq. (4.7). Notably, the noise is added to the sum of updates not after averaging. DP-Noise Adder also tracks the DP budget for the server because it knows the number of clients used in this round and the amplitude of Gaussian noise. Finally, $||\tilde{w}_{global}||$ is compared with $||c^{(t)}||$. If $||\tilde{w}_{global}|| > ||c^{(t)}||$, from which the server deduces that the amount of noise is added too much, the server will scale down $||\tilde{w}_{global}||$ to a smaller value by eq. (4.8). This operation follows the post-processing property of differential privacy so that (ϵ, δ) cannot be influenced. This post-processing is equivalent to adjusting the model learning rate lower after knowing the noise influences too much on the result.

$$\tilde{w}_{global} = \frac{1}{m}\{\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)}) + \mathcal{N}(0, \sigma^2 \cdot (c^{(t)})^2)\} \tag{4.7}$$

$$\tilde{w}_{global} := \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||}) \tag{4.8}$$

4.1.3. Evaluation

The previous sub-section exhibits how individual updates are filtered and ultimately aggregated by the server. This sub-section will reflect the procedures and convince how those modules are combined to provide security and privacy guarantee.

The server requires three things from each client, the unit vector of the update, the unit vector of the last layer's update, and the norm of updates. The update can be gradient, weight, or weight difference (in our experiment), which is the digest from the learning process, indicating the direction and step-size (with learning rate) for the next round. This is why the server requires those three things from clients, as the direction and amplitude can be used as features to distinguish the malicious ones from all updates. The purpose of uploading norm and unit vector separately is to avoid division and square root operations since they are needed when calculating the denominator of cosine distance. Those two operations are computationally costly in most secure multi-party computation and homomorphic encryption protocols. Even though the protocol supports them in our experiment, we want to eliminate them for efficiency and for users who prefer to change to other unsupported protocols. The important thing is the removal is quite simple that only requires clients to send one more encrypted value. The data distribution, namely the label distribution, significantly affects the last layer's neurons that directly link to the predicting scores when the neural performs backward propagation. Therefore, the parameters from the final layer expose more information than the general. The server will use these three metrics for the filtering.

The later SMPC procedures reveal nothing from clients to the server except the indices of suspicious clients, whether the amplitude is greater than the clipping boundary, and the final averaged aggregated updates. Since the direction and amplitude of updates are masked texts for the server, the server can learn nothing from updates. Different SMPC modules marked by the red outline in figure 4.1 require different SMPC operations, and the operations are listed in table 4.1. It should be noticed that the comparison between two secret shares in TSS Auto DBSCAN is different and relatively simpler compared with the comparison between a secret share and a public number in Clipper shown in chapter 2. For instance, $[a]_i$ and $[b]_i$, where $i = 1, 2$, are secret shares of a and b for two parties P_1 and P_2. To compare the relationship of $[a] \leq [b]$, they locally compute $[a]_i - [b]_i$, then cooperatively reveal $c \leftarrow \sum_i [a]_i - [b_i]$, where c can determine which one is greater than the other. Although c is disclosed due to comparison, the server only knows the gap between $[a]$ and $[b]$ from c, and truth values a and b can not be inferred.

Table 4.1: SMPC operations in encrypted FLVoogd

Module	Operation
Cosine Distance	- multiplication between two shares
	- addition between two shares
TSS Auto DBSCAN	- multiplication between two shares
	- addition between two shares
	- comparison between two shares
Clipper	- comparison between a share and a public number
Aggregator	- multiplication between two shares
	- multiplication between a share and a public number
	- addition between two shares

Clipper is a crafty module, much like "kill two birds with one stone". On the one hand, it restrains the abnormally large amplitude and controls the next descent step-size. On the other hand, it provides the $l2$-sensitivity for the DP budget tracer. In addition, the clipper is adaptive learning during the learning process. Using the median of norms [85] as the clipping boundary sounds like a more unadorned alternative since it also fully uses the honest-majority assumption but is not applicable in our setting. The median should be a public number because it will be used for DP noise adding. All the amplitudes are secret shares, and the comparison among them reveals no information. However, secrets will be disclosed if the median after comparison is published because the server knows those gaps. Thus, this efficient and adaptive learning clipper is irreplaceable in our setting.

Till now, the filtering process maintains the robustness of the model, and SMPC preserves the participant's privacy to the server. One more step - DP noise adding - is carried out by the server to assure the participant's privacy to other participants. The DP-Noise Adder module also provides methods for tracking the ϵ-budget for differential privacy. The proportion of clients used in each round is dynamic because of sub-sampling and filtering; the amount of adding noise is adaptively changed and determined by the clipper. Still, the DP accountant can automatically monitor and track the process, then report back to the server.

Different modules in figure 4.1 are elaborately designed to provide privacy and security for the model cooperatively. The algorithm of encrypted FLVoogd is manifested in algorithm 3.

Algorithm 3 Encrypted FLVoogd - server

```
1:  Input:
2:  $\mathcal{C}$, $N$                          ▷ $\mathcal{C}$ is the set of clients, $N = |\mathcal{C}|$
3:  $T$, $q$                          ▷ $T$ is the number of training iteration, $q$ is the sampling ratio
4:  $c^{(0)}$, $\gamma$, $\eta_c$          ▷ $c^{(0)}$ is the initial clipping boundary, $\gamma$ is the expected clipping ratio, $\eta_c$ is Clipper's
    learning rate
5:  $\sigma$, $\delta$                          ▷ $\sigma$ is the coefficient to control the noise strength, $\delta$ is for DP
6:  for round $t$: $1, 2, ..., T$ do
7:      $\mathcal{C}^{(t)}$, $n \leftarrow$ subsample($\mathcal{C}$, $N$, $q$)                          ▷ $n = |\mathcal{C}^{(t)}|$
8:      for $client_i \in \mathcal{C}^{(t)}$ do
9:          $\overline{w}_i^{(t)}$, $||w||_i^{(t)}$, $\overline{v}_i^{(t)} \leftarrow client_i(t, send)$      ▷ $\overline{w}$ is the unit vector of weight difference, $||w||$ is the
    norm of weight difference, $\overline{v}$ is the unit vector of last layer's weight difference
10:     $idx_{f1}^{(t)}$, $n^{(t)'} \leftarrow$ Auto_DBSCAN($\{\overline{w}_1^{(t)}, \overline{w}_2^{(t)}, ..., \overline{w}_n^{(t)}\}$) by algorithm 4          ▷ $n^{(t)'} = |idx_{f1}^{(t)}|$
11:     $idx_{f2}^{(t)}$, $n^{(t)''} \leftarrow$ Auto_DBSCAN($\{\overline{v}_i^{(t)} : i \in idx_{f1}\}$) by algorithm 4          ▷ $n^{(t)''} = |idx_{f2}^{(t)}|$
12:     $w_{global}^{(t)} \leftarrow 0$, $\hat{\gamma}^{(t)} \leftarrow 0$
13:     for index $i \in idx_{f2}^{(t)}$ do
14:         if $||w||_i^{(t)} > c^{(t)}$ then
15:             $||w||_i^{(t)} \leftarrow c^{(t)}$
16:         else
17:             $\hat{\gamma}^{(t)} \leftarrow \hat{\gamma}^{(t)} + 1$
18:         $w_{global}^{(t)} \leftarrow w_{global}^{(t)} + ||w||_i^{(t)} \cdot \overline{w}_i^{(t)}$
19:     $w_{global}^{(t)} \leftarrow \frac{w_{global}^{(t)}}{n^{(t)''}}$, $\hat{\gamma}^{(t)} \leftarrow \frac{\hat{\gamma}^{(t)}}{n^{(t)''}}$
20:     $c^{(t+1)} \leftarrow c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma}^{(t)} - \gamma))$, $\epsilon^{(t)} \leftarrow$ DP_budget($\frac{n^{(t)''}}{N}$, $\sigma$, $\delta$)
21:     $\tilde{w}_{global}^{(t)} \leftarrow w_{global}^{(t)} + \frac{1}{n^{(t)''}}\mathcal{N}(0, \sigma^2(c^{(t)})^2)$          ▷ satisfying $(\epsilon, \delta)$-differential privacy
22:     $\tilde{w}_{global}^{(t)} \leftarrow \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||})$          ▷ satisfying post-processing
23:     $client_i(t, receive) \leftarrow \tilde{w}_{global}^{(t)}$
```

4.2. Unencrypted FLVoogd Server

4.2.1. Assumption

Assumptions regarding the client inherit assumptions in subsection 4.1.1. In contrast, the server should be honest and trusted because this framework does not encrypt clients' uploads. The server can

Algorithm 4 Auto DBSCAN

```
1: Input: $W$    ▷ $W \in \mathbb{R}^{n\times m}$ represents $n \times m$ matrix where each row is a client's unit vector from $n$ clients and the dimension of the vector is $m$
2: Output: $idx$, $|idx|$    ▷ $idx$ is a list of indices of benign clients
3: $M_{cos} \leftarrow$ CosDist($W$) by eq. (4.2)
4: $M_{tss} \leftarrow$ TSSDist($M_{cos}$) by eq. (4.3)
5: for row $i$: $1, 2..., n$ do
6:     $median_i \leftarrow$ quickMedian($M_{tss,i}$)
7: $median \leftarrow \frac{1}{n}\sum_{i=1}^{n} median_i$
8: $Eps \leftarrow median$, $MinPts \leftarrow n//2+1$
9: $idx \leftarrow DBSCAN(M_{tss}, Eps, MinPts, precomputed)$
10: return $idx$, $|idx|$
```

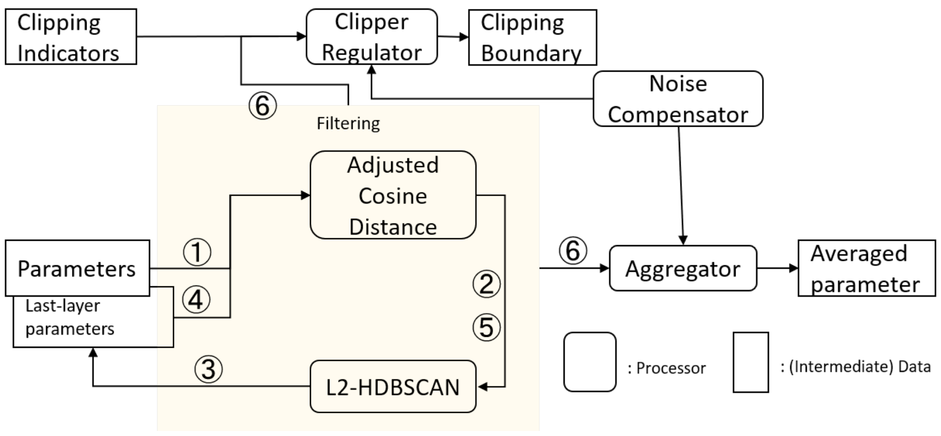


Figure 4.2: Unencrypted FLVoogd server framework

observe the plaintext of each upload, so the server can perform reconstruction attacks on clients.

4.2.2. Pipeline and algorithm

The unecrypted version is shown in figure 4.2. The server receives local model parameters w_i and clipping indicator b_i from each individual. Clients have already decided whether to clip w_i or not to make its norm not larger than $c^{(t)}$. If the client clips the model updates, b_i is set to 0; otherwise, b_i is 1. In addition, clients have already added $\mathcal{N}(0, \frac{\sigma^2(c^{(t)})^2}{n})$ and $\mathcal{N}(0, \frac{\sigma_b^2}{n})$ amount of noise to w_i and b_i, respectively. If no one has been filtered out, according to the property of sum of Gaussian noise, the amount of noise is $\mathcal{N}(0, \sigma^2(c^{(t)})^2)$ for $w_{global}^{(t)}$ and $\mathcal{N}(0, \sigma_b^2)$ for b_{avg} after averaging aggregation. The noise strength (noise multiplier) can be calculated as $(\sigma^{-2} + (2\sigma_b)^{-2})^{-1/2}$ [5] for tracing the DP budget.

Similar to the previous section, the filtering process contains six steps. ①: In the first step, the server uses the whole model parameters w_i from each client to compute the adjusted-cosine distance matrix M_{adjcos} by eq. (4.9), where the diagonal elements of M_{adjcos} are 0. Not only does this distance metric consider the relative direction, but also the relative position [93]. ②: In the second step, the server feeds the adjusted-cosine distance matrix to HDBSCAN, which measuring metric is set to $l2$ and $allow\_single\_cluster$ is set to $True$. ③: As a return, HDBSCAN gives a cluster with the majority. The server selects the corresponding clients' parameters from the last layer according to the returned indices. ④: In the fourth step, similar to step 1, the server again computes the adjusted-cosine distance matrix but only uses the last-layer parameters from the selected clients. ⑤: The fifth step is identical to the second step. ⑥: Finally, HDBSCAN outcomes the indices containing m clients who will be considered benign.

$$cos - dist_{adj}(i,j) = \frac{(w_i - \frac{1}{n}\sum_k w_k) \cdot (w_j - \frac{1}{n}\sum_k w_k)}{||w||_i||w||_j} \tag{4.9}$$

Before using the benign parameters and indicators for updating, Noise Compensator will compensate the noise loss due to the filtering. The compensating noise is $w_{noise} \leftarrow \mathcal{N}(0, \frac{n-m}{n}\sigma^2(c^{(t)})^2)$ and $b_{noise} \leftarrow \mathcal{N}(0, \frac{n-m}{n}\sigma_b^2)$ for parameters and indicators, respectively. However, The compensation is optional. If the subsampling rate is smaller than 1% and the number of clients per round is fewer than 20, the the compensation is unnecessary ($w_{noise} = 0, b_{noise} = 0$). Subsampling can significantly slow down the increase of DP-budget if it is small enough [111]. The proportion of clients who has not clipped their updates is $\hat{\gamma} \leftarrow$ min$\{$max$[\frac{1}{m}(\sum_i^m b_i + b_{noise}), 0], 1\}$. The min and max guarantee $\hat{\gamma}$ to locate in a meaningful scale, which follows DP post-processing. Then, the clipping boundary for the next round can be computed as eq. (4.5). The aggregation follows $\tilde{w}_{global} = \frac{1}{m}(\sum\limits_{i=1}^{m} w_i + w_{noise})$. Finally, $||\tilde{w}_{global}||$ is compared with $||c^{(t)}||$. If $||\tilde{w}_{global}|| > ||c^{(t)}||$, from which the server deduces that the amount of noise is added too much, the server will scale down $||\tilde{w}_{global}||$ to a smaller value by eq. (4.8).

4.2.3. Evaluation

Varying from the encrypted version, the unencrypted one does not use S2PC. Even though the noise adding moves to the client-side, which somewhat deviates the uploaded parameters from the true ones, it cannot provide strong (ϵ, δ)-DP with respect to the server but only guarantees (ϵ, δ)-DP after aggregation. Therefore, it cannot be supposed that the server infers nothing from the uploads if the server is honest-but-curious. However, it has two advantages. The first one is that the computation is fast since no secure computations are needed. The second one is, compared to DBSCAN, HDBSCAN is less sensitive to the noise, so it generates clusters more stably.

4.3. FLVoogd Client

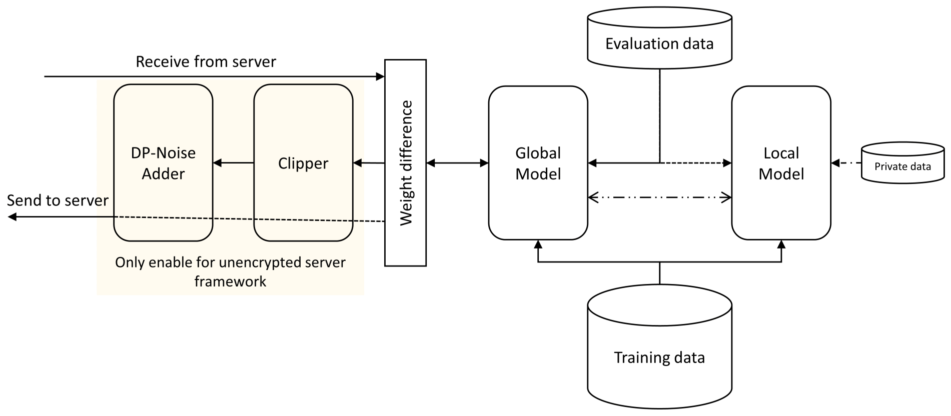


Figure 4.3: FLVoogd client framework

4.3.1. Assumption

Assumptions regarding the client inherit assumptions in subsection 4.1.1. As mentioned, the server tries its best to maintain security and privacy for clients. However, clients cannot entirely rely on those mechanisms executed remotely. Clients can also build a local mechanism for the model's robustness. Since clients are owners of the data, the client can locally conduct a performance-based test to verify whether the model can be accepted or not. In other words, clients can separate some of their data for the model evaluation, e.g., testing the model's accuracy. The number of training samples will reduce because of data splitting, but the number of evaluating samples does not need so many [19].

4.3.2. Pipeline and algorithm

Referencing the idea from Ditto [61], each FLVoogd's client builds two identical models, namely, the global model and local model shown in figure 4.3. In each round, the client receives the averaged aggregated weight difference $\tilde{w}_{global}$ from the server and updates the weight of the global model accordingly by eq. (4.10). In contrast, the local model is not updated in this step. After updating the locally global model, the client tests the model accuracy using evaluation data and obtains the testing accuracy acc_{ref}.

$$W_{global} := W_{global} + \tilde{w}_{global} \tag{4.10}$$

The client starts the training process in the next step. The client feeds the partial training data to the global and local models in each mini-batch iteration. It is supposed that there is a coefficient λ_{ditto} to control the distance of the local model from the global model. The objective function of the local model becomes like eq. (4.11), where $F(\cdot)$ is the objective function for the global model and originally for the local model. The change in the local model's objective function now is that the client adds an additional $l2$-regularization term to force the local model to approximate the global model. Consequently, the local model can learn from the global model, and the gap between them is constrained by λ_{ditto}.

$$\min_{W_{local}} F'(W_{local}; W_{global}) = F(W_{local}) + \frac{\lambda_{ditto}}{2} ||W_{local} - W_{global}||^2 \tag{4.11}$$

Furthermore, eq. (4.11) can be converted into a gradient decent format shown in eq. (4.12), where η_{local} is the client's local learning rate. The formula shown in eq. (4.12) can be easily implemented by PyTorch [90] where the client extracts the gradient and adds the $\lambda_{ditto}(W_{local} - W_{global}))$ term to it before running *optimizer.step()*.

$$g := g - \eta_{local}(\nabla F(W_{local}) + \lambda_{ditto}(W_{local} - W_{global})) \tag{4.12}$$

Till now, the client has λ_{ditto} as a controller to adjust the learning distance between the local and global models, but how to set an appropriate value λ_{ditto} for the local model? Intuitively, if the global model is admirable and exemplary, we expect the local model to learn as much helpful information as possible from the global model; otherwise, we desire the local model to learn less or even not learn from the global model. Then, the client can use the testing accuracy acc_{ref} as a reference to flexibly adjust λ_{ditto} by eq. (4.13). In the formula, λ_{max} and λ_{min} are the maximum and minimum values for λ_{ditto}, η_{ditto} is the learning rate, acc_{local} is the testing accuracy of the local model, and acc_{thres} is the minimum threshold to increase λ_{ditto}. λ_{max} and λ_{min} restrain the coefficient of the $l2$-regularization in a reasonable interval. η_{ditto} controls each mini-batch iteration's growing/decaying speed for λ_{ditto}. acc_{thres} is the threshold to control whether the current global model is worth being learned. In other words, the local model will absorb from the global model, only if acc_{ref} is higher than $acc_{local} + acc_{thres}$.

$$\lambda_{ditto} := \min(\lambda_{max}, \max(\lambda_{min}, \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres}))) \tag{4.13}$$

If the server uses the encrypted framework, the client secretly shares his/her update with servers after the training is completed; otherwise, the client should do extra two steps. Under the unencrypted framework setting, the server broadcasts the clipping boundary in each round. Once the client receives and compares its update's norm with the clipping threshold, he/she decides whether to clip the update or not and sends the indicator b with added noise back to the server. If the update is clipped, the indicator is 0; otherwise, the indicator is 1. Besides, the weight difference will be added by Gaussian noise and sent back to the server. The amount of Gaussian noise is equivalent to the original noise added from the server-side but divided into n parts for n clients. Since the sum of Gaussian distribution is Gaussian distribution and the post-processing property of DP noise, (ϵ, δ)-DP is still provided after the aggregation.

4.3.3. Evaluation

The idea of splitting the model into local and global is originally from Ditto [61]. For each client, we request them to build two models, one for training and communication with others and another for training and self-use. Rather than directly loading the weight difference aggregated by the server, the local model learns from the global model by $l2$-regularization controlled by the coefficient λ_{ditto}. Different from [61] where λ is a fixed value, we use performance-based evaluation to make λ_{ditto} adaptive. As the clients are data holders, they can test the model performance, e.g., predicting accuracy, to deduce whether the global model is admirable for approximating.

Furthermore, varying from [19, 84, 85, 91] where the server fully takes the responsibility of a robust model, clients share that responsibility locally in our setting. Due to the pre-validation, the local model will not learn from the global model if the global one is found being poisoned. For some attacks, they can escape from the server's check at the initial few iterations or the converging stage. Though some are temporary effects, Ditto can prevent them from influencing the local model. Notably, the metric of test-based performance can vary, not limited to the general accuracy. This defensive scheme, to some extent, also offers personalized to users. In figure 4.3, readers may notice that an interface is left for "private data", meaning that clients can also feed the data to the local model, which they do not want to share with others. The local model stands alone from the global model and is only for self-use.

The algorithm of FLVoogd's client is manifested in algorithm 5 and algorithm 6.

Algorithm 5 FLVoogd - client (1)

```
1: **Input**:
2: $\mathcal{D}_{train}$, $\mathcal{D}_{eval}$    ▷ $\mathcal{D}_{train}$ is the training set, $\mathcal{D}_{eval}$ is the testing set
3: $W_{local}$, $W_{global}$, $F$    ▷ $W_{local}$/$W_{global}$ are the local/global parameters, $F$ is the objective function
4: $\eta_{local}$, $\eta_{global}$, $E$    ▷ $\eta_{local}$/$\eta_{global}$ is the learning rate for the local/global model, $E$ is the number of local training epochs
5: $\lambda_{ditto}^{(0)}$, $\eta_{ditto}$, $acc_{thres}$, $\lambda_{min}$, $\lambda_{max}$    ▷ $\lambda_{ditto}^{(0)}$ is the initial value for $\lambda_{ditto}$, $\eta_{ditto}$ is the learning rate for $\lambda_{ditto}$, $acc_{thres}$ is the threshold to start learning, $\lambda_{min}$/$\lambda_{max}$ is the minimum/maximum learning rate of $\lambda_{ditto}$
6: **Output**: $w$    ▷ $w$ is the weight difference between before and after training
7: $\tilde{w}_{global} \leftarrow client(receive)$    ▷ receive the update from the server
8: $W_{global} \leftarrow W_{global} + \tilde{w}_{global}$
9: $W_{temp} \leftarrow$deepCopy($W_{global}$)
10: $acc_{ref} \leftarrow$Eval($W_{global}$,$\mathcal{D}_{eval}$)
11: **for** local epoch $e$: $1, 2, ..., E$ **do**
12:     **for** batch iteration $\mathcal{B} \in \mathcal{D}_{train}$ **do**
13:         $W_{global} \leftarrow W_{global} - \eta_{global} \nabla F(W_{global}, \mathcal{B})$    ▷ train global model
14:         $W_{local} \leftarrow W_{local} - \eta_{local}(\nabla F(W_{local}, \mathcal{B}) + \lambda_{ditto}(W_{local} - W_{global}))$    ▷ train local model
15:         $acc_{local} \leftarrow$Eval($W_{local}$,$\mathcal{D}_{eval}$)
16:         $\lambda_{ditto} \leftarrow \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres})$ by eq. (4.13)
17: $w \leftarrow W_{global} - W_{temp}$
18: **return** $w$
```

Algorithm 6 FLVoogd - client (2)

```
1: **Input**: $w$    ▷ clients will process $w$ according to the server framework
2: $||w|| \leftarrow \sqrt{\sum_j w_j^2}$
3: **if** Encrypted FLVoogd Server **then**
4:     $\overline{w} \leftarrow \frac{w}{||w||}$
5:     $\overline{v} \leftarrow \frac{\{w_j : j=k,...,m\}}{\sqrt{\sum_{j=k}^{m} w_j^2}}$    ▷ $k$ is the starting index of the last layer
6:     $client(send) \leftarrow \overline{w}, ||w||, \overline{v}$
7: **else**
8:     **if** $||w|| > c$ **then**
9:         $w \leftarrow w \cdot \frac{c}{||w||}$, $b \leftarrow 0$    ▷ $b$ is the indicator
10:     **else**
11:         $b \leftarrow 1$
12:     $w \leftarrow w + \mathcal{N}(0, \frac{\sigma^2 c^2}{n})$, $b \leftarrow b + \mathcal{N}(0, \frac{\sigma_b^2}{n})$    ▷ add Gaussian DP noise component
13:     $client(send) \leftarrow w, b$
```

5 Experiment

5.1. Experimental Setup

We conducted all the experiments using PyTorch [90] and the source code was available on GitHub[1].

Datasets and Neural Network. We followed the recent research on Byzantine attacks [33, 109] on FL and chose a typical application scenario - image classification. The datasets in our experiments included MNIST [59], CIFAR-10 [56], and EMNIST [22]. MNIST is a grayscale digit image classification dataset with 10 categories consisting of 60,000 training and 10,000 testing instances. CIFAR-10 is a color image classification dataset that includes 50,000 training examples and 10,000 testing examples. EMNIST is an extending MNIST to handwritten letters dataset constituted by 62 unbalanced classes with 814,255 characters. The non-IID data splitting for MNIST and CIFAR-10 in our experiment followed the method carried out in FLTrust [19], where Deg_{nIID} (q in [19]) controlled the level of non-IID. Specifically, to split clients into I groups, a training instance with label i is assigned to group I_i with probability $q > 0$ and to other groups with probability $(1-q)/(|I|-1)$. In terms of EMNIST, we applied the method from [110] where the smaller α_{sim} was, the more tasks were dissimilar. CNNs were used as our global and local models, where CIFAR-10 was trained by ResNet-20 [46] (269,772 parameters in total), and MNIST & EMNIST were trained by a $2\times$convolutional layers' NN shown in table 5.1.

Table 5.1: $2\times$CNN architectures

Layer	Size (MNIST)	Size (EMNIST)
Input	(1, 28, 28)	(1, 28, 28)
Conv2d+Sigmoid	(1, 6, 5)	(1, 6, 5)
MaxPool2d	(2, 2)	(2, 2)
Conv2d+Sigmoid	(6, 16, 5)	(6, 16, 5)
MaxPool2d	(2, 2)	(2, 2)
Linear+Sigmoid	(256, 120)	(256, 120)
Linear+Sigmoid	(120, 84)	-
Linear	(84,10)	(120, 62)
#Parameters	44,426	40,914

Evaluation Metrics. Main Task Accuracy (MA) represents the accuracy of a model tested by its benign task. It indicates the fraction of correct predictions. Untargeted-attack adversaries aim to reduce MA, while the defensive mechanism should not be designed to influence MA. If the model is under targeted attacks, Backdoor Accuracy (BA) is the metric to reflect how successful the adversaries are. It denotes the fraction of correct predictions for backdoor samples. Adversaries try their best to augment

[1]https://github.com/Timo9Madrid7/maliciousfl

BA, while a defensive mechanism should be designed for deflating BA.

FL Configuration. The total number of clients N was set to 100. Each client received unique training samples and testing samples from the split. ResNet-20 was a pre-trained version[2] to accelerate the training process, while other networks were not pre-trained. The learning rates of global model η_{global} and local model η_{local} were 0.01. The local training epoch E was 1 since clients did not hold an adequate number of samples. The coefficient of $l2$-regularization λ_{ditto} was initialized as 0 and its min-max interval was $[0.0, 2.0]$, where the maximum was suggested by [61]. The threshold acc_{thres} for the local model starting learning from the global was 0.05. The learning rate η_{ditto} for λ_{ditto} was 1. The expected clipping ratio γ, the initial clipping boundary $c^{(0)}$, the clipping learning rate η_c were 0.5, 10, 0.3. Other settings varied for different experiments.

Malicious Configuration. Both model poisoning attacks and data poisoning attacks share a parameter, Poisoned Model Rate (PMR), indicating the fraction of poisoned models **for each round**. If the attacking type is data poisoning, it has one more parameter, Poisoned Data Rate (PDR), representing the poisoned ratio of the data. Due to our assumption, PMR was less than 50% in our experiments. The adversary should carefully select PDR because they are not the higher, the better. Higher PDR can impact BA more adversely, but adversaries' uploads can be detected and filtered out more effortlessly. Settings of attacking parameters, such as PDR, in section 5.2.1, are nearly marginal thresholds, below which the attacking effect is non-significant even if it escapes from being detected. In terms of PMR, it is set with a rigorous value, approximately 50% following the assumption. Common attacking settings are shown in table 5.2.

Table 5.2: Attack settings

Attacks	(E)MNIST	CIFAR-10
A1	mean = 0, std = 1	
A2	Krum's ϵ = 10^{-3}, threshold = 2×10^{-2}	
A5	a 5x5 white square is inserted into the targeted data and labeling it as "0"	the semantic pattern is a car with stripes and labeling "car" to "bird"
A6	by adding Ardis_IV to training and labeling "7" as "1"	by adding Southwest Airline images to training and labeling "airplane" as "truck"

5.2. Experimental Results

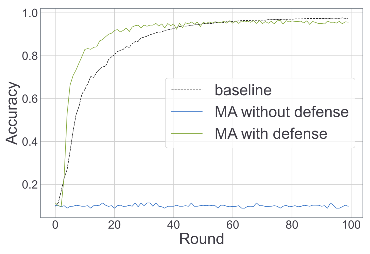
This section will evaluate FLVoogd against different attacks outlined in chapter 3 and demonstrate that our defense mechanism is robust to those byzantine attacks under different malicious configurations. In addition, it will also present the effect on MA because of adding DP noise. Finally, it will assess the impact of the degree of non-IID data on the defensive scheme.

5.2.1. Fending off byzantine attacks

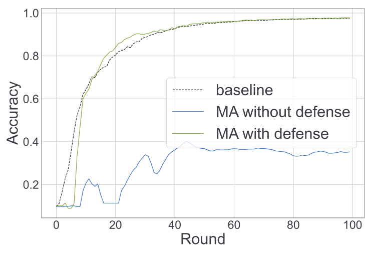
Byzantine attacks in Chapter 3 are separately tested on MNIST, CIFAR-10, and EMNIST, shown in figure 5.1, figure 5.2, and figure 5.3 respectively. All the experiments were run at least five times in case the defense effectively but occasionally works. Results of MNIST will be used as a paradigm for the analysis, and the analysis and conclusion for other datasets are similar.

Figure 5.1 shows MNIST under different attacks from A1 to A6. Training and testing samples are IID split in advance, and each sample merely belongs to one client, such that clients hold unique IID datasets. "Baseline" or "without defense" results from the server ruining naive aggregation. Except for the baseline, where 21 clients are chosen for each round, 40 clients are uniformly randomly selected from 100 participants for other situations. The number of benign clients keeps at 21 to make the result comparable. In figure 5.1a, due to malicious random uploads, the aggregated updates are meaningless, leading to the blue curve with extremely low accuracy; however, the filtering process conducts so effectively that the learning curve - the green one - can behave normally under this attack.

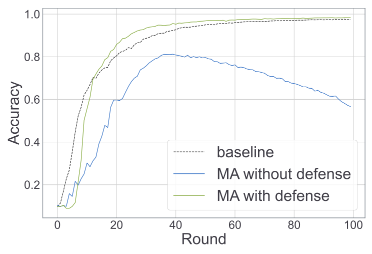
[2]https://github.com/chenyaofo/pytorch-cifar-models



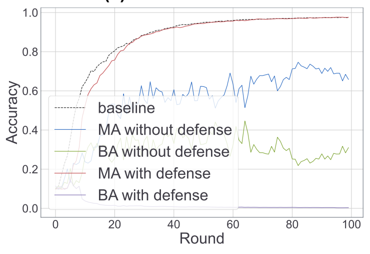
(a) A1: PMR=19/40



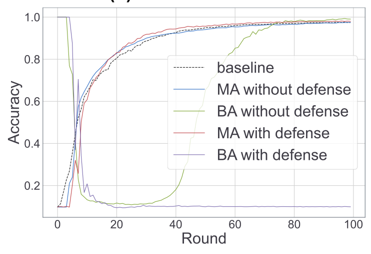
(b) A2: PMR=19/40



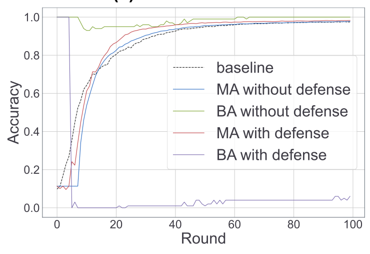
(c) A3: PMR=19/40



(d) A4: PMR=19/40, PDR=1.0



(e) A5: PMR=19/40, PDR=0.2



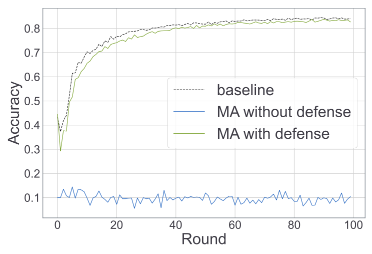
(f) A6: PMR=19/40, PDR=0.33

Figure 5.1: Byzantine attacks on MNIST

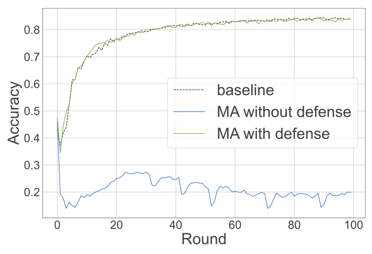
In figure 5.1b, the Krum attack tries its best to upload counter-directional updates to devalue the global accuracy. Consequently, the global accuracy is even worse than a random guess (50%). However, the global accuracy can reach an original level using the defense of FLVoogd. In figure 5.1c, the trimmed-mean attack starts to degrade the model accuracy approximately at midterm and reduces accuracy from higher than 80% to less than 60% within 50 rounds. According to the result, FLVoogd prevents this malicious reduction of accuracy effectively. In figure 5.1d, byzantine clients flip the labels of all training samples, rendering the final prediction like a random guess. Thanks to the honest majority, MA can be slightly better than 50%, and BA can be marginally worse than 50%; still, the result is unacceptable unless the defense is introduced. One notable point is that the filter cannot correctly recognize flipping uploads in several initial rounds, but it can acknowledge and expel malicious updates once the global model learns a little from those majorities. In figure 5.1e, the backdoor triggering attack shows its supremacy at the initial stage, and BA can easily approximate to 100% in the first several rounds. After the model learns sufficient benign samples, BA tends to decrease while MA tends to increase. Without the defense, BA again grows sharply at midterm and finally attains relatively high accuracy. In contrast, under the protection, BA is restrained at low accuracy and can impossibly revive after the filter starts to work. The filter does not work at the initial stage because the model is chaotic, and the updates produced from the model reveal little information about the data distribution. In figure 5.1f, under no defense, BA manifests likewise A5 at the initial stage but does not decline after MA rises. Since the malicious data is sampled from another dataset without intersection with MNIST, learning from the benign samples cannot benefit the model, so BA persists at high accuracy. However, FLVoogd can successfully detect and block these uploads according to the abnormal data distribution, preventing the intrusion of edge-case.

Figure 5.2 shows CIFAR-10 under different attacks from A1 to A6. Similar to MNIST, all the attacks have been eliminated or restrained below comparably low accuracy, but there is a noteworthy point in figure 5.2e, where BA's learning curve notably fluctuates no matter whether there is defense or not. It is because the backdoor images are still the original images but with a naturally specific pattern, different from MNIST, where a pattern is manually added to make the backdoor image vary from the origin. Thereafter, these minor samples are forced to be labeled as another, like label flipping applied to partial images from one class.

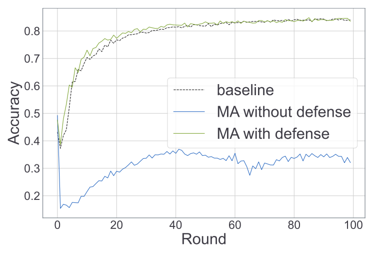
Figure 5.3 shows EMNIST under different attacks from A1 to A6. Similar to MNIST and CIFAR-10, all the attacks have been eliminated or restrained below comparably low accuracy. However, we initially found that FLVoogd could not prevent EMNIST from A6 productively. Firstly, we did not use the whole dataset for the training but followed the advice from [45], where 20% was suggested for 100 clients.



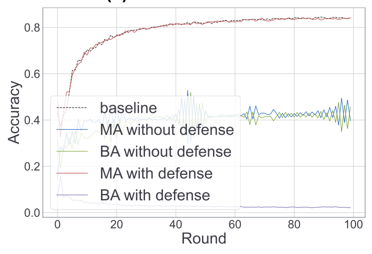
(a) A1: PMR=19/40



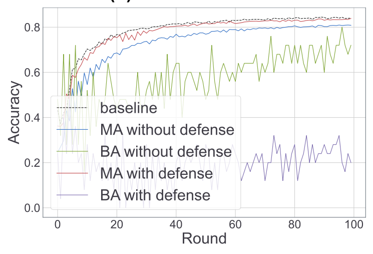
(b) A2: PMR=19/40



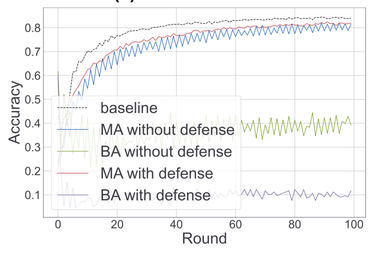
(c) A3: PMR=19/40



(d) A4: PMR=19/40, PDR=1.0



(e) A5: PMR=19/40, PDR=0.33



(f) A6: PMR=19/40, PDR=0.33

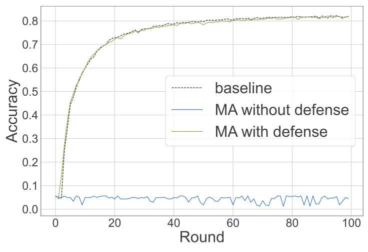
Figure 5.2: Byzantine attacks on CIFAR-10

In addition, there were 62 classes to be classified. Consequently, the model initially required several rounds to figure out what correct “7” and “1” roughly looked like. The model would not rebound those edge cases if edge-case clients instructed the model incorrectly with the mislabelled pictures at the beginning. Therefore, in the first five rounds, we put the model under training with benign-only samples to compensate for this unfairness. After that, the model could successfully filter out those malicious uploads. Nonetheless, pre-training is not mandatory for other scenarios.

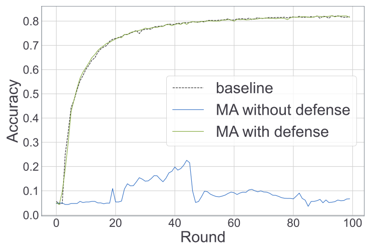
5.2.2. Adding and tracking DP noise

The experiments in subsection 5.2.1 were not added by Gaussian DP noise. As shown in figure 4.1, the DP Noise Adder is independent from the filtering procedure. Therefore, experiments that test the DP effect and monitor the ϵ budget will be independently studied in this subsection. The experiments consist of different combinations of subsampling ratio q and noise strength coefficient σ. The DP noise, to some extent, will adversely affect the convergence and accuracy of the model. In return, this kind of sacrifice gains a differential privacy guarantee. The growth of ϵ after each iteration is estimated by Moments accountant or RDP, where δ is set as a constant ($= 10^{-3}$ [38]) considering 100 is the total number of clients.

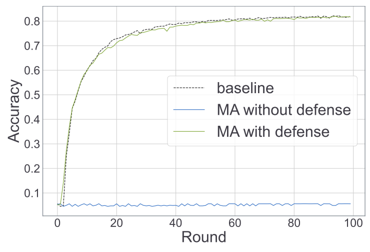
The consequence of DP Gaussian noise is tested by MNIST and shown in figure 5.4. We kept the ratio of noise strength σ and the chosen number of clients for each round n constant, $\frac{\sigma}{n} = 1/20$. For each experiment, we changed the subsampling rate q from 0.2 to 1.0. The total number of training rounds extended to 200, as the noise postponed the convergence. Figures 5.4a, 5.4b, 5.4c, 5.4d, 5.4e nearly exhibit an identical trending. In general, the added noise decreased the final accuracy by 7.0%. In return, almost in all the experimental settings, (ϵ, δ) was better than $(20, 10^{-3})$, and the best one can achieve $(13.29, 10^{-3})$ estimated by RDP. The subsampling did not play an important role in significantly moderating the growth of ϵ, as the total number of clients was inadequate. The subsampling could be more effective if 1,000 or 10,000 clients participated in the FL training [38]. Solid lines and dash lines in figure 5.4f are ϵ estimated by Moment’s accountant and RDP, respectively, where RDP always provides a lower ϵ’s boundary compared with Moments account. Adding more noise can provide a stronger differential privacy guarantee but may harm the final model accuracy and slow the converging speed. The trade-off between ϵ and accuracy is a challenging problem [28]. The FLVoogd framework provides an adaptive supervisor for ϵ, which the server can customize, so the server can stop the training or adjust the sampling ratio and the amount of adding noise in time once ϵ is undesirable.



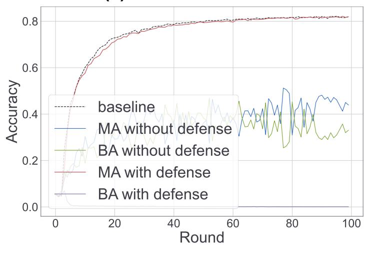
(a) A1: PMR=19/40



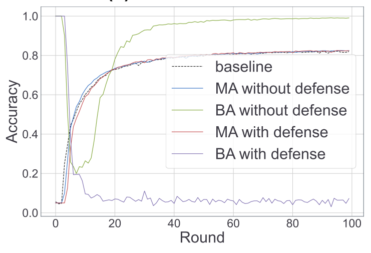
(b) A2: PMR=19/40



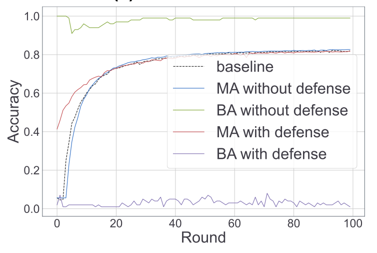
(c) A3: PMR=19/40



(d) A4: PMR=19/40, PDR=1.0



(e) A5: PMR=19/40, PDR=0.2



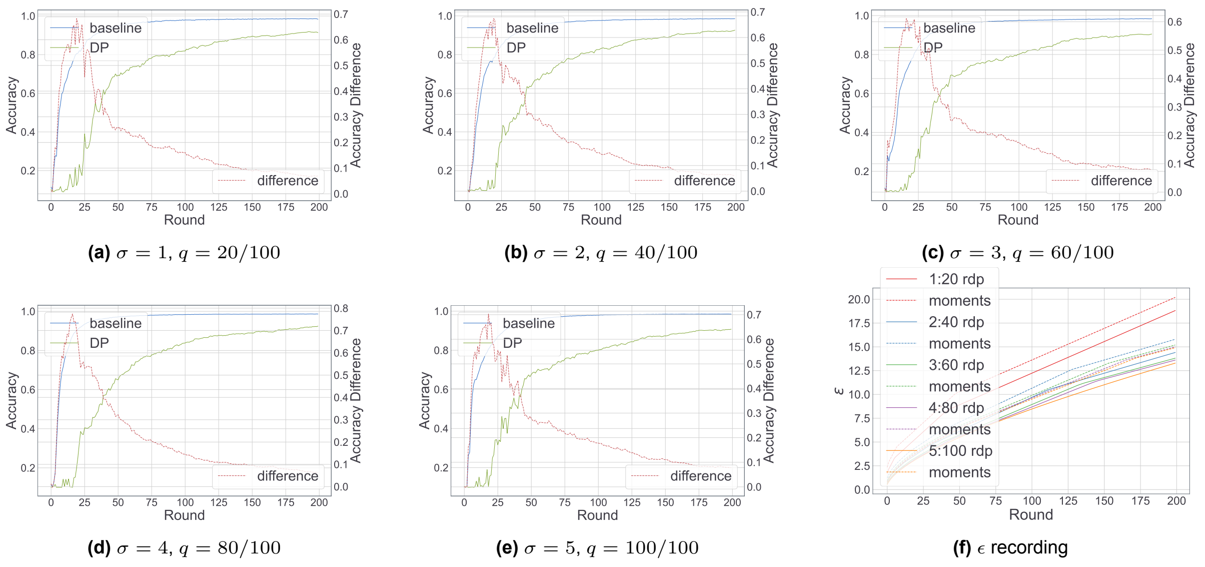
(f) A6: PMR=19/40, PDR=0.33

Figure 5.3: Byzantine attacks on EMNIST

5.2.3. Non-IID interference

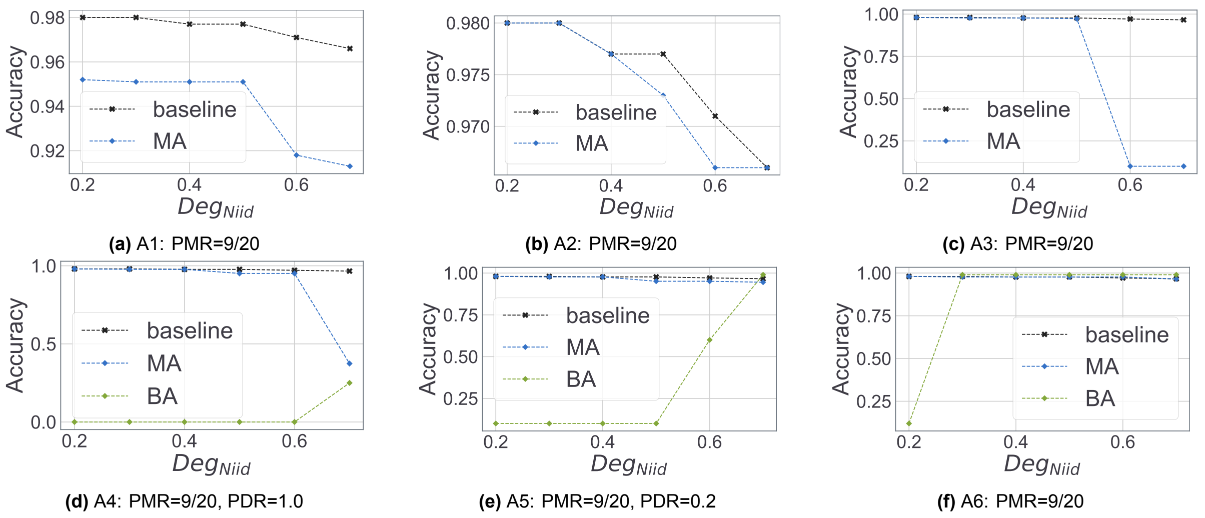
FLVoogd successfully enervates or eliminates the model's attacking effect when clients hold IID datasets. All benign models behave similarly with IID datasets and upload relative weights' updates. However, the defense can be more problematic if those datasets are non-IID because the filter may not distinguish whether the poisoning or the non-IID data cause the directional differences. Using non-IID data to train the model leads to the deviation of the local model's updates. For example, if one class dominates the training set, the backward propagation may adjust some neurons' parameters more than the others. In this subsection, a non-IID data split is used for testing the robustness of FLVoogd under different attacks. The training and testing samples will be split into non-IID datasets. Experiments will test Deg_{nIID} from 0.2 to 0.7. The number of clients per round is reduced from 40 to 20 since DBSCAN is unstable with noise points where the non-IID noise may connect clusters. The defense may collapse if too many clients are selected per round under the non-IID setting.

The defensive effect under the non-IID setting has been tested on the MNIST dataset, shown in figure 5.5. In general, FLVoogd cannot completely tolerate that clients hold extreme non-IID samples. In figure 5.5a, adversaries perform random uploading chosen from $\mathcal{N}(0, 1)$. The filter rejects all these uploads until the model converges. After convergence, the filter hardly distinguishes between the malicious and model uploads because both appear somewhat random behaviors, resulting in fluctuations in the convergence state. However, the accuracy is still above 90% since benign uploads again become meaningful once below this threshold, and the filter once more rejects malicious random uploads. Krum attack shows ineffectual regardless of the non-IID degree in figure 5.5b. The filter is so sensitive to the direction of uploads that uploads with reverse direction can scarcely pass the filtering. In figure 5.5c, the filter relinquishes its duty after the non-IID degree is larger than 0.5. Compared to A1, A3 is a more advanced scheme, which chooses the randomness adaptively, so the attacking effect is more significant. In figure 5.5d, random flipping works after the non-IID ratio is higher than 0.6. After $Deg_{nIID} > 0.5$, one class completely dominates a dataset, leading that each mini-batch iteration contains over 50% samples from the same class. Consequently, the learning process is tampered with by the flipping of one class intermittently once the non-flipped samples of this class miss the training round. In figure 5.5e, backdoor accuracy cannot be constrained if increasing the non-IID degree to more than 0.5, as the filter cannot discriminate whether the non-IID or the backdoor targets cause the directional difference. This situation similarly happens in figure 5.5f where the result is even worse because the defense collapses when the non-IID ratio is higher than 0.2. Contradicting A5, where the backdoor targets are still the samples in the dataset, A6 introduces the backdoor targets from another dataset and aims to compromise the weakness of the model prediction. The filter relies on the model digest. In other words, the filter performs ineptly if the model digest cannot reflect normal/abnormal directions.



(a) $\sigma = 1$, $q = 20/100$ **(b)** $\sigma = 2$, $q = 40/100$ **(c)** $\sigma = 3$, $q = 60/100$ **(d)** $\sigma = 4$, $q = 80/100$ **(e)** $\sigma = 5$, $q = 100/100$ **(f)** ϵ recording

Figure 5.4: DP's effect on MNIST



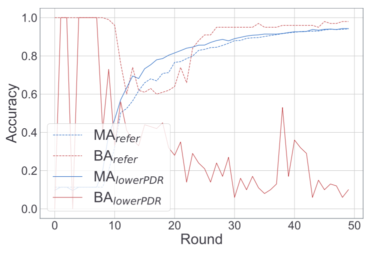
(a) A1: PMR=9/20 **(b)** A2: PMR=9/20 **(c)** A3: PMR=9/20 **(d)** A4: PMR=9/20, PDR=1.0 **(e)** A5: PMR=9/20, PDR=0.2 **(f)** A6: PMR=9/20

Figure 5.5: Non-IID effect on MNIST

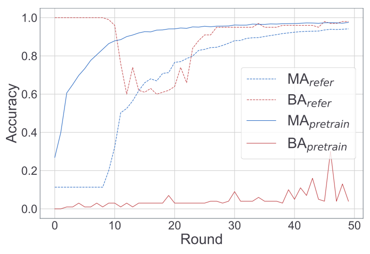
Since the model can never learn those edge cases with true labels, the model cannot provide evidence of abnormal behaviors. When the non-IID ratio is lower than 0.3, the filter can detect those edge cases mainly because of the distribution of uploads. However, after non-IID increases, the upload lacks this kind of information.

Since the FLVoogd performed worse when it suffered from A6's attack, we selected this situation for further study. We wanted to assist FLVoogd somewhat - training the model ahead or decreasing the PMR. As mentioned, pre-training is doable for some application scenarios. In addition, according to [96], PMR≈ 50% is a very pessimistic assumption, so we tried to lower it a little bit to see how our framework would react.

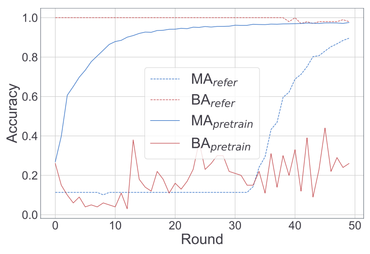
In figure 5.6, it shows more experiments for improving the defense performance. In figure 5.6a, PDR is reduced from 45% to 30%, and the BA learning curve declines once the model has learned the correct direction from the benign uploads. The poisoning effect is weakened because of the lower PDR. In figure 5.6b and figure 5.6c, the model accuracy is trained approximately to 25% before the attacks deploy. The updating directions of models become consistent after the pre-train. Thus, the filter can sift those malicious uploads once it first time meets the upload in an abnormal direction. The results also



(a) $Deg_{nIID} = 0.3$: lowering PDR to 6/20=30%



(b) $Deg_{nIID} = 0.3$: pretraining the model to accuracy around 25%



(c) $Deg_{nIID} = 0.5$: pretraining the model to accuracy around 25%

Figure 5.6: Pretraining and a lower PDR help the defense

verify that defending against targeted attacks depends on the performance of models on the dataset. If the model can separately recognize the poisoned and normal samples, it can output distinguishable model updates. Then, after the filter captures this variance, the defense effectively works.

6 Related Work

Since its inception, Federated Learning has been extensively researched because of its interdisciplinary and pragmatism. The application domain of FL is diverse, as mentioned in chapter 1, and the FL structure is flexible and can be applied in either B2B or B2C modes. Besides, constructing a robust and privacy-preserved framework requires the knowledge of cryptography, machine learning, and deep learning. Many scientists from different fields have participated in this research in the recent half-decade. Furthermore, the commonly used assumption of FL - clients are honest-majority [96] - renders the defense strategies applicable. This chapter will present some related state-of-art or classical FL frameworks and summary their performances.

6.1. Poisoning Defense

This section will provide other research relevant to the defense of poisoning in FL. Those approaches can generally be categorized into statistical distance-based filtering, prior performance testing filtering, and unsupervised machine learning clustering filtering.

Statistical distance-based filtering: The statistical distance-based approach relies on the distances of uploads to distinguish the potential malicious ones from the benign ones. Filtering by distance is relatively efficient because it does not require the server to acquire additional knowledge of the dataset. Section 3.1 analyzes that the uploaded parameters are the high summary of the individual data, so the distances of parameters can be used as a criterion to classify the latent malicious uploads. These methods include but not limit to Krum [13], Geometric Median [73, 115], Auror [97], FoolsGold [36], and AFA [77]. However, these methods have a common issue: they all strongly assume the distribution of clients' datasets. In terms of Krum, Geometric Median/Mean, and Auror, they assume that the datasets of benign clients are almost IID, so they have nearly identical behaviors, while malicious clients behave abnormally to degrade the model. This kind of filter normally selects a reference. Clients who are too far away from the reference will be regarded as potentially malicious. In contrast, the authors of Fllods-Glod and AFA assume that all the malicious clients have a general-purpose, e.g., flipping one class' label, so they behave identically, while benign clients will behave not exactly the same and contribute to each class equally. Besides, statistical distance-based approaches do not have the resilience to the adaptive attack. For instance, section 3.2.2 presents adversaries of Krum and Geometric Mean who can tamper with the model by colluding with each other. Other methods, like Ditto [61], aim to keep a distance between the local and global models to safeguard the local predicting accuracy. However, the gap is not easy to control. Under some extremely pessimistic situations, e.g., the global model is wholly compromised, even learning a little bit from the global model, the local model will be adversely influenced.

Prior performance testing filtering: Prior performance testing requires that the server or participants have some prior knowledge of a correct model. For instance, if the server has some prior knowledge, e.g., a root validation dataset [19], the server can test the upload parameters on the validation set to find malicious ones that degrade the model's accuracy. This root dataset is sometimes feasible for the server. Google hired some workers to type using Gboard to collect data as a root dataset [69]. However, not every server has the ability to collect enough data for the root dataset, and data often includes personal information that may betray the regulations of GDPR [118]. In addition, the validation process occupies the extra time of FL, leading to lower communication efficiency, especially when the number of clients is tremendous. The alternative method is to let clients test the performance of the aggregated model, such as Baffle [4], where partial clients receive the aggregated model and vote to accept or abort the model, which depends on the performance of the global model on the local dataset. However, backdoor attacks are still hard to detect, especially under the Non-IID setting, if the local number of data is not adequate.

Unsupervised machine learning clustering filtering: Similar to distance-based filtering, clustering also computes distances among uploads, employing the unsupervised ML algorithm to handle the distance matrix. It tries to extract feature variance from different uploaded parameters and uses the variances to distinguish malicious and benign clients, such as [84, 85, 91, 120] to name a few. Those papers commonly employ HDBSCAN/DBSCAN introduced in section 2.3 because density-based clustering is suitable under this scenario where the number of clusters is unknown in advance. Besides, the distance measuring metric is flexible because HDBSCAN/DBSCAN accepts precomputed distance matrix. As parameters in each layer can reflect different properties of the dataset, e.g., the last fully-connected layer reveals the distribution of a dataset [91], the filter can process selected layers individually. Furthermore, the procedures of DBSCAN are able to be encrypted so that the output labels can be computed privately.

6.2. Privacy Preserving

Depending on the adversary, either participants or the server, the defending strategies can roughly be divided into two classes, DP based approach and encryption based approach. For the sake of simplicity, both homomorphic encryption and secret sharing are classified as the encryption based approach.

Differential Privacy based approach: DP is an efficient tool to defend membership inference from the client-side and provides a quantified privacy budget. Clipping and adding Gaussian noise are one of the known techniques to achieve differential privacy for deep learning proposed in [80]. By adding the appropriate amount of noise, the honest-but-curious clients cannot distinguish whether a specific person has participated in the FL training or not. At the same time, the paramount task accuracy is not significantly destructed. One ramification of this field is to explore different accountant methods to quantify the privacy loss introduced in section 2.2 so as to lessen the added noise. Another is to lower clipping bound to add noise, such as the adaptive clipping proposed in [5], so as to reduce the magnitude of the added noise. Still, there are other methods to achieve DP. In [63], authors used Canonical sketches to instigate the errors to achieve DP while reducing the communication bandwidth.

Encryption based approach: Encryption is a productive tool to defend the inference from the server-side because once the parameters are encrypted, the server cannot individually observe each upload. Encryption can be performed via either homomorphic encryption or secret sharing. If the server adopts the vanilla FL framework shown in algorithm 1, any encryption scheme with additive homomorphism can be used for the encryption. Secure aggregation requires more than addition; thus, fully homomorphic encryption is often needed when the algorithm operates both addition and multiplication. However, homomorphic encryption causes severe computational overhead, e.g., the lattice-based homomorphic encryption used in Poseidon [94]. In this case, some studies such as [67] try to send the encoded signs rather than the parameters to the server. Nevertheless, the computational overhead is still not significantly improved due to the latency of convergence. Secret sharing is the substitution of homomorphic encryption and has a relatively lower computational overhead, but the cost is the communication rounds [9, 99, 100]. Furthermore, both methods face the issue that clients may drop off due to network congestion or local shutdown. The developers need to design the protocol to deal with such issues carefully; otherwise, the secret parameters cannot be reconstructed. To avoid these issues, instead of using SMPC, some scientists use S2PC [27, 85], where clients send one secret share to the aggregator and another to the external server, which can also be regarded as a third-party supervisor. Consequently, the remaining calculations can be cooperatively done by only two parties - the server and the external server. In addition, the original communication costs from clients are reduced. Recent state-of-art works have also employed this framework, such as [84, 91].

6.3. Other

Many other prospective fields are relevant to FL but not included in this research. This section will present some of them. On the one hand, those studies play an essential role in FL, especially in the application domain. On the other hand, those researches are provided as hints to inspire the readers who want to continue this study.

Fairness: Fairness is another popular field in FL. Referencing an example from [66], several banks collaboratively train an FL model, but one of them owns much more clients' data than others'. It will generate a conflict that the larger bank, which contributes more than other banks during the training, gains fewer returns from the joint training, while small banks benefit more from the model. Consequently, there should be an incentive policy to motivate the participants privately share more data, and in return, the more they share, the more rewards they will get. There are already some papers [62, 66, 123] aiming to solve this kind of issue.

Personalized and Non-IID: Statistical heterogeneity due to non-IID datasets is a cumbersome problem in FL. Some non-IID dataset holders may receive the global model that performs worse locally since their true model is far from the aggregated model. Consequently, contributors do not have identical accuracy on their local datasets, which breaks the promise of fairness. Meanwhile, the parameters behave differently because of deriving from non-IID datasets. As a result, distinguishing malicious parameters from uploads is more complicated. Besides, learning from non-IID datasets influences the convergence of the global model and makes the analysis even harder. Some researches come up with some methods to tackle this issue, e.g., [32, 43, 95, 103]. It should be noted that those scientists who are dedicated to solving non-IID in FL, to some extent, also try to solve the fairness simultaneously due to the common goal - identical local accuracy.

Communication efficiency: FL is a kind of decentralized machine/deep learning, requiring network communication between parties. In addition, FL is also a popular application for IoT devices, the communication efficiency of which is highly demanding [48]. The commonly used strategy is to decrease the information entropy, in other words, to lower the communication bandwidth, such as Sketch [63] mentioned in previous sections and signSGD [11]. Sketch uses Canonical sketches where the gradients are quantized to lower-precision values, while signSGD only requires clients to upload signs of parameters. In addition, some methods use statistical observation from the model parameters to shrink down the number of uploaded parameters. For instance, because the majority of a neural network's parameters are near zero, it is recommended to send only gradients larger than a predefined threshold [102]. Furthermore, by merely broadcasting the last output layer, similar to transfer learning, Bristle greatly minimizes communication overhead [106].

7 Conclusion

In this thesis, we introduce Federated Learning by comparing it with the traditional centralized machine learning. Although Federated Learning claims it is confidential and privacy-preserving, the recent works show that the original paradigm cannot guarantee those because of malicious participants who perform byzantine attacks on the model. In order to filter the byzantine clients out while maintaining personal privacy, we propose FLVoogd, which is a robust and privacy-preserving federated learning framework that restrains the adverse impact of Byzantine attacks within an acceptable level while maintaining the performance of model predictions on the main task.

Our framework applies Differential Privacy to guarantee the aggregated model somewhat deviates from the truth. Hence, adversaries hardly observe the membership's variance from the result. In addition, (H)DBSCAN is used to extract the honest-majority participants for robust aggregation effectively. Meanwhile, secure Multi-party Computation accompanies DBSCAN to make the actual content of uploads invisible to the server, such that the inference from the server side is eliminated. The background knowledge of these three main techniques used from the server side is fully demonstrated. Besides, clients in our framework build dual models, one for the global training and another for the local training, combined with performance-based testing that controls the distance between both models so as to achieve model separation and personalization.

To attack and test the robustness of our framework, we detail six attacking methods, including three untargeted and three targeted attacks. Those attacks can adversely decrease the main task accuracy or achieve a comparably high backdoor accuracy if federated learning adopts a naive aggregation rule. However, FLVoogd can productively defend against those attacks in most situations.

There are two critical differences between our encrypted framework and prior works. Firstly, most procedures are executed adhering to privacy preservation, where operations are doable for mostly popular SMPC protocols. Secondly, we provide adaptive adjustments such that the whole process can run automatically.

The performance of the design is tested in the field of image classification. Experimental results show that our framework can successfully resist state-of-art attacking schemes under an i.i.d. situation. At the same time, it still works effectively when the non-i.i.d. ratio is not such high. The filtering process relies on whether the model can provide a distinguishable and pertinent digest for datasets from different clients.

Future works could include: merging the transfer learning into the current framework to tackle GAN inference and combine it with other efficiently communicative schemes, e.g., sketch, to reduce the communication bandwidth and enhance differential privacy.

Proceedings of Machine Learning Research , 2022

FLVoogd: Robust And Privacy Preserving Federated Learning

Yuhang Tian Y.TIAN-13@STUDENT.TUDELFT.NL
Delft University of Technology

Rui Wang R.WANG-8@TUDELFT.NL
Delft University of Technology

Yanqi Qiao Y.QIAO@TUDELFT.NL
Delft University of Technology

Emmanouil Panaousis E.PANAOUSIS@GREENWICH.AC.UK
University of Greenwich

Kaitai Liang KAITAI.LIANG@TUDELFT.NL
Delft University of Technology

Abstract

In this work, we propose FLVoogd, an updated federated learning method in which servers and clients collaboratively eliminate Byzantine attacks while preserving privacy. In particular, servers use automatic Density-based Spatial Clustering of Applications with Noise (DBSCAN) combined with S2PC to cluster the benign majority without acquiring sensitive personal information. Meanwhile, clients build dual models and perform test-based distance controlling to adjust their local models toward the global one to achieve personalizing. Our framework is automatic and adaptive that servers/clients don't need to tune the parameters during the training. In addition, our framework leverages Secure Multi-party Computation (SMPC) operations, including multiplications, additions, and comparison, where costly operations, like division and square root, are not required. Evaluations are carried out on some conventional datasets from the image classification field. The result shows that FLVoogd can effectively reject malicious uploads in most scenarios; meanwhile, it avoids data leakage from the server-side.

Keywords: federated learning; secure-multi-party computation; differential privacy

1. Introduction

Unlike the centralized learning setting, where a server collects substantial users' data to build a model for predictions, Federated Learning (FL) requires the model parameters that clients train independently with their data and devices. In FL paradigm frameworks, such as FedAvg McMahan et al. (2016), the server iteratively aggregates local models trained by individuals and sends the global one back to clients for their local updates, to achieve collaborative training. Since no actual data is sent to the server, this paradigm was considered privacy-preserving. In the past half-decade, FL has been widely researched and applied in many fields such as image recognition Li et al. (2021a), Natural Language Processing (NLP) Liu et al. (2021), financial system Long et al. (2020), and medical care Dayan et al. (2021).

However, such a framework still faces two main challenges - privacy and security. On the one hand, the conventional FL setting cannot get rid of the disclosure of clients' information and even enlarge the attacking surface Aono et al. (2017). Not only can the server be an adversary to infer the information from local models sent by clients in this scenario, but also every participant can perform the inference attack on the global model constructed by each individual. Therefore, some research employs SMPC to encrypt the uploads, such as Nguyen et al. (2021a). However, the cost of the design or operations is high, especially when dealing with the division and the square root. On the other hand, without countermeasures, adversaries can arbitrarily substitute the data with the poisoned one Wang et al. (2020) or even directly change the upload into a meaningless random number, leading their uploads to betray the regulation. To eliminate the attacking consequence, some research, like Li et al. (2021b) and Rieger et al. (2022), builds a practical defensive framework, but with too unintuitive hyper-parameters to tune for different situations.

To improve the efficiency and save expensive operations, we develop an updated framework that combines SMPC Knott et al. (2021); OpenMined (2021), DBSCAN Ester et al. (1996), differential privacy Geyer et al. (2017), and personalized local model Li et al. (2021b) to eliminate the malicious uploaded parameters without revealing any sensitive information and guarantee (ϵ, δ)-DP after the aggregation. Compared with the past research, our framework 1) filters the abnormal uploads without knowing their sensitive information; 2) performs the training process adaptively, requiring no parameter tuning; 3) uses SMPC operations efficiently supported by most protocols. We leverage the conventional image classification dataset to evaluate the framework. The results show that the filter can reject the Byzantine attacks under most situations without degrading the model performance, and the trade-off between predicting accuracy and DP strength can be customized for different scenarios.

2. Background and Problem Setting

2.1. Adversarial Attack

Six different state-of-art attacking schemes are used to test our FL defenses' robustness and named from A1 to A6.

•**Random upload** (A1): As its name suggests, the adversary substitutes the factual update with a random noise chosen from $X \sim \mathcal{N}(0, 1)$. Consequently, the average of parameters can arbitrarily deviate from $w_{avg} = \frac{1}{n}\sum_{i=1}^{n} w_i$ to $w_{dev} = \frac{1}{m}\sum_{i=1}^{m} w_i + \frac{1}{n-m}\sum_{i=m+1}^{n}\mathcal{N}(0, 1)$, where m is the number of honest updates and $(n - m)$ is the number of malicious updates.

•**Krum attack** (A2): It is designed to crack the Krum aggregation rule. In a nutshell, Krum selects one vector from a set of n vectors that is the most comparable to the rest. Even if a compromised client gives the chosen vector, the impact is limited in this situation. However, when adversaries try to invalidate the Krum aggregation rule, they can conspire to elaborate a set of vectors to make $KR(w_1', ..., w_f', ..., w_n)$ output w_1' such that w_1' mostly inversely differs from the true selected one without being attacked Fang et al. (2020).

•**Trimmed-mean attack** (A3): Trimmed-mean sorts n updates for each j^{th} parameter $sort(w_{1j}, ..., w_{nj})$, eliminates the highest and smallest β amount from the sorted list, and averages the remaining $(n - 2\beta)$ parameters as the global model's j^{th} parameter Yin et al. (2018). To enervate this aggregation rule, adversaries collude to submit deviating models in

the opposite direction that the global model would change in the absence of attacks.
•**Label flipping** (A4): Each adversary converts the label of a sample from l to $L - l - 1$, where l is the truth label of the sample, and L is the total number of classes Muñoz-González et al. (2017). For instance, adversaries label digit "0" as "9" and digit "9" as "0" to label-flip the MNIST data.
•**Backdoor triggering** (A5): This kind of attack is also known as trojan attacks Gu et al. (2017). The adversary inserts a specific pattern into training samples or uses existing ones to render the corresponding testing samples with that pattern classified as the desired class. This pattern functions as a trigger. After the global model learns this pattern, it will be triggered and output the misled prediction. If the adversary uses the existing pattern in the sample, this backdoor attack is a semantic backdoor attack Rieger et al. (2022).
•**Edge-case attack**(A6): Under the edge-case attack setting, adversaries aim to attack the heavy-tail of the prediction Wang et al. (2020). They try to find or manufacture samples that the model predicts correctly but with a comparably low confidence value; then, they label those samples with a label they want. The intuition behind it is that the model cannot assure the correctness of predictions even if the result is correct, as the predicting score is not such high, so it can be easily misled by the attacker who feeds those edge-case samples with wrong labels.

2.2. DBSCAN

DBSCAN is initially designed for clustering and distinguishing the noise from the high dimensional database depending on the variance of density Ester et al. (1996). A non-negligible quantity of samples should be in the cluster if a cluster is formed, while the cluster can hardly be formed in areas where samples are located sparsely. These "depopulated zones" can be used as gaps to separate the different classes and to sift out noisy samples. We will consistently follow some of the concepts and symbols used in Ester et al. (1996). $N_{Eps}(p)$ represents neighbors of a point p within a range with radius Eps (Eps is a preset hyper-parameter). A point p is a *corepoint*, if $|N_{Eps}| \geq MinPts$ ($MinPts$ is a preset hyper-parameter). In addition, a *corepoint* is the centroid of a cluster, so in other words, a cluster is only formed when its centroid is a *corepoint*. A point p is a *borderpoint*, if its neighbours contain at least one *corepoint*. It should be noted that a point can be a *borderpoint* for different clusters, but it will be only assigned to a unique cluster eventually, and it depends on which cluster it assigns the point to first. If a point is neither a *corepoint* nor *borderpoint*, it will be classified as noise.

2.3. SMPC

As mentioned, uploading weights instead of the raw data to the server is not privacy-preserving. As shown in Zhang and Luo (2020), model parameters can disclose some information about individual data. For example, adversaries can use Generative Adversary Networks (GANs) to reconstruct the class representatives from the aggregated parameters. This powerful reconstruction is more harmful if it happens on the server side because the server can steal the class representatives from each individual uploading. To avoid revealing the uploads to the server, SMPC can be used for private aggregation, and the result will only be revealed eventually. Following the structures in Nguyen et al. (2021b,a); Rieger et al. (2022), we will use Secure 2-party Computation (S2PC), a ramification of SMPC, to guarantee that the individual upload will not be plain-text to the server. Under the S2PC setting, each client will not directly send the model parameters to the server but separate the upload into two parts and share one with the server for aggregation and another with the external server. As both servers hold merely one piece of the secret, the secret cannot be known if they do not collude. Based on the secret sharing scheme, each server can do arithmetic operations relying on its own share and through some communication. To achieve this target, two libraries CrypTen[1] Knott et al. (2021) and SyMPC[2] OpenMined (2021) derived from PySyft are used for the experiment. Both of them use secret sharing but with different protocols to achieve S2PC. CrypTen is currently designed only for semi-honest parties, while SyMPC can tolerate minor malicious parties.

2.4. Security Assumption

We primarily consider possible server-side and client-side risks. There are two types of servers in our setting. Firstly, servers can be honest-but-curious who infer the actual data or relevant information from uploads while heeding the regulation. Secondly, if FLVoodg runs under SyMPC-Falcon Wagh et al. (2020), servers can be malicious (minority) who betray the secure aggregation rule and send an incorrect model back to participants. In terms of participants, in each round, less than half of them can be malicious and perform byzantine attacks 2.1 to deteriorate the performance of the global model. In addition, any client can be curious about information from others and performs client-level inference attacks Geyer et al. (2017), inferring whether a particular client participates in the training, given a specific dataset of that client.

3. FLVoogd Overview and Design

3.1. FLVoogd Server

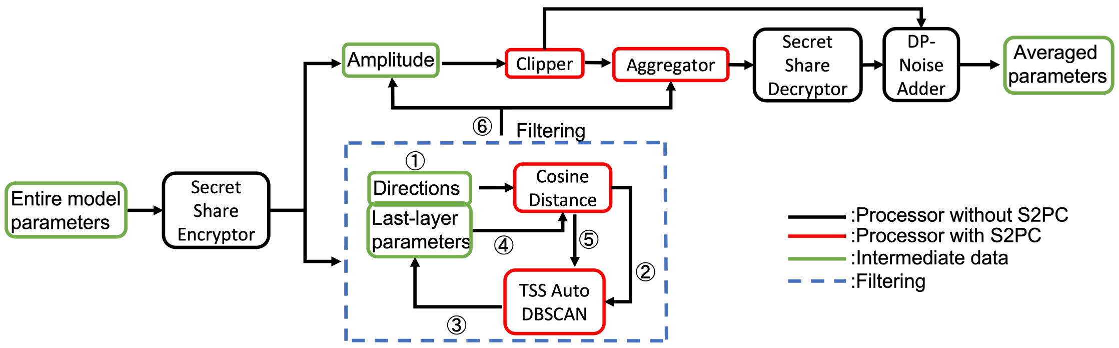


Figure 1: FLVoogd server framework.

1. https://github.com/facebookresearch/CrypTen
2. https://github.com/OpenMined/SyMPC

The overview of our framework, FLVoogd with SMPC, is shown in Fig. 1. Initially, each client computes the $l2$-norm of its uploaded parameters w_i as the amplitude $||w||_i \leftarrow \sqrt{\sum_j w_{ij}^2}$ and unitizes the parameters to $\overline{w}_i \leftarrow \frac{w_i}{||w||_i}$ as the direction. The unitizing can simplify the later SMPC computations, e.g., cosine distance where division and square root operations are saved. Besides, the client performs the same for the parameters from the last layer and obtains the unitized last layer parameters $\overline{v}_i$. The reason for extracting the last layer is that parameters in the final layer reveal more explicit information relevant to the dataset's distribution Rieger et al. (2022) which can be used for distinguishing backdoor uploads. Then, the client secretly shares the direction $\overline{w}_i$, the amplitude $||w||_i$, and last-layer direction $\overline{v}_i$ with two parties, the server for aggregation and external server. If the SMPC protocol uses Falcon, one more server is added and receives the third share from the client.

Once servers receive the uploads from the selected clients, they can perform secure aggregation. It is supposed that the number of received uploads is n, and the clipping boundary for the current round is $c^{(t)}$. The server firstly carries out the filtering process, the block with a light yellow background in Fig. 1, and the procedure can be divided into six steps. ①: In the first step, the server uses the directional vector $\overline{w}$ to compute the cosine distance matrix M_{cos} by Eq. (1), where $M_{cosij} = cos - dist(i, j)$ for $i \neq j$, $M_{cosij} = 0$ for $i = j$, i or j denotes the client's index, and u denotes the parameter's index. Since the vectors are unit vectors, the cosine distance between two vectors can be simplified into a dot production. The computation is collaboratively completed by two/three servers, involving the addition and multiplication among secret shares. ②: In the second step, the server feeds the Total-Sum-of-Square (TSS)-based DBSCAN with the distance matrix from the former step. DBSCAN calculates the TSS by Eq. (2) for each pair of rows to obtain a new distance matrix M_{tss}. M_{cos} provides the directional similarity, while M_{tss} enlarges the variance of counter-directions and narrows the variance of identical directions such that the filter can capture the difference more easily. There are two hyper-parameters for DBSCAN, Eps and $MinPts$. As honest-majority is the basic assumption, $MinPts$ is set to $\lfloor n/2 \rfloor + 1$ and Eps is the average of the median ($\lfloor n/2 \rfloor$) in each row of the distance matrix $M_{tss} \in \mathbb{R}^{n \times n}$. This setting for Eps and $MinPts$ guarantees the DBSCAN can automatically adjust its radius accordingly through the whole process without manual involvement, and the selection makes full use of the honest-majority assumption. This step includes the addition, multiplication, and comparison of shares. ③: In the third step, DBSCAN filters out the noise and minority group and returns indices of the majority group. The server selects the corresponding clients' parameters from the last layer according to the indices. ④: For the next step, similar to step 1, the server again computes the cosine distance matrix but now uses the last-layer parameters. Then, the server obtains cosine distance matrix M'_{cos}. ⑤: The fifth step is identical to the second step. ⑥: In the final steps, DBSCAN outputs the indices that the server will consider as benign.

$$cos - dist(i, j) = \frac{\sum_u \overline{w}_{i,u} \overline{w}_{j,u}}{\sqrt{\sum_u \overline{w}_{i,u}^2} \sqrt{\sum_u \overline{w}_{j,u}^2}} = \sum_u \overline{w}_{i,u} \overline{w}_{j,u} = \overline{w}_i \cdot \overline{w}_j \tag{1}$$

$$tss - dist(i, j) = \sum_u \left(M_{cosi,u} - M_{cosj,u}\right)^2 \tag{2}$$

After knowing which clients are considered as benign ones, the server clips their amplitudes before performs the aggregation by $||w||_i = \min(||w||_i, c^{(t)})$. On the one hand, it restrains the abnormally large amplitude and controls the next descent step size. On the other hand, it provides the $l2$-sensitivity for the DP budget tracer. Furthermore, during the clipping, the clipper records the ratio of clients not being clipped as $\hat{\gamma}$. The expected clipping ratio is set as γ. The next round clipping boundary $c^{(t+1)}$ is updated by Eq. (3) where η_c is the learning rate of the clipper. If the actual non-clipping number of clients is larger than expected, the clipping boundary will decrease to cut more clients in the next round; otherwise, it will increase to be looser. The exponential base guarantees that any adjustment is a positive number. SMPC's operation in this step requires comparing a share with a public number.

$$c^{(t+1)} = c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma} - \gamma)) \tag{3}$$

The aggregation rule is simply averaging benign clients' uploads by Eq. (4). It is supposed that the number of clients after filtering is $m(\leq n)$. After obtaining the merged model w_{global}, servers collaboratively reveal and announce the plain text of w_{global}. The aggregation contains the addition and multiplication of shares, and the multiplication of shares and public numbers.

$$w_{global} = \frac{1}{m}\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)}) \tag{4}$$

The server eventually knows the global parameter till finishing aggregation, and local parameters are already merged into one; thus, the server has no idea of individual local updates. Before sending the global update back to clients, the server adds Gaussian noise to provide a differential privacy guarantee to defend against client-level inference from the client side. The mean is 0, and the standard deviation is the maximum $l2$-sensitivity multiplied by a coefficient σ that represents the strength of DP. Thanks to the clipping, all uploads are bounded into a sphere whose radius is exactly the clipping boundary $c^{(t)}$. Therefore, the noise is added following Eq. (5). Notably, the noise is added to the sum of updates not after averaging. DP-Noise Adder also tracks the DP budget for the server because it knows the number of clients used in this round and the amplitude of Gaussian noise. Finally, $||\tilde{w}_{global}||$ is compared with $||c^{(t)}||$. If $||\tilde{w}_{global}|| > ||c^{(t)}||$, from which the server deduces that the amount of noise is added too much, the server will scale down $||\tilde{w}_{global}||$ to a smaller value by Eq. (6). This operation follows the post-processing property of differential privacy so that (ϵ, δ) cannot be influenced. This post-processing is equivalent to adjusting the model learning rate lower after knowing the noise influences too much on the result. The algorithm of FLVoogd server is manifested in Alg. 1.

$$\tilde{w}_{global} = \frac{1}{m}\{\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)}) + \mathcal{N}(0, \sigma^2 \cdot (c^{(t)})^2)\} \tag{5}$$

$$\tilde{w}_{global} := \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||}) \tag{6}$$

Algorithm 1 FLVoogd server algorithm

```
1:  Input:
2:  $\mathcal{C}$, $N$                                   ▷ $\mathcal{C}$ is the set of clients, $N = |\mathcal{C}|$
3:  $T$, $q$                                ▷ $T$ is the number of training iteration, $q$ is the sampling ratio
4:  $c^{(0)}$, $\gamma$, $\eta_c$   ▷ $c^{(0)}$ is the initial clipping boundary, $\gamma$ is the expected clipping ratio, $\eta_c$ is Clipper's learning rate
5:  $\sigma$, $\delta$                      ▷ $\sigma$ is the coefficient to control the noise strength, $\delta$ is for DP
6:  for round $t$: $1, 2, ..., T$ do
7:      $\mathcal{C}^{(t)}$, $n \leftarrow$ subsample($\mathcal{C}$, $N$, $q$)          ▷ $n = |\mathcal{C}^{(t)}|$
8:      for $client_i \in \mathcal{C}^{(t)}$ do
9:          $\overline{w}_i^{(t)}$, $||w||_i^{(t)}$, $\overline{v}_i^{(t)} \leftarrow client_i(t, send)$     ▷ $\overline{w}$ is the unit vector of weight difference, $||w||$ is the norm of weight difference, $\overline{v}$ is the unit vector of last layer's weight difference
10:     $idx_{f1}^{(t)}$, $n^{(t)\prime} \leftarrow$ Auto_DBSCAN($\{\overline{w}_1^{(t)}, \overline{w}_2^{(t)}, ..., \overline{w}_n^{(t)}\}$) by Alg. 2     ▷ $n^{(t)\prime} = |idx_{f1}^{(t)}|$
11:     $idx_{f2}^{(t)}$, $n^{(t)\prime\prime} \leftarrow$ Auto_DBSCAN($\{\overline{v}_i^{(t)} : i \in idx_{f1}\}$) by Alg. 2     ▷ $n^{(t)\prime\prime} = |idx_{f2}^{(t)}|$
12:     $w_{global}^{(t)} \leftarrow 0$, $\hat{\gamma}^{(t)} \leftarrow 0$
13:     for index $i \in idx_{f2}^{(t)}$ do
14:         if $||w||_i^{(t)} > c^{(t)}$ then
15:             $||w||_i^{(t)} \leftarrow c^{(t)}$
16:         else
17:             $\hat{\gamma}^{(t)} \leftarrow \hat{\gamma}^{(t)} + 1$
18:         $w_{global}^{(t)} \leftarrow w_{global}^{(t)} + ||w||_i^{(t)} \cdot \overline{w}_i^{(t)}$
19:     $w_{global}^{(t)} \leftarrow \frac{w_{global}^{(t)}}{n^{(t)\prime\prime}}$, $\hat{\gamma}^{(t)} \leftarrow \frac{\hat{\gamma}^{(t)}}{n^{(t)\prime\prime}}$
20:     $c^{(t+1)} \leftarrow c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma}^{(t)} - \gamma))$, $\epsilon^{(t)} \leftarrow$ DP_budget($\frac{n^{(t)\prime\prime}}{N}$, $\sigma$, $\delta$)
21:     $\tilde{w}_{global}^{(t)} \leftarrow w_{global}^{(t)} + \frac{1}{n^{(t)\prime\prime}}\mathcal{N}(0, \sigma^2(c^{(t)})^2)$     ▷ satisfying $(\epsilon, \delta)$-differential privacy
22:     $\tilde{w}_{global}^{(t)} \leftarrow \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||})$     ▷ satisfying post-processing
23:     $client_i(t, receive) \leftarrow \tilde{w}_{global}^{(t)}$
```

3.2. FLVoogd Client

Referencing the idea from Ditto Li et al. (2021b), each FLVoogd's client builds two identical models, namely, the global model and the local model. Varying from Cao et al. (2020); Nguyen et al. (2021a,b); Rieger et al. (2022) where the server entirely takes the responsibility of a robust model, clients can share that responsibility locally. This defensive scheme builds a gap between the self-used local and global models to enhance the robustness and offers personalization to users. In each round, the client receives the averaged aggregated weight difference $\tilde{w}_{global}$ from the server and updates the weight of the global model accordingly by $W_{global} := W_{global} + \tilde{w}_{global}$. In contrast, the local model is not updated in this step. After updating the locally global model, the client tests the model accuracy using evaluation data and obtains the testing accuracy acc_{ref}.

Algorithm 2 Auto DBSCAN

```
 1: Input: $W$    ▷ $W \in \mathbb{R}^{n\times m}$ represents $n \times m$ matrix where each row is a client's unit vector from $n$ clients and the dimension of the vector is $m$
 2: Output: $idx$, $|idx|$    ▷ $idx$ is a list of indices of benign clients
 3: $M_{cos} \leftarrow$ CosDist($W$) by Eq. (1)
 4: $M_{tss} \leftarrow$ TSSDist($M_{cos}$) by Eq. (2)
 5: for row $i$: $1, 2..., n$ do
 6:     $median_i \leftarrow$ quickMedian($M_{tss,i}$)
 7: $median \leftarrow \frac{1}{n}\sum_{i=1}^{n} median_i$
 8: $Eps \leftarrow median$, $MinPts \leftarrow n//2+1$
 9: $idx \leftarrow DBSCAN(M_{tss}, Eps, MinPts, precomputed)$
10: return $idx$, $|idx|$
```

The client feeds the partial training data to the global and local models in each mini-batch iteration. It is supposed that there is a coefficient λ_{ditto} to control the distance of the local model from the global model. The objective function of the local model becomes like Eq. (7), where $F(\cdot)$ is the objective function for the global model and originally for the local model. The change in the local model's objective function now is that the client adds an additional $l2$-regularization term to force the local model to approximate the global model. Consequently, the local model can learn from the global model, and the gap between them is constrained by λ_{ditto}.

$$\min_{W_{local}} F'(W_{local}; W_{global}) = F(W_{local}) + \frac{\lambda_{ditto}}{2}||W_{local} - W_{global}||^2 \tag{7}$$

Furthermore, Eq. (7) can be converted into a gradient decent format shown in Eq. (8), where η_{local} is the client's local learning rate. The formula shown in Eq. (8) can be easily implemented by PyTorch where the client extracts the gradient and adds the $\lambda_{ditto}(W_{local} - W_{global})$ term to it before running *optimizer.step()*.

$$g := g - \eta_{local}(\nabla F(W_{local}) + \lambda_{ditto}(W_{local} - W_{global})) \tag{8}$$

Till now, the client has λ_{ditto} as a controller to adjust the learning distance between the local and global models, but how to set an appropriate value λ_{ditto} for the local model? Intuitively, if the global model is admirable and exemplary, we expect the local model to learn as much helpful information as possible from the global model; otherwise, we desire the local model to learn less or even not learn from the global model. Then, the client can use the testing accuracy acc_{ref} as a reference to flexibly adjust λ_{ditto} by Eq. (9). In the formula, λ_{max} and λ_{min} are the maximum and minimum values for λ_{ditto}, η_{ditto} is the learning rate, acc_{local} is the testing accuracy of the local model, and acc_{thres} is the minimum threshold to increase λ_{ditto}. λ_{max} and λ_{min} restrain the coefficient of the $l2$-regularization in a reasonable interval. η_{ditto} controls each mini-batch iteration's growing/decaying speed for λ_{ditto}. acc_{thres} is the threshold to control whether the current global model is worth being learned. In other words, the local model will absorb from the global model, only if acc_{ref} is higher than $acc_{local} + acc_{thres}$. The client secretly shares his/her update with servers after completing the training. The algorithm of FLVoogd's client is manifested in Alg. 3.

$$\lambda_{ditto} := \min(\lambda_{max}, \max(\lambda_{min}, \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres}))) \tag{9}$$

Algorithm 3 FLVoogd client algorithm

```
1: Input:
2: $\mathcal{D}_{train}$, $\mathcal{D}_{eval}$    ▷ $\mathcal{D}_{train}$ is the training set, $\mathcal{D}_{eval}$ is the testing set
3: $W_{local}$, $W_{global}$, $F$    ▷ $W_{local}/W_{global}$ are the local/global parameters, $F$ is the objective function
4: $\eta_{local}$, $\eta_{global}$, $E$    ▷ $\eta_{local}/\eta_{global}$ is the learning rate for the local/global model, $E$ is the number of local training epochs
5: $\lambda_{ditto}^{(0)}$, $\eta_{ditto}$, $acc_{thres}$, $\lambda_{min}$, $\lambda_{max}$    ▷ $\lambda_{ditto}^{(0)}$ is the initial value for $\lambda_{ditto}$, $\eta_{ditto}$ is the learning rate for $\lambda_{ditto}$, $acc_{thres}$ is the threshold to start learning, $\lambda_{min}/\lambda_{max}$ is the minimum/maximum learning rate of $\lambda_{ditto}$
6: $\tilde{w}_{global} \leftarrow client(receive)$    ▷ receive the update from the server
7: $W_{global} \leftarrow W_{global} + \tilde{w}_{global}$
8: $W_{temp} \leftarrow$deepCopy$(W_{global})$
9: $acc_{ref} \leftarrow$Eval$(W_{global}$,$\mathcal{D}_{eval})$
10: for local epoch $e$: $1, 2, ..., E$ do
11:     for batch iteration $\mathcal{B} \in \mathcal{D}_{train}$ do
12:         $W_{global} \leftarrow W_{global} - \eta_{global}\nabla F(W_{global}, \mathcal{B})$    ▷ global train
13:         $W_{local} \leftarrow W_{local} - \eta_{local}(\nabla F(W_{local}, \mathcal{B}) + \lambda_{ditto}(W_{local} - W_{global}))$    ▷ local train
14:         $acc_{local} \leftarrow$Eval$(W_{local}$,$\mathcal{D}_{eval})$
15:         $\lambda_{ditto} \leftarrow \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres})$ by Eq. (9)
16: $w \leftarrow W_{global} - W_{temp}$
17: $||w|| \leftarrow \sqrt{\sum_j w_j^2}$, $\overline{w} \leftarrow \frac{w}{||w||}$, $\overline{v} \leftarrow \frac{\{w_j : j=k,...,m\}}{\sqrt{\sum_{j=k}^{m} w_j^2}}$    ▷ $k$ is the starting index of the last layer
18: $client(send) \leftarrow \overline{w}, ||w||, \overline{v}$
```

4. Experiment

4.1. Experimental Setup

We conducted all the experiments using PyTorch, and the source code was available on GitHub[3].

Datasets and Neural Network. We followed the recent research on Byzantine attacks Fang et al. (2020); Wang et al. (2020) on FL and chose a typical application scenario - image classification. The datasets in our experiments included MNIST, CIFAR-10, and EMNIST. The non-IID data splitting for MNIST and CIFAR-10 in our experiment followed the method carried out in FLTrust Cao et al. (2020), where Deg_{nIID} (q in Cao et al. (2020)) controlled the level of non-IID. In terms of EMNIST, we applied the method from He et al. (2020) where the smaller α_{sim} was, the more tasks were dissimilar. CNNs were used as our global and local models, where CIFAR-10 was trained by ResNet-20 He et al. (2016)

3. https://github.com/Timo9Madrid7/maliciousfl

(269,772 parameters in total), and MNIST & EMNIST were trained by a 2×convolutional layers' NN. ResNet-20 was a pre-trained version[4] to accelerate the training process.

Evaluation Metrics. Main Task Accuracy (MA) represents the accuracy of a model tested by its benign task. It indicates the fraction of correct predictions. If the model is under targeted attacks, Backdoor Accuracy (BA) is the metric to reflect how successful the adversaries are. It denotes the fraction of correct predictions for backdoor samples.

FL Configuration. The total number of clients N was set to 100. Each client received unique training samples and testing samples from the split. The learning rates of global model η_{global} and local model η_{local} were 0.01. The local training epoch E was 1 since clients did not hold an adequate number of samples. The coefficient of $l2$-regularization λ_{ditto} was initialized as 0 and its min-max interval was $[0.0, 2.0]$, where the maximum was suggested by Li et al. (2021b). The threshold acc_{thres} for the local model starting learning from the global was 0.05. The learning rate η_{ditto} for λ_{ditto} was 1. The expected clipping ratio γ, the initial clipping boundary $c^{(0)}$, the clipping learning rate η_c were 0.5, 10, 0.3. Other settings varied for different experiments.

Malicious Configuration. Both model poisoning attacks and data poisoning attacks share a parameter Poisoned Model Rate (PMR), indicating the fraction of poisoned models **for each round**. If the attacking type is data poisoning, it has one more parameter Poisoned Data Rate (PDR), representing the poisoned ratio of the data. Common attacking settings are shown in table 1, where settings are nearly marginal thresholds, below which the attacking effect is non-significant even if it escapes from being detected.

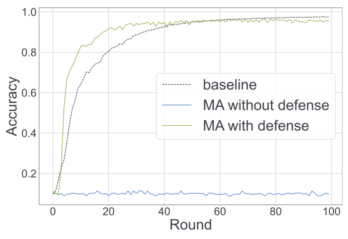
Table 1: Attack settings

Attacks	(E)MNIST	CIFAR-10
A1	mean = 0, std = 1	
A2	Krum's $\epsilon = 10^{-3}$, threshold $= 2 \times 10^{-2}$	
A5	a 5x5 white square is inserted into the targeted data and labeling it as "0"	the semantic pattern is a car with stripes and labeling "car" to "bird"
A6	by adding Ardis_IV to training and labeling "7" as "1"	by adding Southwest Airline images to training and labeling "airplane" as "truck"

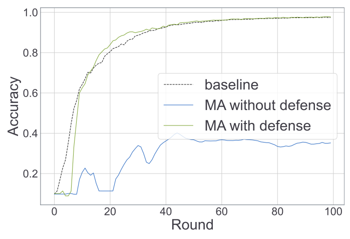
4.2. Fending-off Byzantine Attacks

Fig. 2 shows MNIST under different attacks from A1 to A6. "Baseline" or "without defense" results from the server ruining naive aggregation. 21 clients are uniformly randomly selected from 100 participants for the baseline, while 40 clients for other situations. In Fig. 2(a), due to malicious random uploads, the aggregated updates are meaningless, leading to the blue curve with extremely low accuracy; however, the filtering process conducts so effectively that the learning curve - the green one - can behave normally under this attack. In Fig. 2(b), the Krum attack tries its best to upload counter-directional updates to devalue the global accuracy. Consequently, the global accuracy is even worse than a random guess (50%). However, the global accuracy can reach an original level using the defense of FLVoogd. In Fig. 2(c), the trimmed-mean attack starts to degrade the model accuracy approximately at

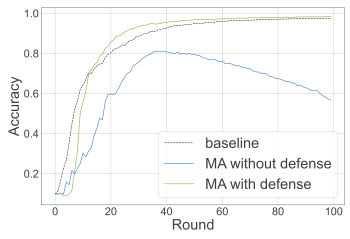
4. https://github.com/chenyaofo/pytorch-cifar-models



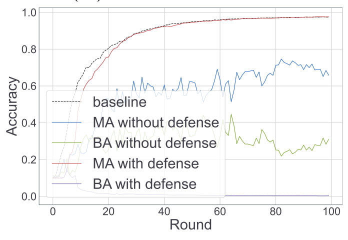
(a) A1: PMR=19/40



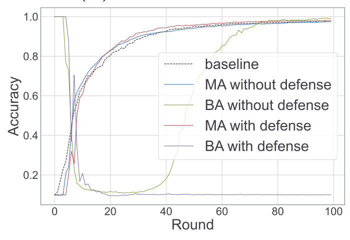
(b) A2: PMR=19/40



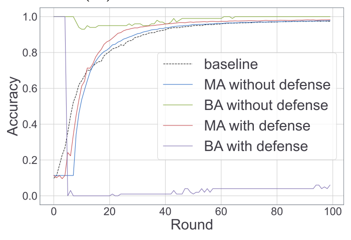
(c) A3: PMR=19/40



(d) A4: PMR=19/40, PDR=1.0



(e) A5: PMR=19/40, PDR=0.2

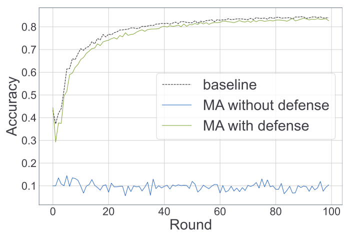


(f) A6: PMR=19/40, PDR=0.33

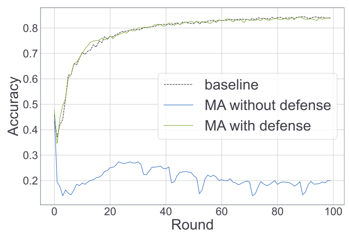
Figure 2: Byzantine attacks on MNIST

midterm and reduces accuracy from higher than 80% to less than 60% within 50 rounds. FLVoogd prevents this malicious reduction of accuracy effectively. In Fig. 2(d), Byzantine clients flip the labels of all training samples, rendering the final prediction like a random guess. One notable point is that the filter cannot correctly recognize flipping uploads in several initial rounds, but it can acknowledge and expel malicious updates once the global model learns a little from those majorities. In Fig. 2(e), the backdoor triggering attack shows its supremacy at the initial stage, and BA can easily approximate to 100% in the first several rounds. After the model learns sufficient benign samples, BA tends to decrease while MA tends to increase. Without the defense, BA again grows sharply at midterm and finally attains relatively high accuracy. In contrast, under the protection, BA is restrained at low accuracy and can impossibly revive after the filter starts to work. The filter does not work at the initial stage because the model is chaotic, and the updates produced from the model reveal little information about the data distribution. In Fig. 2(f), under no defense, BA manifests likewise A5 at the initial stage but does not decline after MA rises. Since the malicious data is sampled from another dataset without intersection with MNIST, learning from the benign samples cannot benefit the model, so BA persists at high accuracy. However, FLVoogd can successfully detect and block these uploads according to the abnormal data distribution, preventing the intrusion of edge-case.

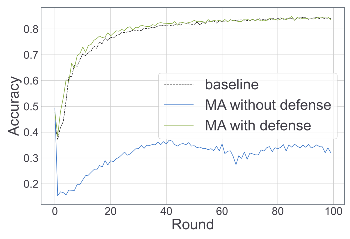
Fig. 3 shows CIFAR-10 under different attacks from A1 to A6, where all the attacks have been eliminated or restrained below comparably low accuracy. Fig. 4 shows EMNIST under various attacks from A1 to A6. Similar to MNIST and CIFAR-10, all the attacks have been eliminated or restrained below comparably low accuracy. However, we initially found that FLVoogd could not prevent EMNIST from A6 productively since 1) we did not use the whole dataset for the training but followed the advice from He et al. (2020), where 20% was suggested for 100 clients; 2) there were 62 classes to be classified. Consequently, the model initially required several rounds to figure out what correct "7" and "1" roughly



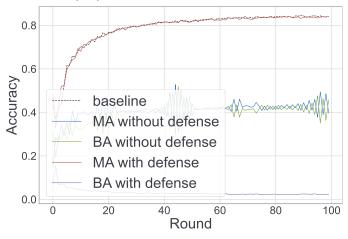
(*a*) A1: PMR=19/40



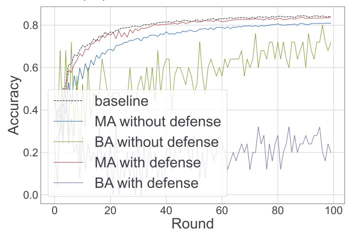
(*b*) A2: PMR=19/40



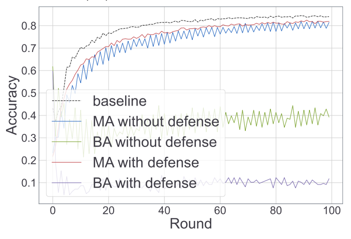
(*c*) A3: PMR=19/40



(*d*) A4: PMR=19/40, PDR=1.0

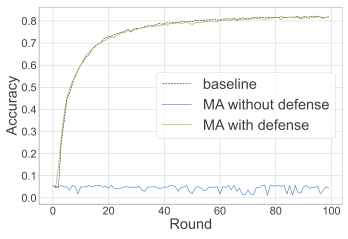


(*e*) A5: PMR=19/40, PDR=0.33

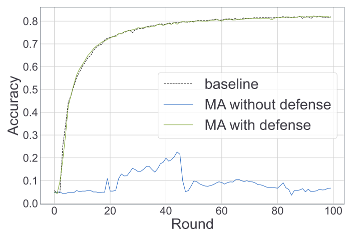


(*f*) A6: PMR=19/40, PDR=0.33

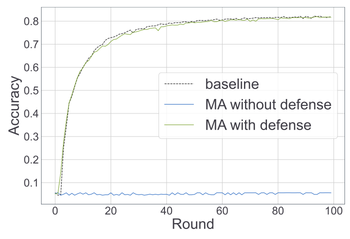
Figure 3: Byzantine attacks on CIFAR-10



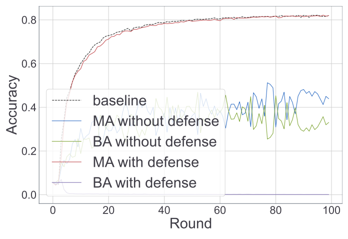
(*a*) A1: PMR=19/40



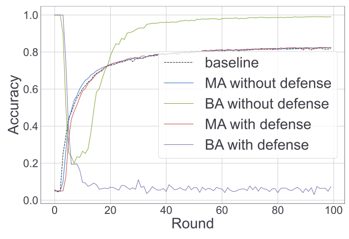
(*b*) A2: PMR=19/40



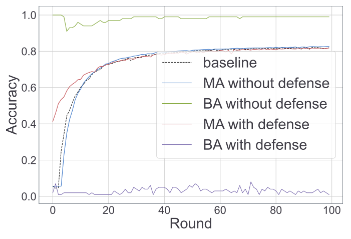
(*c*) A3: PMR=19/40



(*d*) A4: PMR=19/40, PDR=1.0



(*e*) A5: PMR=19/40



(*f*) A6: PMR=19/40, PDR=0.33

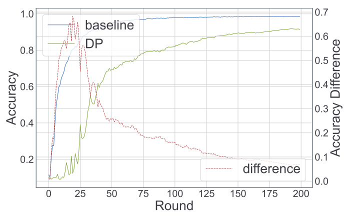
Figure 4: Byzantine attacks on EMNIST

looked like. The model would not rebound those edge cases if edge-case clients instructed the model incorrectly with the mislabelled pictures at the beginning. Therefore, in the first five rounds, we put the model under training with benign-only samples to compensate for this unfairness. After that, the model could successfully filter out those malicious uploads.

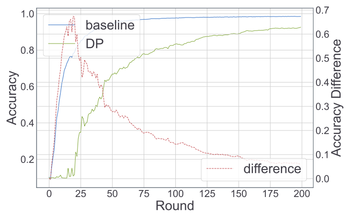
4.3. Adding and Tracking DP

Experiments that test the DP effect and monitor the ϵ budget will be independently studied in this subsection. The experiments consist of different combinations of subsampling ratio

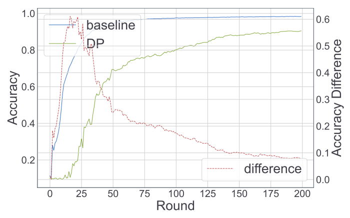
q and noise strength coefficient σ. The DP noise, to some extent, will adversely affect the convergence and accuracy of the model. In return, this kind of sacrifice gains a differential privacy guarantee. The growth of ϵ after each iteration is estimated by Moments accountant or Rényi-DP (RDP) Wang et al. (2019), where δ is set as a constant ($= 10^{-3}$ Geyer et al. (2017)) considering 100 is the total number of clients.



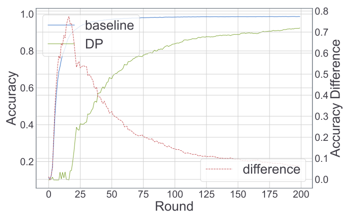
(a) $\sigma = 1$, $q = 20/100$



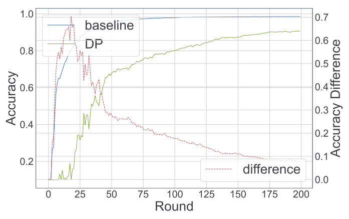
(b) $\sigma = 2$, $q = 40/100$



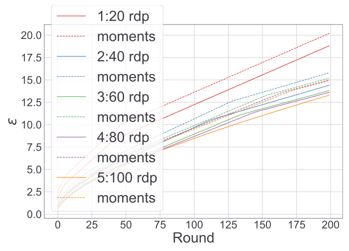
(c) $\sigma = 3$, $q = 60/100$



(d) $\sigma = 4$, $q = 80/100$



(e) $\sigma = 5$, $q = 100/100$



(f) ϵ recording

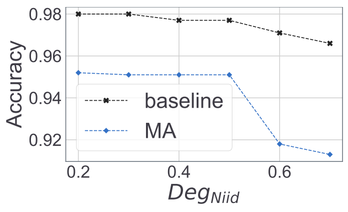
Figure 5: DP's effect on MNIST

The consequence of DP Gaussian noise is tested by MNIST and shown in Fig. 5. We kept the ratio of noise strength σ and the chosen number of clients for each round n constant, $\frac{\sigma}{n} = 1/20$. Each experiment used a different subsampling rate q from 0.2 to 1.0. The total number of training rounds extended to 200, as the noise postponed the convergence. Figures nearly exhibit a similar trend. In general, the added noise decreased the final accuracy by 7.0%. In return, almost in all the experimental settings, (ϵ, δ) was better than $(20, 10^{-3})$, and the best one can achieve $(13.29, 10^{-3})$ estimated by RDP. Solid lines and dash lines in Fig. 5(f) are ϵ estimated by Moment's accountant and RDP, respectively, where RDP always provides a lower ϵ's boundary. The FLVoogd framework provides an adaptive supervisor for ϵ, which the server can customize, so the server can stop the training or adjust the sampling ratio and the amount of adding noise in time once ϵ is undesirable.

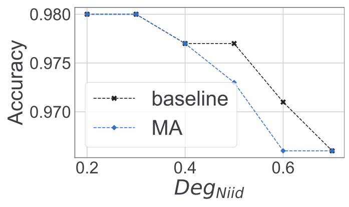
4.4. Non-IID Interference

Experiments will test Deg_{nIID} from 0.2 to 0.7. The number of clients per round is reduced from 40 to 20 since DBSCAN is unstable with noisy points where the non-IID noise may connect clusters. The defense may collapse if too many clients are selected per round under the non-IID setting.

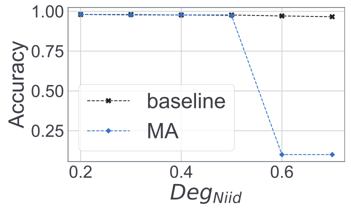
The defensive effect under the non-IID setting has been tested on the MNIST dataset, shown in Fig. 6. In general, FLVoogd cannot ultimately tolerate that clients hold extreme non-IID samples. In Fig. 6(a), the filter rejects all these uploads until the model converges. After convergence, the filter hardly distinguishes between the malicious and model uploads



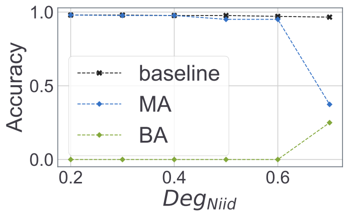
(a) A1: PMR=9/20



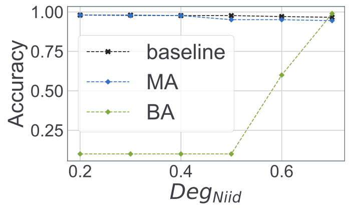
(b) A2: PMR=9/20



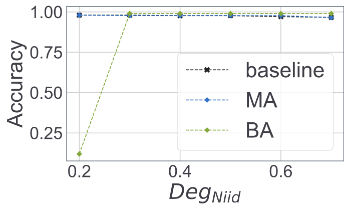
(c) A3: PMR=9/20



(d) A4: PMR=9/20, PDR=1.0



(e) A5: PMR=9/20, PDR=0.2

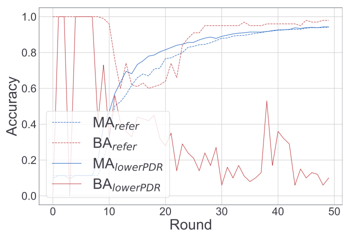


(f) A6: PMR=9/20, PDR=0.33

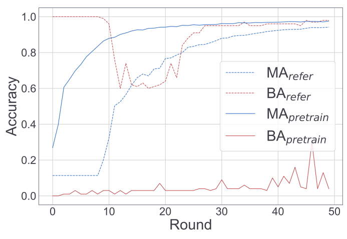
Figure 6: Non-IID effect on MNIST

because both appear somewhat random behaviors, resulting in fluctuations in the convergence state. However, the accuracy is still above 90% since benign uploads again become meaningful once below this threshold, and the filter once more rejects malicious random uploads. Krum attack shows ineffectual regardless of the non-IID degree in Fig. 6(b). The filter is so sensitive to the direction of uploads that uploads with reverse direction can scarcely pass the filtering. In Fig. 6(c), the filter relinquishes its duty after the non-IID degree is more extensive than 0.5. Compared to A1, A3 is a more advanced scheme, which chooses the randomness adaptively, so the attacking effect is more significant. In Fig. 6(d), random flipping works after the non-IID ratio is higher than 0.6. After $Deg_{nIID} > 0.5$, one class completely dominates a dataset, leading that each mini-batch iteration contains over 50% samples from the same class. Consequently, the learning process is tampered with by the flipping of one class intermittently once the non-flipped samples of this class miss the training round. In Fig. 6(e), backdoor accuracy cannot be constrained if increasing the non-IID degree to more than 0.5, as the filter cannot discriminate whether the non-IID or the backdoor targets cause the directional difference. This situation similarly happens in Fig. 6(f) where the result is even worse because the defense collapses when the non-IID ratio is just higher than 0.2. Contradicting A5, where the backdoor targets are still the samples in the dataset, A6 introduces the backdoor targets from another dataset and aims to compromise the weakness of the model prediction. The filter performs ineptly if the model digest cannot reflect normal/abnormal directions. Since the model can never learn those edge cases with true labels, the model cannot provide evidence of deviant behaviors. When the non-IID ratio is lower than 0.3, the filter can detect those edge cases mainly because of the distribution of uploads. However, after non-IID increases, the upload lacks this kind of information.

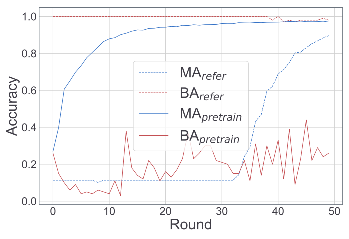
Since the FLVoogd performed worse when it suffered from A6's attack, we selected this situation for further study. We wanted to assist FLVoogd somewhat - training the model ahead or decreasing the PMR. As mentioned, pre-training is doable for some application scenarios. In addition, according to Shejwalkar et al. (2021), PMR≈ 50% is a very pessimistic



(a) Deg_{nIID}=0.3: lowering PDR to 6/20=30%



(b) Deg_{nIID}=0.3: pretraining the model to accuracy around 25%



(c) Deg_{nIID}=0.5: pretraining the model to accuracy around 25%

Figure 7: Pretraining and a lower PDR help the defense

assumption, so we tried to lower it a little bit to see how our framework would react. In Fig. $7(a)$, PDR is reduced from 45% to 30%, and the BA learning curve declines once the model has learned the correct direction from the benign uploads. The poisoning effect is weakened because of the lower PDR. In Fig. $7(b)$ and Fig. $7(c)$, the model accuracy is trained approximately to 25% before the attacks deploy. The updating directions of models become consistent after the pre-train. Thus, the filter can sift those malicious uploads once it first time meets the upload in an abnormal direction. The results also verify that defending against targeted attacks depends on the performance of models on the dataset. If the model can separately recognize the poisoned and normal samples, it can output distinguishable model updates. Then, after the filter captures this variance, the defense effectively works.

5. Conclusion

We introduce FLVoogd, a robust and privacy-preserving federated learning framework that restrains the adverse impact of Byzantine attacks within an acceptable level while maintaining the performance of model predictions on the main task. There are two critical differences between our design and prior works. Firstly, most procedures are executed under privacy preservation, where operations are doable for mostly popular SMPC protocols. Secondly, we provide adaptive adjustments such that the whole process can run automatically. Future works could include: merging the transfer learning into the current framework to tackle GAN inference and combine it with other efficiently communicative schemes, e.g., sketch, to reduce the communication bandwidth and enhance the differential privacy.

References

Yoshinori Aono, Takuya Hayashi, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE TIFS*, pages 1333–1345, 2017.

Xiaoyu Cao, Minghong Fang, et al. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

Ittai Dayan, Holger R Roth, et al. Federated learning for predicting clinical outcomes in patients with covid-19. *Nature medicine*, pages 1735–1743, 2021.

Martin Ester, Hans-Peter Kriegel, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

Minghong Fang, Xiaoyu Cao, et al. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622, 2020.

Robin C Geyer, Tassilo Klein, et al. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

Tianyu Gu, Brendan Dolan-Gavitt, et al. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Chaoyang He, Songze Li, et al. Fedml: A research library and benchmark for federated machine learning. *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop*, 2020.

Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Brian Knott, Shobha Venkataraman, et al. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

Qinbin Li, Bingsheng He, et al. Model-contrastive federated learning. In *CVPR*, pages 10713–10722, 2021a.

Tian Li, Shengyuan Hu, et al. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021b.

Ming Liu, Stella Ho, Mengqi Wang, et al. Federated learning meets natural language processing: A survey. *arXiv preprint arXiv:2107.12603*, 2021.

Guodong Long, Yue Tan, et al. Federated learning for open banking. In *Federated learning*, pages 240–254. Springer, 2020.

H Brendan McMahan, Eider Moore, et al. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

Luis Muñoz-González, Battista Biggio, et al. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 27–38, 2017.

Thien Duc Nguyen, Phillip Rieger, et al. Flguard: secure and private federated learning. *arXiv preprint arXiv:2101.02281*, 2021a.

Thien Duc Nguyen, Phillip Rieger, et al. Flame: Taming backdoors in federated learning. *Cryptology ePrint Archive*, 2021b.

OpenMined. Sympc: A smpc companion library for syft. *GitHub repository*, 2021. URL `https://github.com/OpenMined/SyMPC`.

Phillip Rieger, Thien Duc Nguyen, et al. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. *arXiv preprint arXiv:2201.00763*, 2022.

Virat Shejwalkar, Amir Houmansadr, et al. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. *arXiv preprint arXiv:2108.10241*, 2021.

Sameer Wagh, Shruti Tople, et al. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.

Hongyi Wang, Kartik Sreenivasan, et al. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33:16070–16084, 2020.

Yu-Xiang Wang, Borja Balle, et al. Subsampled rényi differential privacy and analytical moments accountant. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1226–1235. PMLR, 2019.

Dong Yin, Yudong Chen, et al. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.

Xianglong Zhang and Xinjian Luo. Exploiting defenses against gan-based feature inference attacks in federated learning. *arXiv preprint arXiv:2004.12571*, 2020.

References

[1] Martin Abadi et al. “Deep learning with differential privacy”. In: (2016), pp. 308–318.

[2] Naman Agarwal et al. “cpSGD: Communication-efficient and differentially-private distributed SGD”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[3] Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. “FeO2: Federated Learning with Opt-Out Differential Privacy”. In: *arXiv preprint arXiv:2110.15252* (2021).

[4] Sebastien Andreina et al. “Baffle: Backdoor detection via feedback-based federated learning”. In: (2021), pp. 852–863.

[5] Galen Andrew et al. “Differentially private learning with adaptive clipping”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[6] Giuseppe Ateniese et al. “Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers”. In: *International Journal of Security and Networks* 10.3 (2015), pp. 137–150.

[7] Eugene Bagdasaryan et al. “How to backdoor federated learning”. In: (2020), pp. 2938–2948.

[8] Donald Beaver. “Efficient multiparty protocols using circuit randomization”. In: *Annual International Cryptology Conference*. Springer. 1991, pp. 420–432.

[9] James Henry Bell et al. “Secure single-server aggregation with (poly) logarithmic overhead”. In: (2020), pp. 1253–1269.

[10] Mihir Bellare and Silvio Micali. “Non-interactive oblivious transfer and applications”. In: (1989), pp. 547–557.

[11] Jeremy Bernstein et al. “signSGD: Compressed optimisation for non-convex problems”. In: (2018), pp. 560–569.

[12] Stefano A Bini. “Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care?” In: *The Journal of arthroplasty* 33.8 (2018), pp. 2358–2361.

[13] Peva Blanchard et al. “Machine learning with adversaries: Byzantine tolerant gradient descent”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[14] Elette Boyle. *FSS Part 2 - Elette Boyle (The 12th BIU Winter School on Cryptography)*. 2022. URL: `https://www.youtube.com/watch?v=Zm-MUVve2_w&list=LL&index=2&t=2938s&ab_channel=TheBIUResearchCenteronAppliedCryptographyandCyberSecurity` (visited on 04/20/2022).

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function secret sharing”. In: (2015), pp. 337–367.

[16] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function secret sharing: Improvements and extensions”. In: (2016), pp. 1292–1303.

[17] Elette Boyle et al. “Function secret sharing for mixed-mode and fixed-point secure computation”. In: (2021), pp. 871–900.

[18] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. “Density-based clustering based on hierarchical density estimates”. In: (2013), pp. 160–172.

[19] Xiaoyu Cao et al. “Fltrust: Byzantine-robust federated learning via trust bootstrapping”. In: *arXiv preprint arXiv:2012.13995* (2020).

[20] Mingqing Chen et al. “Federated learning of out-of-vocabulary words”. In: *arXiv preprint arXiv:1903.10635* (2019).

[21] Kai-Min Chung, Yael Kalai, and Salil Vadhan. “Improved delegation of computation using fully homomorphic encryption”. In: (2010), pp. 483–501.

[22] Gregory Cohen et al. “EMNIST: Extending MNIST to handwritten letters”. In: (2017), pp. 2921–2926.

[23] Ronald Cramer, Ivan Damgård, and Yuval Ishai. “Share conversion, pseudorandom secret-sharing and applications to secure computation”. In: (2005), pp. 342–362.

[24] Ivan Damgård et al. “Multiparty computation from somewhat homomorphic encryption”. In: (2012), pp. 643–662.

[25] Ivan Damgård et al. “Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits”. In: (2013), pp. 1–18.

[26] Konstantinos G Derpanis. “Mean shift clustering”. In: *Lecture Notes* 32 (2005).

[27] Ye Dong et al. “FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority”. In: *Cryptology ePrint Archive* (2021).

[28] Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. “Differential privacy in practice: Expose your epsilons!” In: *Journal of Privacy and Confidentiality* 9.2 (2019).

[29] Cynthia Dwork, Aaron Roth, et al. “The algorithmic foundations of differential privacy.” In: *Found. Trends Theor. Comput. Sci.* 9.3-4 (2014), pp. 211–407.

[30] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. “Boosting and differential privacy”. In: (2010), pp. 51–60.

[31] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: 96.34 (1996), pp. 226–231.

[32] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. “Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 3557–3568.

[33] Minghong Fang et al. “Local Model Poisoning Attacks to $\{$Byzantine-Robust$\}$ Federated Learning”. In: (2020), pp. 1605–1622.

[34] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: (2015), pp. 1322–1333.

[35] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: (2015), pp. 1322–1333.

[36] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. “The limitations of federated learning in sybil settings”. In: (2020), pp. 301–316.

[37] Jun Furukawa et al. “High-throughput secure three-party computation for malicious adversaries and an honest majority”. In: (2017), pp. 225–255.

[38] Robin C Geyer, Tassilo Klein, and Moin Nabi. “Differentially private federated learning: A client level perspective”. In: *arXiv preprint arXiv:1712.07557* (2017).

[39] Niv Gilboa and Yuval Ishai. “Distributed point functions and their applications”. In: (2014), pp. 640–658.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[41] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. “Badnets: Identifying vulnerabilities in the machine learning model supply chain”. In: *arXiv preprint arXiv:1708.06733* (2017).

[42] Xu Han, Haoran Yu, and Haisong Gu. “Visual inspection with federated learning”. In: *International Conference on Image Analysis and Recognition* (2019), pp. 52–64.

[43] Filip Hanzely et al. “Lower bounds and optimal algorithms for personalized federated learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 2304–2315.

[44] Shirin Hasavari and Yeong Tae Song. “2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)”. In: (2019), pp. 71–75.

[45] Chaoyang He et al. “FedML: A Research Library and Benchmark for Federated Machine Learning”. In: *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop* (2020).

[46] Kaiming He et al. “Deep residual learning for image recognition”. In: (2016), pp. 770–778.

[47] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. “Deep models under the GAN: information leakage from collaborative deep learning”. In: (2017), pp. 603–618.

[48] Hanpeng Hu, Dan Wang, and Chuan Wu. “Distributed machine learning through heterogeneous edge systems”. In: 34.05 (2020), pp. 7179–7186.

[49] Goodfellow Ian et al. “Generative adversarial nets.” In Advances in neural information processing systems”. In: (2014).

[50] Peter Kairouz et al. “Advances and open problems in federated learning”. In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[51] Amin Karami and Ronnie Johansson. “Choosing DBSCAN parameters automatically using differential evolution”. In: *International Journal of Computer Applications* 91.7 (2014), pp. 1–11.

[52] Marcel Keller, Emmanuela Orsini, and Peter Scholl. “MASCOT: faster malicious arithmetic secure computation with oblivious transfer”. In: (2016), pp. 830–842.

[53] B. Knott et al. “CrypTen: Secure Multi-Party Computation Meets Machine Learning”. In: (2021).

[54] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. “Federated optimization: Distributed optimization beyond the datacenter”. In: *arXiv preprint arXiv:1511.03575* (2015).

[55] Jakub Konečnỳ et al. “Federated optimization: Distributed machine learning for on-device intelligence”. In: *arXiv preprint arXiv:1610.02527* (2016).

[56] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[57] Huseyin Kusetogullari et al. “ARDIS: a Swedish historical handwritten digit dataset”. In: *Neural Computing and Applications* 32.21 (2020), pp. 16505–16518.

[58] Wenhao Lai et al. “A new DBSCAN parameters determination method based on improved MVO”. In: *IEEE Access* 7 (2019), pp. 104085–104095.

[59] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[60] Li Li et al. “A review of applications in federated learning”. In: *Computers & Industrial Engineering* 149 (2020), p. 106854.

[61] Tian Li et al. “Ditto: Fair and robust federated learning through personalization”. In: (2021), pp. 6357–6368.

[62] Tian Li et al. “Fair resource allocation in federated learning”. In: *arXiv preprint arXiv:1905.10497* (2019).

[63] Zaoxing Liu et al. “Enhancing the privacy of federated learning with sketching”. In: *arXiv preprint arXiv:1911.01812* (2019).

[64] Lingjuan Lyu, Han Yu, and Qiang Yang. “Threats to federated learning: A survey”. In: *arXiv preprint arXiv:2003.02133* (2020).

[65] Lingjuan Lyu et al. “Privacy and robustness in federated learning: Attacks and defenses”. In: *arXiv preprint arXiv:2012.06337* (2020).

[66] Lingjuan Lyu et al. “Towards fair and privacy-preserving federated deep models”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.11 (2020), pp. 2524–2541.

[67] Xu Ma et al. “Privacy-preserving Byzantine-robust federated learning”. In: *Computer Standards & Interfaces* 80 (2022), p. 103561.

[68] Leland McInnes, John Healy, and Steve Astels. “hdbscan: Hierarchical density based clustering”. In: *The Journal of Open Source Software* 2.11 (2017), p. 205.

[69] Brendan McMahan and Daniel Ramage. “Federated learning: Collaborative machine learning without centralized training data”. In: *Google Research Blog* 3 (2017).

[70] H Brendan McMahan et al. “Federated learning of deep networks using model averaging”. In: *arXiv preprint arXiv:1602.05629* (2016).

[71] Frank D McSherry. “Privacy integrated queries: an extensible platform for privacy-preserving data analysis”. In: (2009), pp. 19–30.

[72] Luca Melis et al. “Exploiting unintended feature leakage in collaborative learning”. In: (2019), pp. 691–706.

[73] Stanislav Minsker. “Geometric median and robust estimation in Banach spaces”. In: *Bernoulli* 21.4 (2015), pp. 2308–2335.

[74] Ilya Mironov. “Rényi differential privacy”. In: (2017), pp. 263–275.

[75] Payman Mohassel and Peter Rindal. “ABY3: A mixed protocol framework for machine learning”. In: (2018), pp. 35–52.

[76] Viraaji Mothukuri et al. “A survey on security and privacy of federated learning”. In: *Future Generation Computer Systems* 115 (2021), pp. 619–640.

[77] Luis Muñoz-González, Kenneth T Co, and Emil C Lupu. “Byzantine-robust federated machine learning through adaptive model averaging”. In: *arXiv preprint arXiv:1909.05125* (2019).

[78] Luis Muñoz-González et al. “Towards poisoning of deep learning algorithms with back-gradient optimization”. In: (2017), pp. 27–38.

[79] Arvind Narayanan and Vitaly Shmatikov. “Robust de-anonymization of large sparse datasets”. In: *2008 IEEE Symposium on Security and Privacy* (2008), pp. 111–125.

[80] Milad Nasr, Reza Shokri, et al. “Improving deep learning with differential privacy using gradient encoding and denoising”. In: *arXiv preprint arXiv:2007.11524* (2020).

[81] Milad Nasr, Reza Shokri, and Amir Houmansadr. “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning”. In: (2019), pp. 739–753.

[82] T.M. Navamani. *Chapter 7 - Efficient Deep Learning Approaches for Health Informatics*. Ed. by Arun Kumar Sangaiah. Academic Press, 2019, pp. 123–137. ISBN: 978-0-12-816718-2.

[83] Joseph P. Near and Chiké Abuah. *Programming Differential Privacy*. Vol. 1. 2021. URL: `https://uvm-plaid.github.io/programming-dp/`.

[84] Thien Duc Nguyen et al. “FLAME: Taming Backdoors in Federated Learning”. In: (2022).

[85] Thien Duc Nguyen et al. “FLGUARD: secure and private federated learning”. In: *arXiv preprint arXiv:2101.02281* (2021).

[86] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. “Smooth sensitivity and sampling in private data analysis”. In: (2007), pp. 75–84.

[87] OpenMined. “SyMPC: A SMPC companion library for Syft”. In: *GitHub repository* (2021). URL: `https://github.com/OpenMined/SyMPC`.

[88] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: (1999), pp. 223–238.

[89] Nicolas Papernot et al. “Towards the science of security and privacy in machine learning”. In: *arXiv preprint arXiv:1611.03814* (2016).

[90] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: 32 (2019). Ed. by H. Wallach et al. URL: `https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

[91] Phillip Rieger et al. “DeepSight: Mitigating Backdoor Attacks in Federated Learning Through Deep Model Inspection”. In: *arXiv preprint arXiv:2201.00763* (2022).

[92] Théo Ryffel et al. “Ariann: Low-interaction privacy-preserving deep learning via function secret sharing”. In: *Proceedings on Privacy Enhancing Technologies* 2022.1 (2020), pp. 291–316.

[93] Badrul Sarwar et al. “Item-based collaborative filtering recommendation algorithms”. In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 285–295.

[94] Sinem Sav et al. “POSEIDON: privacy-preserving federated neural network learning”. In: *arXiv preprint arXiv:2009.00349* (2020).

[95] Aviv Shamsian et al. “Personalized federated learning using hypernetworks”. In: (2021), pp. 9489–9502.

[96] Virat Shejwalkar et al. “Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning”. In: *arXiv preprint arXiv:2108.10241* (2021).

[97] Shiqi Shen, Shruti Tople, and Prateek Saxena. “Auror: Defending against poisoning attacks in collaborative deep learning systems”. In: (2016), pp. 508–519.

[98] Reza Shokri et al. “Membership inference attacks against machine learning models”. In: (2017), pp. 3–18.

[99] Jinhyun So, Başak Güler, and A Salman Avestimehr. “Byzantine-resilient secure federated learning”. In: *IEEE Journal on Selected Areas in Communications* 39.7 (2020), pp. 2168–2181.

[100] Jinhyun So, Başak Güler, and A Salman Avestimehr. “Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning”. In: *IEEE Journal on Selected Areas in Information Theory* 2.1 (2021), pp. 479–489.

[101] Artur Starczewski, Piotr Goetzen, and Meng Joo Er. “A new method for automatic determining of the DBSCAN parameters”. In: *Journal of Artificial Intelligence and Soft Computing Research* 10 (2020).

[102] Nikko Strom. “Scalable distributed DNN training using commodity GPU cloud computing”. In: (2015).

[103] Canh T Dinh, Nguyen Tran, and Josh Nguyen. “Personalized federated learning with moreau envelopes”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21394–21405.

[104] Aleksei Triastcyn and Boi Faltings. “Federated learning with bayesian differential privacy”. In: (2019), pp. 2587–2596.

[105] Stacey Truex et al. “Demystifying membership inference attacks in machine learning as a service”. In: *IEEE Transactions on Services Computing* (2019).

[106] Joost Verbraeken, Martijn de Vos, and Johan Pouwelse. “Bristle: Decentralized Federated Learning in Byzantine, Non-iid Environments”. In: *arXiv preprint arXiv:2110.11006* (2021).

[107] Sameer Wagh, Divya Gupta, and Nishanth Chandran. “SecureNN: 3-Party Secure Computation for Neural Network Training.” In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 26–49.

[108] Sameer Wagh et al. “Falcon: Honest-majority maliciously secure framework for private deep learning”. In: *arXiv preprint arXiv:2004.02229* (2020).

[109] Hongyi Wang et al. “Attack of the tails: Yes, you really can backdoor federated learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16070–16084.

[110] Hongyi Wang et al. “Federated learning with matched averaging”. In: *arXiv preprint arXiv:2002.06440* (2020).

[111] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. “Subsampled rényi differential privacy and analytical moments accountant”. In: (2019), pp. 1226–1235.

[112] Wikipedia. *Facebook–Cambridge Analytica data scandal*. 2018. URL: `https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal` (visited on 03/29/2022).

[113] Louis F Williams Jr. “A modification to the half-interval search (binary search) method”. In: (1976), pp. 95–101.

[114] Xindong Wu et al. “Data mining with big data”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.1 (2014), pp. 97–107.

[115] Jian Xu et al. “SignGuard: Byzantine-robust Federated Learning through Collaborative Malicious Gradient Filtering”. In: *arXiv preprint arXiv:2109.05872* (2021).

[116] Xinyi Xu and Lingjuan Lyu. “Towards building a robust and fair federated learning system”. In: *arXiv e-prints* (2020), arXiv–2011.

[117] Jyoti Yadav and Monika Sharma. “A Review of K-mean Algorithm”. In: *Int. J. Eng. Trends Technol* 4.7 (2013), pp. 2972–2976.

[118] Qiang Yang et al. “Federated machine learning: Concept and applications”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[119] Wensi Yang et al. “Ffd: A federated learning based method for credit card fraud detection”. In: *International conference on big data* (2019), pp. 18–32.

[120] Abbas Yazdinejad et al. “Block hunter: Federated learning for cyber threat hunting in blockchain-based iiot networks”. In: *arXiv preprint arXiv:2204.09829* (2022).

[121] Dong Yin et al. “Byzantine-robust distributed learning: Towards optimal statistical rates”. In: (2018), pp. 5650–5659.

[122] Han Yu et al. “A fairness-aware incentive scheme for federated learning”. In: (2020), pp. 393–399.

[123] Jingfeng Zhang et al. “Hierarchically fair federated learning”. In: *arXiv preprint arXiv:2004.10386* (2020).

[124] Xianglong Zhang and Xinjian Luo. “Exploiting defenses against GAN-based feature inference attacks in federated learning”. In: *arXiv preprint arXiv:2004.12571* (2020).

[125] Hangyu Zhu et al. “Federated learning on non-IID data: A survey”. In: *Neurocomputing* 465 (2021), pp. 371–390.

[126] Ligeng Zhu, Zhijian Liu, and Song Han. “Deep leakage from gradients”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[127] Alexander Ziller et al. “Pysyft: A library for easy federated learning”. In: (2021), pp. 111–139.