

Pernille Kopperud  
Dilawar Mahmood

# Privacy-Preserving Federated Learning Applied to Decentralized Data

Bachelor's project in Computer Engineering  
Supervisor: Ole Christian Eidheim  
May 2021



Pernille Kopperud  
Dilawar Mahmood

# **Privacy-Preserving Federated Learning Applied to Decentralized Data**

Bachelor's project in Computer Engineering  
Supervisor: Ole Christian Eidheim  
May 2021

Norwegian University of Science and Technology  
Faculty of Information Technology and Electrical Engineering  
Department of Computer Science





## Preface

The following report will discuss research related to *federated learning applied to decentralized data*. The purpose of this report is to explore how federated learning can increase data privacy for decentralized data, and to examine statistical and cryptographic methods for enhancing security in federated learning environments, while still attempting to preserve the model performance achieved in centralized learning. Furthermore, the report will describe the methodology behind implementing federated learning for local simulations, and present the experiments and results obtained with federated learning and the different methods used for enhancing privacy and security.

The content presented in this report is highly theoretical, especially the description of federated learning and the description of the different statistical and cryptographic methods for enhancing security in federated learning. The authors of this report have taken extra courses in artificial intelligence, statistical learning and pure mathematics which are not offered by the Applied Information Technology Department at NTNU. These courses provided the authors with the knowledge required to give a thorough introduction to concepts such as the use of group theory in homomorphic encryption with federated learning. Nevertheless, the only prerequisites for understanding this report are:

- Knowledge about machine learning concepts covered by the course TDAT3025, Applied Machine Learning with Project, at NTNU.
- Knowledge about statistics and probability theory covered by the course TDAT2001, Natural sciences and Statistics, at NTNU.
- Knowledge about cryptography concepts covered by the course TDAT2002, Mathematics 2, at NTNU.

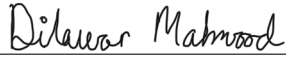
Dilawar Mahmood, one of the authors of this report, currently works for a company named Infiniwell. Infiniwell provides artificial intelligence powered diagnostics tools, technologies and processes to healthcare providers. The solutions that Infiniwell utilizes today are largely based around centralized machine learning which is not ideal considering the privacy issues related to the approach. In order to explore a more secure alternative to centralized learning, the authors decided to collaborate with Infiniwell in exploring federated learning applied to privacy-sensitive medical data.

Sincere gratitude to those who made this project possible:

- Ole Christian Eidheim, Associate Professor at NTNU, as the authors' academic adviser.
- Odd Sandbekkhaug, CEO at Infiniwell, for providing the authors with an office and technical resources to complete the project.

The authors of this report:

  
Pernille Kopperud

  
Dilawar Mahmood

## Summary

In today's society, technology is constantly evolving, and as a result the amount of data being collected and stored is increasing massively. In recent years, many companies have collected large amounts of data from various data sources such as hospitals and other health institutions. Today, many companies use machine learning in order to retain valuable information from the data they have collected due to its high-availability and large quantity. The process of collecting, storing and using machine learning on this data, is known as centralized learning. The challenge with this approach concerns the collection and storage of the data. The data collection process is governed by strict rules such as GDPR, and the companies that store data take on a great responsibility in terms of preserving the privacy of the data. When privacy-sensitive data leaves its data source, it can potentially be intercepted by an adversary which threatens the data privacy. Furthermore, the data collected has to be stored in a database. This requires the responsible companies to maintain a certain level of security in the database in order to prevent any attacks towards the privacy of the data being stored.

This report considers a more privacy-preserving approach to remedy the privacy concerns related to centralized learning. This approach is known as *federated learning*. Federated learning is a relatively new approach which aims to preserve the privacy of the data-owners, which are referred to as clients in federated learning. Unlike centralized learning where the model is trained at the server, federated learning distributes a global model to all participating clients. The distributed models are trained locally at the clients. Once the clients have trained their individual models, the updated models are communicated back to the central server where they are aggregated. The main privacy advantage to this approach is that the raw data never has to leave the clients, making the data less vulnerable to potential attacks. Moreover, no data is stored by a third party, thus alleviating the responsibility of storing privacy-sensitive data in a database.

While federated learning appears to be the better approach for training models on privacy-sensitive decentralized data, the approach is not completely secure. Even though no data is being communicated or stored, the model is still being shared between the clients and the server which causes new privacy-related challenges. Therefore, this report explores how statistical and cryptographic methods can further enhance security in federated learning environments. In addition, this report observes how combining such methods with federated learning affects the model performance since model performance is of high priority when performing machine learning. Finally, this report studies the trade-off between model-performance and achieving a secure federated learning environment. The report also describes an implementation of federated learning applied to sensitive medical data.

# Contents

<b>List of Figures</b>	<b>6</b>
<b>List of Tables</b>	<b>8</b>
<b>1 Introduction</b>	<b>10</b>
1.1 Background	10
1.2 The Research Questions	10
1.3 Structure	11
1.4 Acronyms	12
<b>2 Related Work</b>	<b>13</b>
2.1 Distributed Deep Learning	13
2.1.1 Concurrent Training	13
2.1.1.1 Model Parallelism	13
2.1.1.2 Layer Pipelining	14
2.1.1.3 Data Parallelism	14
2.1.1.4 Hybrid Parallelism	15
2.1.2 Consistency	15
2.1.2.1 Synchronous updates	16
2.1.2.2 Asynchronous updates	17
2.1.2.3 Decentralized updates	17
2.2 Federated Learning	17
2.2.1 Aggregation Methods	19
2.2.1.1 FedSGD	20
2.2.1.2 Federated Averaging	21
2.2.1.3 Secure Aggregation	23
2.2.2 Communication Efficiency in Federated Learning	23
2.3 Attacks on Federated Learning	24
2.3.1 Data and model poisoning attacks	25
2.3.2 Inference attacks	26
2.3.2.1 Generative Adversarial Networks Attack	27
2.4 Robust Federated Aggregation	28
2.4.1 Secure Average Oracle	28
2.4.2 Corruption Model	29
2.4.3 Robust Aggregation using the Geometric Median	29
2.4.3.1 Theoretical Assumptions	32
2.5 Memorization	33
2.5.1 Eidetic Memorization	33
2.5.2 Memorization in Federated Learning	34
2.6 Differential Privacy	34
2.6.1 Definition of Differential Privacy	35
2.6.1.1 Relaxed Definition of Differential Privacy	37
2.6.1.2 Moments Accountant	37
2.6.2 Federated Learning with Differential Privacy	38
2.7 Homomorphic Encryption	39
2.7.1 Binary operators, Groups, and Rings	39
2.7.1.1 Binary operators	39
2.7.1.2 Groups	40
2.7.1.3 Rings	40
2.7.1.4 Homomorphisms and Homomorphic Encryption	40
2.7.2 A formal definition of Homomorphic Encryption	41

2.7.2.1	Paillier Cryptosystem . . . . .	42
2.7.3	Homomorphic Encryption in Federated Learning . . . . .	42
<b>3</b>	<b>Method</b>	<b>45</b>
3.1	Process . . . . .	45
3.1.1	Research . . . . .	45
3.1.2	Data Collection . . . . .	45
3.1.3	Data Analysis . . . . .	45
3.2	Execution . . . . .	48
3.2.1	Experiment Process . . . . .	48
3.2.2	Experimentation Pipeline . . . . .	49
3.2.3	Data Preprocessing . . . . .	50
3.2.4	Overview of Machine Learning Models . . . . .	52
3.2.4.1	Softmax Regression . . . . .	52
3.2.4.2	Artificial Neural Network . . . . .	53
3.2.4.3	1D Convolutional Neural Network Model . . . . .	53
3.2.5	Implementation of Federated Learning . . . . .	55
3.2.6	Hyperparameters . . . . .	55
3.2.7	Training the Model . . . . .	56
3.2.7.1	Selecting model, optimizers, and hyperparameters . . . . .	56
3.2.7.2	Training loop . . . . .	56
3.2.8	Analyzing the Experiments . . . . .	56
3.2.8.1	Assessing model performance . . . . .	56
3.2.8.2	Privacy Preservation . . . . .	57
3.3	Choice of Technologies . . . . .	57
3.3.1	TensorFlow . . . . .	57
3.3.1.1	TensorFlow Federated . . . . .	57
3.3.1.2	TensorBoard . . . . .	57
3.3.1.3	Keras . . . . .	58
3.3.2	Jupyter Notebook . . . . .	58
3.3.3	Python Paillier . . . . .	59
3.3.4	CUDA-enabled GPU card . . . . .	59
3.3.5	NumPy . . . . .	59
3.3.6	Pandas . . . . .	59
3.3.7	Scikit-learn . . . . .	59
3.3.8	Matplotlib . . . . .	59
3.3.9	Plotly . . . . .	59
<b>4</b>	<b>Results</b>	<b>60</b>
4.1	Overview . . . . .	60
4.2	Preliminary Experiments . . . . .	61
4.2.1	Centralized Learning . . . . .	61
4.2.1.1	Centralized Learning with ANN . . . . .	61
4.2.1.2	Centralized Learning with CNN . . . . .	63
4.2.2	Federated Stochastic Gradient Descent . . . . .	66
4.2.2.1	FedSGD with ANN . . . . .	67
4.2.2.2	FedSGD with CNN . . . . .	69
4.2.3	Federated Averaging . . . . .	71
4.2.3.1	FedAvg with ANN . . . . .	72
4.2.3.2	FedAvg with CNN . . . . .	75
4.3	Experiments regarding Privacy Issues in Federated Learning . . . . .	82
4.3.1	Federated Averaging with Static Data Poisoning . . . . .	82
4.3.2	Memorization in Federated and Centralized Learning . . . . .	83
4.3.2.1	FedAvg with ANN . . . . .	84



4.3.2.2	FedAvg with CNN . . . . .	85
4.3.2.3	Centralized Learning with ANN . . . . .	87
4.3.2.4	Centralized Learning with CNN . . . . .	88
4.3.3	Model Extraction in Federated Learning . . . . .	89
4.4	Privacy-Preserving Experiments in Federated Learning . . . . .	92
4.4.1	Robust Federated Aggregation with Static Data Poisoning . . . . .	92
4.4.2	Differential Privacy in Federated Learning . . . . .	93
4.4.2.1	DP-FedAvg with ANN . . . . .	94
4.4.2.2	DP-FedAvg with CNN . . . . .	96
4.4.2.3	Forced Memorization in ANN . . . . .	98
4.4.2.4	Forced Memorization in CNN . . . . .	100
4.4.3	Model Extraction in Federated Learning with Differential Privacy . . . . .	102
4.4.4	Federated Learning with Homomorphic Encryption . . . . .	104
<b>5</b>	<b>Discussion</b>	<b>107</b>
5.1	Federated and Centralized Learning . . . . .	107
5.1.1	Model Performance . . . . .	107
5.1.2	Memorization as a Privacy Issue . . . . .	108
5.1.3	Privacy Benefits in using Federated Learning . . . . .	109
5.2	Robustness in Federated Learning . . . . .	109
5.3	Model Extraction . . . . .	110
5.4	Differential Privacy . . . . .	111
5.4.1	Model performance . . . . .	112
5.4.2	Memorization . . . . .	112
5.4.3	Model Extraction . . . . .	113
5.5	Federated Learning with Homomorphic Encryption . . . . .	113
5.6	Summary . . . . .	114
<b>6</b>	<b>Conclusion</b>	<b>116</b>
<b>7</b>	<b>Future Work</b>	<b>118</b>
<b>8</b>	<b>Broader Impact</b>	<b>119</b>
	<b>Bibliography</b>	<b>120</b>
	<b>Attachments</b>	<b>123</b>

# List of Figures

1	A Visualization of the Model Parallelism Architecture . . . . .	14
2	A Visualization of the layer Pipelining Architecture . . . . .	14
3	A Visualization of the Data Parallelism Architecture . . . . .	15
4	A Visualization of the DistBelief Architecture . . . . .	15
5	Parameter Server Architecture . . . . .	16
6	GossipGraD . . . . .	17
7	Illustration of the Federated Learning Workflow . . . . .	18
8	Illustration of Model and Data Poisoning in Federated Learning . . . . .	26
9	Illustration of Inference Attack . . . . .	26
10	Illustration of Generative Adversarial Networks . . . . .	27
11	Geometric Median vs. Arithmetic Mean . . . . .	30
12	Robust Federated Aggregation . . . . .	31
13	Overview of Differential Privacy . . . . .	35
14	The Laplace Distribution . . . . .	36
15	Moments Accountant vs. The Strong Composition Theorem . . . . .	37
16	Homomorphic Encryption Scenario . . . . .	39
17	Homomorphic Encryption Timeline . . . . .	41
18	Paillier cryptosystem benchmarked with different key sizes. . . . .	43
19	Distribution of the data in the MIT-BIH Arrhythmia Database . . . . .	47
20	Plots of 2D Histogram for each class . . . . .	48
21	Illustration of the Hypothetico-Deductive Model . . . . .	49
22	Illustration of the Implemented Machine Learning Pipeline. . . . .	50
23	Data Distribution after resampling . . . . .	51
24	Illustration of the Softmax Regression Model . . . . .	52
25	Illustration of the Artificial Neural Network . . . . .	53
26	Illustration of the 1D CNN Model . . . . .	54
27	TensorBoard . . . . .	58
28	Confusion matrix for the centralized learning experiment with the ANN model. . . . .	62
29	Graph illustrating accuracy obtained during the centralized learning experiment with the ANN model . . . . .	63
30	Graph illustrating the loss obtained during the centralized learning experiment with an ANN model . . . . .	63
31	Confusion matrix for the centralized learning experiment with the CNN model . . . . .	65
32	Graph illustrating accuracy obtained during the centralized learning experiment with an CNN model . . . . .	65
33	Graph illustrating the loss obtained during the centralized learning experiment with an CNN model . . . . .	66
34	Confusion matrix for the centralized learning experiment with the ANN model . . . . .	68
35	Graph illustrating accuracy obtained during the centralized learning experiment with the ANN model . . . . .	68
36	Graph illustrating the loss obtained during the FedSGD experiment with an ANN model . . . . .	69
37	Confusion matrix for the centralized learning experiment with the CNN model . . . . .	70
38	Graph illustrating accuracy obtained during the centralized learning experiment with a CNN model . . . . .	71
39	Graph illustrating the loss obtained during the FedSGD experiment with an CNN model . . . . .	71
40	Confusion matrix for the FedAvg experiment with the ANN model . . . . .	73
41	Graph illustrating accuracy obtained during the FedAvg experiment with the ANN model . . . . .	74
42	Graph illustrating the loss obtained during the FedAvg experiment with the ANN model . . . . .	75
43	Confusion matrix for the FedAvg experiment with the CNN model trained on Non-IID data . . . . .	76
44	Graph illustrating accuracy obtained during the FedAvg experiment with the CNN model trained on Non-IID data . . . . .	77

45	Graph illustrating the loss obtained during the FedAvg experiment with the CNN model trained on Non-IID data . . . . .	78
46	Confusion matrix for the FedAvg experiment with the CNN model trained on the uniform data distribution . . . . .	79
47	Confusion matrix for the FedAvg experiment with the CNN model trained on class distributed data . . . . .	81
48	Confusion matrix for the FedAvg experiment with static data poisoning . . . . .	83
49	Confusion matrix for the memorization experiment using the FedAvg algorithm with the ANN model . . . . .	85
50	Confusion matrix for the memorization experiment using the FedAvg algorithm with the CNN model . . . . .	86
51	Confusion matrix for the memorization experiment using the FedAvg algorithm with the ANN model . . . . .	88
52	Confusion matrix for the memorization experiment using centralized learning with the CNN model . . . . .	89
53	Model Extraction for the Normal Beats class . . . . .	90
54	Model Extraction for the Supraventricular Beats class . . . . .	91
55	Confusion matrix for the RFA experiment with static data poisoning . . . . .	93
56	Confusion matrix for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model . . . . .	95
57	Moments accountant for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model . . . . .	96
58	Confusion matrix for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model . . . . .	97
59	Moments accountant for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model . . . . .	98
60	Confusion matrix for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model . . . . .	100
61	Confusion matrix for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model . . . . .	102
62	Model Extraction for the Normal Beats class in a DP setting . . . . .	103
63	Model Extraction for the Supraventricular Beats class in a DP setting . . . . .	104
64	Graph illustrating the validation loss for doing federated learning with homomorphic encryption. . . . .	106

## List of Tables

1	Excerpt of the Raw Data . . . . .	46
2	Overview of the training configuration used for each experiment in Section 4. . . . .	55
3	Training configuration for the centralized learning experiment with the ANN model. . . . .	61
4	Accuracy, loss and training time for the centralized learning experiment with the ANN model. . . . .	62
5	Classification report for the centralized learning experiment with the ANN model . . . . .	62
6	Accuracy, loss and training time for the centralized learning experiment with the CNN model. . . . .	64
7	Classification report for the centralized learning experiment with the CNN model . . . . .	64
8	Training configuration for experiments with FedSGD. . . . .	66
9	Accuracy, loss and training time for the FedSGD experiment with the ANN model. . . . .	67
10	Classification report for the FedSGD experiment with the ANN model . . . . .	67
11	Accuracy, loss and training time for the FedSGD experiment with the CNN model . . . . .	69
12	Classification report for the FedSGD experiment with the CNN model . . . . .	70
13	Training configuration for the FedAvg experiment. . . . .	72
14	Accuracy, loss and training time for the FedAvg experiment with the ANN model . . . . .	72
15	Classification report for the FedAvg experiment with the ANN model . . . . .	73
16	Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on Non-IID data. . . . .	75
17	Classification report for the FedAvg experiment with the CNN model trained on Non-IID data. . . . .	76
18	Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on the uniform data distribution. . . . .	78
19	Classification report for the FedAvg experiment with the CNN model trained the uniform data distribution. . . . .	79
20	Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on class distributed data . . . . .	80
21	Classification report for the FedAvg experiment with the CNN model trained on class distributed data. . . . .	80
22	Training configuration for the federated learning experiment using the FedAvg algorithm with static data poisoning. . . . .	82
23	Accuracy, loss and training time for the FedAvg experiment with static data poisoning. . . . .	82
24	Classification report for the FedAvg experiment with static data poisoning . . . . .	83
25	Training configuration for the memorization experiment using the FedAvg algorithm. . . . .	84
26	Accuracy, loss and training time for the memorization experiment using the FedAvg algorithm with the ANN model . . . . .	84
27	Classification report for the memorization experiment using the FedAvg algorithm with the ANN model . . . . .	85
28	Accuracy, loss and training time for the memorization experiment using the FedAvg algorithm with the CNN model . . . . .	86
29	Classification report for the memorization experiment using the FedAvg algorithm with the CNN model . . . . .	86
30	Training configuration for the memorization experiment using centralized learning. . . . .	87
31	Accuracy, loss and training time for the memorization experiment using centralized learning with the ANN model . . . . .	87
32	Classification report for the memorization experiment using centralized learning with the ANN model . . . . .	87
33	Accuracy, loss and training time for the memorization experiment using centralized learning with the CNN model . . . . .	88
34	Classification report for the memorization experiment using centralized learning with the CNN model . . . . .	89
35	Training configuration for the model extraction experiment using the FedAvg algorithm. . . . .	90

36	Training configuration for the federated learning experiment using the RFA algorithm with static data poisoning. . . . .	92
37	Accuracy, loss and training time for the RFA experiment with static data poisoning . . . . .	92
38	Classification report for the RFA experiment with static data poisoning . . . . .	93
39	Training configuration for the differential privacy experiment using the DP-FedAvg algorithm.	94
40	DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model. . . . .	94
41	Accuracy, loss and training time for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model . . . . .	94
42	Classification report for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model . . . . .	95
43	DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model. . . . .	96
44	Accuracy, loss and training time for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model . . . . .	96
45	Classification report for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model . . . . .	97
46	DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model while forcing memorization. . . . .	98
47	Accuracy, loss and training time for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model . . . . .	99
48	Classification report for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model . . . . .	99
49	DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model while forcing memorization. . . . .	100
50	Accuracy, loss and training time for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model . . . . .	101
51	Classification report for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model . . . . .	101
52	Training configuration for the model extraction experiment using the DP-FedAvg algorithm.	102
53	Differential privacy parameters for the model extraction experiment using the DP-FedAvg algorithm. . . . .	103
54	Training configuration for homomorphic encryption with federated averaging. . . . .	105
55	Accuracy, loss and training time after performing federated learning with homomorphic encryption . . . . .	105
56	Size of a weight and its encrypted value in number of bytes . . . . .	105
57	The results discussed in Section 5.1 . . . . .	107
58	The results discussed in Section 5.2 . . . . .	109
59	The results discussed in Section 5.4 . . . . .	111

# 1 Introduction

This chapter will provide an introduction to the report. This introduction will include the background as to why federated learning was chosen as the subject of the report and the research questions formulated. Furthermore, the structure of the report and the acronyms used in the report will also be presented.

## 1.1 Background

Infiniwell is an artificial intelligence company founded in 2018. The company specializes in the interpretation of biometric waveforms and medical data, and aims to help healthcare providers deliver better and more efficient care to patients by equipping them with AI-powered diagnostic tools, technology and processes<sup>1</sup>. The company is building a cloud-based remote patient monitoring and diagnostic platform which uses AI to assist in clinical decision-making. To achieve this, they are collaborating with companies like *Clarity Medical*, *Telenor* and *Microsoft*. Clarity Medical produces devices to measure vital signs continuously. These devices are distributed amongst patients around the globe, and they record patients' vital signs remotely. The recorded decentralized patient data is streamed via the 5G network provided by Telenor to secure servers offered by Microsoft.

Infiniwell has developed several deep learning models to analyze the patient data stored in Microsoft Azure in real-time. These models are all trained on the centrally stored data with stochastic gradient descent, meaning that Infiniwell applies centralized learning on the data. The diagnostics retrieved from the deep learning models are then sent to medical personnel for further analysis. This centralized machine learning pipeline allows patients to be monitored by hospitals remotely, thus avoiding being hospitalized for a longer period of time. This reduces the amounts of in-patients, allowing the hospitals to use their beds for more serious and emergent cases.

Infiniwell is not currently satisfied with their machine learning pipeline. As previously stated, Infiniwell collects decentralized data and stores it in a database. Unfortunately, the data collection process can be vulnerable to attacks, and can in turn jeopardize the privacy of the patients. Furthermore, the storage of privacy-sensitive data comes with a large responsibility to protect that data. As mentioned, Infiniwell works with sensitive patient data which makes it extremely important that they are able to provide a secure machine learning pipeline that can both preserve the privacy of its clients, and simultaneously train well-performing models.

In recent years, a new machine learning approach, called *Federated Learning*, has emerged. This approach is known to train deep learning models on decentralized data in a way that preserves the privacy of the involved data-owners. Since Infiniwell is working with decentralized data where privacy-preservation is of high importance, federated learning seemed like an adequate approach to solve the issues concerning privacy in the centralized learning pipeline.

## 1.2 The Research Questions

The purpose of this report is to research how federated learning would perform in an environment similar to Infiniwell's. In addition, the report will explore different statistical and cryptographic methods to further enhance privacy and security in federated learning environments. The goal of the research is to determine whether federated learning combined with different methods for enhancing privacy and security can solve the privacy-issues in centralized learning, while still being able to train well-performing models. The results of this report could potentially help Infiniwell create an even more secure machine learning pipeline, which can further increase the privacy of their consumers. Therefore, the research is focused on the following three subjects:

- **Privacy.** In regards to federated learning, we want to explore how data privacy is obtained. Data privacy in federated learning concerns protecting the data held by the clients in the federation from being accessed by anyone but the data-owner holding the data.

---

<sup>1</sup>Infiniwell - <https://www.norwayhealthtech.com/member/infiniwell/>

- **Security.** In federated learning, security entails protecting the clients, their data and the models from potential threats and attacks. This is done by applying appropriate protection methods to secure those resources.
- **Model performance.** In this report, we want to research how federated learning along with a variety of protection methods, affect model performance. This entails looking at how the model fits to the training data, how well the model classifies new data, and how long it takes to train the model.

With these subjects in mind, we wanted to formulate research questions that could help determine whether federated learning combined with different methods for enhancing privacy and security, would provide a well-performing model. Federated learning is known to be a privacy-preserving approach to machine learning tasks, and in this report we wanted observe how the approach increases privacy and which trade-offs are made in order to achieve this. Therefore, the first research question formulated was:

*How does federated learning increase privacy when applied to decentralized data?*

The research question presented above exclusively concerns federated learning. In order to further evaluate federated learning as a privacy-preserving approach, we wanted to observe the learning algorithm in combination with different statistical and cryptographic methods. This would provide insight into how successful the different methods were in enhancing security in federated learning environments, as well as how the methods affected model-performance. With this in mind, we formulated a second research question:

*How can different methods enhance security in federated learning environments, and how do these methods affect model performance?*

### 1.3 Structure

This report is divided into the following ten chapters:

- **Introduction** - This chapter contains an introduction to the project. In the introduction, the background, the research questions, the structure of the report will be described, and the acronyms will be presented.
- **Related Work** - This chapter contains explanations of related work and theory. This chapter will provide the basis for the experiments presented in the Results chapter.
- **Method** - This chapter includes a description of the research process and the methodology behind executing a series of experiments. In addition, this chapter will provide an overview of the technologies used in the project and explain why these were chosen.
- **Results** - This chapter will provide an overview of the experiments performed. Furthermore, the results of the experiments will be presented alongside the training configuration used.
- **Discussion** - This chapter will discuss the results obtained during the project in regards to the related work. It will also discuss the meaning of the results with respect to the research questions.
- **Conclusion** - This chapter will attempt to answer the research questions presented in the Introduction. The conclusion will be based on the results and the discussion.
- **Future Work** - This chapter will present suggestions for future work in relation to the report.
- **Broader Impact** - This chapter will attempt to give an overview of the broader impact of this report.
- **Bibliography** - This chapter will provide the complete bibliography for the report.
- **Attachments** - This chapter will include all relevant attachments related to the report.

## 1.4 Acronyms

- **AM** - *Arithmetic Mean*
- **ANN** - *Artificial Neural Network*
- **CNN** - *Convolutional Neural Network*
- **DNN** - *Deep Neural Network*
- **DP** - *Differential Privacy*
- **DP-FedAvg** - *Differentially-Private Federated Averaging*
- **ECG** - *Electrocardiogram*
- **FedAvg**- *Federated Averaging*
- **FedSGD** - *Federated Stochastic Gradient Descent*
- **FHE** - *Fully Homomorphic Encryption*
- **FL** - *Federated Learning*
- **GAN** - *Generative Adversarial Network*
- **GDPR** - *General Data Protection Regulation*
- **GM** - *Geometric Median*
- **HE** - *Homomorphic Encryption*
- **Non-IID** - *Non Independent and Identically Distributed*
- **PHE** - *Partially Homomorphic Encryption*
- **RFA** - *Robust Federated Aggregation*
- **SAO** - *Secure Average Oracle*
- **SGD** - *Stochastic Gradient Descent*
- **SMC** - *Secure Multiparty Computation*
- **SWHE** - *Somewhat Homomorphic Encryption*
- **TFF** - *TensorFlow Federated*



## 2 Related Work

The following sections will present relevant theory concerning federated learning. First, the chapter will provide an introduction to distributed deep learning which is the basis for federated learning. Furthermore, the chapter will present federated learning and related privacy-concerns. Finally, the chapter is going to present different statistical and cryptographic methods for enhancing security in federated learning environments. The theory provided will largely be based on related research papers, but also on the prior knowledge presented in the Preface of this report. The theory covered in this chapter, will be relevant for answering the research questions described in Chapter 1.

### 2.1 Distributed Deep Learning

Deep learning has become a popular method for solving optimization problems when working with large datasets, but little to none domain knowledge is available. The use of deep neural networks (DNNs) require a huge amount of computational power and memory. Some examples of DNNs are:

- *AlexNet* is a convolutional neural network (CNN) that consists of 60 million parameters. AlexNet was used to recognize objects on the ImageNet dataset [1].
- *I3D* is a CNN that consists of 25 million parameters. I3D is used for action classification in videos [2].
- *Transformer-XL* is a Transformer model which consists of 460 million parameters. Transformer-XL is used for capturing longer-term dependencies on text data [3].

The deep neural networks listed above have large memory and computation requirements. Training these models in a sequential manner is not scalable. Distributed deep learning methods use hardware in a more efficient manner, thus increasing scalability in the training of deep neural networks.

#### 2.1.1 Concurrent Training

Concurrent training methods split the deep neural network and the data between the compute-nodes in a cluster. To make deep learning more scalable, and to utilize multiple CPUs and GPUs, there have been efforts to parallelize the training of deep neural networks. This section will look at methods for obtaining training parallelism of deep neural networks.

##### 2.1.1.1 Model Parallelism

In the paper *Large Scale Distributed Deep Networks* model parallelism is discussed as the distribution of the neurons in a DNN  $M$  among different compute-nodes in a cluster [4]. Each node is responsible for calculating the activation function

$$a_i = g(\mathbf{W}_i^T \mathbf{X} + \mathbf{b}_i) \tag{1}$$

for the neurons on machine  $i$ , and passing it to the next layer in the network. In Equation 1,  $g$  is the activation function used in the DNN. The neurons which are connected in the model  $M$  must pass their activation's to each other, and if such neurons are not on the same node, then the nodes must communicate the activations  $a_i$  with each other. This could lead to higher communication cost in the compute cluster, which is a drawback with the method. Moreover, the method is synchronous which is another drawback because all the activations must reach the last layer of the network before the optimization process can proceed with back-propagation. Another drawback is that some nodes cannot start the computation of the activations for their neurons before they have received the activations from the other nodes. Thus, the bottleneck in this method is the slowest node  $i$ , since all the other nodes have to wait for the slowest node before proceeding with the next mini-batch. The mini-batches must also be copied to each node in the network, since each set of neurons must train on the same data, leading to an even higher communication cost. Model Parallelism is illustrated in Figure 1.

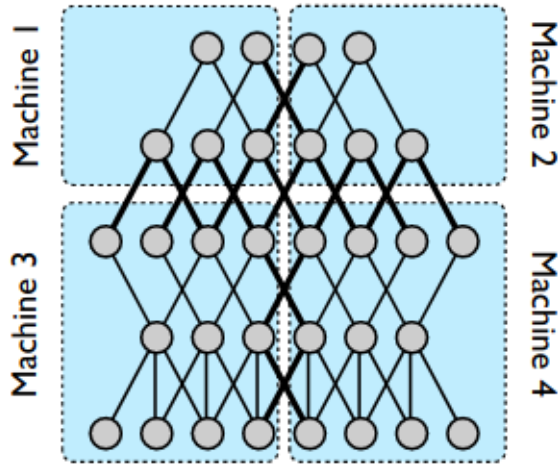


Figure 1: A visualization of the model parallelism architecture. The neurons in the network are distributed among 4 nodes in a local cluster. Machines 1 and 2 are dependent on the activations from Machines 3 and 4 [4].

### 2.1.1.2 Layer Pipelining

A similar technique to the one discussed in Section 2.1.1.1 is distributing the layers  $L$  of the network to the compute nodes instead of the neurons. In the *neuron distribution*-technique, the problem of the nodes having to communicate with many other nodes occurred, since one neuron can be connected to many other neurons. Given two adjacent layers  $l_i, l_{i+1} \in L$ , and if these two layers are distributed on two different nodes, the node containing  $l_i$  only have to communicate with the node  $l_{i+1}$ . This reduces the communication cost between compute nodes discussed in Section 2.1.1.1, but the mini-batches must still be copied to every node in the compute cluster [5]. Layer Pipelining is illustrated in Figure 2.

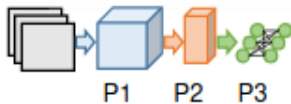


Figure 2: A visualization of the layer pipelining architecture. This method greatly reduces the communication cost compared to the Model Parallelism technique. In the figure, the DNN is partitioned according to depth, and each layer is assigned to a processor. [5]

### 2.1.1.3 Data Parallelism

Sections 2.1.1.1 and 2.1.1.2 discussed methods that concerned partitioning and distributing the model. In this section, another method called data parallelism will be discussed. Data parallelism is a method that partitions and distributes the training data instead of the model. The data is partitioned into  $N$  subsets, and distributed among the compute nodes. In this method, the compute nodes contain the whole model. These nodes draw a mini-batch from their local data partition, and run forward- and backward-passes through the whole network. The weight updates  $\Delta w$  from each node are reduced with protocols such as MapReduce or Message Passing Interface [5]. The majority of the operations in Stochastic Gradient Descent (SGD) over mini-batches are independent, which makes this technique highly scalable compared to model parallelism and layer pipelining. The communication cost is also reduced, since the only communication that happens between the compute-nodes is when they are reducing their  $\Delta w$ . A drawback with

this method is that the whole model has to fit in the memory of the compute nodes. Figure 3 illustrates Data Parallelism.



Figure 3: A visualization of the data parallelism architecture. The data is distributed among the different nodes in the compute cluster. The independent nature of SGD over mini-batches makes this technique highly scalable [5].

#### 2.1.1.4 Hybrid Parallelism

Until now, the different parallelism schemes have been discussed in an isolated way. However, there exists methods which applies several of the parallelism schemes at the same time. *DistBelief* is such a method [4]. This method uses Model Parallelism, Layer Pipelining, and Data Parallelism all at once. Model Parallelism and Layer Pipelining are combined in such a way that the neurons belonging to a layer are contained together in a compute node. This method is illustrated in Figure 4. Another method which utilizes all three concurrency schemes is *Project Adam*, which uses fewer compute nodes than *DistBelief* [6]. Both methods use the notion of *Parameter Server*, which helps synchronize the optimization across the different compute nodes. This will be discussed in greater detail in Section 2.1.2.

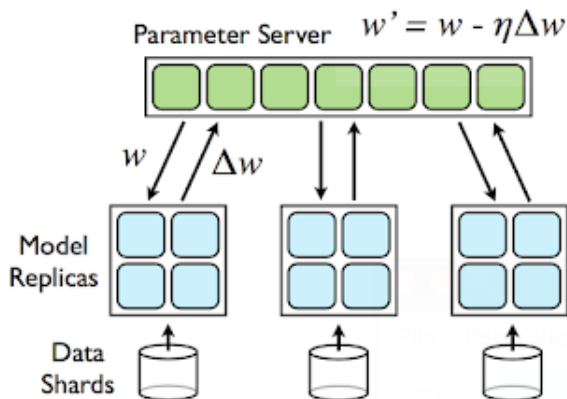


Figure 4: A visualization of the *DistBelief* architecture. This scheme uses Model Parallelism, Layer Pipelining, and Data Parallelism all at once, thus making it a hybrid scheme [5].

#### 2.1.2 Consistency

Consistency concerns initializing the compute nodes with the same parameters after an iteration, i.e., weights, hyperparameters, and other parameters which define a DNN. This is important for converging the compute nodes. If all the initialized weights on the compute nodes were different, the output from the compute nodes would differ. In later training rounds this could result in an inconsistent model. Thus, the compute nodes need a way to write their model updates to a global server. The *parameter server* acts as an external orchestrator to synchronize the compute nodes across the cluster [7].

As visualized in Figure 5, the parameter server consists of different modules to make the nodes consistent. The *server manager* maintains the assignment of parameter partitions, and metadata about the

nodes, for example node liveness. The *server node* maintains a partition of the globally shared parameters, and these nodes are used for replicating and migrating parameters for reliability and scalability. Each of the worker groups, which may consist of several worker nodes, or compute nodes, have a *task scheduler*. The task scheduler assigns tasks to the different worker nodes. To obtain model consistency, the worker nodes communicate directly with the server node to retrieve parameters instead of communicating within a worker group. Thus, the worker nodes obtain the most recent parameters from the parameter server. After doing some rounds of optimization, the worker nodes write the updated parameter to the parameter server, where they are aggregated. The parameter server represents the parameters as *key-value* vectors, such that the worker nodes can do linear algebra operations on these parameters [7]. The write operation can be synchronous or asynchronous.

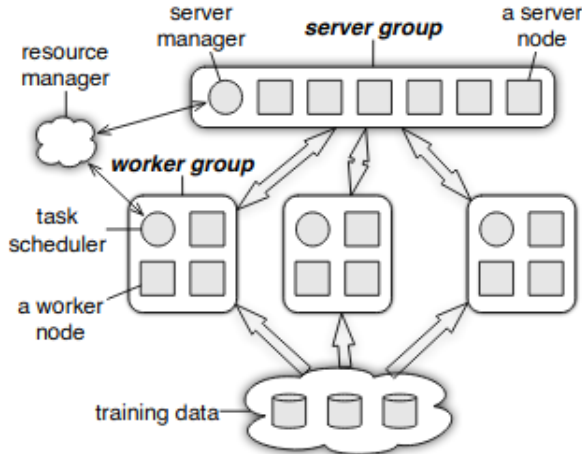


Figure 5: A visualization of the parameter server. The parameter server uses modules such as a server manager, server nodes and task scheduler for synchronizing nodes across the cluster. The parameters are distributed across a set of server nodes. Each server node has the ability to push out its local parameters and to pull in remote parameters. [7]

### 2.1.2.1 Synchronous updates

When updating the parameter server in a synchronous fashion, the parameter server waits for all the updates from the compute nodes before aggregating them. This will lead to high consistency, but this is not a scalable approach, since the parameter server has to wait for the slowest compute node before aggregating the updates and storing them on the server node. The parameter server aggregates the weight updates from the servers by taking the average over all the compute nodes:

$$\Delta w_t = \frac{1}{n} \sum_{i=1}^n \Delta w_i. \quad (2)$$

In Equation 2,  $\Delta w_i$  are the individual weight updates from each of the  $n$  compute nodes.  $t$  denotes the current timestep. Equation 2 describes how the parameter server averages the weight updates calculated in the current timestep  $t$ . After the aggregation, the parameter server updates the weights for the next timestep  $t + 1$ :

$$\Delta w_{t+1} = w_t - \eta \Delta w_t, \quad (3)$$

where  $\eta$  is the global learning rate, and  $w_t$  are the weights of the global model. The compute nodes in the local cluster are going to use  $w_{t+1}$  for the next optimization rounds, before repeating the processes described by Equations 2 and 3.

### 2.1.2.2 Asynchronous updates

When updating the parameter server in an asynchronous fashion, the parameter server does not wait for the updates from all of the compute nodes before storing the latest parameters in the server node. Unlike synchronous updates described in Section 2.1.2.1, this method reduces model consistency, but is scalable since the parameter server does not have to wait on the slowest compute node. Asynchronous updates can lead to *stale* parameters because of updates coming from slow nodes and the parameters on the parameter server getting overwritten. Examples of distributive learning algorithms which use asynchronous updates are the HOGWILD! algorithm [8] and Downpour SGD [4]. The Stale Synchronous Parallel (SSP) model described in the paper *More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server* [9], attempts to reduce parameter staleness by introducing a staleness parameter  $\tau$ . This staleness parameter makes the learning rate a function of the staleness in the parameters. If the most recent update was made at time  $t = t_0$ , the parameter server gets a new update at  $t = t_1$ , and the staleness parameter is calculated as

$$\tau = t_1 - t_0. \tag{4}$$

The learning rate is then defined as

$$\eta = \begin{cases} \eta_0/\tau & \text{if } \tau \neq 0 \\ \eta_0 & \text{otherwise.} \end{cases} \tag{5}$$

### 2.1.2.3 Decentralized updates

A decentralized update method does not require a parameter server, since the worker nodes communicate with each other. This leads to lower communication costs, since the nodes do not have to write and read parameters from an external server. Examples of such algorithms are *gossip algorithms*, which communicate and aggregate updates between each other in an exponential way [10]. GossipGraD, which is a gossip algorithm, is illustrated in Figure 6.

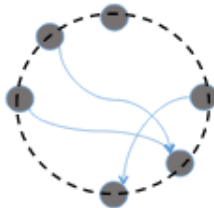


Figure 6: A Visualization of the GossipGraD Algorithm. There is no parameter server involved here, which can be seen by the nodes communicating updates directly to each other [10].

There are also methods that require no communication at all between worker nodes, and one ends up with an ensemble of models. Averaging many different models can slow down inference on new data. This problem can be solved by using *knowledge distillation*. Knowledge distillation requires a new DNN, a mimic network, which trains on the labels provided by the ensemble model [11]. Another disadvantage to decentralized updates is that the communication cost is much higher compared to applying synchronous or asynchronous updates. The reason for the high communication cost is that the number of times the updates are shared is higher with decentralized updates, since all the nodes have to communicate with each other.

## 2.2 Federated Learning

Federated learning is a relatively new machine learning approach as it became an important research question as late as 2015 [12]. Federated learning was introduced as an extension to distributed machine learning, and offers a way to train models in the client’s domain on distributed data which is owned by indi-

vidual clients [13]. As explained in *Communication-Efficient Learning of Deep Networks from Decentralized Data*, the basic idea behind federated learning is that learning tasks are solved with a loose federation of clients which are managed by a central server. The clients consists of a multitude of participating devices. In federated learning, clients are able to download the current global model from the server and train the model on their local data. Once the clients have trained their individual models, each client's updated model is sent back to the central server where the local models are aggregated [14]. These updates are only a means to improving the global model, and are therefore not stored once aggregated.

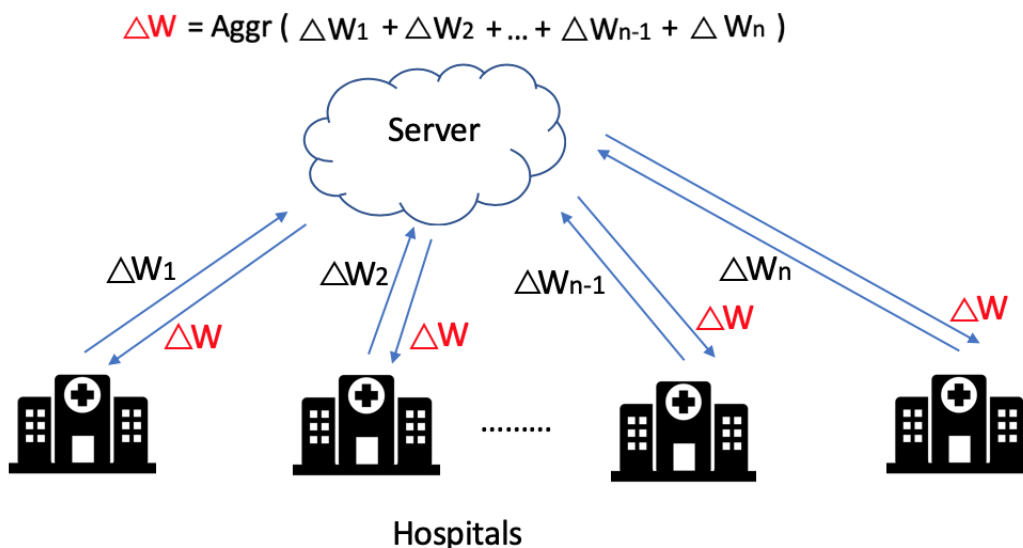


Figure 7: The figure illustrates four hospitals that each downloads the global model from the central server. The hospitals train the downloaded model on their respective local data, and sends their individual updates back to the central server. The central server aggregates these updates, resulting in an updated global model <sup>2</sup>.

Figure 7 further illustrates the workflow in federated learning. In this illustration each hospital is considered a different client, and are coordinated by the central server. As illustrated, the current global model's weights  $\Delta W$  is sent to each of the participating clients at time  $t$ . The participating clients then train the current global model on their individual local data. This results in each individual client having unique models with updated model weights  $\{\Delta W_i | 1 \leq i \leq n\}$ . These weights are then sent back to the central server where each of the clients individual updated weights are aggregated into *one* updated global model with weights  $\Delta W_{t+1}$ . This process is repeated until the model has converged.

This learning approach is quite unique in the sense that it allows clients to collectively reap the benefits of a shared model trained on large amounts of data without needing to disclose their local data. From a privacy perspective, this is a huge stride from traditional, centralized machine learning as it decentralizes learning by removing the need to pool data into one single location. This decentralization allows for data minimization in the sense that both the global model and the clients only have access to the data that is necessary, and once the updated weights are aggregated the individual weights are forgotten [14]. The characteristics of federated learning makes the approach well-suited for supervised learning tasks where the volume and sensitivity of the data is significant, and where the tasks will benefit greatly from training on real-world data from distributed devices [14].

<sup>2</sup><https://medium.com/@vaikkunthmugunthan/a-laymans-introduction-to-privacy-preserving-federated-learning-8ca0e6c73ad4>

### 2.2.1 Aggregation Methods

As described in section 2.2, each participating client downloads the global model and trains it on their individual local data. Once training is completed, the updated model weights are sent back to the server where they are aggregated. In order to aggregate the weights the central server receives, federated learning utilizes an aggregation method. The aggregation algorithm’s first task is to have each client  $k$  compute the average gradient on the client’s local data at the current state of the global model  $w_t$  [14]. This task can be described by the following equation

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta \nabla f(w_t), \tag{6}$$

where  $w_{t+1}^k$  are the parameters of client  $k$  and  $\eta$  is the learning rate at which the parameters are calculated. The central server then aggregates these parameters  $w_{t+1}^k$  by calculating a weighted average of the updates from each client in an attempt to improve the global model. This can be described by the following equation [14]

$$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k. \tag{7}$$

where  $w_{t+1}$  is the updated state of the global model,  $n_k$  is the number of the data points at client  $k$  and  $n$  is the number of clients.

There currently exists several different methods for aggregation in federated learning, and all of these have advantages and disadvantages which will be covered in the following subsections. However, one thing they all have in common is that they aim to address the key properties of federated optimization. [14]. In *Communication-Efficient Learning of Deep Networks from Decentralized Data* [14], federated optimization is described as optimization problems that is implicit in federated learning. The key properties of federated optimization are considered to be the following:

- **Non-IID Data:**

In federated learning, models are often trained on non-IID data. Non-IID data is data which is neither independent nor identically distributed. The reason why federated learning often has non-IID data is because the participating devices often consists of data specific to the use of one particular device. The data held by one specific client will therefore not be representative for the entire federation, thus introducing an optimization issue that needs to be addressed by the federated aggregation algorithm.

- **Unbalanced Data:**

Another concern in federated learning is that data often is heavily unbalanced. This issue arises as a result of clients using their devices differently, especially in terms of how much a device is utilized. Clients that use their device more often than others, will gather more training data. This is an issue since the data held by each client may vary in quantity, and can result in a global model which is biased towards a specific client’s data.

- **Profoundly Distributed Data:**

Data used in federated learning tasks is often massively distributed. Using a federated learning approach, one should expect that the number of clients participating in the optimization is much larger than the average number of examples each client holds. This means that the aggregation methods used in federated learning must pay attention to the distribution of the data to create a well-performing and unbiased global model.

- **Communication Constraints:**

In federated learning one relies on a loose federation of participating devices. This can lead to optimization issues such as devices in the federation being offline or experiencing communication constraints. It is therefore essential that the aggregation method used in federated learning is able to handle varying participation, and is able to select clients that can participate in training.

With these key properties in mind, consider an objective function of the form

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) = \frac{1}{n} \sum_{i=1}^n f_i(w). \tag{8}$$

$f_i(w) = l(x_i, y_i; w)$  is the prediction loss of one example  $(x_i, y_i)$  where the prediction is made based on the model weights  $w \in \mathbb{R}^d$ . Any machine learning algorithm designed to minimize such an objective function is suitable for training with multiple clients [14]. Now consider that the data is distributed across  $K$  clients, and client  $k$  has  $P_k$  which is the set of data points where  $n_k = |P_k|$ . The problem in Equation 8 can then be translated to a federated learning task with the following equation

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \quad , \tag{9}$$

and describes the local loss  $F_k$  of one single client [14]. Equation 9, illustrates a solution to the key property in federated optimization described as *Profoundly Distributed Data*.

As discussed in this section, the key properties of federated optimization pose significant challenges in comparison to standard distributed learning. In the following sections, several aggregation methods will be described, and the first method will present the simplest solution to the problem behind Equation 9.

### 2.2.1.1 FedSGD

Federated Stochastic Gradient Descent, or FedSGD, is the baseline algorithm in federated learning. The mechanism of action behind the algorithm consists of performing a single step gradient descent for each of the participating clients and updating their respective weights accordingly. Finally, the algorithm aggregates the gradients calculated and updates the state of the global model. This process is described in Section 2.2.1.

The following pseudocode describes the FedSGD algorithm.



---

**Algorithm 1:** FedSGD [15]

$w_0$  are the weights of the initial global model.  $K$  are all the participating clients.  $\eta$  is the learning rate.

---

**Procedure** SERVERinitialize  $w_0$ ;**for**  $t = 0, 1, 2, \dots, T$  **do**    **for** all  $k$  in the  $K$  nodes in parallel **do**         $g_k \leftarrow \text{ClientUpdate}(k, w_t)$ ;    **end**     $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^K \frac{n_k}{n} g_k$ ;**end**return  $w_T$ ;**Procedure** ClientUpdate( $k, w$ ) $\mathcal{B} \leftarrow$  split  $\mathcal{P}_k$  to set of batches; $G \leftarrow$  List of gradients for each mini-batch;**for** all  $b \in \mathcal{B}$  **do**     $g \leftarrow \nabla l(b; w)$ ;     $G.append(g)$ ;**end**return Average( $G$ );

---

For each client  $k$  in  $K$ , FedSGD calculates the gradients. This is done by FedSGD randomly selecting  $b$  examples from the client and evaluating the  $b$ 's at the same  $w$ . This is called batching, and is utilized in order to enhance data parallelism. Once the gradients have been calculated, the gradients are sent back to the central server where they are aggregated into *one* updated global model. The way FedSGD solves the optimization-issue that is *profoundly distributed data* described in Section 2.2.1, is by utilizing Equation 9 in the aggregation of the client gradients. In other words, FedSGD uses weighted averaging to aggregate the gradients of each client which can contribute to building an unbiased global model.

The reason why FedSGD is described as the simplest method to solve the optimization-issue of massively distributed data is because it only performs a single step gradient descent. This means that FedSGD will compute the gradients for each client once, and then immediately send the gradients back to the server where they are aggregated. In contrast to FedAvg, this can be inefficient as it will take longer to train a well-performing model. In addition, an adversary can use *gradient inversion* to extract information about the training data from the clients [16].

### 2.2.1.2 Federated Averaging

Federated averaging, also known as FedAvg, is a more advanced aggregation algorithm compared to FedSGD. The following pseudocode describes the FedAvg algorithm.

---

**Algorithm 2:** FedAvg [14]

$E$  is the number of client epochs.  $B$  is the local mini-batch size.

---

```
Procedure SERVER
initialize  $w_0$ ;
for each round  $t = 0, 1, 2, \dots, T$  do
   $m \leftarrow \max(C \cdot K, 1)$ ;
   $S_t \leftarrow$  (random set of  $m$  clients);
  for each client  $k \in S_t$  in parallel do
     $w_{t+1}^k \leftarrow$  ClientUpdate( $k, w_t$ );
  end
   $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ ;
end
return  $w_T$ ;

Procedure ClientUpdate( $k, w$ )
 $\mathcal{B} \leftarrow$  split  $\mathcal{P}_k$  to set of batches of size  $B$ ;
for each local epoch  $i$  from 1 to  $E$  do
  for batch  $b \in \mathcal{B}$  do
     $w \leftarrow w - \eta \nabla l(b; w)$ ;
  end
end
return  $w$ ;
```

---

Algorithm 2 describes the mechanism behind the FedAvg aggregation method. Firstly, the initial state of the global model  $w_0$  is initialized. Furthermore, a random set  $S_t$  of  $m$  clients is chosen. Each of these clients then performs gradient descent steps on their respective data. Finally, the updated weights of each client is sent to the central server where the weights are aggregated with a weighted arithmetic mean.

The algorithm works similarly to the FedSGD algorithm described in Section 2.2.1.1. Both algorithms use Equation 6 to compute the local gradient average for each client  $k$  and use Equation 7 to aggregate the parameters. However, the difference is that FedAvg can apply multiple steps of gradient descent before sending the clients parameters back to the central server to be aggregated. In the FedAvg algorithm, the parameters of the model are aggregated instead of the gradients produced by the clients. This is because the clients run several epochs of stochastic gradient descent, and the gradients are recomputed for every update to the local model. The number of local updates per round  $u_k$  is decided by the number of rounds each clients makes over its local data  $E$ , the local mini-batch size  $B$  and the number of local examples  $n_k$  for client  $k$ . Equation 10 describes the correlation.

$$u_k = E \frac{n_k}{B} \tag{10}$$

Equation 10 can further explain the connection between FedAvg and FedSGD, as inserting  $B = \infty$  and  $E = 1$  into the equation would result in  $u_k \rightarrow 0$ . This means that the number of local updates per round at client  $k$  converges towards 0, which corresponds to FedSGD.

By examining Algorithm 2, it becomes clear that one can fine-tune a multitude of parameters to optimize the performance of the model. This makes the algorithm more flexible than FedSGD. An advantage FedAvg provides, is that it makes it possible to adjust  $E$  and  $B$  which in turn can decrease the communication cost. This is due to the fact that running more local SGD updates per round will lead to less communication as it more rarely sends updates to the central server where the updates are aggregated. This will cost less in terms of communication [14]. However, there are some risks in adjusting these parameters. For example, by utilizing a large  $E$  during training, the training time will increase significantly. In addition, a substantially large  $E$  could also cause the models at each client to become specialized due to clients

training on the same data for an extended period of time. This can result in high variance between the clients and their respective weights, and eventually decrease the performance of the global model. It can also cause increased memorization (see Section 2.5).

### 2.2.1.3 Secure Aggregation

Federated learning increases privacy since the server does not need to store privacy-sensitive data before doing learning tasks on the data. In addition, federated learning does not require the clients to send more data to the server, since they have a model available on their local device. However, federated learning still requires the clients to send their updates to the central server, which allows the server see all the updates before aggregating them. This can be a problem since the weight matrices can contain privacy-sensitive information about the local data of a client, which can be extracted by looking at how sensitive some weights are given some input data. Moreover, the updates of one client are visible to other participating clients due to the fact that the updates are aggregated and sent back to the clients. Thus, the weights of the final model can still reveal information about the sensitive data used to fit the model.

The *Secure Aggregation* algorithm was introduced by Bonawitz et. al [17], and this algorithm uses the *Secure Multiparty Computation* (SMC) protocol to protect the privacy of client updates. The clients can act as parties in the SMC protocol and calculate the aggregated weights between each other before sending the aggregated weights to the server. Thus, the server will only see the aggregated weights, and update the global model based on this value. The SMC protocol lets the parties

$P_1, P_2, \dots, P_n \in \mathcal{P}$  with inputs  $x_1, x_2, \dots, x_n$  calculate a function  $f(x_1, x_2, \dots, x_n)$  such that their inputs stay private. Even if a subset of  $\mathcal{C} \subset \mathcal{P}$  collude, these parties can only see  $\{x_i, f(x_1, \dots, x_i, \dots, x_n) : P_i \in \mathcal{C}\}$ , and nothing more [18]. In the federated learning setting, the  $P_i$ 's are the different clients participating in a round,  $x_i$  are their updates, and  $f$  is the aggregation function to be computed by the clients. The secure computation of the aggregation function  $f$  can be achieved by for example using *Homomorphic Encryption* (see Section 2.7) or *Secret Sharing*.

Although Secure Aggregation achieves privacy by *hiding* the single updates from the server, it does come at a cost of both computational and communication complexity. The clients have to communicate with each other using the SMC protocol, and if there are many clients participating in a federated learning round this will be expensive with respect to communication. Moreover, it will take longer to aggregate the weights if there are many participating clients. As mentioned above, the final model can still reveal information about the data used to fit the model. This is a problem after using Secure Aggregation, since the aggregated weights are still being sent back to all the clients. Differentially-Private FedAvg (see Section 2.6) mitigates this problem by clipping the updates and adding noise to the aggregated weights. Another disadvantage with the Secure Aggregation scheme is that the algorithm is vulnerable to poisoning attacks because the server cannot inspect and filter the individual updates.

## 2.2.2 Communication Efficiency in Federated Learning

The federated learning approach provides a new, more secure way to train models in the sense that the participating devices are capable of training a model together without having to share any raw local data with the central server. The clients are able to train a local model which is then aggregated by the central server, improving the overall global model. In addition to the privacy benefits, this learning approach allows for efficient use of network bandwidth and limits latency. The efficient use of bandwidth in federated learning comes as a result of clients only sending the updated model weights rather than communicating the raw data. By limiting the data that is transmitted, the communication cost will be far less and the bandwidth will be better utilized. Moreover, latency can be limited since the models are consistently being trained and updated. In addition, federated learning allows for real-time predictions as they are made locally on the client's device [19].

However, federated learning presents some problems that in turn can lead to an increase in communication cost. In large scaled networks there are a multitude of clients participating in training. The computation

power of these and their ability to participate in training may vary. In addition, most mobile devices experience some constraints such as bandwidth, battery and computation limitations. The heterogeneity of the participating devices and the limitation that most mobile devices encounter, constitutes a genuine concern with regards to communication cost in federated learning [19]. In order to address these concerns, it is necessary to explore how these limitations are affected by the main principles of the federated learning algorithms. As stated in Section 2.2.1, the fundamentals of these algorithms involves having each client compute the average gradient on the client’s local data by using the current state of the global model, and then having the central server calculate a weighted average of these parameters in order to improve the global model. Each client computes their gradients by performing a given amount of steps of stochastic gradient descent. According to *Evaluating the Communication Efficiency in Federated Learning Algorithms* [19], each step of stochastic gradient descent is fairly expensive in regards to battery usage on the participating devices. Therefore, it is important to limit the iterations of stochastic gradient descent in the algorithms to prevent clients from being unable to participate in training due to limitations in battery capacity. Furthermore, the amount of memory that stochastic gradient descent uses or references while running, grows linearly with the batch size given in the federated learning algorithm [20]. This can cause issues in the sense that participating devices, e.g. mobile devices, might have limited memory. This can force a decrease in batch size, which further can result in an increase in communication cost.

In *Robust and Communication-Efficient Federated Learning from Non-IID Data* [20], it is concluded that the following conditions needs to be addressed in order to obtain a communication-efficient federated learning algorithm:

- **Robust to non-IID data, small batch sizes and unbalanced data:**

The federated learning algorithm is considered robust to non-IID data if the training converges without any regard towards local distribution of client data.

- **Robust to large number of clients and partial client participation:**

The federated learning algorithm is considered robust to partial client participation if the effect of reduced participation is not critical in terms of model performance.

- **Communication compression in both directions between clients and central server:**

The federated learning algorithm needs to be able to adequately compress the communication between clients and server, in both directions. The compression is considered strong if the compression rate is greater than  $\times 32$ , and weak if the compression rate is smaller or equal to  $\times 32$ .

The FedAvg algorithm described in Section 2.2.1.2, is not able to satisfy all these conditions [20]. Using the FedAvg algorithm in training while there is partial participation can result in the optimization process moving away from the minimum, as well as causing the model to forget previously learned concepts. The algorithm is not considered to be robust to non-IID data either, as it is vastly sensitive to the degree of IID of the client data. However, FedAvg supports communication compression either way, and is considered to be able to achieve strong compression [20].

## 2.3 Attacks on Federated Learning

In recent years, federated learning has emerged as a new approach designed to address the issue of privacy in machine learning. The issue of privacy has become increasingly important as technology has progressed and larger quantities of data is being collected. New legal restrictions concerning privacy preservation are constantly being formulated, e.g. GDPR, and these are making centralized machine learning less feasible [21].

The main advantage in utilizing federated learning in contrast to centralized learning, is that it allows clients to keep their data local and therefore more secure. However, the distributed nature of federated learning also makes room for new privacy concerns and threats. There is a multitude of attacks that could

pose a potential threat to the learning approach, and the attacks are often classified as either an insider attack or an outsider attack. Insider attacks are attacks that are initiated by the central server and the participating clients in the federated learning network. Outsider attacks, however, are attacks that are performed by individuals outside the federated learning network. Such attacks often include individuals eavesdropping on the communication that occurs between the central server and the clients. Both attack types pose a significant threat to federated learning, but one could argue that insider attacks constitute the stronger threat. This is due to insider attacks strictly enhancing the capability of adversaries, as well as being more difficult to prevent [21]. Outsider threats can be greatly prevented by utilizing a secure, encrypted communication line, while insider threats can only be moderately reduced by adjusting the federated learning algorithms to take such attacks into account when aggregating. Consequently, the most problematic privacy concerns that arise while training with federated learning is the potential threat of insiders, also known as malicious participants. Two major insider threats in regards to federated learning are [21]

- ***Data and model poisoning attacks***

Attacks that aims to poison the global model in an attempt to either prevent it from learning or to create a biased model that benefits the adversary.

- ***Inference attacks***

Attacks that normally does not aim to poison the global model. Instead they attempt to either produce wrong outputs or collect information about the model’s properties. Such attacks target the privacy of participants.

Generally, attacks in federated learning can be difficult to detect due to the non-IID data distribution which is common in federated learning. This type of distribution allows enough room for the adversary to hide malicious data and updates without being easily detected [22].

### **2.3.1 Data and model poisoning attacks**

As described in Section 2.3, data and model poisoning attacks attempt to corrupt the global model in some way. These types of attacks are carried out in the training phase of federated learning, and can either be random or targeted attacks. Adversaries performing random poisoning attacks attempts to decrease the model performance by reducing the accuracy of the model, while adversaries performing targeted attacks aim to mislead the model to output target labels defined by the adversaries themselves.

Poisoning attacks can be carried out either by targeting the training data (*data poisoning*) or the model (*model poisoning*). Data poisoning attacks are executed during the local data collection process, and is often categorized as either clean-label or dirty-label. Clean-label data poisoning attacks requires the adversary to introduce an input-label pair that does not seem to be mislabeled. This is because one assumes that there is a process where data and labels are validated as a plausible pair. Dirty-label data poisoning attacks are, on the other hand, attacks where the adversary does not have to concern themselves with this process. The adversary can introduce numerous input-label pairs with the target labels of their choosing in order to hopefully cause the model to eventually misclassify data [21]. Previous research regarding data poisoning in federated learning has demonstrated that such attacks can cause a considerable reduction in the model accuracy, regardless of the number of malicious clients participating. However, the effectiveness of poisoning attacks increases significantly if malicious clients are highly available to participate in training and also choose to participate in later rounds [23].

Model poisoning attacks differ slightly from data poisoning in the sense that model poisoning attacks are executed during the local training process at the client. An adversary will attempt to manipulate the training process by either poisoning the local model updates or by inserting hidden backdoor into the updates. These poisoned updates will be sent back to the central server where they are aggregated, and will in turn poison the global model. This can lead to the model misclassifying certain inputs with high certainty [21].

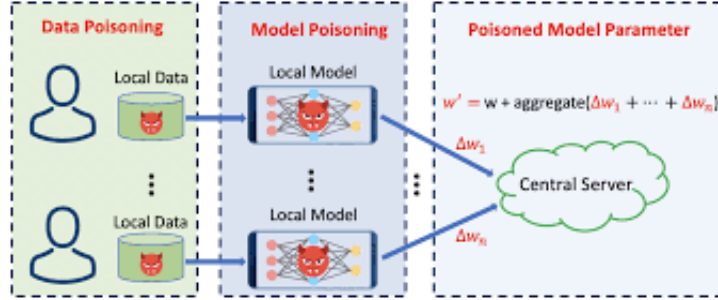


Figure 8: The figure illustrates where data and model poisoning occurs in the federated learning workflow. Data poisoning is executed during data collection phase, while model poisoning is performed during model training [24].

Figure 8 illustrates data and model poisoning in a federated learning setting. The illustration describes where in the federated learning protocol each poisoning attack is performed. Data poisoning attacks are executed on the local data by a malicious participant with the purpose of manipulating their data to eventually cause the global model to misclassify data. Model poisoning attacks are executed while training the local model on the malicious participant’s data. The adversary aims to poison the local model updates that are sent back to the central server.

### 2.3.2 Inference attacks

As described in Section 2.2, the clients partaking in the federated learning protocol will receive a global model from the central server. Each client then locally trains this model on their individual data before the updated model is sent back to be aggregated at the central server. In this process the clients are openly communicating updates which consist of weights. There are certain privacy risks in communicating these weights even though federated learning enforces the principle of data minimization. The model updates can leak information about participants training data due to models, especially deep learning models, memorizing outlier data. Moreover, an adversary can execute an inference attack by saving snapshots of the parameters of the federated learning model. This allows the adversary to observe the difference between the snapshots which can be exploited in the sense that the adversary can pinpoint the aggregated updates from all participating clients [21]. Such attacks can result in the adversary being able to recover significant amounts of data and details surrounding it. The adversary can also retrieve the original training data.



Figure 9: Illustration of Inference Attack. The figure describes two clients participating in federated learning. Both clients download the global model, train the model on their local data and send their individual updates back to the central server. An adversary saves an image of the aggregated model, and calculates the difference between the new image and the last image saved. Based on the difference of the gradients, the adversary can infer information <sup>3</sup>.

Figure 9 illustrates a general overview of how an inference attack is executed. First, participating clients receive the weights of the global model from the central server. Each client trains the global model on their local data, and sends their gradients back to the server. An adversary then saves a snapshot of the gradients belonging to the aggregated model. The adversary saves such snapshots each time clients' gradients are aggregated. This allows the adversary to calculate the difference between consecutive aggregated models, and to obtain information about the clients training data.

### 2.3.2.1 Generative Adversarial Networks Attack

An example of an inference attack used on deep federated learning models are generative adversarial networks (GAN) attacks. Generative adversarial networks comprise of two neural network modules, a generator and a discriminator. The generator's task is to generate samples based on random noise input which should approximate the training data. The task of the discriminator is to observe the difference between the samples it collects from both the generator and the actual training data. These two modules are trained concurrently, and will progress and learn in order to ensure that the generator will eventually be able to generate realistic training samples [25]. Such networks can be used to execute an inference attack in federated learning because the distributed, real-time nature of federated learning allows an adversary to train a GAN which can generate realistic samples of training data which should have been private [21]. Generative Adversarial Networks Attack is illustrated in Figure 10.

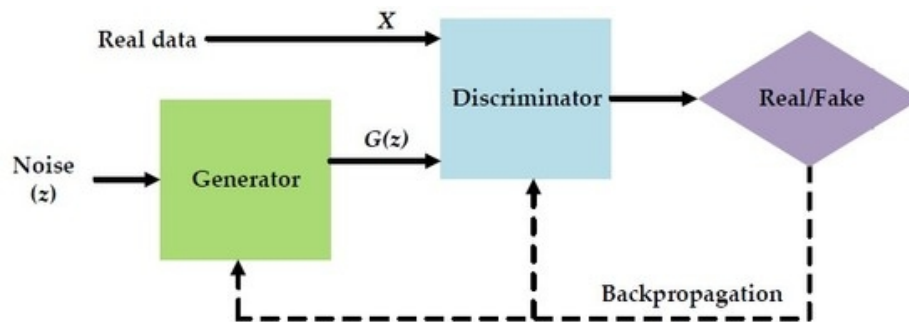


Figure 10: The figure illustrates how a generative adversarial network is trained. The network comprise of a generator  $G$  and a discriminator  $D$ . First, the generator  $G$  takes noise  $z$  as input. The generator uses the noise  $z$  to generate training samples which is then transmitted to the discriminator  $D$ . The discriminator  $D$  compares the training samples from the generator with the real training data, and will determine how far the training samples are from the real data. Using backpropagation, the network will train and learn in order to ensure that the generator will eventually be able to generate realistic training samples [26].

<sup>3</sup>[https://www.researchgate.net/figure/Inference-attacks-against-federated-learning-passive-adversary-by-Melis-et-al-35\\_fig5\\_341478640](https://www.researchgate.net/figure/Inference-attacks-against-federated-learning-passive-adversary-by-Melis-et-al-35_fig5_341478640)

## 2.4 Robust Federated Aggregation

This section is going to look at the *aggregation step* in federated learning. As mentioned in Section 2.2, the server broadcasts the model to the participating devices. The devices apply local updates to the global model, and then the updated models are sent back to the server to be aggregated. In Section 2.2.1, one can observe that the aggregation methods use a weighted arithmetic average to aggregate the updates. This is described by Equation 7.

Suppose that a client  $i$  sends an update which is a statistical outlier. This can happen because the client has outlier data, or if the client is sending corrupted updates to the server as a result of an adversarial attack, as discussed in Section 2.3. This outlier update will then have a significant impact on the aggregation, since statistical mean is not a robust method. An example is that if  $n_i w_{t+1}^i \gg n_j w_{t+1}^j$ , where  $1 \leq i, j \leq K$  and  $i \neq j$ , and all the clients  $j$ 's updates have low variance, then

$$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k \approx \frac{n_i}{n} w_{t+1}^i. \tag{11}$$

Equation 11 shows that the aggregated model  $w_{t+1}$  will be strongly influenced by the updates of client  $i$ , which makes the global model vulnerable to corrupted updates.

The paper *Robust Aggregation for Federated Learning* [27] remedies the problem described above by using geometric median instead of arithmetic average. The solution is presented in Section 2.4.3. The goal of Krishna et al. [27] was to develop an aggregation procedure for federated learning which is both robust against corrupted updates sent by client devices and privacy-preserving. Privacy-preservation can be expensive, since the system can require more than just sending the updates to the server [27]. This may lead to more communication overhead, and may require a more complex implementation that differs from already existing solutions. Krishna et al. required that the aggregation should be communication-efficient and practical, i.e., require minimal engineering overhead relative to existing systems.

### 2.4.1 Secure Average Oracle

Bonawitz et al. [17] were the first to implement a Secure Average Oracle, which is a failure-robust protocol for secure aggregation of high-dimensional data. This aggregation method computes the weighted arithmetic average

$$\sum_{k=1}^m \alpha_k w_k \tag{12}$$

such that client  $k$ 's weight-update  $w_k$  is not revealed to any other client or to the server. In Equation 12,  $m$  is the number of selected clients, and  $\alpha_k$  is the weight of the client  $k$ , i.e., how significant the client is when aggregating the updates. Thus, the Secure Average Oracle aggregates the updates in a privacy-preserving manner, and is typically implemented using cryptographic protocols. Krishna et al. decided to use the Secure Average Oracle as a module in their robust aggregation method. By using a small number of calls to a Secure Average Oracle, the goals stated in the last paragraph of Section 2.4 would be fulfilled. By definition, this would make the aggregation method privacy-preserving since the weights are not shared to the clients or to the server. Moreover, communication-efficiency would be achieved if the number of calls to the Secure Average Oracle is small. Since Krishna et al. reused the Secure Average Oracle, which was already introduced by Bonawitz et al. in 2017, practicality was also achieved as it requires minimal divergence from existing solutions [27].

This aggregation scheme requires multiple calls to a Secure Average Oracle, making the aggregation *iterative*. Multiple calls are required because a single call to the Secure Average Oracle only returns the mean as described in Equation 7. In Section 2.4, it was established that the mean is not robust which means that the aggregation method has to be iterative.



### 2.4.2 Corruption Model

Before aggregation, each device computes an update. If the device is *corrupted*, the update may be arbitrary. If not, the update is in line with the stochastic gradient descent on the local data. During aggregation, all devices behave nominally, i.e., they give their update to the Secure Average Oracle, even if it is corrupted [27]. In other words, corrupted and non-corrupted devices are treated equally during the iterative aggregation. This happens because the Secure Average Oracle does not know if a device has been corrupted, or if it is operating with outlier data.

The term *corrupted* may be ambiguous, since a device can be subject to several types of corruption, and each corruption type depends on the capability of the adversary. Krishna et al. allowed these corruption types in their robust aggregation algorithm:

- **Non-adversarial:**

This corruption type may rise from bugs in the hardware or software of the client device, bugs in the pipeline, etc. As the name suggests, there is no adversary involved here, and as long as the device behaves as expected during aggregation, this type of corruption is allowed.

- **Static data poisoning:**

Static data poisoning is when the adversary can modify the training data of a client, but the modification is independent of the training of the federated model. This is allowed under the corruption model. This is explained in greater detail in Section 2.3.1.

- **Adaptive data poisoning:**

Adaptive data poisoning extends static data poisoning by modifying the training data, while being able to read the federated model. This makes the adversary able to read the model, and then modify the training data such that it *hurts* the global model the most.

- **Update poisoning:**

Update poisoning is an extension of both static and adaptive data poisoning. The adversary is able to write the update of the model directly, without having to change the data to get the desired update. This is the most general setting that is allowed in this corruption model. This is explained in greater detail in Section 2.3.1.

The corruption model described above does not allow an adversary to modify the aggregation method. This is possible in the Byzantine adversarial setting [27]. Thus, for each round of the iterative aggregation, the adversary can't modify the calculated average. Therefore, Byzantine robustness is not compatible with the Secure Average Oracle. Nevertheless, static and adaptive data poisoning, as well as update poisoning, are allowed in this corruption model [27].

### 2.4.3 Robust Aggregation using the Geometric Median

As shown in Section 2.4, the statistical mean is not robust against outliers. Therefore, Krishna et al. had to find a new way to aggregate the updates from the clients, since the Secure Average Oracle does not address this problem. A substitute for the statistical mean is the statistical median. This method is robust to outliers since all the values are sorted, and the value of the middle is returned as the median. In 1991, Lopuhaa and Rousseeuw [28] found that the method *geometric median* is robust. The geometric median is a method for finding the multivariate median of a set of points in a Euclidean space, and is defined as the minimizer of

$$g(z) := \sum_{k=1}^m \alpha_k \|z - w_k\|_2, \quad (13)$$

where  $w_k \in \mathbb{R}^d$  are the weight-updates from the different clients, and  $\alpha_k > 0$  defines how much each client is weighted. Therefore, a  $z \in \mathbb{R}^d$  has to be found such that the sum of the  $l_2$ -distances from  $w_k$  to  $z$  is

minimized. By comparing Equation 7 to Equation 13, one can observe that the first equation computes the average of the sum of the  $l_2$ -distances *squared*, while the second equation has no quadratic term. As shown in Figure 11, the mean tends to move towards the outlier, whereas the geometric median does not do that. For this reason, geometric median is a popular method in robust machine learning, starting with the seminal work of Nemirovski and Yudin in 1983 [29] [27].

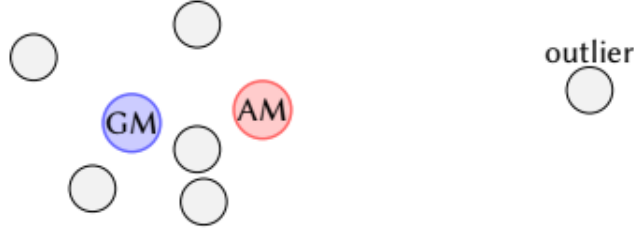


Figure 11: The arithmetic mean (AM) converges towards the outlier, while the geometric median (GM) does not [27].

The geometric median can be computed algorithmically with the Smoothed Weiszfeld algorithm from 1937 [30]

$$z = \frac{\sum_{k=1}^m \beta_k w_k}{\sum_{k=1}^m \beta_k} \quad \text{where} \quad \beta_k = \frac{\alpha_k}{\max\{\nu, \|z - w_k\|_2\}}, \quad (14)$$

where  $\nu$  is a threshold value for the max operator. In each iteration, the Secure Average Oracle receives the weight vectors  $w_k$  from each client  $k$ , and calculates an estimate of the geometric median  $z$ , where initial client weights are  $\beta_k = \alpha_k$ .  $z$  is broadcasted to the clients, where the clients calculate  $\beta_k$ . In other words, the clients adjust how much they should be weighted during the calculation of the geometric median. This can be seen in Equation 14, since the initial weight  $\alpha_k$  is divided by  $\max\{\nu, \|z - w_k\|_2\}$ . Suppose two clients  $i$  and  $j$  with their respective weight updates  $w_i$  and  $w_j$  and initial weights  $\alpha_i = \alpha_j$ . If

$$\|z - w_i\|_2 > \|z - w_j\|_2, \quad (15)$$

the  $l_2$ -distance of client  $i$ 's update from the estimated median is larger than that of client  $j$ , then

$$\frac{\alpha_i}{\max\{\nu, \|z - w_i\|_2\}} < \frac{\alpha_j}{\max\{\nu, \|z - w_j\|_2\}} \implies \beta_i < \beta_j. \quad (16)$$

Assume that  $\nu < \|z - w_j\|_2$ . Then,  $\beta_i < \beta_j$ , which means that client  $i$  is less significant than client  $j$ . This is a consequence of client  $i$ 's updates being further away from the geometric median, which could potentially be outliers. This process is also described by Figure 12. Each iteration is implemented with one call to Secure Average Oracle. Krishna et al. [27] showed that 3-5 iterations, or calls, to the Secure Average Oracle suffice to weight the clients appropriately. Therefore, this method is communication efficient.

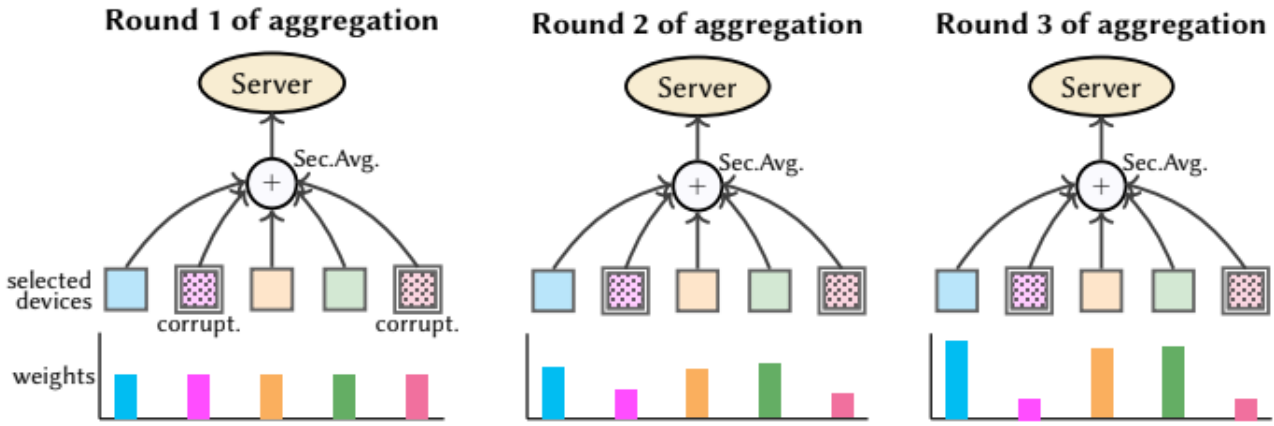


Figure 12: 3 rounds of Robust Federated Aggregation, where the corrupted devices get scaled down. At round 1, each client is weighted equally. For each round that passes, the corrupted clients are weighted less and less. In round 3 one can observe that the corrupted clients are mostly disregarded [27].

By substituting the arithmetic mean in the Federated Averaging algorithm with the geometric median, the Robust Federated Aggregation (RFA) algorithm is obtained. This algorithm can be observed in Algorithm 3.

---

**Algorithm 3:** Robust Federated Aggregation (RFA) [27]

---

**Procedure** SERVER  
initialize  $w_0$ ;  
**for** each round  $t = 0, 1, 2, \dots$  **do**  
     $m \leftarrow \max(C \cdot K, 1)$ ;  
     $S_t \leftarrow$  (random set of  $m$  clients);  
    **for** each client  $k \in S_t$  **in parallel do**  
         $w_{t+1}^k \leftarrow$  ClientUpdate( $k, w_t$ );  
    **end**  
     $w_{t+1} \leftarrow$  GeometricMedian( $w_{t+1}^k$ )  
**end**  
return  $w_T$ ;

**Procedure** ClientUpdate( $k, w$ )  
 $\mathcal{B} \leftarrow$  split  $\mathcal{P}_k$  to set of batches of size  $B$ ;  
**for** each local epoch  $i$  from 1 to  $E$  **do**  
    **for** batch  $b \in \mathcal{B}$  **do**  
         $w \leftarrow w - \eta \nabla l(b; w)$ ;  
    **end**  
**end**  
return  $w$ ;

**Procedure** GeometricMedian( $w^k$ )  
initialize  $\alpha_k$  as client weights;  
initialize  $\nu$  as  $l_2$  threshold;  
initialize  $N$  as total number of iterations;  
 $\beta_k = \alpha_k$ ;  
**for** each iteration  $i$  from 1 to  $N$  **do**  
     $z \leftarrow \frac{\sum_{k=1}^m \beta_k w^k}{\sum_{k=1}^m \beta_k}$ ;  
    **for** each client  $k \in S_t$  **in parallel do**  
         $\beta_k \leftarrow \frac{\alpha_k}{\max\{\nu, \|z - w_k\|_2\}}$ ;  
    **end**  
**end**  
return  $z$ ;

---

### 2.4.3.1 Theoretical Assumptions

The goal of the Robust Federated Aggregation (RFA) algorithm, and all the other federated aggregation algorithms, is to minimize

$$\min_{w \in \mathbb{R}^d} \left[ F(w) = \sum_{k=1}^K \alpha_k F_k(w) \right], \quad \text{with } F_k(w) = \mathcal{L}(w, x, y), \quad (17)$$

where  $(x, y)$  are covariate-response pairs, and  $\mathcal{L}$  is a loss function. In other words, the goal is to minimize the weighted average of per-device objectives  $F_k$ . To show the theoretical convergence rate bound of the RFA algorithm, Krishna et al. [27] assumed that

$$\mathcal{L}(w, x, y) = \mathbb{E}_{x, y \sim P} (y - w^T \phi(x))^2, \quad (18)$$

where  $\phi(x)$  is a feature representation of the covariate  $x$ . This is the least squares objective, which is both convex and quadratic. Further, each  $x, y$  are drawn from the same distribution  $P$  for all devices, and each  $\alpha_k = 1/K$ . Hence, the data has to be IID to prove the theoretical bound of the RFA algorithm, but it will

still converge for other loss functions  $\mathcal{L}$ , and for non-IID data [27]. Another assumption is that  $F$  is a  $\mu$ -strongly convex,  $L$ -smooth function, which are classical assumptions in convex optimization when proving theoretical bounds [31]. The fraction of corrupted devices has to be  $\rho < \frac{1}{2}$ , which means that the majority of devices has to be non-corrupted devices. This assumption corresponds to the properties of geometric median which has a breakdown point of 0.5, thus up to half of the points may be arbitrary. Other assumptions by Krishna et al. were that the feature representation  $\phi(x)$  is bounded, and that the noise variance is  $\sigma^2$ , where  $\sigma$  is the standard deviation in the probability distribution  $P$  [27].

## 2.5 Memorization

Machine learning models can leak information. As mentioned in Section 2.3.2, information leakage in federated learning is often associated with the model updates sent back to the central server and pose a severe threat as the information that is leaked can potentially be privacy-sensitive data. Information leakages in machine learning models are frequently related to the models memorizing data from the training dataset. Model memorization is often indicated by models overfitting. Overfitting is phenomenon that often occurs during model training, and is observed as lower test accuracy and higher training accuracy. However, this is not always the case. Some models such as large language models rarely indicate overfitting, but can nevertheless memorize outlier data [32].

There currently exists numerous inference attacks that an adversary can utilize in order to retrieve information directly connected to the training data, e.g. membership inference, GAN attacks and training data extraction attacks. Training data extraction attacks are relatively similar to GAN attacks in the sense that both attack types attempt to retrieve the original training data. However, the difference is that training data extraction attacks does not aim to only reconstruct representative training data like GAN attacks, but aims to obtain the verbatim training data [32]. The threat of training data extraction attacks are therefore much more significant than GANs, as such attacks can reveal actual sensitive information in the training dataset.

### 2.5.1 Eidetic Memorization

As described in Section 2.5, privacy leakages in machine learning models often occur as a result of the models having memorized data. In the context of training data extraction, eidetic memorization is defined as a model memorizing data that only appears in a small fraction of the training data, or if the datapoint is an outlier. [32]. In other words, models that retain unintended data are subject to eidetic memorization. We can define  $k$ -eidetic memorization with the following equation

$$X : |\{x \in X : t \subseteq x\}| \leq k. \quad (19)$$

A training example  $t$  is  $k$ -eidetic memorized for  $k \geq 1$  by a model  $f_\theta$  if  $t$  is extractable from  $f_\theta$  and  $t$  appears at most  $k$  times in the training data  $X$ . If  $k$  is small, and  $t$  is still extractable from the model  $f_\theta$ , then we have *strong* eidetic memorization [32]. In language models, a training example  $t$  is extractable by a model  $f_\theta$  if  $\exists p$  such that

$$t \leftarrow \operatorname{argmax}_{t':|t'|=N} f_\theta(t'|p). \quad (20)$$

In Equation 20, one can observe a prefix  $p$  which is the input to the model, where the model predicts the next string to be the training example  $t$  [32]. The notion of querying models for memorized training data can also be generalized to other deep generative models, and the challenge is only to find the right input  $p$  such that the desirable data can be extracted from the model. Methods for doing these type of extractions and training data reconstructions on non-generative models are discussed in Section 2.3.

The paper *What Do Compressed Deep Neural Networks Forget?* [33], describes a strong correlation between memorization and the size of a neural network. In a deep neural network (DNN), weights are allocated in sets according to which patterns they can recognize. This means that each set of weights can

recognize different patterns in the data. Each set of weights may differ in size because one set can cover more training examples than another set. The training examples recognized by smaller sets of weights, can be considered outliers. If the DNN is not able to find a pattern, outliers can end up being memorized due to outliers being assigned to specific weights. Therefore, the larger the DNN is, the more examples it can assign to weights, thus increasing its ability to memorize data [33]. The memorization can be a problem since an attacker can use forward-propagation to look at which weights are activated. A weight that is activated rarely may indicate that it has memorized data. An attacker can therefore execute an inference attack, which is described in Section 2.3.2, in order to obtain data representative to the original training data [33].

Sara et al. introduced a method for distilling neural networks such that the model does not memorize outliers in the data. The method is called *model distillation*, and concerns fitting the knowledge of a large neural network into a smaller one. This is done by having a *student* network learn from a larger *teacher* network, and finally pruning the student network by setting a subset of weights or filters to zero [33]. The distillation will cause the model to lose performance on outlier data. The student network will then have less ability to memorize the outliers, since it has fewer weights to allocate for pattern recognition on the data [33].

### 2.5.2 Memorization in Federated Learning

Federated learning is an approach that differs from centralized learning, and the issue of unintended memorization in federated learning is no exception. The nature of federated learning makes for an approach that is naturally more resilient against unintended memorization. Federated learning allows its clients to keep their data local which results in federated learning instinctively grouping the data according to the heterogeneous clients. The fact that data is grouped by clients, makes it conceivable that training examples restricted to a small group of clients may be encountered less frequently during training in federated learning. In turn, this can decrease memorization [34].

## 2.6 Differential Privacy

This section is going to look at federated learning from a privacy perspective. To reiterate, the federated learning loop consists of a server broadcasting the weights of the global model to participating clients, the clients perform SGD-steps on the weights with their local data, and send the updates back to the server to be aggregated. This is repeated until model convergence. In this loop, the aggregation step is the only time information is communicated from the clients to the server. Federated learning does not aim to collect privacy-sensitive data from the clients, but rather distribute the model so that it can be trained in the vicinity of the data-owners. However, these client-updates can contain privacy-sensitive data [35]. It is still substantially less information from an information theoretic point of view than if the client sent the whole raw dataset, but an attacker can still infer information about the privacy-sensitive data by looking at the single updates before aggregation. The inference of the client data from the weight updates depends on what kind of model is being trained, and can range from looking at non-zero coefficients in the weights from the clients, to more complex methods.

Before introducing *differential privacy* to the federated learning process, some important properties regarding the client-updates will be presented [35]:

- **The updates are ephemeral:**

The updates from the clients are never stored by the server, and are discarded immediately after aggregation. In non-federated machine learning, it is normal to hold the data from the clients in order to train several models on the data. Storing privacy-sensitive data comes with a high responsibility to protect that data. This is not the case in federated learning.

- **The updates are focused:**

The updates are minimal since the clients are only sending information that is strictly necessary to improve the current model. Instead of sending the whole dataset, only the weight updates are sent

to the server. This improves privacy and security by minimizing the *attack surface*.

- **The updates are in aggregate:**

When the server improves the global model, it only needs to know the aggregated updates, by for example using a weighted average or geometric median. The server does not consider every single update.

It would be desirable if the server only saw the aggregate of the updates, and did not have to process every single update from the clients. Methods like *Secure Multi-Party Computation* and *Secure Aggregation* try to achieve this, but the main challenge with these methods is that they are not robust to clients failing to deliver updates, which is often the case in *Cross-Device* federated learning. Even if one assumes that the server is only able to see the aggregated updates, the model might memorize a particular clients data, using *eidetic memorization* as discussed in 2.5. Differential privacy tries to solve this problem by *clipping* the updates from the clients, and adding *noise* to the aggregated weights.

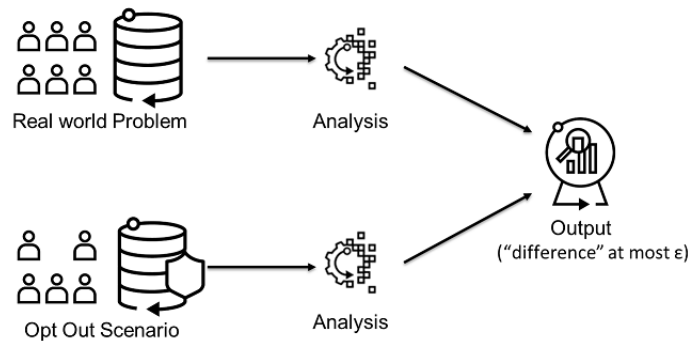


Figure 13: An overview of differential privacy. The output from a query with or without a client should be almost the same. In the figure, a third party queries two databases differing by one row, and receives an output with a *difference* of at most  $\epsilon$ <sup>4</sup>.

### 2.6.1 Definition of Differential Privacy

Suppose two databases  $d$  and  $d'$ , where  $d' \subset d$ , and  $|d| - |d'| = 1$ . In other words, suppose two databases which only differ by one element.  $\epsilon$ -differential privacy, or strict differential privacy, is achieved when querying on  $d$  and  $d'$ , and the result has a difference of at most  $\epsilon$ .  $\epsilon$  can be considered a *privacy budget*, i.e. if  $\epsilon$  is large the privacy is weaker than if  $\epsilon$  was small. A privacy mechanism  $\mathcal{M}$  gives  $\epsilon$ -differential privacy if

$$\Pr[\mathcal{M}(d) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(d') \in S], \quad (21)$$

where  $\mathcal{M}$  is a randomized function, where the input is a database, and the output is the released information. This means that  $\mathcal{M}$  can be considered a privacy-preserving query function [36].  $S \subseteq \text{Range}(\mathcal{M})$  is the subset where two queries are defined to give almost the same result. Equation 21 can be rewritten as

$$\frac{\Pr[\mathcal{M}(d) \in S]}{\Pr[\mathcal{M}(d') \in S]} \leq \exp(\epsilon). \quad (22)$$

$\epsilon \rightarrow 0 \implies \exp(\epsilon) \rightarrow 1$ , which means that the probability of the query  $\mathcal{M}$  being in the same subset  $S$  for both  $d$  and  $d'$  is going to be relatively equal when  $\epsilon$  is small.

From Equation 22, one can observe that it does not matter if a row is present or not in the database. One will get almost the same result from the query either way. Differential privacy provides privacy in the sense that it is not possible to determine if a row is present in the database which is queried, since  $d$  and

<sup>4</sup>Overview of Differential Privacy - [https://www.infosysblogs.com/infosysdigital/2020/07/differential\\_privacy\\_the\\_privacy.html](https://www.infosysblogs.com/infosysdigital/2020/07/differential_privacy_the_privacy.html)

$d'$  are interchangeable, and the queries will give statistically indistinguishable results. This is also explained by Figure 13, where querying on the databases differing by one element will give almost the same results.

Differential privacy is a perturbative privacy method, meaning that privacy is obtained by adding noise to the query function  $f$  which gives the *true* result of a query. For  $\epsilon$ -differential privacy, the noise is commonly drawn from a Laplace distribution [36], and is determined by the *sensitivity* of the true query function  $f$ . This sensitivity of the query function  $f : \mathcal{D} \rightarrow \mathbf{R}^n$  is defined as

$$\Delta f = \max_{d, d'} \|f(d) - f(d')\|_1, \quad (23)$$

$\forall d, d' \in \mathcal{D}$  such that  $d' \subset d$  and  $|d| - |d'| = 1$  [36].  $\mathcal{D}$  is the set of all databases, and  $\mathbf{R}^n$  is the  $n$ -dimensional set of results from the query. Thus, the sensitivity of the query function  $f$  is defined as the maximal difference between two databases differing by only one element. Now one can define how the noise is added to the query function  $f$ :

$$\mathcal{M}(d) = f(d) + \text{Laplace}\left(0, \frac{\Delta f}{\epsilon}\right). \quad (24)$$

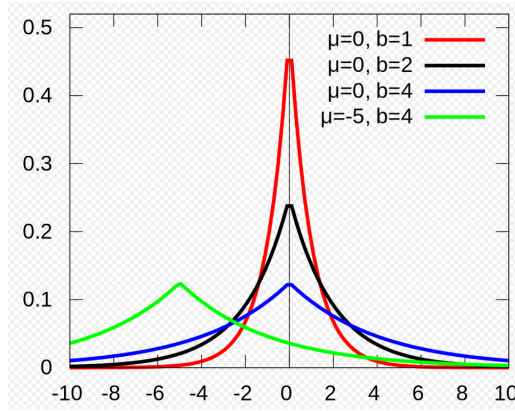


Figure 14: PDFs of Laplace distribution with different locations  $\mu$  and scales  $b$ <sup>5</sup>.

Figure 14 shows how much noise is added to the query function  $f$ . Since  $\text{Laplace}\left(0, \frac{\Delta f}{\epsilon}\right)$  is added to the query function, the location, or mean, of the noise is 0, and the standard deviation of the noise is

$$\sqrt{2}b = \sqrt{2} \left(\frac{\Delta f}{\epsilon}\right). \quad (25)$$

By looking at Equation 26 which describes the probability distribution function (PDF) of the Laplace distribution

$$\frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right), \quad (26)$$

one can observe that it is proportional to  $\exp\left(-|x - \mu|\frac{\epsilon}{\Delta f}\right)$ , since  $b = \frac{\Delta f}{\epsilon}$ . If one decreases  $\epsilon$ , the curve will become more flat, resulting in a large magnitude of noise. In addition, if the sensitivity of the query function  $f$  is high, the curve will become even flatter, giving more noise.

<sup>5</sup>The pdf of the Laplace distribution - [https://en.wikipedia.org/wiki/File:Laplace\\_pdf\\_mod.svg](https://en.wikipedia.org/wiki/File:Laplace_pdf_mod.svg)



### 2.6.1.1 Relaxed Definition of Differential Privacy

The relaxed definition of differential privacy can be used to obtain a more flexible privacy preserving mechanism:

$$\Pr [\mathcal{M}(d) \in S] \leq \exp(\epsilon) \cdot \Pr [\mathcal{M}(d') \in S] + \delta. \quad (27)$$

The  $\delta$  in Equation 27 is added to the  $\epsilon$ -differential privacy definition described in Section 2.6.1, and accounts for the probability that the privacy guarantee of  $\epsilon$ -differential privacy is broken. This relaxed definition is also known as  $(\epsilon, \delta)$ -differential privacy [36]. The sensitivity of the query function  $f$  is now calculated in the following way:

$$\Delta_2 f = \max_{d, d'} \|f(d) - f(d')\|_2. \quad (28)$$

Equation 23 and 28 are relatively equal, the only difference is that the noise is scaled from the  $l_1$ -norm to the  $l_2$ -norm. The noise in  $(\epsilon, \delta)$ -differential privacy is drawn from the Gaussian distribution. Equation 29 describes how the noise is added to the query function [36].

$$\mathcal{M}(d) = f(d) + \frac{\Delta_2 f}{\epsilon} \mathcal{N}\left(0, 2 \ln\left(\frac{1.25}{\delta}\right)\right). \quad (29)$$

### 2.6.1.2 Moments Accountant

In Section 2.6.1,  $\epsilon$  is described as a privacy-budget. To find out how much  $\epsilon$  is being used for each epoch of training, the notion of *privacy loss* is utilized. The paper *Deep Learning with Differential Privacy* [37] introduced a method for tracking privacy loss. Abadi et al. [37] framed the privacy loss as a random variable  $X$ , and used its moments-generating function

$$M_X(t) := \mathbb{E}[e^{tX}], \quad t \in \mathbb{R} \quad (30)$$

to determine the distribution of  $X$ . This accounting method is stronger than methods like *the strong composition theorem*, as moments accountant takes the noise distribution into consideration[37]. The difference between these two accounting methods is illustrated in Figure 15.

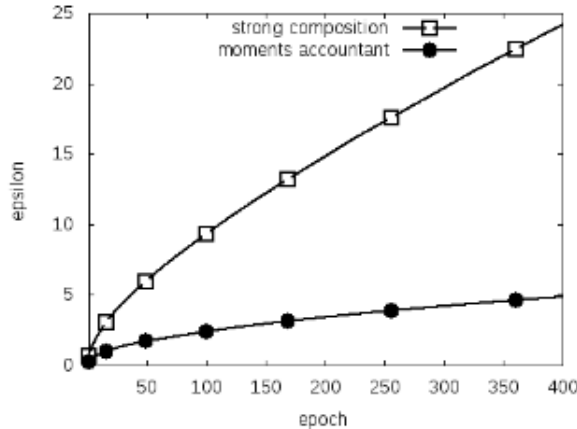


Figure 15: Accumulated  $\epsilon$  graphed as a function of number of epochs using the strong compositional theorem and moments accountant. The moments accountant gives a much tighter estimation of the privacy loss compared to the strong compositional theorem [37].

### 2.6.2 Federated Learning with Differential Privacy

Differential privacy (DP) can be used in a federated learning setting in order to enhance privacy. By comparing federated learning to DP as explained in Section 2.6.1, one can look at the server as the query function  $f$ , and the clients as the databases  $d_k$ . After the server has collected the updates, or queried the updates, one can add noise drawn from a distribution to the aggregated weights, where the noise is proportional to the client’s update. To get a DP-guarantee, there has to be some changes to the FedAvg algorithm described in Algorithm 2:

- The first step in Federated Averaging is to select a random subset of size  $C$  from the available clients. In the DP-setting, the clients are selected independently with a probability  $q$  where the size is an expected value  $\mathbb{E}[C]$ . Because of the addition of noise, there has to be selected a larger number of clients each round. This is because the noise is proportional to one clients update, and the relative effect of the noise on the average value can be decreased by adding more clients [35].
- In the DP-setting, the clients return a clipped version of the updates to the server such that one client’s data does not influence the final average. This is done by bounding the maximum  $l_2$ -norm of any client’s update. The clipping can either be fixed [35], or it can be done adaptively as described in the paper *Differentially Private Learning with Adaptive Clipping* [38]. In this paper, the authors apply adaptive clipping to the median update  $l_2$ -norm, thus eliminating the tuning of any clipping hyperparameter.
- As described in the first paragraph in this section, the server can be viewed as the query function  $f$ . To be more exact, the query function  $f$  is the aggregator, which in the case of federated averaging is a weighed arithmetic mean. The sensitivity  $\Delta f$  of this function must be calculated so that the appropriate amount of noise can be added to the average. The calculation of the sensitivity and the choice of the distribution to draw the noise from depends on the choice of using  $\epsilon$ -DP or  $(\epsilon, \delta)$ -DP [35].

After integrating the changes mentioned in the list above with the Federated Averaging algorithm, one will have obtained the Differentially-Private Federated Averaging algorithm:

---

**Algorithm 4:** Differentially-Private FedAVG [35]

---

**Procedure** SERVER  
initialize  $w_0$ ;  
**for** each round  $t = 0, 1, 2, \dots$  **do**  
    Select each client independently with probability  $q$ ;  
    **for** each client  $k$  **in parallel** **do**  
         $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ ;  
    **end**  
     $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k + \text{Noise}$ ;  
**end**  
return  $w_T$ ;

**Procedure** ClientUpdate( $k, w$ )  
 $\mathcal{B} \leftarrow \text{split } \mathcal{P}_k \text{ to set of batches of size } B$ ;  
**for** each local epoch  $i$  from 1 to  $E$  **do**  
    **for** batch  $b \in \mathcal{B}$  **do**  
         $w \leftarrow w - \eta \nabla l(b; w)$ ;  
    **end**  
**end**  
return Clip( $w$ );

---

The reason for using FedAvg with DP instead of FedSGD, is that FedAvg is more communication-efficient since the clients run several epochs of SGD locally before sending the updates to the server. From a pri-

privacy standpoint, the server has to query the decentralized dataset fewer times compared to FedSGD. Thus, using an algorithm which is more communication-efficient will also be beneficial from a privacy point of view. The disadvantage of using DP is that it costs more in terms of computation, since the server has to select more clients compared to ordinary FedAvg. DP also introduces more hyperparameters to tune, such as  $\epsilon$ ,  $\delta$  and  $q$ . Using DP will also force a trade-off between privacy and model performance due to the fact that too much noise and clipping could decrease the accuracy of the final model.

## 2.7 Homomorphic Encryption

Homomorphic encryption (HE) is an encryption technique which allows calculations on encrypted data without having access to the private key. The result of the calculations are also encrypted values, and only the holder of the private key can see the results. This technique is useful when there is a third party, for example a cloud service, doing computations on sensitive data provided by a user. If the cloud service were to decrypt the data before performing calculations and return the result in plaintext, that would be considered a loss of utility, especially when the user wants to hold their data private<sup>6</sup>. Homomorphic encryption can also be utilized in a federated learning environment. If the aggregation server does not want the clients to know the parameters of the shared global model, the server can use HE to encrypt the parameters, let the clients do optimization on encrypted values, and get encrypted updates back to the server. This cycle is illustrated in Figure 16.

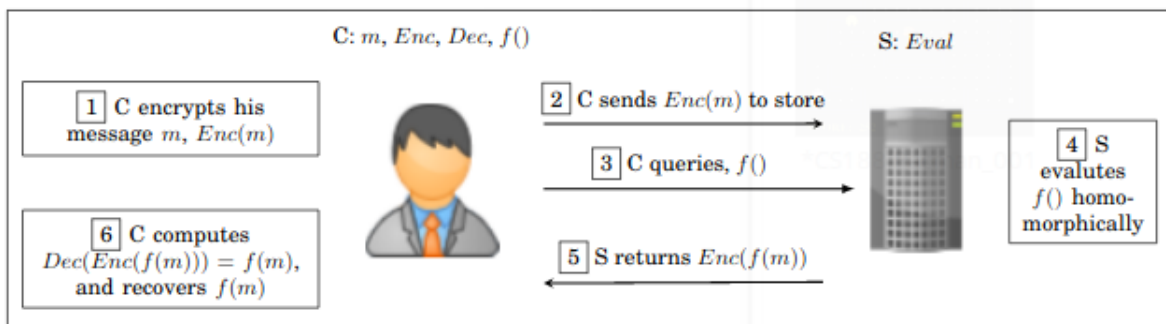


Figure 16: A client  $C$  uses HE to send encrypted data  $m$  to a server  $S$ , which returns  $f(m)$ , which  $C$  can decrypt. [39]

This section will provide a mathematical definition of HE, look at some HE schemes and some applications of HE within the field of machine learning.

### 2.7.1 Binary operators, Groups, and Rings

To describe homomorphic encryption in a precise manner, some definitions in abstract algebra must be reviewed. Therefore, the following sections are going to give an introduction to binary operators, groups and rings. With these definitions in mind, one can define homomorphisms and homomorphic encryption.

#### 2.7.1.1 Binary operators

As described in the book *A First Course in Abstract Algebra, 7th Edition* by John B. Fraleigh [40], a *binary operator*  $*$  is an operator used on a set  $G$  such that  $\forall a, b \in G$ , then  $a * b \in G$  too. This means that  $*$  can be viewed as a mapping

$$* : G \times G \rightarrow G \quad \text{such that} \quad \forall (a, b) \in G \times G \implies *((a, b)) = a * b \in G. \quad (31)$$

<sup>6</sup>Microsoft - <https://www.microsoft.com/en-us/research/project/homomorphic-encryption/>

In other words, if one were to take two elements in the set  $G$ , and apply a binary operator on those elements, then the result will also be an element in the set  $G$ . An example is the set of integers  $\mathbb{Z}$  and the operator  $+$ . Since the result after adding two integers is an integer, then  $+$  is a binary operator with respect to the set  $\mathbb{Z}$  [40].

### 2.7.1.2 Groups

A *group*  $(G, *)$  is a set  $G \neq \emptyset$  with a binary operator  $*$  where the following properties are satisfied:

1. **Associativity.**  $\forall a, b, c \in G$

$$(a * b) * c = a * (b * c) \quad (32)$$

2. **Identity Element.**  $\exists e \in G$  such that  $\forall a \in G$

$$a * e = a = e * a. \quad (33)$$

$e$  is called the identity element in the group  $(G, *)$ .

3. **Inverse.**  $\forall a \in G, \exists a^{-1} \in G$  such that

$$a * a^{-1} = e = a^{-1} * a. \quad (34)$$

$a^{-1}$  is the inverse of  $a$  in  $G$ .

An example of a group is  $(\mathbb{Z}, +)$ .  $(G, *)$  is *abelian* if  $a * b = b * a$  for all  $a, b \in G$  [40].

### 2.7.1.3 Rings

A *ring*  $(R, +, \cdot)$  is a set  $R \neq \emptyset$  with two binary operations  $+$  and  $\cdot$  (addition and multiplication) such that

1.  $(R, +)$  is an abelian group.
2. Multiplication is associative:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c), \quad (35)$$

where  $a, b, c \in R$ .

3. The distributive laws are defined:

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad (36)$$

$$(a + b) \cdot c = a \cdot c + b \cdot c \quad (37)$$

where  $a, b, c \in R$ .

An example of a ring is  $(\mathbb{Z}, +, \cdot)$ . [40]

### 2.7.1.4 Homomorphisms and Homomorphic Encryption

A *homomorphism* is a mapping

$$\phi : A \rightarrow B \quad (38)$$

such that

$$\forall x, y \in A : \phi(x * y) = \phi(x) *' \phi(y), \quad (39)$$

where  $*$  and  $*'$  are binary operations in the algebraic structures (for example groups or rings)  $A$  and  $B$ , respectively [40]. This is where the term *homomorphic* in homomorphic encryption comes from. By looking at the informal definition of homomorphic encryption in Section 2.7, one can observe that one wishes to perform calculations on encrypted values, and get a result which is *consistent* when decrypted. Section 2.7.2 uses Equation 39 and the theory from Section 2.7.1.3 to give a formal definition of homomorphic encryption.

### 2.7.2 A formal definition of Homomorphic Encryption

Let  $(\mathcal{P}, +, \cdot)$  be the ring of plaintexts, and  $(\mathcal{C}, \oplus, \otimes)$  the ring of ciphertexts. Then the mapping

$$E : (\mathcal{P}, +, \cdot) \rightarrow (\mathcal{C}, \oplus, \otimes) \quad (40)$$

is a function that maps plaintexts to ciphertexts, i.e., an encryption function. The decryption function  $D$  is the inverse of  $E$ :

$$E^{-1} = D : (\mathcal{C}, \oplus, \otimes) \rightarrow (\mathcal{P}, +, \cdot). \quad (41)$$

Let  $a, b \in \mathcal{P}$  be two plaintexts, and let  $*$  and  $*'$  be binary operators with respect to  $\mathcal{P}$  and  $\mathcal{C}$ . This encryption scheme can be defined as homomorphic if

$$D(E(a) *' E(b)) = D(E(a * b)) = a * b. \quad (42)$$

Equation 42 uses the homomorphism property defined by Equation 39. Since

$$D(E(a) *' E(b)) = a * b, \quad (43)$$

the binary operation  $*'$  between the ciphertexts corresponds to the binary operation  $*$  between the plaintexts, thus illustrating consistency.

The main challenge with homomorphic encryption is to construct the functions  $e$  and  $d$  such that Equation 42 is true. Through the history (see Figure 17), there have been developed different schemes for solving this problem [39], namely

- **Partially Homomorphic Encryption (PHE)**

PHE only allows either addition or multiplication between the encrypted values. Therefore, this encryption scheme is limited when it comes to evaluating functions on encrypted values. PHE schemes are used in applications like e-voting and Private Information Retrieval. Section 2.7.2.1 describes a cryptosystem which is partially homomorphic.

- **Somewhat Homomorphic Encryption (SWHE)**

SWHE allows both addition and multiplication between the encrypted values, so  $*' \in \{+, \cdot\}$ . In other words, this encryption scheme is more flexible than PHE. A problem with both PHE and SWHE is that the size of ciphertexts increases with each operation, meaning that there is only a limited number of homomorphic operations that can be performed before it is computationally infeasible.

- **Fully Homomorphic Encryption (FHE)**

FHE allows an unlimited number of operations on the encrypted values for an unlimited number of times. This means that FHE solves both the problem with limited evaluation on encrypted values, and the problem with performing homomorphic operations for a limited number of times.

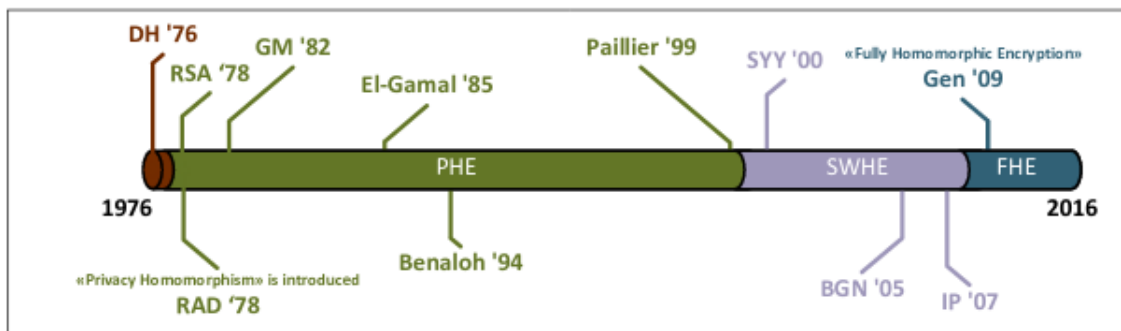


Figure 17: A timeline of the different HE schemes, from PHE to FHE. [39]

### 2.7.2.1 Paillier Cryptosystem

Paillier cryptosystem is a probabilistic encryption scheme based on the *composite residuosity problem*. This problem states that it is difficult to find a  $y$  such that

$$z \equiv y^n \pmod{n^2}, \quad (44)$$

where  $z, n \in \mathbb{Z}$  and  $n$  is composite [41].

The keys are generated by choosing two large prime numbers  $p$  and  $q$  such that

$$\gcd(pq, (p-1)(q-1)) = 1. \quad (45)$$

Then let  $n = pq$  and  $\lambda = \text{lcm}(p-1, q-1)$ . A random number  $g$  is chosen from  $\mathbb{Z}_{n^2}$  such that

$$\gcd(n, L(g^\lambda \pmod{n^2})) = 1, \quad (46)$$

where

$$L(u) = \frac{u-1}{n}. \quad (47)$$

Thus, the public key is  $(n, g)$  and the private key is  $(p, q)$ .

The encryption function  $E : \mathcal{P} \rightarrow \mathcal{C}$  is given by

$$c = E(m) = g^m r^n \pmod{n^2}, \quad (48)$$

where  $m$  is the plaintext, and  $r$  is a random integer [39].

The decryption function  $D : \mathcal{C} \rightarrow \mathcal{P}$  is given by

$$m = D(c) = \frac{L(c^\lambda \pmod{n^2})}{L(g^\lambda \pmod{n^2})} \pmod{n}. \quad (49)$$

Let  $a, b \in \mathcal{P}$  be two plaintexts, and let  $*$  be a binary operation between the corresponding ciphertexts  $E(a)$  and  $E(b)$ . Then

$$E(a) * E(b) = (g^a r_a^n \pmod{n^2}) * (g^b r_b^n \pmod{n^2}) = g^{a+b} (r_a * r_b)^n \pmod{n^2} = E(a+b). \quad (50)$$

As illustrated above, the Paillier Cryptosystem is homomorphic with respect to addition, since it has the same form as Equation 39.

### 2.7.3 Homomorphic Encryption in Federated Learning

As mentioned in Section 2.7, homomorphic encryption can also be applied to federated learning. The following algorithm illustrates FedAvg (see Section 2.2.1.2) with fully homomorphic encryption (FHE), because it is desirable to calculate non-linear activation functions, such as the sigmoid function or tanh,

which is not defined in PHE or SWHE.

---

**Algorithm 5:** FedAvg with HE

---

**Procedure** SERVER  
initialize cryptosystem;  
initialize  $w_0$ ;  
 $w_0 \leftarrow e_K(w_0)$ ;  
**for** each round  $t = 0, 1, 2, \dots, T$  **do**  
     $m \leftarrow \max(C \cdot K, 1)$   
     $S_t \leftarrow$  (random set of  $m$  clients);  
    **for** each client  $k \in S_t$  **in parallel do**  
         $w_{t+1}^k \leftarrow$  ClientUpdate( $k, w_t$ );  
    **end**  
     $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ ;  
**end**  
 $w_T \leftarrow d_K(w_T)$ ;  
return  $w_T$ ;

**Procedure** ClientUpdate( $k, w$ );  
 $\mathcal{B} \leftarrow$  split  $\mathcal{P}_k$  to set of batches of size  $B$ ;  
**for** each local epoch  $i$  from 1 to  $E$  **do**  
    **for** batch  $b \in \mathcal{B}$  **do**  
         $w \leftarrow w - \eta \nabla l(b; w)$ ;  
    **end**  
**end**  
return  $w$ ;

---

As Algorithm 5 illustrates, the server encrypts the weights  $w_0$  before broadcasting the weights to the  $m$  clients, and the server decrypts the weights after the  $T$  rounds are completed. This means that  $w_t$  broadcasted to the clients are encrypted, and the clients do stochastic gradient descent on encrypted weights. Only the server is able to decrypt the weights, and the clients are not able to see the actual weight updates from the server.

A paper from 2020, *BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning* [42], shows that when utilizing federated learning with homomorphic encryption the majority of time spent is on performing calculations between encrypted values, and not the main task which is to train a model with machine learning. This happens because homomorphic encryption operates with long integers, in other words,  $E$  maps the plaintexts to large integers in  $\mathcal{C}$ .

Key size	Plaintext	Ciphertext	Encryption	Decryption
1024	6.87MB	287.64MB	216.87s	68.63s
2048	6.87MB	527.17MB	1152.98s	357.17s
3072	6.87MB	754.62MB	3111.14s	993.80s

Figure 18: Paillier Cryptosystem benchmarked with different key sizes. The Figure describes how the size of the ciphertext increases with the key size. It also shows that the ciphertext is much larger than the plaintext. The increase in ciphertext size leads to an increase in encryption and decryption time [42].

Figure 18 shows how the ciphertext grows with the different key sizes. With a key size of 2048, the ciphertext is almost 100 times larger than the plaintext. This means that homomorphic encryption has both computational and communication overhead, which makes it impractical to train state-of-the-art models in production. On the other side, homomorphic encryption offers strong privacy and no accuracy loss. The strong privacy comes as a result of the encryption and the fact that the clients will never be able to see

the model updates. Furthermore, since the homomorphic encryption scheme is loss-less [42], no information is lost when encrypting and decrypting the weights.



## 3 Method

Based on theory from Chapter 2, we conducted several experiments in order to answer the research questions described in Chapter 1. This chapter is going to give an overview of the methodology behind the experiments, such that the results in Chapter 4 can be reproduced.

### 3.1 Process

This section will describe the process of the project prior to the implementation of federated learning and the experiments performed accordingly. It will cover how the topic was researched, why the dataset used in the experiments was chosen and an analysis of the dataset.

#### 3.1.1 Research

In recent years, the machine learning approach known as federated learning emerged. Due to the current nature of the approach, there is still limited amounts of research concerning the topic. However, papers and frameworks in regards to federated learning are constantly being published. In addition, many large companies such as Google have hosted recorded seminars and talks which allow other interested parties to gain insight into the mechanisms behind federated learning.

As federated learning is relatively new, this project required extensive research in relation to the topic prior to performing the experiments discussed in Section 4. This research consisted of utilizing the information that was currently available such as papers and seminars on federated learning, and federated learning in regards to privacy, communication efficiency and model performance. Furthermore, the information gathered from the research phase of the project was discussed thoroughly, often with the academic adviser, in order to truly understand it. The research phase of the project process, was most intense at the start of the project. However, some research was done as the project advanced due to uncovering new, relevant topics.

#### 3.1.2 Data Collection

Following the research, the search for a dataset that could be used in the implementation of federated learning started. The ideal dataset would contain medical data, preferably electrocardiogram (ECG) data. This was due to the fact that we wished to test what benefits federated learning resulted in applied to medical data, compared to centralized learning. Furthermore, Infiniwell works with medical data, specifically ECG data, which encouraged exploration concerning the effect of federated learning in regards to such data. It was eventually decided that the *MIT-BIH Arrhythmia Database* <sup>7</sup> should be utilized in the experiments. This dataset contains two-channel ambulatory ECG recordings with a sampling frequency of 125Hz, gathered from 47 different people over a time period of four years. The dataset consists of five classes, which are illustrated in Figure 19.

The motivation behind choosing this dataset was that the dataset contained a relatively large amount of training and test data, resulting in a total of 109446 examples. In respect to the limitations in memory and processing capacities of the computers used in the project, this was a sufficient amount of data. The dataset also consisted of data from a decent amount of subjects, as well as being gathered over an acceptable time period which made it even more desirable. Furthermore, this dataset was chosen because Infiniwell, our external client, previously had trained deep learning models using centralized learning with this exact dataset. This made the dataset suitable for our project as it was beneficial to find a dataset that could simulate how federated learning would work in environments such as those Infiniwell works with.

#### 3.1.3 Data Analysis

To obtain sufficient information about the MIT-BIH Arrhythmia Database, data analysis was performed. The data analysis gave insight into how large the dataset was, the distribution of data between the classes

---

<sup>7</sup>MIT-BIH Arrhythmia Database - <http://ecg.mit.edu/george/publications/mitdb-embs-2001.pdf>

and an overview of what the average data example looked like for each class. Moreover, the data analysis allowed observation of the raw data. This was an important part of the process as it gave insight into how the data was structured and what needed to be altered during data preprocessing. Table 1 illustrates an excerpt of the raw data in the *MIT-BIH Arrhythmia Database*. Each row in the table describes an ECG-wave, and the last column describes the class of the ECG recording for each row.

0	1	2	3	4	5	184	185	186	187
0.000000	0.135593	0.409201	0.736077	0.832930	0.765133	0.0	0.0	0.0	0.0
1.000000	0.816742	0.042986	0.027149	0.131222	0.135747	0.0	0.0	0.0	0.0
1.000000	0.924670	0.706215	0.416196	0.180791	0.103578	0.0	0.0	0.0	1.0
0.650190	0.581749	0.536122	0.501901	0.490494	0.486692	0.0	0.0	0.0	4.0
0.120907	0.211587	0.375315	0.471033	0.556675	0.672544	0.0	0.0	0.0	0.0

Table 1: Excerpt of the raw data. The first 187 columns contain the voltages in the interval  $[0, 1]$  for the ECG recordings, and the last column contains the class of the ECG recording. The classes are encoded as 0.0, 1.0, 2.0, 3.0, and 4.0. The classes are shown in Figure 27, and the encodings correspond to the order the ECG recordings are shown (from left to right).

While performing data analysis on the *MIT-BIH Arrhythmia* dataset, it was discovered that the dataset was heavily imbalanced. 83% of the dataset consisted of the class *Normal beats*, while the remaining 17% of the dataset was distributed between the remaining four classes *Unknown beats*, *Fusion beats*, *Ventricular beats* and *Supra-ventricular beats*. The distribution of the dataset is illustrated in Figure 19.

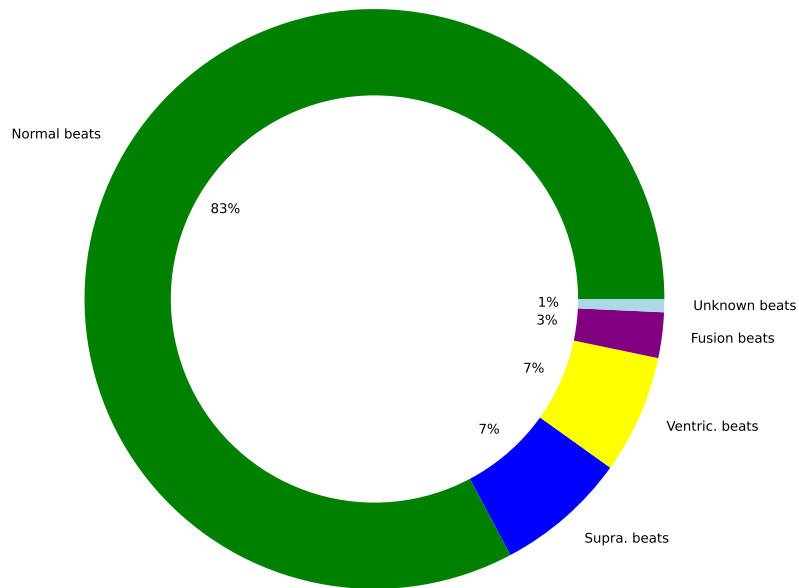


Figure 19: Distribution of the data in the MIT-BIH Arrhythmia Database. The dataset is heavily imbalanced, where the *Normal beats* is the majority class.

In order to determine the quality of the data, we wanted to observe the relationship of intensities at exact positions between the ECG waves for each class. If the 2D histogram shows significant overlap between the ECG's of one class, it would imply that the examples of one class are consistent. In addition, this would allow us to gain insight regarding the variance between the ECG's in the different classes. The 2D histograms for each class are illustrated in Figure 27.

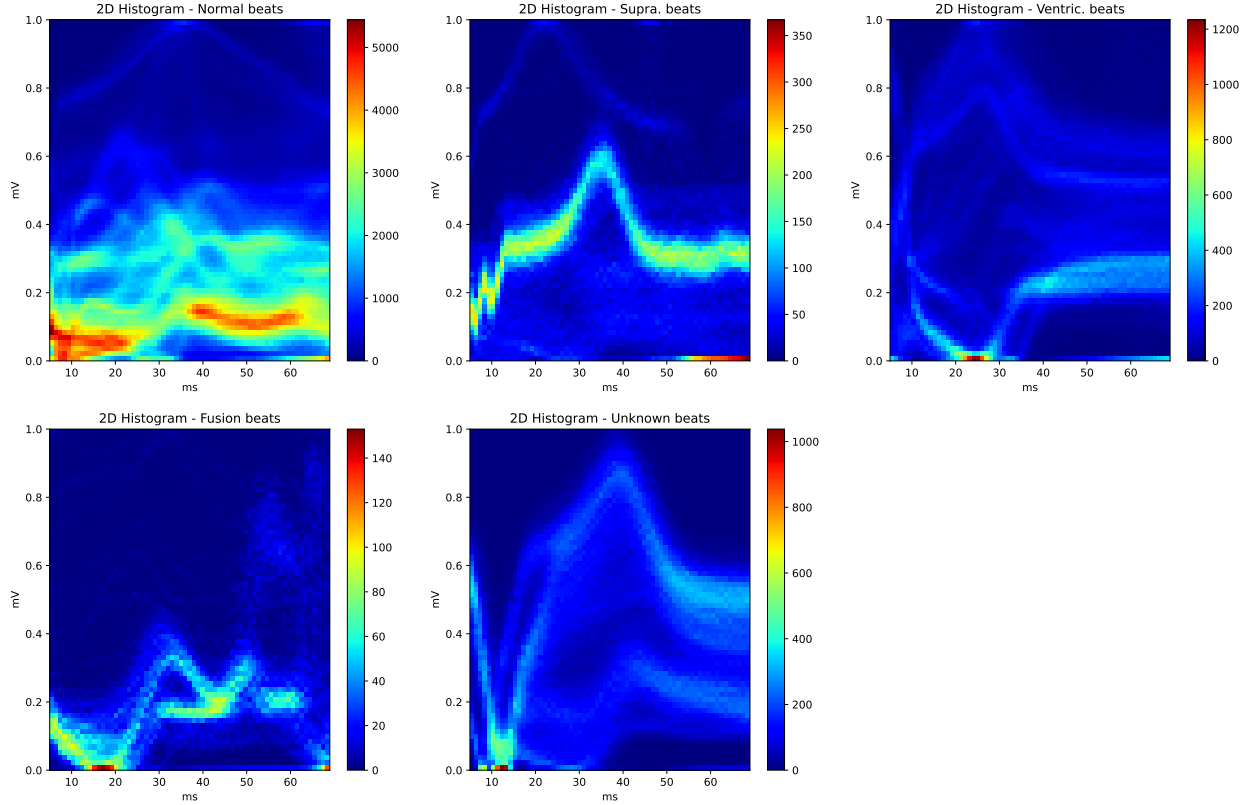


Figure 20: Plots of 2D Histogram for each class. The colorbar shows how many points fall into the same bin. The  $x$ -axis describes the milliseconds (ms) from when the recording started, and the  $y$ -axis describes the recorded millivoltage (mV).

## 3.2 Execution

This section will describe how the experiments with federated and centralized learning were implemented in the project. This section will also explain the methodology behind the experiments.

### 3.2.1 Experiment Process

During the execution phase of the project, we utilized the method described in Figure 21 in order to provide empirical support for the experiments performed. Figure 21, illustrates the hypothetico-deductive model. This model is considered to be a description of a traditional scientific method, and aims to gain empirical support for a hypothesis [43].

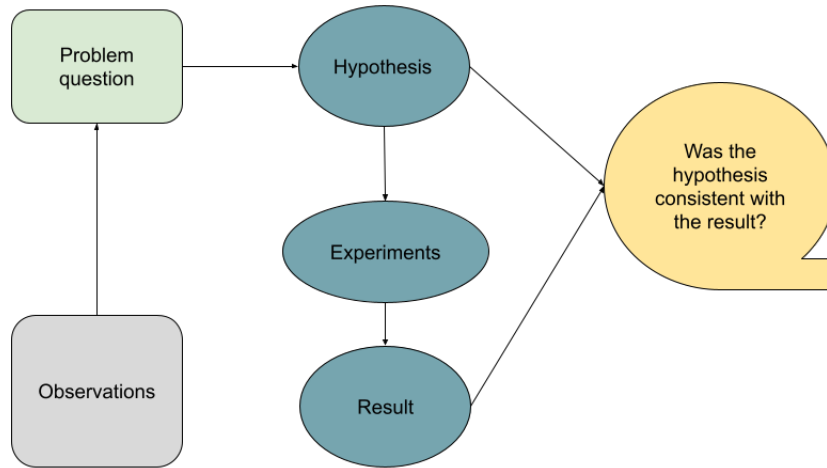


Figure 21: Illustration of the Hypothetico-deductive model. The model describes a scientific method for exploring a research question. First, an observation is made. The observation is used to formulate a research question, which provides the foundation for creating a hypothesis. Based on the hypothesis, experiments are performed and results are obtained. The results are compared to the hypothesis in order to determine if they were consistent.

Some experiments performed during the project, loosely utilized this method. When we discovered something we were curious about, we tried to structure our experiments using the philosophy behind this method. We found that method was most useful in regards to experiments that concerned robust federated aggregation, memorization, differential privacy and homomorphic encryption.

All results found during this process were logged thoroughly. The experiments were logged in a directory in the project where the different parameters concerning the experiments were stored. Due to the logging, we were later able to compare experiments in great detail, and to visualize the results using technologies such as TensorBoard, which is described in Section 3.3.1.2. All random functions such as the dataset shuffling, weight initialization of the models and the random client selection were initiated from a random seed. These choices increased reproducibility of the experiments conducted.

### 3.2.2 Experimentation Pipeline

In order to run the experiments, we needed a robust pipeline to perform data preprocessing, model initialization and other components necessary for the training loop. Finally, the pipeline was used to perform model analysis and model evaluation. To implement the pipeline, the technologies listed in Section 3.3 were used.

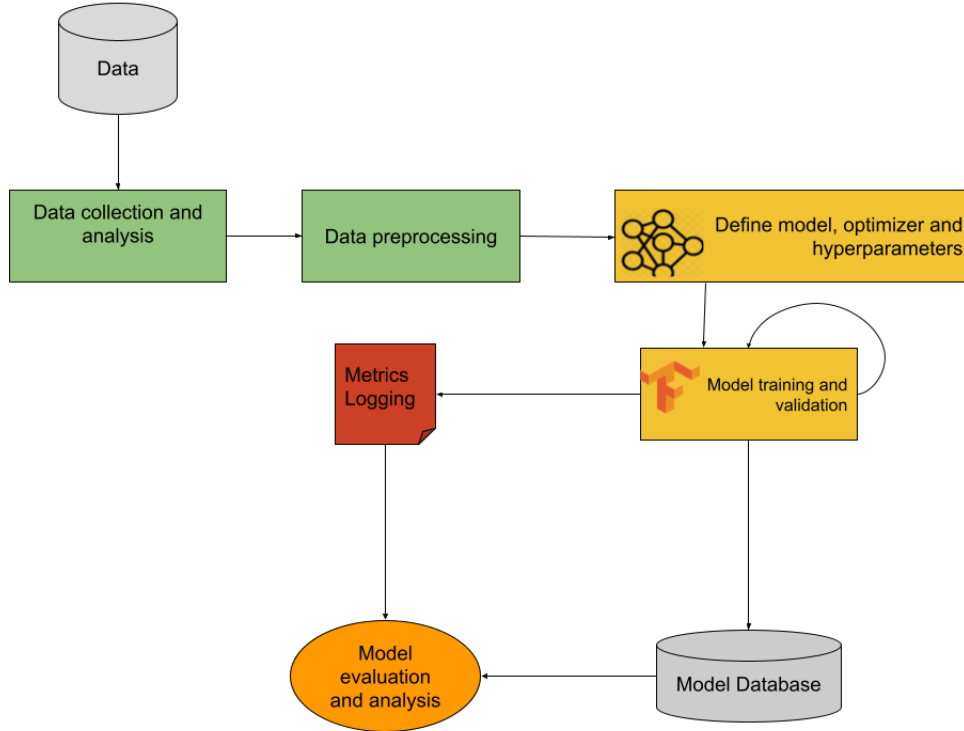


Figure 22: Illustration of the implemented machine learning pipeline. From the figure, one can observe that the first step in the pipeline is to collect and analyze the data retrieved from the database. Once the data has been collected and analyzed, the data will be processed and prepared for model training. Moreover, the model and its parameters are defined before the model starts training. The metrics calculated during training are logged, and the trained model is stored in the model database. Finally, the stored model and its metrics are used in model evaluation and model analysis.

Figure 22 illustrates the experimentation pipeline used in the experiments described in Section 4. The pipeline is representative for both the centralized and the federated pipeline. This is because the federated learning pipeline implemented is not production-ready, and uses TensorFlow Federated. This means that the clients are initialized locally which requires the pipeline to collect the data in order to distribute it to the clients. In a production-ready federated learning pipeline, the data would not have been collected. Therefore, we first collected the data from the database and analyzed it. Once this had been done, the data was cleaned and processed in order to prepare it for the training loop. Furthermore, the model, optimizer and hyperparameters were defined, and sent into the model training loop. The training loop was either centralized or federated. While the model was training, the model was validated using validation data. All metrics obtained during and after training were stored, which made accessing the results of an experiment easy. After a model had been trained, it was stored in the model database. From this database, any trained model could be imported into a Jupyter Notebook for further evaluation and analysis. All modules in the pipeline were fully automated, and we only had to adjust a few parameters in order to define an experiment run.

### 3.2.3 Data Preprocessing

As stated in Section 3.1.2, the *MIT-BIH Arrhythmia Database* dataset was selected to be utilized in the experimentation pipelines. Figure 19 in Section 3.1.2, illustrates that the dataset was heavily imbalanced. Imbalance can cause a biased model, thus provoking a decrease in validation or test accuracy. In order to correct this issue, we had to resample the data to obtain a balanced dataset. Figure 23 illustrates the data distribution after it was resampled.

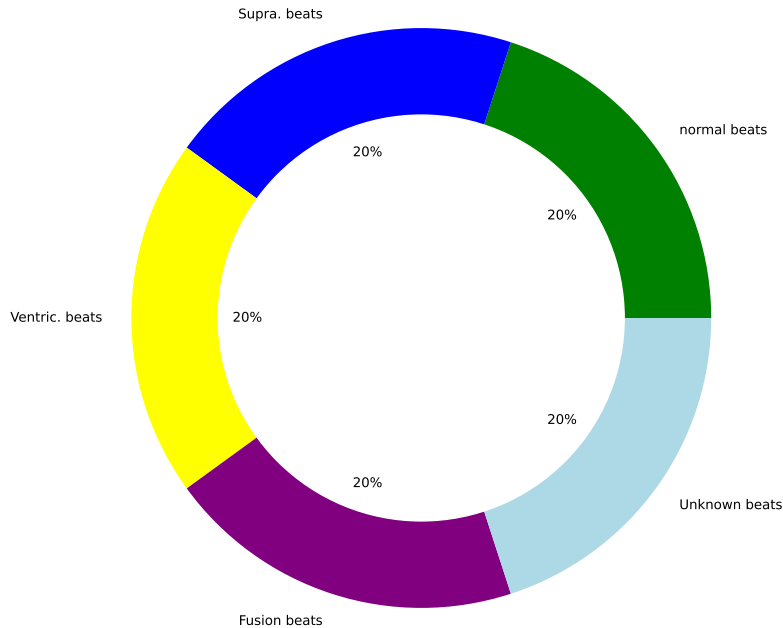


Figure 23: The figure describes the training data distribution after resampling. The training data after resampling consists of 20000 examples per class.

The dataset used in this project was originally organized by class which means that if the data was split into training, test and validation sets, the data in each set would not be representative of the overall data. In order to create a representative distribution, the data was shuffled with a buffer size of 10000 prior to being split in order to reduce the variance, and to ensure generalized models that would be less prone to overfitting. Next, consecutive examples in the dataset were combined into batches where each batch consisted of 32 examples. The reason why we chose to utilize batching was because it would decrease the computation time due to not having to compute the *true* gradient for the entire dataset. Using batching allows for less computation time, while still calculating a sufficient estimate of the gradient. The training data consisted of 100000 resampled examples, where each class had 20000 examples. The test data consisted of 21892 examples. 20% of the training data was used as the validation dataset for testing the model while training.

Once the data had been shuffled and batched, the data that were to be applied in centralized learning was ready to be used in its respective experimentation pipeline. However, in federated learning the data must be located at each client and therefore requires more preprocessing. In order to simulate data at each client, we created methods specific to federated learning where the data was distributed on multiple clients in different ways. Three separate methods were constructed for this purpose:

- 1. Class Distributed:**

Each client received data from one class exclusively. This distribution was created because it gave an overview of how federated learning worked, and how it responded to having high variance amongst clients. However, this distribution is highly unlikely in reality.

- 2. Non-IID Distributed:**

Each client received an approximation of non-IID data. To acquire a non-identical data distribution, we shuffled the data and assigned it randomly amongst the clients. Due to the nature of the dataset, it was difficult to determine dependencies between the data points as we only had access to the raw

ECG-data, and not any other metadata. This distribution was created due to non-IID being a common data distribution in federated learning.

### 3. Uniform Distributed:

Each client received equal amounts of data from each class. This distribution was created to observe how federated learning responded to training on data that was equally distributed. This is also an unlikely distribution, but more plausible than *class distributed*.

These methods were utilized because we wanted to observe how a model trained with federated learning was effected by different data distributions. See Section 4.2.3.2 for the experiments with the different data distributions. The methods also converted the data from Pandas DataFrame to TensorFlow Federated ClientData objects, allowing the data to be directly inputted into the experimentation pipeline for federated learning.

#### 3.2.4 Overview of Machine Learning Models

The experimentation pipeline illustrated in Figure 22, describes the need to define a model in order to proceed to model training. In this project, three different models were built in order to look at how different models perform in federated learning compared to centralized learning. The weights of all three neural networks were initialized with *The Glorot uniform initializer*. The weights are drawn from a uniform distribution within:

$$\left[ -\sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}}, \sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}} \right]. \quad (51)$$

In Equation 51, fan\_in is the number of input units in the weight tensor, and fan\_out is the number of output units. The following subsections will describe the models built and why they were explored.

##### 3.2.4.1 Softmax Regression

The Softmax Regression model displayed in Figure 24, was the smallest model implemented, and can be described with the following equation:

$$\text{softmax}(xW + b). \quad (52)$$

This model was more interpretable than the more complex neural networks described in Sections 3.2.4.2 and 3.2.4.3, making it useful for performing inference attacks on the data used to train the model. This made the model suitable for experiments regarding model extraction and homomorphic encryption, since we could easily display the weights of the model after optimization and obtain an interpretable representation of the training data. As shown in Figure 24, the model only consisted of one fully-connected layer. Therefore, the model could not make a very high-dimensional representation of the data, making it easier to get a representation of the training data.

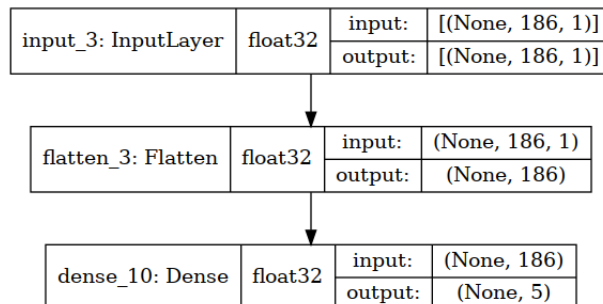


Figure 24: Illustration of the Softmax Regression model. The model has softmax activation in the last layer of the network. The model has 935 parameters.



### 3.2.4.2 Artificial Neural Network

The artificial neural network (ANN) used in the experiments, was implemented as a baseline model with only fully-connected layers. We wanted to build a model which was fairly easy to implement, and that could train quickly in order to be able to test new features without having to train a more complex model, e.g. convolutional neural network. In addition, the ANN model consist of fewer trainable parameters which made it possible to test theories such as reduced memorization in smaller models. Figure 25 illustrates the structure of the ANN.

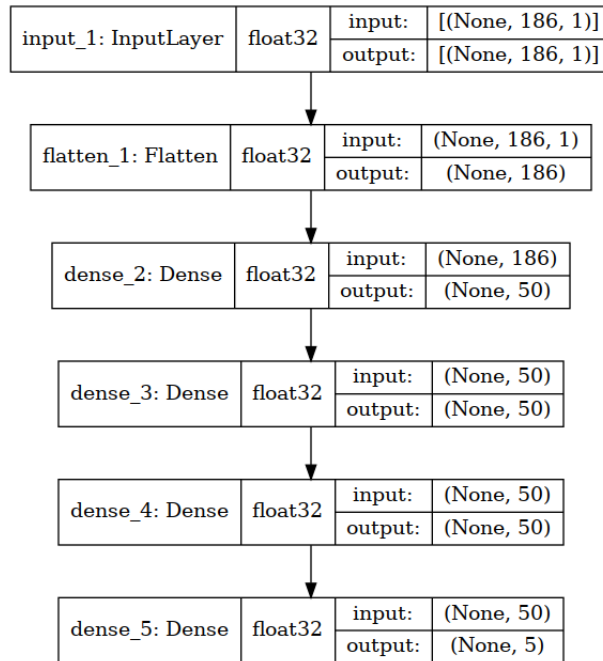


Figure 25: This neural network consists of 3 fully-connected hidden layers, and the activation function applied between each layer was the ReLU activation function. Softmax was applied to the logits of the neural network. The model has 14705 parameters.

### 3.2.4.3 1D Convolutional Neural Network Model

The 1D Convolutional Neural Network (CNN) displayed in Figure 26, was implemented because we wanted to test a more complex model with federated learning, and to compare it with the ANN described in Section 3.2.4.2. Another reason for implementing the CNN model was that Infiniwell have used a similar architecture for their models. The 1D CNN can perform automatic feature extraction from time series data by using convolutions, resulting in effective pattern recognition. This property made the model well-suited for performing classification tasks on the ECG-dataset.

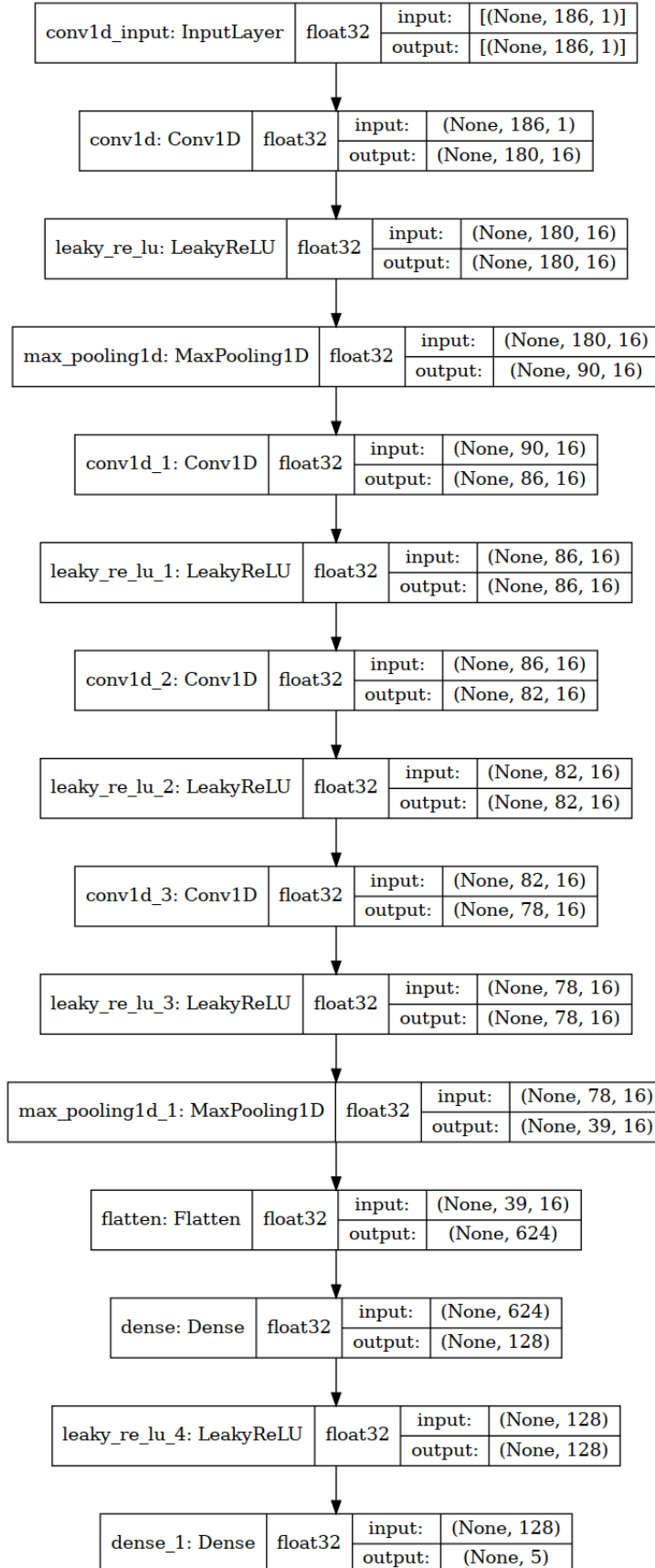


Figure 26: This neural network consists of 4 convolutional layers with 16 filters each. After each convolutional layer, maxpooling was applied where the pooling size was 2. Furthermore, the LeakyReLU activation function was used between the layers. Softmax was applied to the logits of the neural network. The model has 84661 parameters.

### 3.2.5 Implementation of Federated Learning

Federated learning (FL) is a relatively new method for doing distributive machine learning, and implementing FL from scratch would require a comprehensive technology stack. Therefore, we implemented federated learning using TensorFlow Federated (TFF), which is described in Section 3.3.1.1. To initialize a FL experiment, we defined an *Iterative Process* object. This object is defined by the aggregation method, the server optimizer and the client optimizer. Furthermore, we had to define the number of clients to distribute the data between, and how many clients to select per round. The clients selected for each round were randomly sampled from the total client pool. These clients were weighted as  $n_k/n$ , where  $n_k$  was the number of datapoints at client  $k$ , and  $n$  was the size of the training dataset.

To distribute the model to the clients, the machine learning model written in TensorFlow Keras (see Section 3.3.1.3) had to be translated to a Tensorflow Federated Keras model. Once the Iterative Process object had been defined and the machine learning model was ready to be distributed to the clients, the model was ready to be trained using FL. Listing 1 shows pseudocode for how federated learning was performed.

```
for t = 1...T {
  state, metrics = trainer.next(state, train_data)
}
```

Listing 1: Federated learning pseudocode. The variable *trainer* is the Iterative Process object, and the *.next()* method distributes the model (*state*) to the participating clients in order to allow the clients to train the model locally. The *metrics* contains accuracy and loss values after a round, and *train\_data* is a list of the data at the clients.

### 3.2.6 Hyperparameters

The hyperparameters used in the experiments were selected after testing different values. In order to tune the hyperparameters, we used the validation dataset described in Section 3.2.3, and looked at how the model performed using different values. If the model was underfitting or overfitting on the data, we changed the hyperparameters accordingly. For each experiment in Section 4, we will present the training configuration including the hyperparameters using the format shown in Table 2.

Training Configuration	
Learning algorithm:	Federated or Centralized learning
Aggregation method:	Method for aggregating the client updates
Data distribution:	One of the distributions described in Section 3.2.3
Epochs:	Number of epochs to train the global model
Client Epochs:	Number of epochs to train the local model
Total number of clients:	The number of clients where the data has been distributed
Number of participating clients per round:	The number of clients which are participating in FL
Server optimizer:	Optimizer used to train the global model
Server learning rate:	Learning rate used to train the global model
Client optimizer:	Optimizer used to train the local model
Client learning rate:	Learning rate used to train the local model
Loss function:	Loss function used to train both the global and local model

Table 2: Overview of the training configuration used for each experiment in Section 4.

### 3.2.7 Training the Model

To train one of the models described in Section 3.2.4, we implemented two modules, one for centralized learning and the other for federated learning. These modules were used for fitting the model to the training data provided by the data preprocessing module. The main difference between the implementation of centralized and federated learning is the core algorithm for performing gradient descent and updating the weights. The following subsections will provide a description of the methodology used for training machine learning models with centralized and federated learning. These procedures were followed for each of the different experiments performed. The results of the experiments were saved in a folder named with the experiment name, making the experiments reproducible.

#### 3.2.7.1 Selecting model, optimizers, and hyperparameters

Regardless of the module used in any experiment, we had to select the machine learning model, an optimizer and hyperparameters such as the learning rate and the number of epochs. For centralized learning, these are the only parameters that needs to be defined. However, federated learning requires more parameters to be defined before fitting the model as described in Section 3.2.5. The model and its defined parameters, are saved in the folder belonging to the current experiment. The model and the associated parameters are sent to the component for fitting the models with the training data.

#### 3.2.7.2 Training loop

This component is responsible for fitting the models with either centralized or federated learning. Before performing any learning task, the training data and the validation data were retrieved from the data preprocessing module. The reason for using a validation dataset was to avoid overfitting or underfitting while training the model. Moreover, files for storing accuracy and loss metrics in the training loop were created, and TensorBoard was initialized. While the model was being trained with either centralized or federated learning, we were able to view the metrics in TensorBoard. This made it possible to assess the performance of the model based on how the model performed on the validation dataset compared to the training dataset.

After the model had been trained with one of the two learning algorithms, the model object was serialized and saved in the folder belonging to the experiment. This was done so that we could reinitialize the trained models for further analysis and evaluation. The training time was also saved in order to study how different methods, such as FedAvg or RFA, affected the training time of the model.

### 3.2.8 Analyzing the Experiments

The analysis of the experiments was done manually in Jupyter Notebook. We had previously implemented automatic storage of the machine learning models and the different metrics while running experiments. This made it easy to import files belonging to an experiment into Jupyter Notebook. All experiments performed consisted of training a model under different conditions, for example different aggregation methods or different client distributions. These experiments and conditions are explained in greater detail in Chapter 4. Therefore, the main part of the experiment analysis module was to evaluate and analyze the machine learning model using Jupyter Notebook.

#### 3.2.8.1 Assessing model performance

In order to assess the performance of a model, we imported the test dataset from the data preprocessing module and calculated the model performance. After running the model on the test data, we obtained the test accuracy and the test loss of the model. The training and validation curves for the different metrics were also analyzed using TensorBoard in order to determine if the model was affected by overfitting or other problems. We also created a confusion matrix. The test accuracy obtained during model analysis, does not describe the performance of the model on the different classes in the dataset. A confusion matrix, on the other hand, allowed us to extract more descriptive, statistical measures regarding the model for each class:

- **Precision:** The ratio between true positives and all the positives.
- **Recall:** The ratio between true positives and the number of examples.
- $F_1$ -**score:** The harmonic mean of the precision and recall.
- **Support:** The number of samples of the true classifications.

### 3.2.8.2 Privacy Preservation

The experiments concerning outlier-memorization, differential privacy and homomorphic encryption, were analyzed in a more qualitative way. However, to assess outlier-memorization we also utilized the confusion matrix and the statistical measures listed in Section 3.2.8.1 in order to observe how well the model performed on outlier data. To analyze the results after applying differential privacy and homomorphic encryption on the data, the weights of the model were extracted and displayed in Jupyter Notebook. These weights were then studied qualitatively. We observed how much information it was possible to extract concerning the training data. These experiments and results are explained in greater detail in Chapter 4.

## 3.3 Choice of Technologies

This section will provide an overview of technologies used to implement the experiments described in Section 3.2. The programming language used to implement the experiments was Python 3.

### 3.3.1 TensorFlow

TensorFlow<sup>8</sup> is an open source machine learning platform that allows consumers to develop and train machine learning models. The platform offers a variety of software tools and libraries for implementing machine learning pipelines, from data preprocessing to model deployment. TensorFlow was utilized instead of other machine learning platforms, because Infiniwell uses the same platform for training and deploying their models. We wished to choose a platform that would be transferable to Infiniwell. In addition, we chose this platform due to having extensive experience in utilizing TensorFlow to build machine learning solutions. The platform also integrates well with other Python libraries for performing data analysis and linear algebra, such as the Pandas library and the NumPy library. Moreover, TensorFlow offers GPU support and can dynamically allocate variables between the CPU and the GPU. This would prove beneficial as a GPU was used to train models during the experimentation process.

#### 3.3.1.1 TensorFlow Federated

TensorFlow offers a library called TensorFlow Federated (TFF)<sup>9</sup>. This library provides high-level interfaces for implementing federated learning algorithms and tools for working with decentralized data. TensorFlow Federated allowed for easy implementation of the federated learning pipeline using several different aggregation methods, including FedAvg and FedSGD. However, the library does not support more complex aggregation methods such as Robust Federated Aggregation. Instead we implemented RFA using pure TensorFlow. In contrast to TensorFlow, TFF is a strongly-typed functional programming environment, and offers wrappers for performing federated computations. The strongly-typed environment made it easy to control whether the computations were being performed on the client-side or the server-side.

#### 3.3.1.2 TensorBoard

TensorBoard<sup>10</sup> is TensorFlow's visualization toolkit. It provides a dashboard for visualizing and tracking metrics such as loss and accuracy, in real-time. This allowed for tracking of the model performance while training. TensorBoard was also utilized to track the communication metrics between the server and the clients. The visualizations provided by TensorBoard made it easy to share the metrics with the academic adviser and the CEO of Infiniwell.

<sup>8</sup>TensorFlow - <https://www.tensorflow.org/>

<sup>9</sup>TensorFlow Federated - <https://www.tensorflow.org/federated>

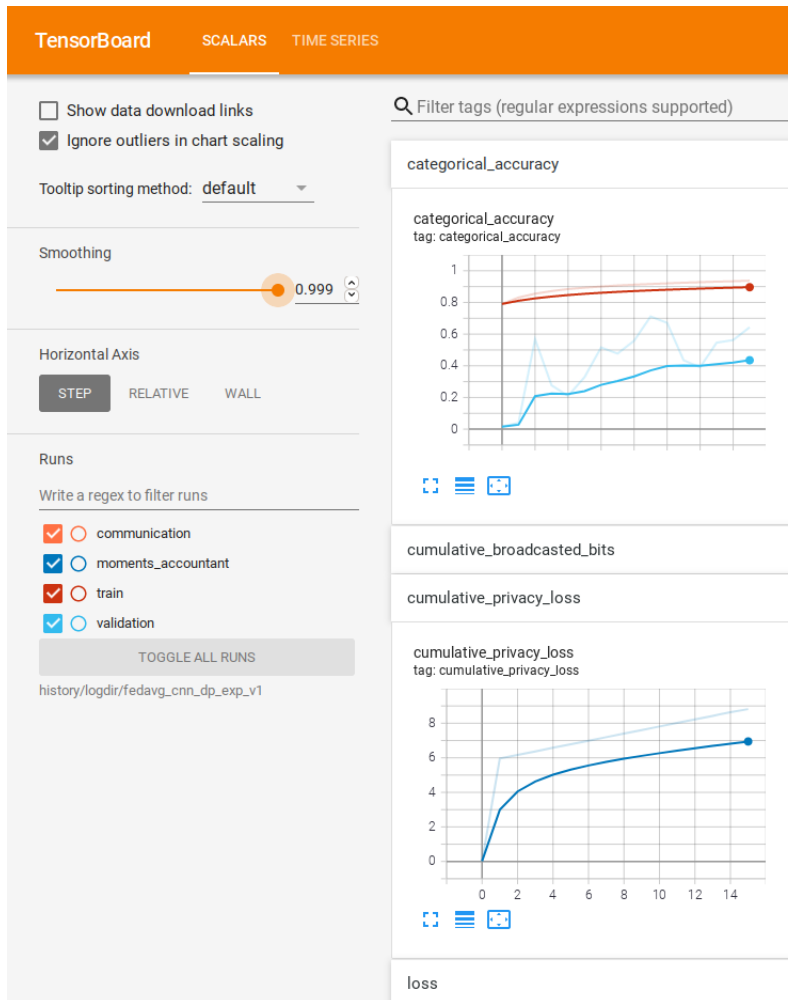


Figure 27: A screenshot of a TensorBoard example. Here one can see the different metrics visualized in a dashboard.

### 3.3.1.3 Keras

Keras<sup>11</sup> is TensorFlow’s high-level API for doing machine learning. Keras offers a multitude of implementations of commonly used neural network building blocks such as layers, activation functions and optimizers. We chose to use this API because it made it easy to build machine learning models and to analyze them.

### 3.3.2 Jupyter Notebook

Jupyter Notebook<sup>12</sup> is an open-source tool for combining executable code with rich text in a single document. This tool made it possible to analyze machine learning models after training, and to visualize data and results from various experiments. Jupyter Notebook permits consumers to export notebooks as HTML- and PDF-files which made it easy to share analysis and results from the experiments.

<sup>10</sup>TensorBoard - <https://www.tensorflow.org/tensorboard>

<sup>11</sup>Keras - [https://www.tensorflow.org/api\\_docs/python/tf/keras](https://www.tensorflow.org/api_docs/python/tf/keras)

<sup>12</sup>Jupyter - <https://jupyter.org/>

### 3.3.3 Python Paillier

Python Paillier<sup>13</sup> is a library which implements the Paillier Partially Homomorphic Encryption scheme. This library was utilized due to neither TensorFlow nor TensorFlow Federated supporting model training with homomorphic encryption. Python Paillier is integrable with the NumPy library which made it easy to encrypt and decrypt the model parameters in federated learning with homomorphic encryption.

### 3.3.4 CUDA-enabled GPU card

Infiniwell provided a GPU card, specifically an NVIDIA GeForce GTX 1070. The GPU card was CUDA-enabled which made it easy to integrate with TensorFlow. The GPU-card was utilized in the project due to it allowing faster model training of different models. CUDA is a platform for doing parallel computations on NVIDIA hardware.

### 3.3.5 NumPy

NumPy<sup>14</sup> is a Python library which was used to perform linear algebra operations. The NumPy library proved helpful in implementing federated learning with homomorphic encryption as we had to build a neural network from scratch in Python.

### 3.3.6 Pandas

The Pandas<sup>15</sup> library was used to perform data manipulation and data preprocessing on the dataset. The library helped convert the raw data to a DataFrame object, a representation which is equivalent to database tables. The Pandas library is integrable with NumPy. This made it easy to execute linear algebra operations on the data.

### 3.3.7 Scikit-learn

Scikit-learn<sup>16</sup> is a lightweight machine learning framework, and was used to perform analysis on machine learning models. With this library, we were able to calculate different statistical measures of model performance. In addition, we used this library to resample the dataset.

### 3.3.8 Matplotlib

Matplotlib<sup>17</sup> is a library for creating visualizations in Python. The visualizations can be static, animated, and interactive. This library helped visualize the data analysis performed, and to illustrate the results of the experiments executed.

### 3.3.9 Plotly

Plotly<sup>18</sup> is an interactive, open-source, and browser-based graphing library for Python. With this library, we were able to interact with the loss and accuracy graphs of an experiment. The accuracy and loss graphs showed in Chapter 4 were produced with Plotly.

---

<sup>13</sup>Python Paillier - <https://github.com/data61/python-paillier>

<sup>14</sup>NumPy - <https://numpy.org/>

<sup>15</sup>Pandas - <https://pandas.pydata.org/>

<sup>16</sup>Scikit-learn - <https://scikit-learn.org/stable/>

<sup>17</sup>Matplotlib - <https://matplotlib.org/>

<sup>18</sup>Plotly - <https://plotly.com/>

## 4 Results

The purpose of this chapter is to present the results obtained while executing different experiments in relation to federated and centralized learning. The experiments presented in this chapter all aim to help answer the research questions described in Chapter 1. Each experiment performed is based on theory from Chapter 2 and the methodology described in Chapter 3. In addition, this chapter will provide a description of each experiment’s training configuration before elaborating on the quantitative and qualitative results of the experiments.

### 4.1 Overview

This section will provide an overview of the experiments presented in this chapter. The experiments are divided into three primary sections:

- **Preliminary Experiments:**

The experiments in this section were all executed in order to establish how federated learning performed compared to centralized learning. The results presented in Section 4.2 will provide information concerning model performance in traditional, centralized learning and in federated learning using two different aggregation methods.

1. **Centralized Learning.** In this experiment we explored how centralized learning performed on the MIT-BIH Arrhythmia Database. Both the ANN model and the CNN model were trained using centralized learning. The results of this experiment can be observed in Section 4.2.1.
2. **Federated Stochastic Gradient Descent (FedSGD).** In this experiment we investigated the performance of federated learning using the FedSGD algorithm applied to the MIT-BIH Arrhythmia Database. Both the ANN model and the CNN model were trained using FedSGD. The results of this experiment can be observed in Section 4.2.2.
3. **Federated Averaging (FedAvg).** This experiment explored the performance of the federated learning algorithm FedAvg applied to the MIT-BIH Arrhythmia Database. Both the ANN model and the CNN model were trained using FedAvg. The results of this experiment can be observed in Section 4.2.3.

- **Experiments regarding Privacy Issues:**

All experiments presented in Section 4.3 explore different privacy issues in machine learning. The main focus of these experiments will be on privacy issues in federated learning, but we will also explore the problem of memorization in regards to centralized learning.

1. **Federated averaging with static data poisoning.** In this experiment, we observed how the CNN model performed with federated averaging when the training data had been manipulated. The results of this experiment can be observed in Section 4.3.1.
2. **Memorization.** In this experiment, we forced memorization in both centralized and federated learning in order to see the effect memorization had on model performance. This experiment utilized both the ANN model and the CNN model. The results of this experiment can be observed in Section 4.3.2.
3. **Model extraction with Federated averaging.** In this experiment, we demonstrated model extraction with the softmax regression model in the sense that we displayed the weights from the participating clients. The results of this experiment can be observed in Section 4.3.3.

- **Privacy-Preserving Experiments:**

The experiments in Section 4.4 will explore different privacy-preserving techniques in relation to federated learning. The techniques used in this section will be directly linked to the privacy issues explored in Section 4.3.



1. **Robust federated aggregation with static data poisoning.** In this experiment, we looked at how the CNN model performed with robust federated aggregation when the training data had been manipulated. The results of this experiment can be observed in Section 4.4.1.
2. **Differential Privacy in Federated Learning.** In this experiment, we applied differential privacy to federated learning, and observed how it affected model performance and memorization. The results of this experiment can be viewed in Section 4.4.2.
3. **Model Extraction in Federated Learning with Differential Privacy.** In this experiment, we demonstrated model extraction with the softmax regression model while training with differentially-private federated averaging. The results of this experiment can be observed in Section 4.4.3.
4. **Federated Learning with Homomorphic Encryption** In this experiment, we performed federated learning with homomorphic encryption. The results of this experiment can be observed in Section 4.4.4.

The experiments listed above were all executed according to the experimentation pipeline illustrated in Section 3.2.2, and the different models utilized are described in Section 3.2.4.

## 4.2 Preliminary Experiments

The results presented in this section will provide information concerning model performance in traditional, centralized learning and in federated learning using two different aggregation methods, FedSGD and FedAvg. These aggregation methods are described in Section 2.2.1.

### 4.2.1 Centralized Learning

The first experiment performed during the project was in regards to centralized learning. We wished to obtain information about how a basic, centralized model would perform on the selected dataset described in Section 3.1.2. The information gathered could be useful in comparing centralized learning with federated learning in terms of model performance. In order to provide a more holistic overview, the experiment was divided into two parts. The first part involved training the artificial neural network using centralized learning, while the second part involved training the convolutional neural network using centralized learning. The experiments utilized the training configuration described in Table 3.

Training Configuration	
Learning algorithm:	Centralized
Data distribution:	Non-IID
Epochs:	15
Learning rate:	0.01
Server Optimizer:	SGD
Loss function:	Categorical Cross-Entropy

Table 3: Training configuration for the centralized learning experiment with the ANN model.

#### 4.2.1.1 Centralized Learning with ANN

During the first part of the experiment, the artificial neural network was trained using centralized learning. We chose to train a less complex model to start with. This was due to the fact that it would be useful to observe how well a model with fewer parameters would perform using centralized learning. It would also provide a good point of comparison when training with federated learning in other experiments. This section will illustrate the results achieved when the ANN model was trained with centralized learning.

Metrics	
Test Accuracy:	92.8%
Training Accuracy:	98.5%
Test Loss:	0.26
Training Loss:	0.04
Training Time:	227 s

Table 4: Accuracy, loss and training time for the centralized learning experiment with the ANN model. This table describes a well-performing model that has a relatively short training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.93	0.96	18118
Supra Ventricular	0.61	0.78	0.69	556
Ventricular	0.78	0.94	0.86	1448
Fusion	0.28	0.90	0.43	162
Unknown	0.79	0.99	0.88	1608

Table 5: Precision, recall, F1-Score and support values for the centralized learning experiment with the ANN model. The F1-score shown in the table describes a model that performed well on every class. This is due to the F1-score being relatively high for all classes which indicates a good true positive rate and a good true negative rate. However, one can observe that the model performed worse on the *Fusion* and the *Supra Ventricular* class compared to the remaining classes.

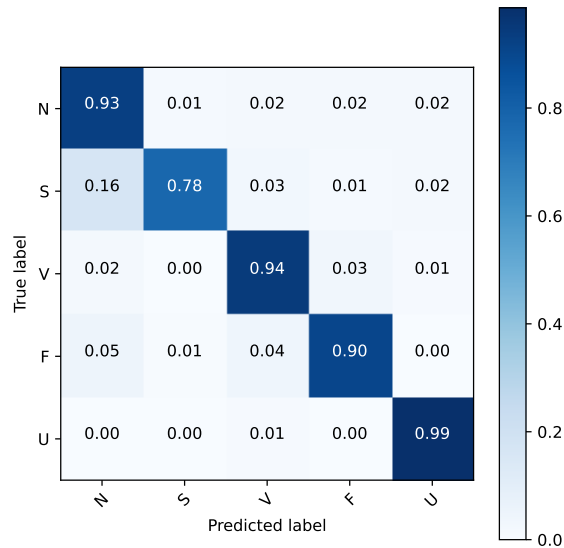


Figure 28: Confusion matrix for the centralized learning experiment with the ANN model. The confusion matrix shows a clear diagonal indicating that the model had a high true positive and true negative rate.

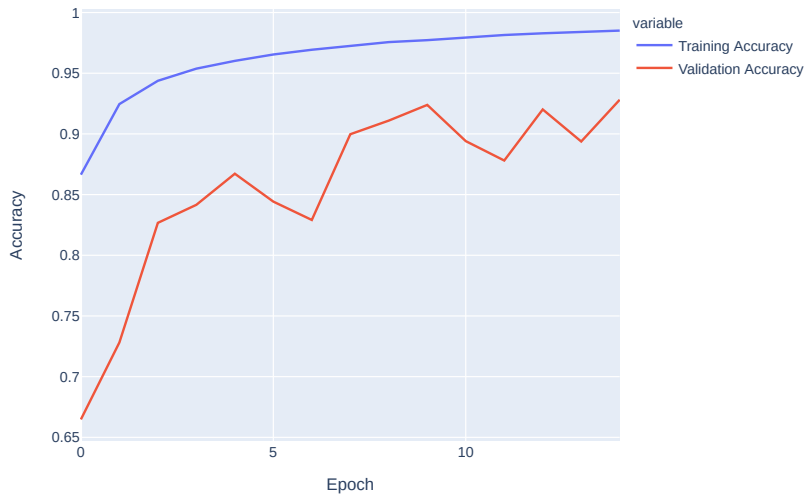


Figure 29: Graph illustrating the training and validation accuracy of the centralized learning experiment with the ANN model. From this graph one can observe that the validation accuracy and the training accuracy converges. This indicates that the model did not overfit on the training data.

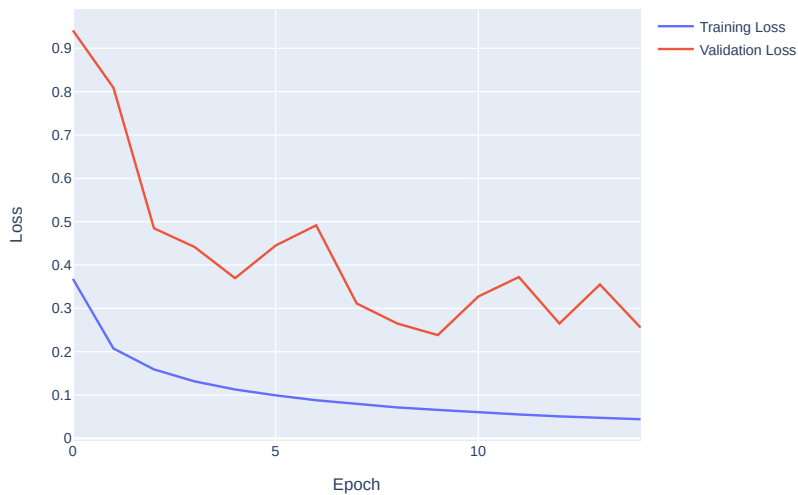


Figure 30: The graph illustrates the training and validation loss of the centralized learning experiment with the ANN model. From this graph one can observe that the validation loss and the training loss converges, indicating that the model did not overfit on the training data.

#### 4.2.1.2 Centralized Learning with CNN

During the second part of the experiment, the convolutional neural network was trained using centralized learning. This model is more complex and comprise of more trainable parameters. We wanted the second part of the experiment to convey how well the CNN would perform using centralized learning. This section will provide the results achieved when the CNN model was trained with centralized learning.

Metrics	
Test Accuracy:	97.1%
Training Accuracy:	99.3%
Test Loss:	0.14
Training Loss:	0.02
Training Time:	471 s

Table 6: Accuracy, loss and training time for the centralized learning experiment with the CNN model. This table describes a well-performing model that had a higher test accuracy and lower test loss than the ANN model described in Table 4. However, the training time for this experiment was more than twice as long as the ANN model.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.98	0.98	18118
Supra Ventricular	0.67	0.76	0.72	556
Ventricular	0.94	0.92	0.93	1448
Fusion	0.72	0.70	0.71	162
Unknown	0.97	0.99	0.98	1608

Table 7: Precision, recall, F1-Score and support values for the centralized learning experiment with the CNN model. The F1-scores shown in the table describes a model that performed well on every class, and that was higher for every class compared to the ANN model described in Table 5. This means that the true positive rates and the true negative rates are high. Similarly to the ANN model, this table also illustrates that the model performed worse on the *Fusion* and the *Supra Ventricular* class compared to the remaining three classes.

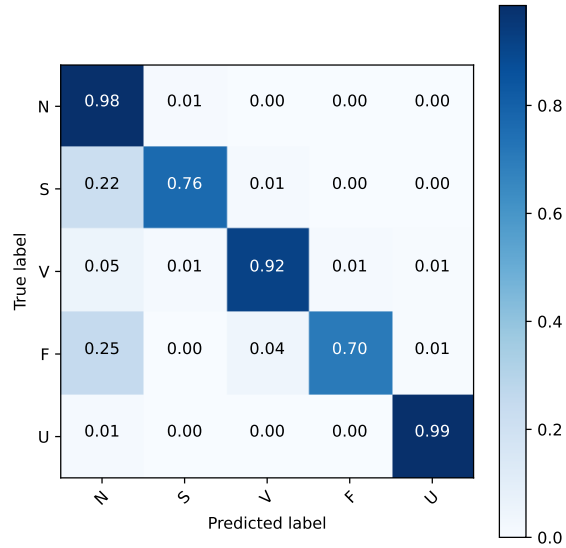


Figure 31: Confusion matrix for the centralized learning experiment with the CNN model. The confusion matrix illustrates a clear diagonal indicating that the model had a high true positive and true negative rate. Compared to the confusion matrix for the ANN model illustrated in Figure 28, there are more false positives and false negatives for the *Supra Ventricular* and the *Fusion* class in the CNN model.

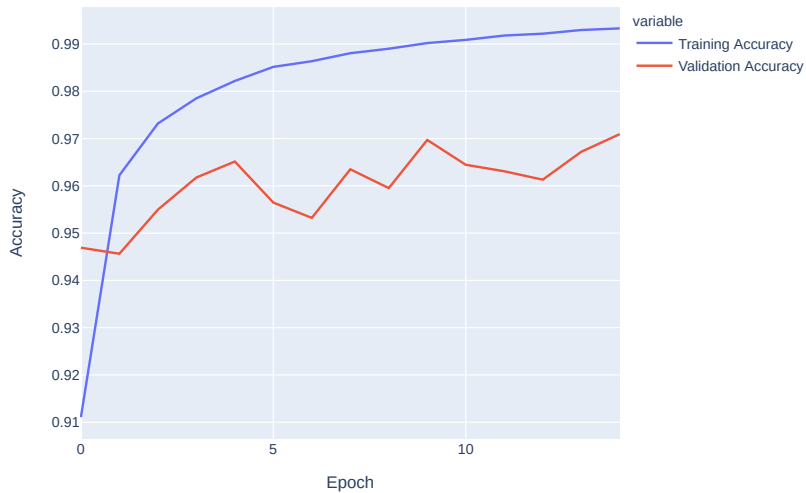


Figure 32: Graph illustrating the training and validation accuracy of the centralized learning experiment with the CNN model. From this graph one can observe that the validation accuracy and the training accuracy converges. This indicates that the model did not overfit on the training data.

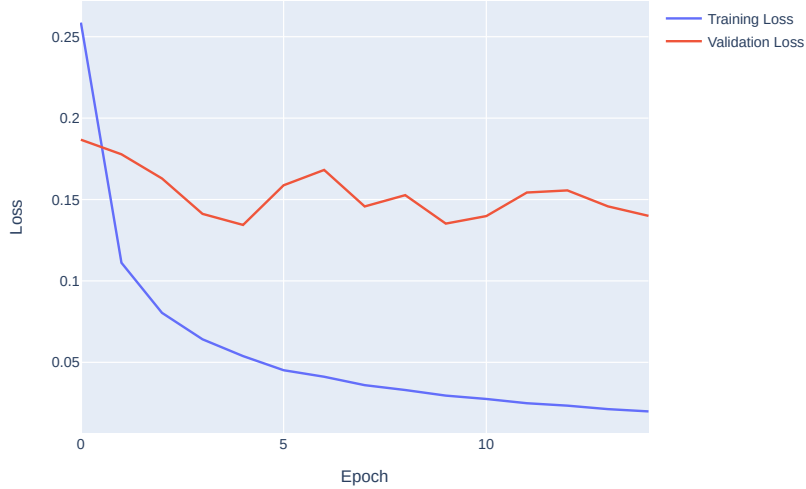


Figure 33: The graph illustrates the training and validation loss of the centralized learning experiment with the CNN model. From this graph one can observe that the validation loss and the training loss converges. This indicates that the model did not overfit on the training data.

#### 4.2.2 Federated Stochastic Gradient Descent

The second experiment performed was in regards to federated learning. To obtain a baseline model with federated learning, we implemented the Federated Stochastic Gradient Descent (FedSGD) aggregation method described in Algorithm 1. FedSGD is the simplest aggregation method described in this project, and the results from this experiment can be used as a baseline for the more complex aggregation methods. To provide sufficient information about the model performance using FedSGD, we decided to train both the ANN model and the CNN model. Table 8 shows the training configuration used in the experiment with FedSGD.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Stochastic Gradient Descent (FedSGD)
Data distribution:	Non-IID
Epochs:	15
Number of clients:	10
Number of participating clients:	10
Server Optimizer:	Adam
Learning rate:	0.01
Loss function:	Categorical Cross-Entropy

Table 8: Training configuration for experiments with FedSGD.

#### 4.2.2.1 FedSGD with ANN

During the first part of the experiment, the artificial neural network was trained using FedSGD. This section will present the results obtained while performing the experiment.

Metrics	
Test Accuracy:	62.1%
Training Accuracy:	75.0%
Test Loss:	0.91
Training Loss:	0.72
Training Time:	428 s

Table 9: Accuracy, loss and training time for the FedSGD experiment with the ANN model. The table describes a model that performed mediocre, and that had a relatively high training time. Compared to the accuracy and loss for centralized learning described in Figure 4, the FedSGD algorithm performed significantly worse.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.96	0.50	0.66	18118
Supra Ventricular	0.07	0.72	0.12	556
Ventricular	0.41	0.72	0.53	1448
Fusion	0.07	0.86	0.13	162
Unknown	0.67	0.88	0.76	1608

Table 10: Precision, Recall, F1-Score and Support values for the FedSGD experiment with the ANN model. The F1-score illustrated in the table describes a model that performed decent on every class, except the *Fusion* and the *Supra Ventricular* class. Compared to the F1-scores for centralized learning described in Figure 5, FedSGD obtained a lower true positive rate and a lower true negative rate.

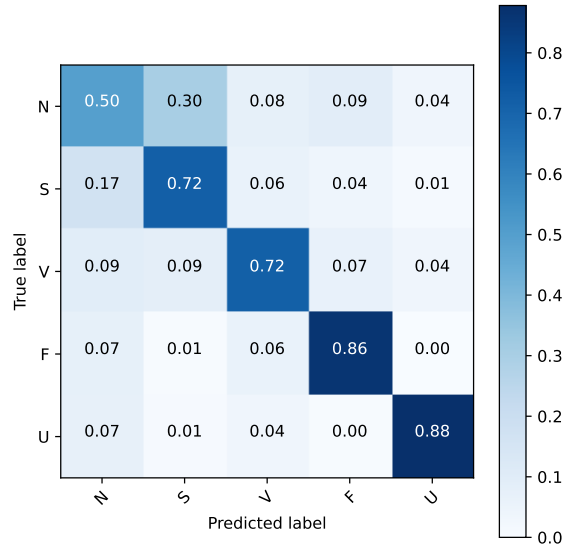


Figure 34: Confusion matrix for the centralized learning experiment with the ANN model. The confusion matrix shows a fairly clear diagonal indicating a decent true positive and true negative rate. However, there are more false positives and false negatives compared to centralized learning which is illustrated in Figure 28.

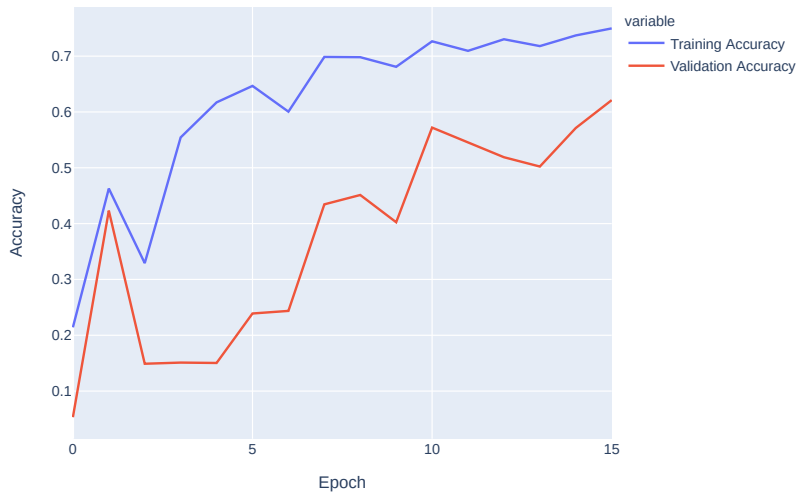


Figure 35: Graph illustrating the training and validation accuracy of the FedSGD experiment with the ANN model. The graph shows that the validation accuracy and the training accuracy are converging which indicates that the model did not overfit on the training data.



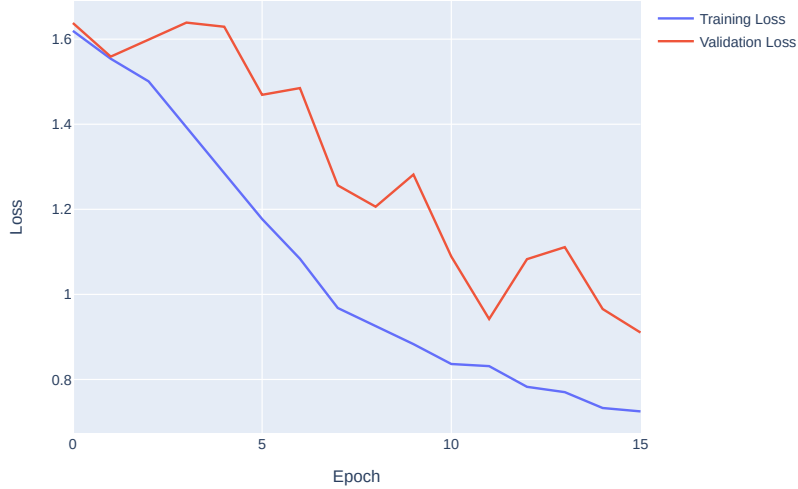


Figure 36: The graph illustrates the training and validation loss of the FedSGD experiment with the ANN model. The graph shows that the validation loss and the training loss are converging which indicates that the model did not overfit on the training data

#### 4.2.2.2 FedSGD with CNN

In this part of the experiment the CNN model was trained using FedSGD. This section will present the results obtained.

Metrics	
Test Accuracy:	50.4%
Training Accuracy:	64.5%
Test Loss:	1.09
Training Loss:	0.93
Training Time:	741 s

Table 11: Accuracy, loss and training time for the FedSGD experiment with the CNN model. The table describes a model that performed mediocre, and that had a high training time. Compared to the accuracy and loss for centralized learning described in Figure 6, the FedSGD algorithm performed significantly worse. In addition, the test accuracy and test loss for the CNN model using FedSGD appears to be lower than for the ANN model with FedSGD, illustrated in Table 9.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.95	0.41	0.57	18118
Supra Ventricular	0.08	0.59	0.14	556
Ventricular	0.18	0.47	0.26	1448
Fusion	0.06	0.86	0.11	162
Unknown	0.38	0.90	0.54	1608

Table 12: Precision, Recall, F1-Score and Support values for the FedSGD experiment with the CNN model. The F1-scores illustrated in the table describes a model that performed decently on the *Normal* and the *Unknown* class. However, the model performed poorly on the remaining three classes. Compared to the F1-scores for centralized learning described in Figure 7, FedSGD obtained a lower true positive rate and a lower true negative rate.

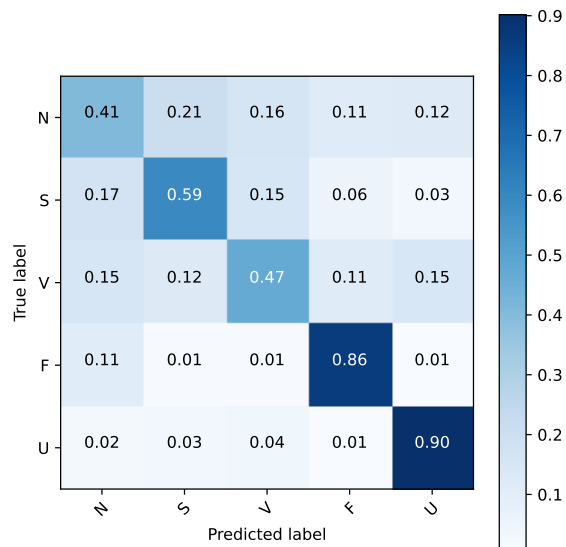


Figure 37: Confusion matrix for the centralized learning experiment with the CNN model. The confusion matrix demonstrates a diagonal indicating that the model provides more true positives and true negatives than false positives and false negatives. However, one can observe that the model classifies wrong more often on the *Normal*, the *Supra Ventricular* and the *Ventricular* class compared to the *Fusion* and the *Unknown* class.

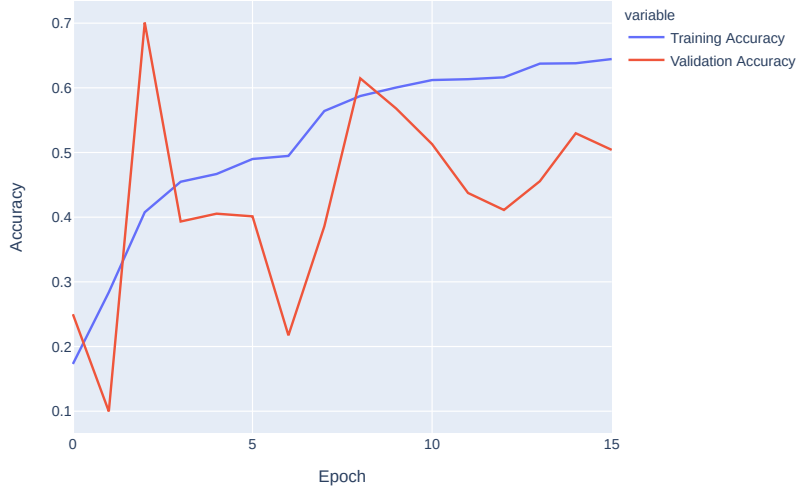


Figure 38: Graph illustrating the training and validation accuracy of the FedSGD experiment with the CNN model. From the graph one can observe that the validation accuracy fluctuates, while the training accuracy improves steadily. However, the accuracies converges indicating that the model did not overfit on the training data.

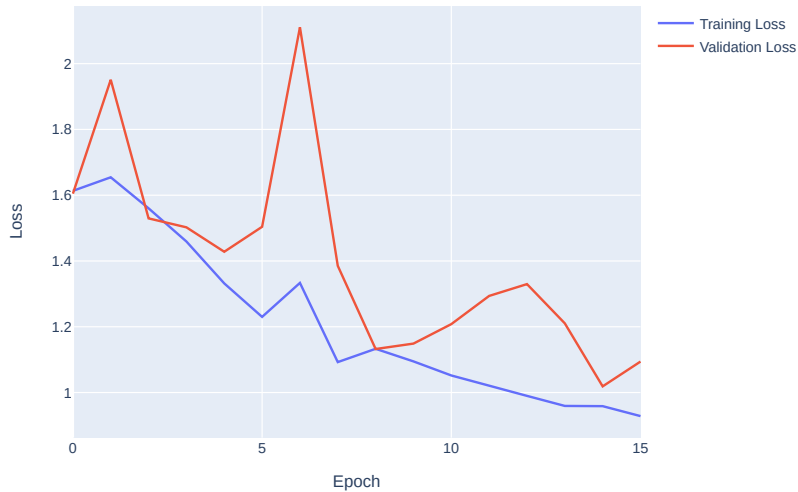


Figure 39: The graph illustrates the training and validation loss of the FedSGD experiment with the CNN model. From the graph one can observe that the training and validation loss converges despite the validation loss fluctuating.

### 4.2.3 Federated Averaging

The Federated Averaging (FedAvg) experiment was executed in order to test how well a more complex federated learning algorithm would perform on the MIT-BIH Arrhythmia Database. We wanted to perform an experiment that utilized suitable parameters, and that could be comparable with centralized learn-

ing in terms of model performance. In this experiment we tested the FedAvg algorithm, described in Algorithm 2, on both the ANN model and the CNN model, using the same hyperparameters for each model. We chose to test both models because we wanted create a point of comparison between federated learning and centralized learning, and had already trained both the ANN and the CNN using centralized learning. Both models were tested using the training configuration presented in Table 13.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Averaging (FedAvg)
Epochs:	15
Client Epochs:	10
Total number of clients:	10
Number of participating clients per round:	10
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.02
Loss function:	Categorical Cross-Entropy

Table 13: Training configuration for the FedAvg experiment.

#### 4.2.3.1 FedAvg with ANN

During the first part of the experiment, we trained the artificial neural network described in Section 3.2.4.2 using the federated learning algorithm FedAvg. We chose to train this model only using the standard non-IID data distribution described in Section 3.2.3. The results of the experiment are illustrated below.

Metrics	
Test Accuracy:	93.4%
Training Accuracy:	98.6%
Test Loss:	0.21
Training Loss:	0.04
Training Time:	333 s

Table 14: Accuracy, loss and training time for the FedAvg experiment with the ANN model. The metrics shown in this table describes a well-performing model with a fairly low training time. Compared to the accuracy for centralized learning described in Figure 4, the FedAvg algorithm performed slightly better, but had a higher training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.95	0.97	18118
Supra Ventricular	0.49	0.82	0.61	556
Ventricular	0.86	0.95	0.90	1448
Fusion	0.43	0.86	0.57	162
Unknown	0.94	0.98	0.96	1608

Table 15: Classification report for the FedAvg experiment with the ANN model. The table describes the precision, recall, F1-Score and support values for the experiment. The F1-scores illustrated in the table describes a model that performed well on every class. However, the F1-score is slightly worse for the *Fusion* class. Compared to the F1-scores for centralized learning described in Figure 5, FedAvg obtained overall higher F1-scores.

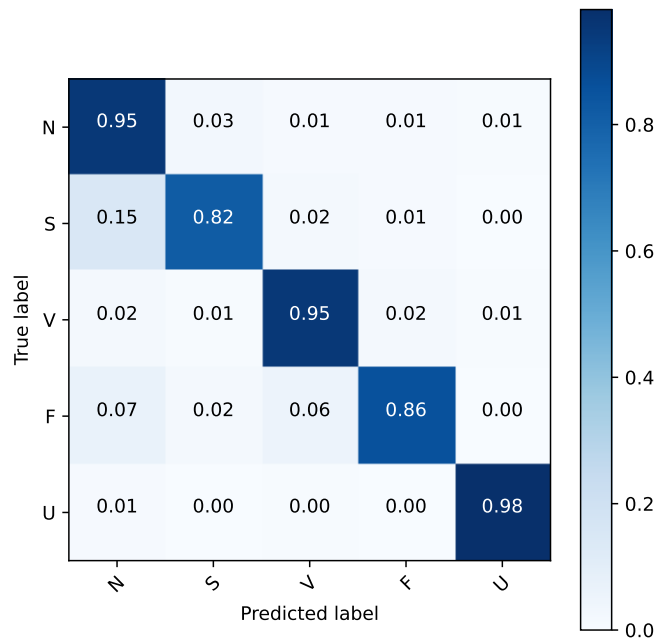


Figure 40: Confusion matrix for the FedAvg experiment with the ANN model. The confusion matrix shows a clear diagonal indicating a high true positive and true negative rate in the model. Compared to the centralized learning experiment with the ANN model illustrated in Figure 28, the FedAvg algorithm with the ANN model performed similarly.

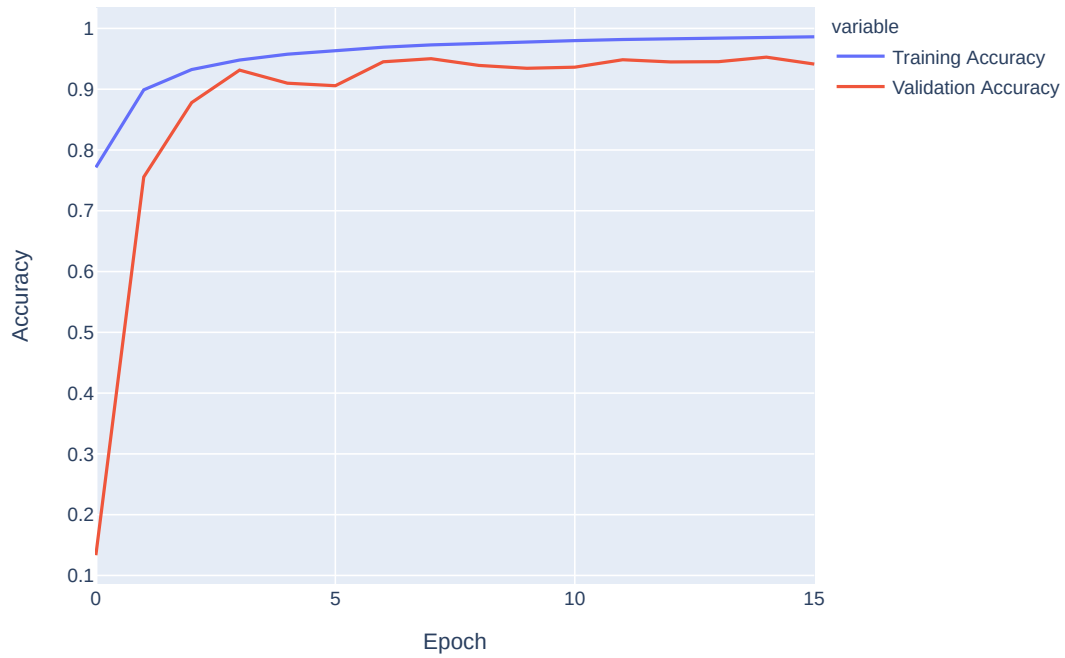


Figure 41: Graph illustrating the training and validation accuracy of the FedAvg experiment with the ANN model. From the graph one can observe that the training accuracy and the validation accuracy converges, indicating that the model did not overfit on the training data.

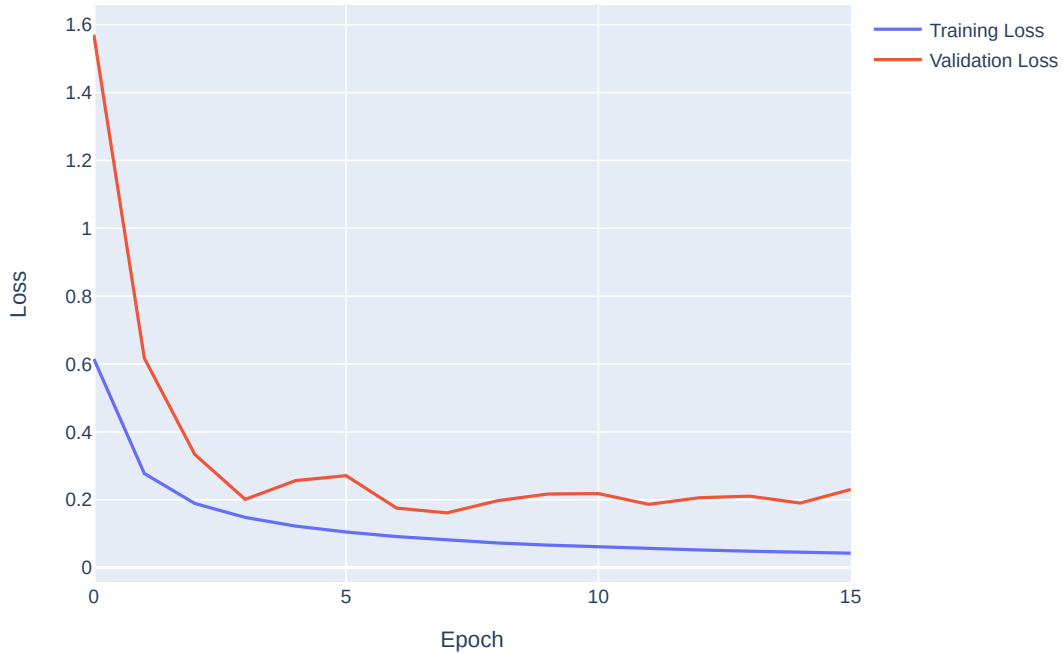


Figure 42: The graph illustrates the training and validation loss of the FedAvg experiment with the ANN model. The graph shows a training and validation loss that converges, and that is relatively low.

#### 4.2.3.2 FedAvg with CNN

During the second part of the experiment, we trained the convolutional neural network described in Section 3.2.4.3 using the federated learning algorithm FedAvg. In addition to training this model using the non-IID data distribution that was used when experimenting with the artificial neural network, we also wanted to test the convolutional neural network in regards to the two other data distributions described in Section 3.2.3.

#### Non-IID Distribution

The following results illustrate the performance of the CNN model applied to the non-IID data distribution of the dataset. The distribution is described in Section 3.2.3.

Metrics	
Test Accuracy:	96.2%
Training Accuracy:	99.3%
Test Loss:	0.20
Training Loss:	0.02
Training Time:	687 s

Table 16: Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on Non-IID data. The metrics presented in this table indicate a well-performing model with a relatively high training time. Compared to the metrics for the centralized learning experiment described in Table 6, the CNN model trained with FedAvg performed slightly worse and had a higher training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.97	0.98	18118
Supra Ventricular	0.61	0.84	0.71	556
Ventricular	0.87	0.94	0.90	1448
Fusion	0.49	0.90	0.64	162
Unknown	0.97	0.99	0.98	1608

Table 17: Classification report for the FedAvg experiment with the CNN model trained on Non-IID data. The table describes the precision, recall, F1-Score and support values for the experiment. The F1-scores illustrated in this table show a model that performed extremely well on every class, except the *Fusion* class where it performed slightly worse. Compared to the F1-score for the centralized learning experiment described in Table 7, the CNN model trained with FedAvg performed similarly on every class.

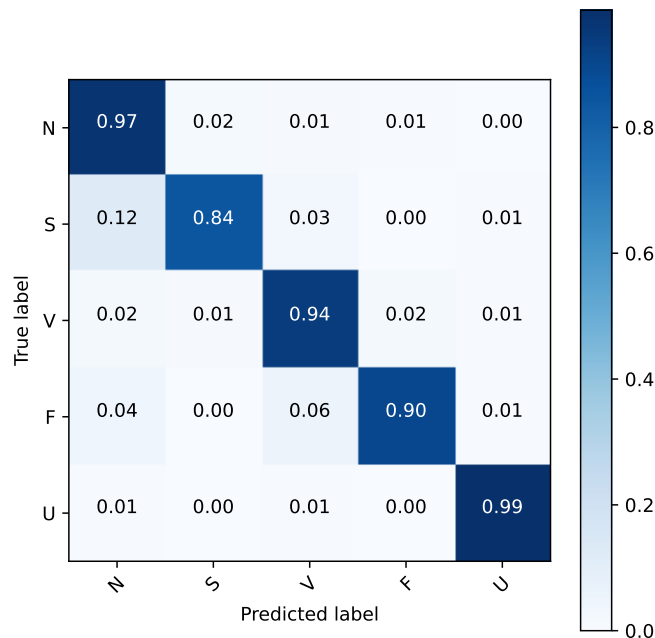


Figure 43: Confusion matrix for the FedAvg experiment with the CNN model trained on Non-IID data. The confusion matrix shows a clear diagonal indicating a high rate of true positives and true negatives in the model. The diagonal is slightly more evident for the CNN model with FedAvg compared to the CNN model with centralized learning. The confusion matrix for the CNN model with centralized learning is illustrated in Figure 31.



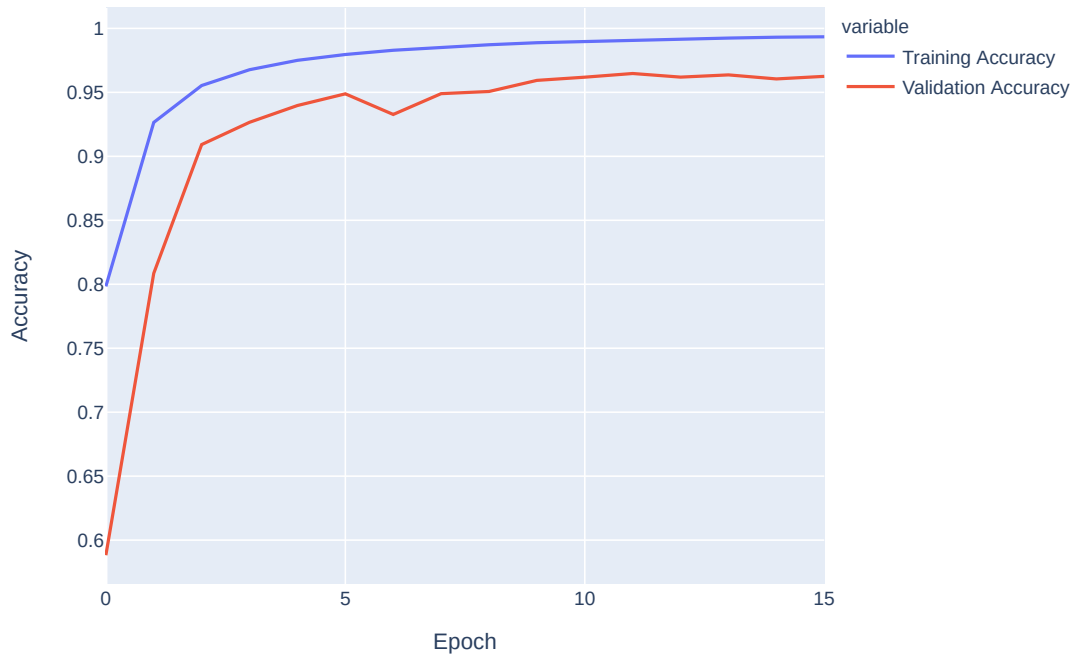


Figure 44: Graph illustrating the training and validation accuracy of the FedAvg experiment with the CNN model trained on Non-IID data. From the graph one can observe that the training and validation accuracy converges, indicating that the model did not overfit on the training data.

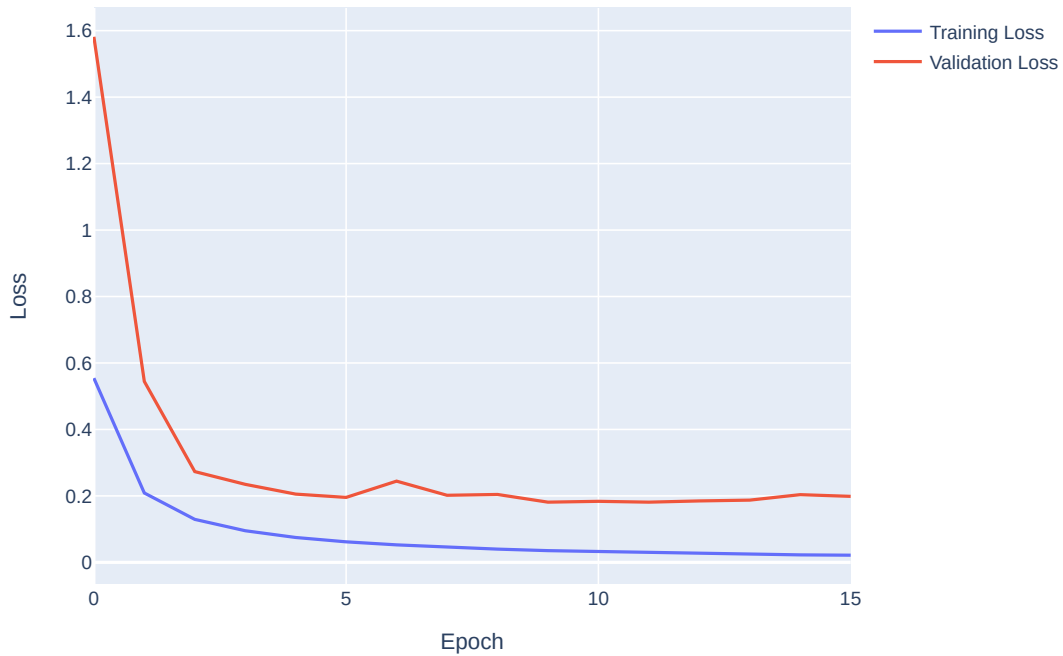


Figure 45: The graph illustrates the training and validation loss of the FedAvg experiment with the CNN model trained on Non-IID data. The graph illustrates that the validation and training loss converges.

### Uniform Distribution

The following results illustrate the performance of the CNN model applied to a uniform data distribution of the dataset. The distribution is described in Section 3.2.3.

Metrics	
Test Accuracy:	95.2%
Training Accuracy:	98.5%
Test Loss:	0.20
Training Loss:	0.05
Training Time:	598 s

Table 18: Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on the uniform data distribution. The metrics shown in this table describe a well-performing model with a relatively high training time. Compared to the CNN model trained with FedAvg on the Non-IID data distribution described in Table 16, the model performed a bit worse. However, the training time was lower for the uniform data distribution.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.95	0.97	18118
Supra Ventricular	0.52	0.84	0.64	556
Ventricular	0.86	0.94	0.90	1448
Fusion	0.41	0.88	0.56	162
Unknown	0.97	0.98	0.96	1608

Table 19: Classification report for the FedAvg experiment with the CNN model trained on uniform data distribution. The table describes the precision, recall, F1-Score and support values for the experiment. From the table one can observe that the F1-score is high for every class, except the *Fusion* class where it is slightly lower. Compared to the CNN model trained with FedAvg and the Non-IID data distribution, the performance of the model trained on uniform data was worse for every class.

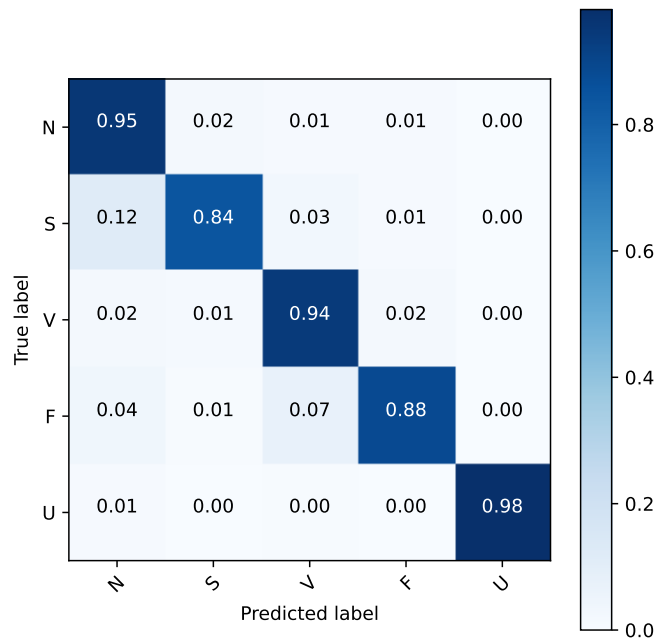


Figure 46: Confusion matrix for the FedAvg experiment with the CNN model trained on the uniform data distribution. The confusion matrix illustrates a clear diagonal indicating that the model had few false positives and few false negatives.

### Class Distribution

The following results illustrate the performance of the CNN model applied to the class distribution of the dataset. This distribution is described in Section 3.2.3. The class distribution only allows for 5 clients in total due to there only being 5 classes. This also means that there will only be 5 participating clients per round. This is the only experiment that used a different number of clients from what is described in Table 13.

Metrics	
Test Accuracy:	9.4%
Training Accuracy:	99.9%
Test Loss:	3.95
Training Loss:	0.000314
Training Time:	564 s

Table 20: Accuracy, loss and training time for the FedAvg experiment with the CNN model trained on class distributed data. The metrics shown in this table describe a model that performed extremely poorly. It also has a relatively high training time. Both compared to the Non-IID data distribution (Table 16) and the uniform data distribution (Table 18), this model performed much worse.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.00	0.00	0.00	18118
Supra Ventricular	0.00	0.00	0.00	556
Ventricular	0.06	0.81	0.11	1448
Fusion	0.06	0.46	0.11	162
Unknown	0.66	0.50	0.57	1608

Table 21: Classification report for the FedAvg experiment with the CNN model trained on class distributed data. The table describes the precision, recall, F1-Score and support values for the experiment. From the table one can observe that the F1-scores are extremely low for every class, except the *Unknown* class where it is decent. This indicates that the model had low class precision and low recall.

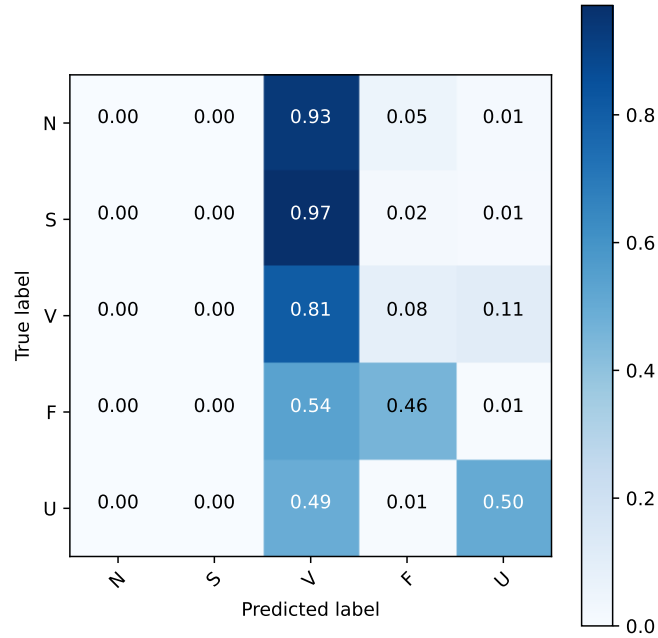


Figure 47: Confusion matrix for the FedAvg experiment with the CNN model trained on class distributed data. The figure shows a line straight down the middle of the confusion matrix, illustrating that the model classified nearly every ECG recording as *Ventricular beats*. This indicates a high false positive and false negative rate.

### 4.3 Experiments regarding Privacy Issues in Federated Learning

The results of the experiments presented in this section, aim to determine the threat of vulnerabilities in federated learning. This section will look at how some of the privacy issues in federated learning also occur in centralized learning.

#### 4.3.1 Federated Averaging with Static Data Poisoning

The purpose of conducting this experiment was to find out how the Federated Averaging (FedAvg) algorithm described in Section 2.2.1.2 performed in an adversarial setting. To create an adversarial setting, we inserted corrupted data into one of the participating clients. This is known as static data poisoning, and is explained in Section 2.4.2. As described in Section 3.1.3, each datapoint in an ECG recording is in the interval  $[0, 1]$ . In order to insert corrupted data, we sampled 20000 random values in the interval  $[20, 40]$  and labeled them as *Normal beats*. Furthermore, these values were inserted into client 1 which caused client 1 to become an outlier. When this client sends its local updates to the central server, these updates will be corrupted. The 9 remaining clients had non-poisoned local data. This experiment was conducted by training the CNN model with the FedAvg algorithm. Table 22 describes the training configuration used in this experiment.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Averaging (FedAvg)
Data distribution:	Non-IID
Epochs:	15
Client Epochs:	10
Total number of clients:	10
Number of participating clients per round:	10
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.02
Loss function:	Categorical Cross-Entropy

Table 22: Training configuration for the federated learning experiment using the FedAvg algorithm with static data poisoning.

The results of the static data poisoning experiment will be presented below.

Metrics	
Test Accuracy:	82.6%
Training Accuracy:	40.9%
Test Loss:	2663.8
Training Loss:	NaN
Training Time:	530 s

Table 23: Accuracy, loss and training time for the FedAvg experiment with static data poisoning. The training loss became NaN as a result of gradient explosion, causing the model to stop training. This failure happened because the model was not robust against the static data poisoning.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.83	1.00	0.90	18118
Supra Ventricular	0.00	0.00	0.00	556
Ventricular	0.00	0.00	0.00	1448
Fusion	0.06	0.01	0.01	162
Unknown	0.00	0.00	0.00	1608

Table 24: Classification report for the FedAvg experiment with static data poisoning. The table describes the precision, recall, F1-Score and support values for the experiment. The *Normal* class had the highest F1-score, which indicates that the model performed well on this class. This is a result of the model being strongly influenced by the corrupted client, which only had data with label *Normal* beats.

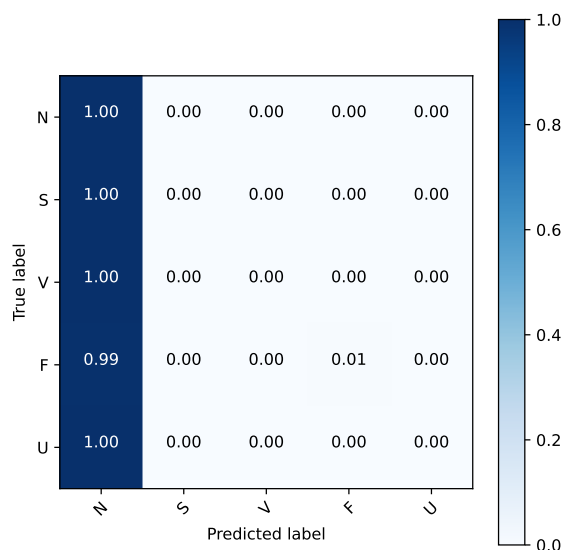


Figure 48: Confusion matrix for the FedAvg experiment with static data poisoning. As indicated in Table 24, the model was strongly influenced by the corrupt client, making it classify almost the whole test-dataset as *Normal* beats.

### 4.3.2 Memorization in Federated and Centralized Learning

This section will describe an experiment performed to observe the occurrence of memorization in federated and centralized learning. This experiment was inspired by the fact that information leakage is a well-known issue in machine learning models, often due to memorization as described in Section 2.5. We wanted to explore if the issue of memorization posed the same threat in federated learning as in centralized learning. In addition, we wanted to see how the size of the model used in training would affect the degree of memorization. To obtain results about memorization in federated and centralized learning, we decided to train both learning approaches with the artificial neural network and the 1D convolutional neural network. The experiment would only provide accurate results if we were to make sure that one type of the data would only occur a few times in the dataset. In order to achieve this, we manipulated the dataset

by reducing the *Unknown* class from 20000 examples to 100 examples. The remaining classes contained 20000 examples each.

### Federated Learning

Table 25 describes the training configuration used in the memorization experiment using federated learning.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Averaging (FedAvg)
Data distribution:	Non-IID
Epochs:	15
Client Epochs:	10
Total number of clients:	10
Number of participating clients per round:	10
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.02
Loss function:	Categorical Cross-Entropy

Table 25: Training configuration for the memorization experiment using the FedAvg algorithm.

#### 4.3.2.1 FedAvg with ANN

This section will present the results achieved when the ANN model was trained with the FedAvg algorithm while forcing memorization.

Metrics	
Test Accuracy:	92.9%
Training Accuracy:	98.3%
Test Loss:	0.29
Training Loss:	0.05
Training Time:	281 s

Table 26: Accuracy, loss and training time for the memorization experiment using the FedAvg algorithm with the ANN model. The metrics illustrated in this table describe a well-performing model with a relatively short training time.



Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.98	0.95	0.96	18118
Supra Ventricular	0.44	0.83	0.57	556
Ventricular	0.79	0.94	0.86	1448
Fusion	0.41	0.90	0.56	162
Unknown	0.99	0.71	0.83	1608

Table 27: Classification report for the memorization experiment using the FedAvg algorithm with the ANN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores, one can observe that the model performed well on the *Unknown* class despite the model only having seen training examples of the class a hundred times.

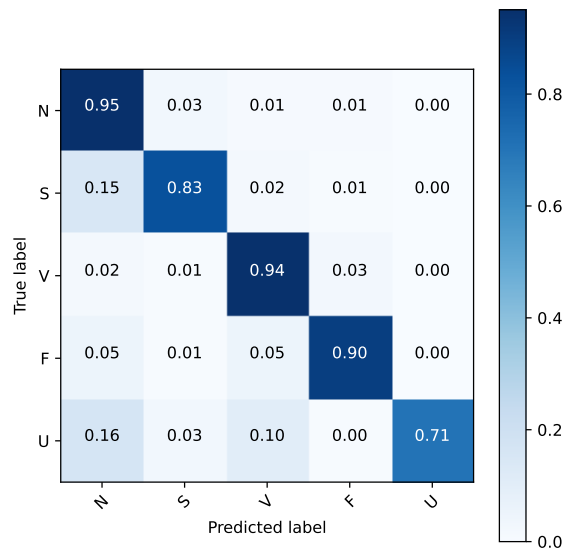


Figure 49: Confusion matrix for the memorization experiment using the FedAvg algorithm with the ANN model. The confusion matrix illustrates a clear diagonal, indicating a high number of true positives and true negatives. One can also observe that the model mostly classified correctly for the *Unknown* class despite it not having seen more than a hundred training examples of the class.

#### 4.3.2.2 FedAvg with CNN

This section will present the results achieved while training the 1D CNN model with the FedAvg algorithm.

Metrics	
Test Accuracy:	95.0%
Training Accuracy:	99.3%
Test Loss:	0.31
Training Loss:	0.02
Training Time:	602 s

Table 28: Accuracy, loss and training time for the memorization experiment using the FedAvg algorithm with the CNN model. The metrics shown in this table describe a well-performing model with a relatively long training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.98	0.97	0.98	18118
Supra Ventricular	0.59	0.84	0.69	556
Ventricular	0.81	0.95	0.87	1448
Fusion	0.56	0.85	0.67	162
Unknown	1.00	0.78	0.88	1608

Table 29: Classification report for the memorization experiment using the FedAvg algorithm with the CNN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores, one can observe that the model performed well on the *Unknown* class despite it only having seen training examples of the class a hundred times. The model did also have high F1-scores for the other class, indicating a high true positive rate and a high true negative rate.

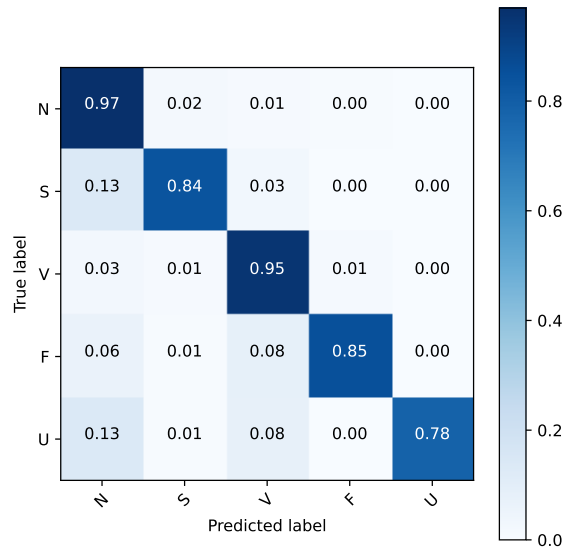


Figure 50: Confusion matrix for the memorization experiment using the FedAvg algorithm with the CNN model. The confusion matrix shows a clear diagonal. One can observe that the model largely classified correctly for the *Unknown* class despite it only having seen a hundred training examples of the class.

## Centralized Learning

Table 30 describes the training configuration used in the memorization experiment using centralized learning.

Training Configuration	
Learning algorithm:	Centralized
Epochs:	15
Server optimizer:	SGD
Learning rate:	0.01
Loss function:	Categorical Cross-Entropy

Table 30: Training configuration for the memorization experiment using centralized learning.

### 4.3.2.3 Centralized Learning with ANN

This section will present the results achieved while training the ANN model with centralized learning.

Metrics	
Test Accuracy:	92.7%
Training Accuracy:	98.2%
Test Loss:	0.27
Training Loss:	0.05
Training Time:	190 s

Table 31: Accuracy, loss and training time for the memorization experiment using centralized learning with the ANN model. The metrics illustrated in this table describe a well-performing model with a low training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.96	0.96	0.96	18118
Supra Ventricular	0.55	0.78	0.65	556
Ventricular	0.83	0.92	0.87	1448
Fusion	0.33	0.88	0.48	162
Unknown	1.00	0.59	0.74	1608

Table 32: Classification report for the memorization experiment using centralized learning with the ANN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model performed well on the *Unknown* class despite it only having seen training examples of the class a hundred times. However, the model performed worse on the *Unknown* class compared to the memorization experiment using FedAvg described in Table 27.

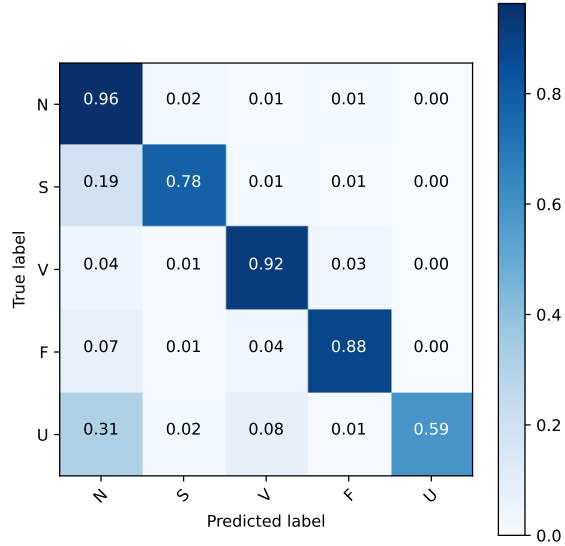


Figure 51: Confusion matrix for the memorization experiment using the FedAvg algorithm with the ANN model. The confusion matrix shows a relatively clear diagonal. One can observe that the model classified correctly for the *Unknown* class far less than for the other classes, but it still performed decently on this class despite only having seen training examples of the class a hundred times.

#### 4.3.2.4 Centralized Learning with CNN

This section will present the results achieved while training the 1D CNN model with centralized learning.

Metrics	
Test Accuracy:	94.2%
Training Accuracy:	99.2%
Test Loss:	0.30
Training Loss:	0.02
Training Time:	403 s

Table 33: Accuracy, loss and training time for the memorization experiment using centralized learning with the CNN model. The metrics illustrated in this table describe a well-performing model that had a decent training time.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.95	0.97	18118
Supra Ventricular	0.46	0.85	0.59	556
Ventricular	0.80	0.95	0.87	1448
Fusion	0.52	0.87	0.65	162
Unknown	0.99	0.85	0.92	1608

Table 34: Classification report for the memorization experiment using centralized learning with the CNN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model performed extremely well on the *Unknown* class, despite only having seen training examples of this class a hundred times. The F1-score for the *Unknown* class is slightly higher for this experiment compared to the memorization experiment using FedAvg with the CNN model described in Table 29.

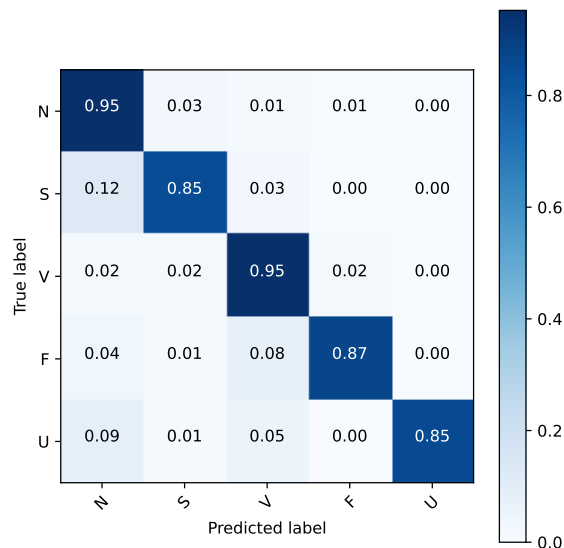


Figure 52: Confusion matrix for the memorization experiment using centralized learning with the CNN model. The confusion matrix shows a clear diagonal. One can observe a high rate of true positives and true negatives for the *Unknown* class despite it only having seen training examples of this class a hundred times.

### 4.3.3 Model Extraction in Federated Learning

The purpose of this experiment was to perform model extraction in federated learning, and show a representation of the local training data at the clients. This was done by displaying the weight updates from the participating clients in the federated learning loop. By using the softmax regression model described in Section 3.2.4.1, we were able to show a relatively exact estimation of the client’s training data. Furthermore, to make it simple to identify the training data of a client, we inserted 20000 examples of *Normal beats* ECG recordings into client 1. The remaining clients received non-IID data. This made client 1 an outlier, as it only had one type of data. In other words, client 1 had low variance within its local data, and high variance from the data of the other clients. In contrast to the other experiments in Chapter 4, the purpose of this experiment was not to train a well-performing model, but to perform model extraction. The training configuration used in this experiment can be viewed in Table 35.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Averaging (FedAvg)
Data distribution:	Non-IID
Epochs:	5
Client Epochs:	10
Total number of clients:	5
Number of participating clients per round:	5
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.1
Loss function:	Categorical Cross-Entropy

Table 35: Training configuration for the model extraction experiment using the FedAvg algorithm.

The following figures will illustrate the weights extracted during this experiment.

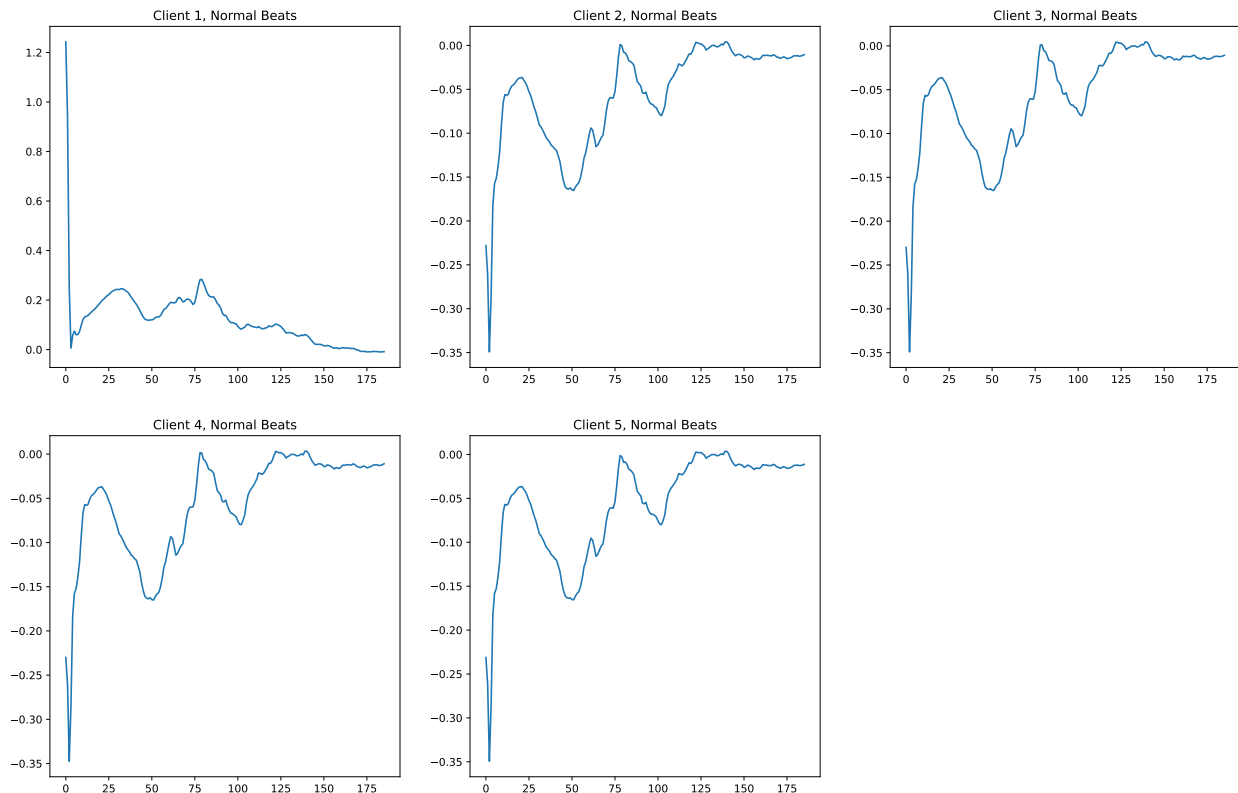


Figure 53: Model Extraction for Normal Beats class from the 5 participating clients. These weights were extracted after the last round of the federated averaging loop. The weights of client 1 stand out from the rest of the clients. In addition, one can observe that the curve has a strong resemblance to the *Normal* beats class displayed in the 2D-histogram in Figure 27.

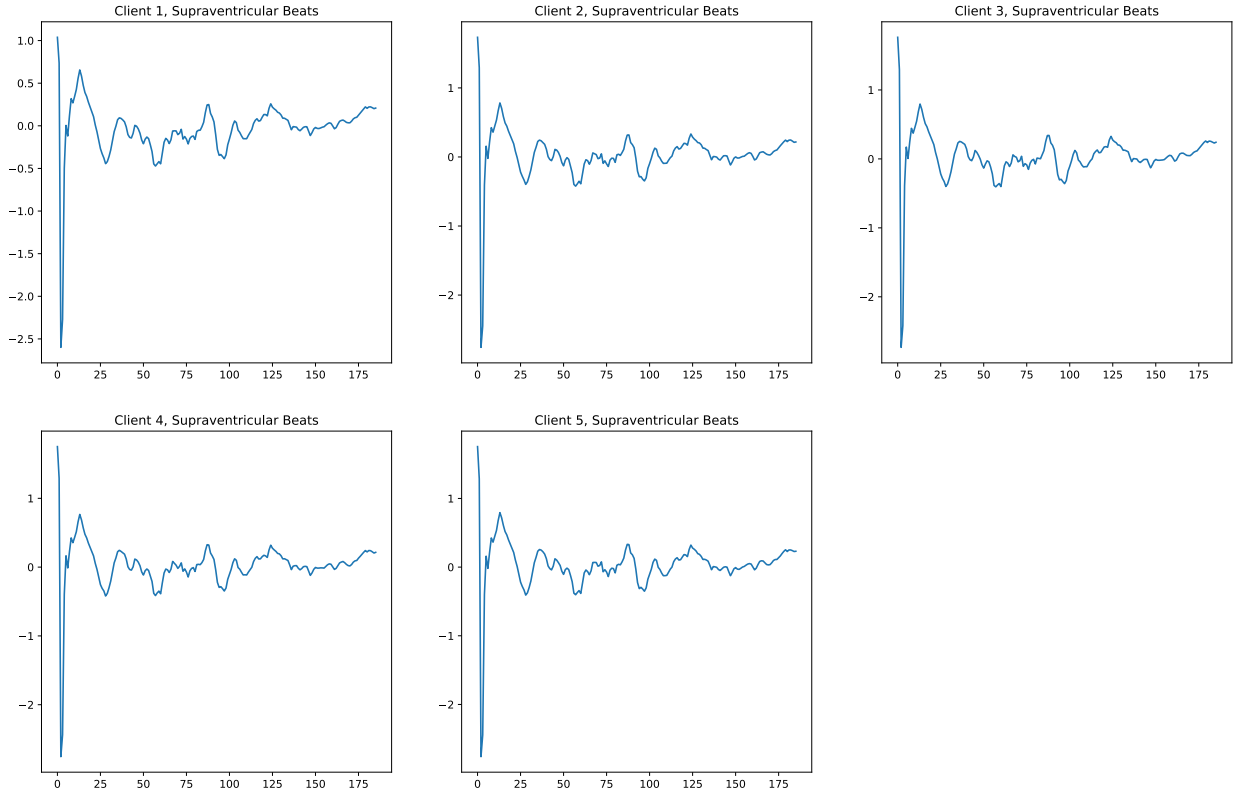


Figure 54: Model Extraction for Supraventricular Beats class from the 5 participating clients. These weights were extracted after the last round of the federated averaging loop. The weights of the clients are relatively equal, and this is a result of the model being aggregated after each training loop.

## 4.4 Privacy-Preserving Experiments in Federated Learning

The results of the experiments described in this section will attempt to explore the effect of utilizing different privacy-preserving techniques.

### 4.4.1 Robust Federated Aggregation with Static Data Poisoning

The purpose of conducting this experiment was to find out how the Robust Federated Aggregation (RFA) algorithm described in Algorithm 3 performed with static data poisoning. The results from this experiment can be compared to other federated learning runs where static data poisoning was present. The setup for this experiment was the same as the experiment described in Section 4.3.1. The only difference is the aggregation algorithm used on the updates from the clients. In the experiment described in Section 4.3.1, the federated learning algorithm FedAvg was used. This algorithm uses the arithmetic mean to aggregate the client updates. In the experiment presented in this section, the robust federated aggregation (RFA) method was utilized. RFA uses the geometric median to aggregate the client updates. This experiment was conducted by training the CNN model using the RFA algorithm. The training configuration for this experiment is presented in Table 36.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Robust Federated Aggregation (RFA)
Data distribution:	Non-IID
Epochs:	15
Client Epochs:	10
Total number of clients:	10
Number of participating clients per round:	10
Number of calls to Secure Average Oracle:	3
L2-threshold:	$10^{-6}$
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.02
Loss function:	Categorical Cross-Entropy

Table 36: Training configuration for the federated learning experiment using the RFA algorithm with static data poisoning.

Metrics	
Test Accuracy:	92.0%
Training Accuracy:	97.9%
Test Loss:	0.28
Training Loss:	0.06
Training Time:	501 s

Table 37: Accuracy, loss and training time for the RFA experiment with static data poisoning. Compared to Table 23, this model did not suffer from gradient explosion. The model was also able to obtain a test accuracy of 92% in 501 seconds, making it both a well-performing and robust model.



Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.99	0.92	0.95	18118
Supra Ventricular	0.41	0.84	0.55	556
Ventricular	0.81	0.92	0.86	1448
Fusion	0.22	0.92	0.35	162
Unknown	0.94	0.98	0.96	1608

Table 38: Classification report for the RFA experiment with static data poisoning. The table describes the precision, recall, F1-Score and support values for the experiment. Compared to Table 24, this model obtained higher F1-scores for all the 5 classes.

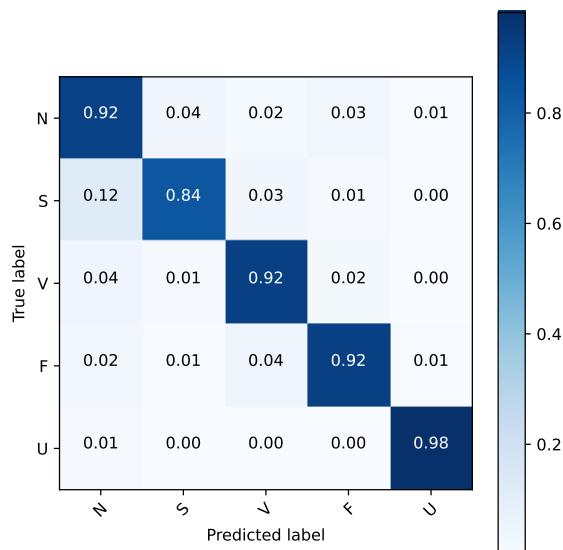


Figure 55: Confusion matrix for the RFA experiment with static data poisoning. Compared to the confusion matrix in Figure 48, this model was able to correctly classify the other classes in the test dataset, not only the *Normal* class.

#### 4.4.2 Differential Privacy in Federated Learning

This section will describe an experiment performed to observe the effect of applying  $(\epsilon, \delta)$ -differential privacy (see Section 2.6.1.1) while training models using federated learning. We wanted to see how differential privacy affected the performance and memorization in different models. To obtain results regarding differential privacy in federated learning, we trained both the artificial neural network and the convolutional neural network with the Differentially-Private Federated Averaging (DP-FedAvg) algorithm described in Algorithm 4. This was done twice, once without forced memorization and the other with forced memorization. We forced memorization in the models in the same way as in the memorization experiment described in Section 4.3.2. Table 39 shows the training configuration used in this experiment.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Differentially-private Federated Averaging (DP-FedAvg)
Data distribution:	Non-IID
Epochs:	15
Client Epochs:	10
Total number of clients:	10
Number of participating clients per round:	10
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.02
Loss function:	Categorical Cross-Entropy

Table 39: Training configuration for the differential privacy experiment using the DP-FedAvg algorithm.

#### 4.4.2.1 DP-FedAvg with ANN

This section will present the results obtained while training the ANN model with differential privacy using the DP-FedAvg algorithm. Table 40 describes the differential privacy parameters used in the experiment.

Differential Privacy Parameters	
Differential privacy mechanism:	Gaussian fixed
Delta ( $\delta$ ):	$10^{-5}$
Noise multiplier:	0.45
Clipping norm:	0.80

Table 40: DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model.

Metrics	
Test Accuracy:	69.3%
Training Accuracy:	91.7%
Test Loss:	0.79
Training Loss:	0.22
Training Time:	336 s

Table 41: Accuracy, loss and training time for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model. The metrics shown in this table describe a model that overall performed decently. The model managed to achieve a test accuracy of approximately 69% while only using 336 seconds to train. However, compared to the FedAvg experiment with the ANN model described in Table 14, this is significantly worse.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.98	0.68	0.80	18118
Supra Ventricular	0.12	0.77	0.21	556
Ventricular	0.40	0.89	0.55	1448
Fusion	0.09	0.88	0.16	162
Unknown	1.00	0.64	0.78	1608

Table 42: Classification report for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model performed well on the *Normal* and the *Unknown* class, but significantly worse for the remaining three classes. Compared to the F1-scores of the FedAvg experiment described in Table 15, this model performed worse on every class.

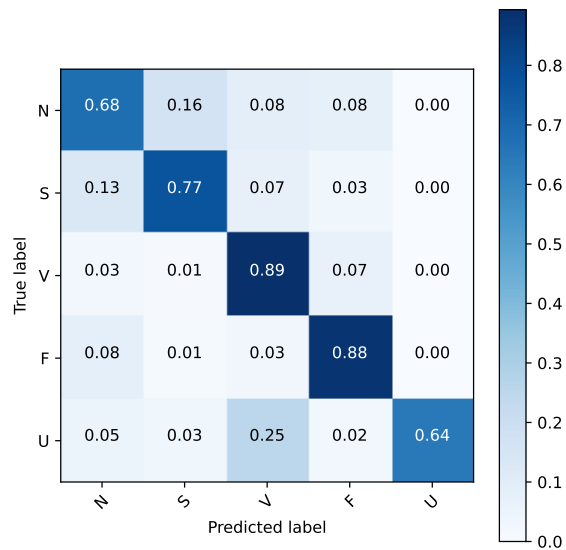


Figure 56: Confusion matrix for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model. The confusion matrix shows a fairly clear diagonal indicating a decently high rate of true positives and true negatives. However, one can observe that the model was more uncertain in regards to the *Normal* and the *Unknown* classes. In comparison to the confusion matrix of the FedAvg experiment illustrated in Figure 40, this model classified incorrectly more often.

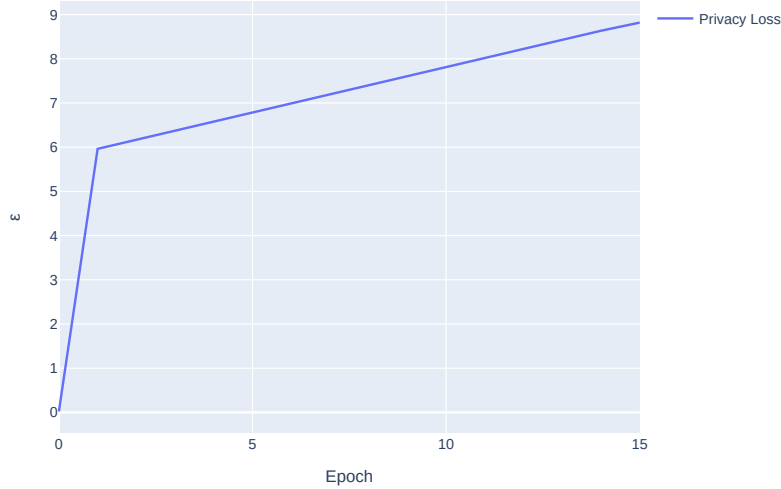


Figure 57: The figure illustrates the moments accountant for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model. It shows cumulative privacy loss. From this graph one can observe that  $\epsilon$  was approximately 9 when the model was done training.

#### 4.4.2.2 DP-FedAvg with CNN

This section will present the result of training the CNN model with the DP-FedAvg algorithm. Table 43 describes the differential privacy parameters used in this experiment.

Differential Privacy Parameters	
Differential privacy mechanism:	Gaussian fixed
Delta ( $\delta$ ):	$10^{-5}$
Noise multiplier:	0.45
Clipping norm:	0.80

Table 43: DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model.

Metrics	
Test Accuracy:	64.8%
Training Accuracy:	93.7%
Test Loss:	1.34
Training Loss:	0.18
Training Time:	671 s

Table 44: Accuracy, loss and training time for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model. The metrics illustrated in this table describe a model that performed decently. However, compared to the FedAvg experiment with the CNN model described in Table 16, DP-FedAvg performed significantly worse.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.98	0.61	0.75	18118
Supra Ventricular	0.14	0.74	0.23	556
Ventricular	0.28	0.86	0.42	1448
Fusion	0.07	0.74	0.13	162
Unknown	0.88	0.85	0.86	1608

Table 45: Classification report for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model performed best for the *Unknown* and the *Normal* class, but the F1-scores are overall worse than for the FedAvg experiment with the CNN model described in Table 17.

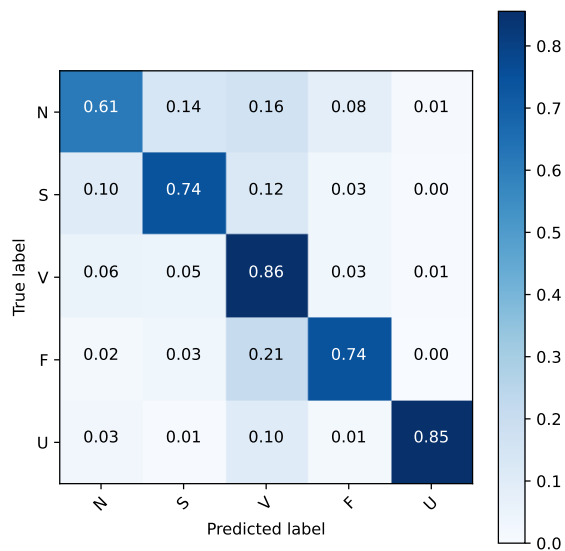


Figure 58: Confusion matrix for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model. The confusion matrix illustrates a relatively clear diagonal indicating that the model is more than capable of classifying correctly on every class. However, the diagonal is more faint compared to the FedAvg experiment with the CNN model as illustrated in Figure 43.

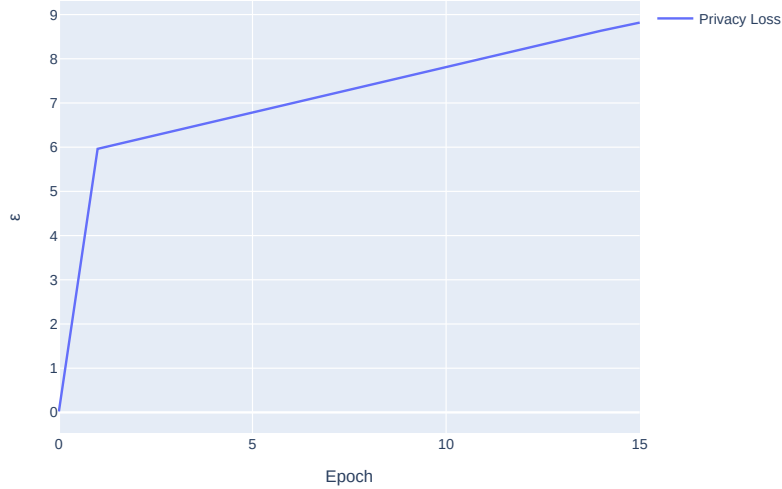


Figure 59: The figure illustrates the moments accountant for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model. It shows cumulative privacy loss. From this graph one can observe that  $\epsilon$  was approximately 9 when the model was done training.

#### 4.4.2.3 Forced Memorization in ANN

This section will present the result of training the ANN model with differential privacy using the DP-FedAvg algorithm while forcing memorization in the model. Table 46 describes the differential privacy parameters used in this experiment.

Differential Privacy Parameters	
Differential privacy mechanism:	Gaussian fixed
Delta ( $\delta$ ):	$1.25 \cdot 10^{-5}$
Noise multiplier:	0.5
Clipping norm:	0.75

Table 46: DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the ANN model while forcing memorization.

Metrics	
Test Accuracy:	74.8%
Training Accuracy:	90.7%
Test Loss:	0.93
Training Loss:	0.25
Training Time:	281 s

Table 47: Accuracy, loss and training time for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model. The metrics illustrated in this table describe a model that performed decently with an accuracy of approximately 75%. The training time of the model was also quite low. Compared to the metrics of the memorization experiment described in Table 26, the test accuracy in the DP-FedAvg experiment with forced memorization, is a fair bit lower.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.94	0.80	0.87	18118
Supra Ventricular	0.17	0.78	0.28	556
Ventricular	0.42	0.88	0.56	1448
Fusion	0.16	0.81	0.26	162
Unknown	0.00	0.00	0.00	1608

Table 48: Classification report for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model was not able to classify the *Unknown* class when applying differential privacy. In comparison to the stats of the memorization experiment described in Table 27, this is an enormous shift because both models only saw the training examples of the *Unknown* class a hundred times.

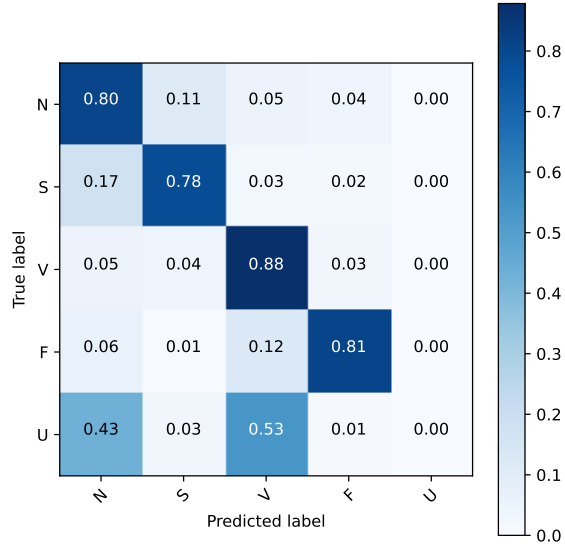


Figure 60: Confusion matrix for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the ANN model. The confusion matrix shows a clear diagonal for all classes except the *Unknown* class. One can observe that each time the model received an ECG of the *Unknown* class, it classified everything but the *Unknown* class.

#### 4.4.2.4 Forced Memorization in CNN

This section will present the result of training the CNN model with the DP-FedAvg algorithm while forcing memorization in the model. Table 49 describes the differential privacy parameters used in this experiment.

Differential Privacy Parameters	
Differential privacy mechanism:	Gaussian fixed
Delta ( $\delta$ ):	$1.25 \cdot 10^{-5}$
Noise multiplier:	0.5
Clipping norm:	0.75

Table 49: DP Parameters for the differential privacy experiment using the DP-FedAvg algorithm with the CNN model while forcing memorization.



Metrics	
Test Accuracy:	75.9%
Training Accuracy:	92.2%
Test Loss:	0.85
Training Loss:	0.22
Training Time:	555 s

Table 50: Accuracy, loss and training time for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model. The metrics shown in this table describe a model that performed decently on the test dataset. Compared to the memorization experiment with the CNN model, the accuracy illustrated in this table is significantly lower. This indicates that the model trained with DP-FedAvg performed worse on the test data compared to the FedAvg algorithm.

Classification Report				
Class	Precision	Recall	F1-Score	Support
Normal	0.91	0.82	0.86	18118
Supra Ventricular	0.08	0.41	0.13	556
Ventricular	0.69	0.70	0.69	1448
Fusion	0.19	0.84	0.31	162
Unknown	0.97	0.24	0.39	1608

Table 51: Classification report for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model. The table describes the precision, recall, F1-Score and support values for the experiment. From the F1-scores one can observe that the model performed badly for the *Supra Ventricular*, the *Fusion* and the *Unknown* class. Compared to the memorization experiment with FedAvg, this model had a notable reduction in precision and recall for the *Unknown* class.

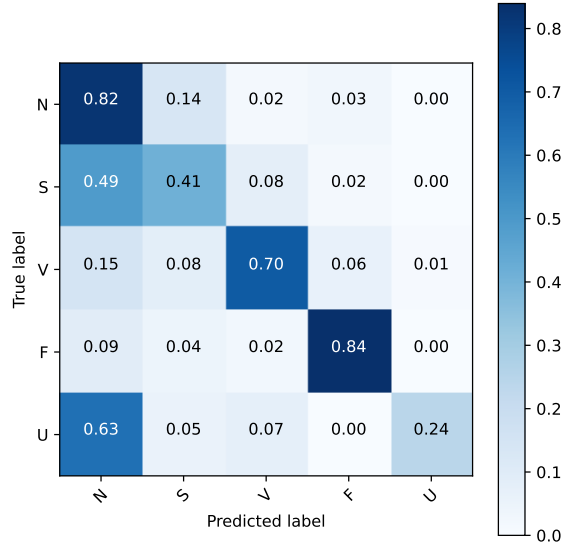


Figure 61: Confusion matrix for the differential privacy experiment with memorization using the DP-FedAvg algorithm with the CNN model. The confusion matrix illustrates a slight diagonal. However, one can easily observe that the model struggled with classifying the *Supra Ventricular Beats* and the *Unknown Beats*. Compared to the memorization experiment with FedAvg illustrated in Figure 50, the diagonal of the DP-FedAvg experiment with forced memorization is significantly less distinct.

#### 4.4.3 Model Extraction in Federated Learning with Differential Privacy

The purpose of this experiment was to perform model extraction in federated learning with differential-private federated averaging (DP-FedAvg), and to show a representation of the local training data at the clients. The setup for this experiment was the same as for the experiment described in Section 4.3.3. However, the aggregation algorithm used on the updates from the clients was different. The experiment described in Section 4.3.3 used the FedAvg algorithm, while this experiment utilized the differentially-private federated averaging to aggregate. This experiment trained the softmax regression model. The training configuration used in this experiment is presented in Table 52.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Differentially-private Federated Averaging (DP-FedAvg)
Data distribution:	Non-IID
Epochs:	5
Client Epochs:	10
Total number of clients:	5
Number of participating clients per round:	5
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.1
Loss function:	Categorical Cross-Entropy

Table 52: Training configuration for the model extraction experiment using the DP-FedAvg algorithm.

Table 53 describes the differential privacy parameters used in this experiment.

Differential Privacy Parameters	
Differential privacy mechanism:	Gaussian fixed
Delta ( $\delta$ ):	$10^{-5}$
Noise multiplier:	0.5
Clipping norm:	0.75

Table 53: Differential privacy parameters for the model extraction experiment using the DP-FedAvg algorithm.

### Model Extraction

While training the softmax regression model, the updates were collected from the 5 participating clients. These updates are displayed in Figures 62 and 63 in order to illustrate the difference between client 1 and the rest of the clients.

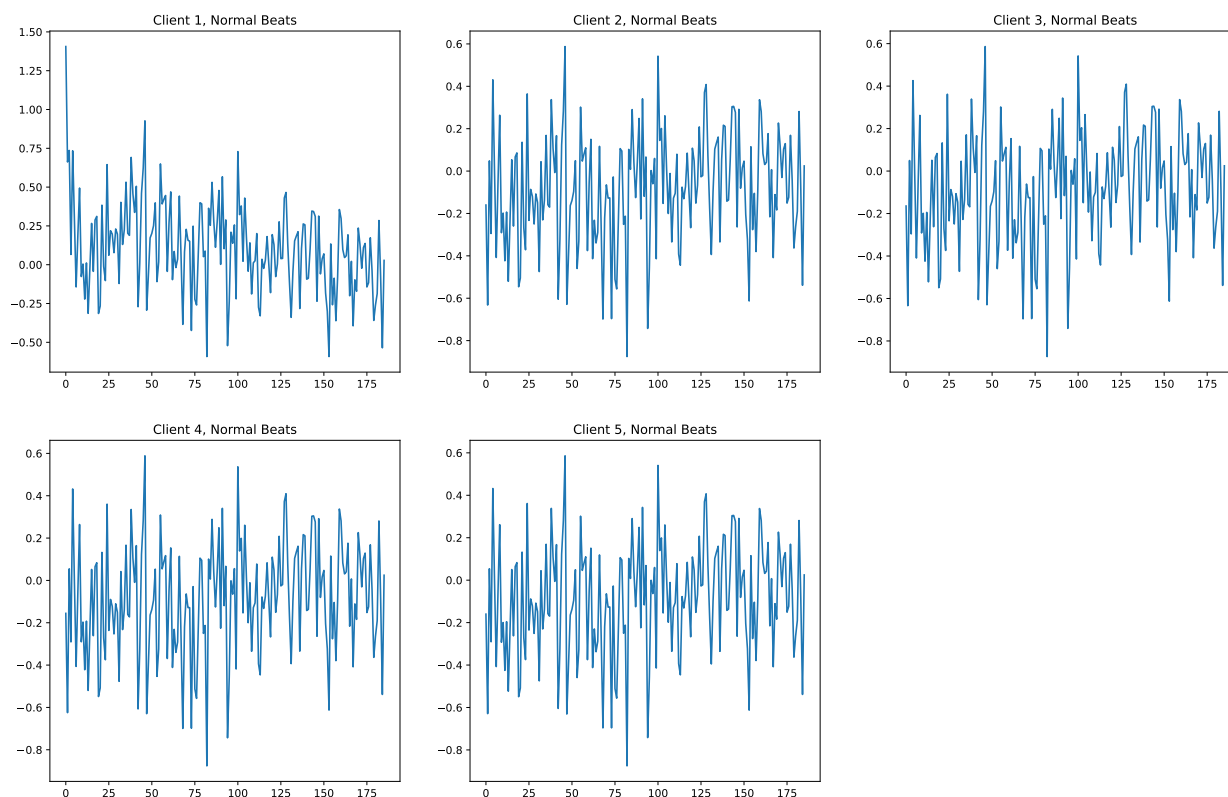


Figure 62: Model Extraction for Normal Beats class from the 5 participating clients. These weights were extracted after the last round of the differential-private federated averaging loop. One can observe that the *Normal Beats* at client 1 stands out from the rest of the clients. However, all of the client's weights consist of a lot of noise.

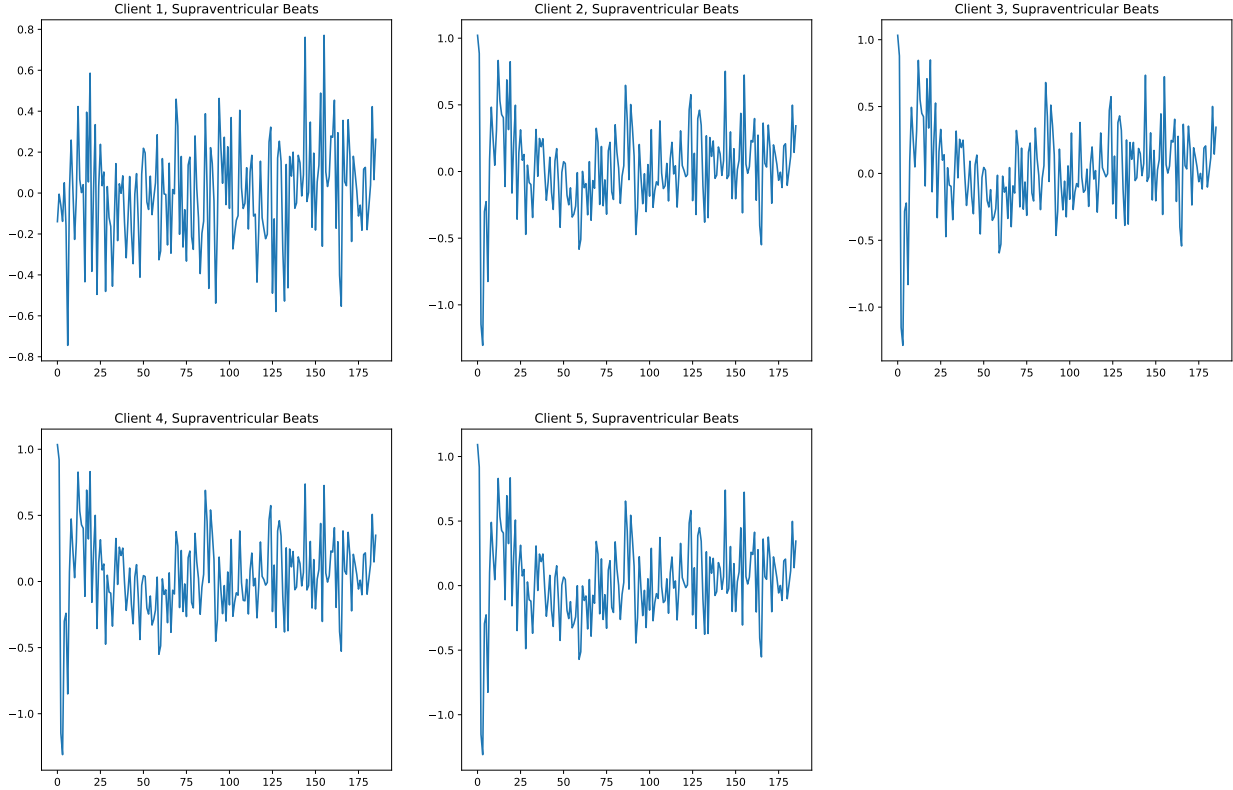


Figure 63: Model Extraction for Supraventricular Beats class from the 5 participating clients. These weights were extracted after the last round of the differential-private federated averaging loop. One can observe that the weights consist of a lot of noise for all the clients, making it difficult to distinguish the weights.

#### 4.4.4 Federated Learning with Homomorphic Encryption

The purpose of this experiment was to demonstrate the use of homomorphic encryption (HE) in federated learning. We implemented Algorithm 5, but replaced fully homomorphic (FHE) encryption with partially homomorphic encryption (PHE), and used the Paillier Cryptosystem which is described in Section 2.7.2.1. Neither TensorFlow or TensorFlow Federated supported machine learning with homomorphic encryption. Therefore, we had to implement federated learning from scratch using only NumPy (see Section 3.3.5). Furthermore, we used the Python Paillier library (see Section 3.3.3) to implement PHE. The aggregation algorithm in this experiment was federated averaging, and the model to be trained was the softmax regression model presented in Section 3.2.4.1. The purpose of this experiment was not to train a well-performing model, only to conduct federated learning with encrypted weights. As explained in Section 2.7.3, homomorphic encryption with federated learning is computationally expensive. Therefore, we did not train the model with as many epochs and client epochs as the other experiments in Chapter 4. The training configuration used in this experiment can be observed in Table 54.

Training Configuration	
Learning algorithm:	Federated
Aggregation method:	Federated Averaging with HE
Data distribution:	Non-IID
Epochs:	3
Client Epochs:	3
Total number of clients:	5
Number of participating clients per round:	5
Key size:	1024
Server optimizer:	SGD
Server learning rate:	1.0
Client optimizer:	SGD
Client learning rate:	0.1
Loss function:	Categorical Cross-Entropy

Table 54: Training configuration for homomorphic encryption with federated averaging.

Metrics	
Test Accuracy:	8.01%
Training Accuracy:	21.8%
Test Loss:	3.15
Training Loss:	2.77
Training Time:	796 s

Table 55: Accuracy, loss and training time after performing federated learning with homomorphic encryption. The metrics shown in this table describe a model that performed badly. The model did also have an extremely high training time. The training time is enormous considering that the model only trained using 3 epochs.

Weight Size	
Weight:	24 B
Encrypted Weight:	300 B

Table 56: Size of a weight and its encrypted value in number of bytes. The size of the encrypted weight is 12.5 larger than the original value.

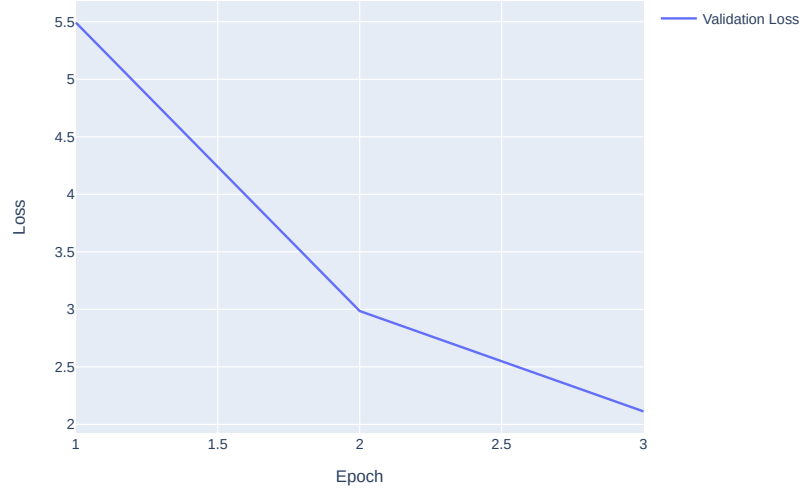


Figure 64: Graph illustrating the validation loss for doing federated learning with homomorphic encryption. Since the validation loss is decreasing for each epoch, this model was able to learn from the training data using Federated Averaging with Homomorphic Encryption.

## 5 Discussion

This chapter will discuss the results of the experiments presented in Chapter 4. The results will be discussed in regards to the related work introduced in Chapter 2. This chapter will also elaborate on why the results are important in terms of privacy, security and model performance in federated learning. The discussion will provide the basis for answering the research questions described in Chapter 1.

### 5.1 Federated and Centralized Learning

In this report we wanted to assess how federated learning compares to centralized learning, both in terms of privacy and performance. We also wanted to explore the occurrence of memorization in both learning approaches. With the intention of doing this, we conducted several experiments in regards to both centralized and federated learning. The results of the different experiments can be found in the sections listed in Table 57.

Overview of the Experiments	
Centralized Learning:	Section 4.2.1
FedSGD:	Section 4.2.2
FedAvg:	Section 4.2.3
Memorization:	Section 4.3.2

Table 57: The results discussed in Section 5.1

#### 5.1.1 Model Performance

Before discussing the privacy metrics of the two learning approaches, we wanted to compare the model performance of each approach. Starting with centralized learning, we achieved a test accuracy of 92.8% and 97.1% with the ANN model and the CNN model, respectively. The confusion matrices for these two models, displayed in Figures 28 and 31, illustrate a clear diagonal. This indicates that both models classified many true positives and true negatives for each class. Both the test accuracy and F1-scores for the CNN model are higher than the test accuracy and F1-scores achieved with the ANN model, making the CNN model the best performing model trained during the centralized learning experiment in terms of accuracy.

As for federated learning, we started with testing the simplest aggregation method presented in this report, namely FedSGD. This aggregation method obtained relatively poor results. The test accuracy achieved with the ANN model using FedSGD was 62.1% , and 50.4% with the CNN model. Such poor results were expected because, as described in Section 2.2.1.1, FedSGD only performs one step of gradient descent on the client data before sending the updates back to the central server to be aggregated.

To further evaluate federated learning, we also performed experiments using the federated learning algorithm FedAvg. This aggregation method performed almost equally well as the models trained with centralized learning. As illustrated in Section 4.2.3, training with FedAvg resulted in test accuracies of 93.4% and 96.2% for the ANN and CNN models, respectively. The F1-scores presented in Tables 15 and 17 reveal that the F1-scores obtained with FedAvg are relatively similar to the values achieved with centralized learning. This indicates that the two algorithms performed similarly on the five classes in the dataset. As described in Section 2.2.1.2, FedAvg performs several rounds of optimization on the client datasets, thus providing better models than those obtained with FedSGD.

The results concerning model performance in federated and centralized learning, illustrate that both learning approaches applied to the MIT-BIH Arrhythmia Dataset were able to provide well-performing models.

However, it became clear that the FedSGD aggregation method was not optimal as it converged slower due to it only performing one step of gradient descent on the clients data before aggregating. The results obtained during these preliminary experiments supported the related work within this field described in Section 2.2.1. Moreover, the results gathered during the preliminary experiments reveal a slight benefit to training the CNN model with centralized learning compared to FedAvg in terms of model performance. This can be explained with the fact that federated learning allows clients to keep their data local while training, while centralized learning collects all the data and stores it in a database. As described in Section 2.2, when training a model using federated learning, the global model is sent from the central server to the individual clients. The individual clients then train this model using their own local data which causes the model to become specialized. The aggregation performed on the client updates attempts to create a generalized model. However, the aggregated model might not become as generalized as one would like. This issue is unique to federated learning, as centralized learning does not train several models that needs to be aggregated after each global epoch.

The training time for the CNN model was longer when the model was trained with federated averaging compared to centralized learning. The training time for the CNN model trained with centralized learning was 471 seconds, while the training time for the CNN model with FedAvg was 687 seconds. Both models trained for 15 epochs. As mentioned in Section 3.3.4, the hardware used to train the models was a NVIDIA GeForce GTX 1070 with CUDA which made model training efficient. However, if federated averaging was performed on *real* decentralized clients on separate devices instead of using TensorFlow Federated (TFF), the difference in training times would be much larger. This is because TFF initiates the clients locally, thus making the communication time between the server and the clients negligible.

### 5.1.2 Memorization as a Privacy Issue

In Section 4.3.2, the results concerning memorization in federated and centralized learning were presented. In these experiments, we forced memorization in the different models by significantly reducing the number of training examples within the *Unknown* class. We then observed how the models performed on this class. The results demonstrated a high degree of memorization in both federated and centralized learning using the ANN and the CNN model. In the confusion matrices for FedAvg and centralized learning using the ANN model, one can observe that the values for the *Unknown* class are relatively high despite this class only having 100 training examples. These confusion matrices are shown in Figures 51 and 52. The same phenomenon can be observed in the confusion matrices for training the CNN model with FedAvg and centralized learning. However, when training the CNN model, both learning approaches showed an even higher degree of memorization than with the ANN model. Since the CNN model is larger than the ANN, which can be seen in the figures in Section 3.2.4, the CNN model had an increased ability to memorize outlier data. This is coherent with the theory presented in Section 2.5.1, where it was shown that larger models are more inclined to memorize outlier data.

As explained previously, the ANN models trained with both centralized and federated learning showed less memorization than the CNN models. Memorization is considered to be a larger problem in big models, and from the results it is clear that federated learning has less memorization in the CNN model compared to centralized learning with the CNN model. This could imply that memorization is a slightly smaller issue in federated learning. The reduction in memorization in federated learning could be explained by the aggregation. When the weights are aggregated, the average of the weights is calculated, which could potentially cause the weights responsible for memorizing the outlier data to loose information. Regardless, memorization is a significant privacy issue in both centralized and federated learning as it enables adversaries to use forward-propagation to look at which of the weights have been activated. An attacker could conduct an inference attack that would reveal the weights who are rarely activated and can infer memorized data. This is described in Section 2.5.1.

We used the confusion matrix to determine the presence of memorization when training models with different learning algorithms. When we first conducted the experiments regarding memorization, we found this method to be uncertain. This is because we did not directly study the weights of the models to check the presence of memorization. After running some more experiments, we realized that the confusion ma-



trix was adequate for checking memorization, since we could see how well the model performed on the class with only 100 training examples.

### 5.1.3 Privacy Benefits in using Federated Learning

Even though federated learning might sacrifice some model performance, it provides a significant privacy benefit when training machine learning models on sensitive data. This is due to the fact that when using federated learning to train models, it is not required to collect data from the clients. Instead, federated learning allows its clients to train a model locally and then send the weights of the updated model back to the central server. The only time the weights are sent to the server is when the clients’ updates are aggregated. This means that the information that needs to be communicated between the clients and the server, is sent far less often and contains fewer details concerning the original training data. This is considered to be a significant privacy benefit, especially when handling sensitive data such as medical records, as the risk of data leakage is greatly reduced. The server never stores the updates from the clients, and the updates are discarded immediately after aggregation. In order to train a model with centralized learning, all the data has to be in the vicinity of the model. In centralized learning, it is normal for the server to hold the data collected from the clients in order to make it possible to train several models on the data later. Storing privacy-sensitive data comes with high responsibility to protect that data. This is not the case in federated learning.

In comparing Algorithm 1 for FedSGD and Algorithm 2 for FedAvg, one can observe that FedSGD shares the gradients, whilst FedAvg shares the model weights. This means that an adversary could use *gradient inversion* to see the information shared through the gradients by the FedSGD algorithm [16]. Therefore, FedAvg offers a more secure method compared to FedSGD.

## 5.2 Robustness in Federated Learning

To assess the robustness in federated learning, we conducted two experiments, which are listed in Table 58. We wanted to see how FedAvg performed compared to robust federated aggregation (RFA) when the data had been poisoned. To reiterate, we sampled random values in the interval  $[20, 40]$ , labeled them as *Normal* beats, and inserted the values into client 1. The values of the other clients were in the interval  $[0, 1]$ , making client 1 an outlier. The values for client 1 were 20 to 40 times larger than the values of the other clients, causing the updates to become much larger. This approach is known as static data poisoning, and is explained in greater detail in Section 2.4.2.

Overview of the Experiments	
FedAvg with Static Data Poisoning:	Section 4.3.1
RFA with Static Data Poisoning:	Section 4.4.1

Table 58: The results discussed in Section 5.2

First, we tested the ANN model trained with FedAvg on the unbalanced data distribution. As described in Section 2.4, FedAvg uses the arithmetic mean to aggregate the updates from the clients. This method is not robust because the arithmetic mean is easily affected by outlier values. The results obtained after testing FedAvg on poisoned data, support the fact that FedAvg is not a robust method. This can be seen in the confusion matrix illustrated in Figure 48, where the model almost exclusively classified *Normal* beats, which is precisely the label that was used on the poisoned data. These classifications can be explained by the updates received from the poisoned client, because these updates were much larger than the updates from the other clients. Since FedAvg was used to aggregate the updates, and the updates from the outlier client were large, the global model went from having small weights to large weights. This shift is illustrated in Figure 11. Table 23 shows the metrics obtained after using FedAvg on poisoned data. In this

table, one can observe that the training loss is NaN (not a number). This is a result of *exploding gradients*, which is a phenomenon that can occur in deep neural networks when they have weight values larger than 1.0. In this experiment, the weight values became so large that the error gradient used to update the weights of the model overflowed, causing an undefined training loss. Having an undefined training loss entails that the model stops learning from the data, resulting in a broken federated averaging process. The results in Section 4.3.1 illustrates the impact of FedAvg being a non-robust method, since we ended up with a corrupt model suffering from gradient explosion, and classifying almost the whole dataset as *Normal* beats.

In order to increase robustness, we implemented the robust federated aggregation (RFA) algorithm, and tested the CNN model on the unbalanced data described in Section 4.3.1. To reiterate, the RFA algorithm uses the geometric median instead of the arithmetic mean to aggregate the client updates. The geometric median is a more robust method compared to the arithmetic mean, and is not as inclined to converge towards outliers. RFA and geometric median are explained in greater detail in Section 2.4.3. After replacing FedAvg with RFA to aggregate the updates from the clients, the model became robust. This can be seen in Table 37, where the training loss is no longer NaN. Thus, the model does not suffer from gradient explosion. The confusion matrix in Figure 55 displays a clear diagonal, showing that the model trained with RFA performed well on every class. Even though data poisoning was present in this experiment, we still managed to get a test accuracy of 92.0%. This is due to the fact that RFA gives low priority to outlier clients. In addition, the training time for RFA was 501 seconds, which was 29 seconds faster than the training time for FedAvg. This could be a consequence of the model trained with FedAvg overflowing, since it had to calculate error gradient with larger weights. This experiment illustrated how RFA increased robustness in federated learning, and how this aggregation algorithm can protect the model from static data poisoning.

From the results, one can observe that there is a significant benefit to using RFA compared to FedAvg when doing federated learning with poisoned data. First, the model trained with RFA did not suffer from gradient explosion, and unlike FedAvg, the federated learning process was not broken. This is an important result, because when using RFA, an adversary cannot use static data poisoning or another type of corruption (see Section 2.4.2) to destroy the federated learning process. In the results from the experiments, RFA was robust against static data poisoning, which is coherent with the related work described in Section 2.4.2. Furthermore, the model trained with RFA was relatively well-performing, both in terms of being able to classify on the test dataset, and the training time. Unlike the model trained with FedAvg, which classified the *Normal* class on almost the whole test dataset, the model trained with RFA was able to make good classifications in all the 5 ECG classes, even though the data was severely manipulated with values 20 to 40 times larger than the ordinary datapoints.

In summary, RFA increased security in the federated learning process, as it was able to make the model more robust against data poisoning attacks. However, the aggregation method can also have unintentional consequences if a non-malicious client possesses outlier data. In this situation, the RFA algorithm would disregard the non-malicious client that holds the outlier data in the sense that the updates of this client would not be weighted as much as the other clients' updates. In turn, this can cause a decrease in model performance for this particular client. However, in doing this, RFA preserves a well-performing model for the majority of the clients.

### 5.3 Model Extraction

In order to further evaluate the privacy issues in federated learning, we conducted an experiment where we attempted to simulate a model extraction attack on the softmax regression model. To make it simple to identify the training data of one specific client, we manipulated the distribution of the data in such a way that client 1 had low variance within its own local data, and high variance from the data of the other clients. This experiment is described in greater detail in Section 4.3.3.

The results of this experiment illustrate the weight updates that were collected whilst training the softmax regression model with FedAvg. In Figure 53, one can observe the weights of the 5 participating clients for

the *Normal* class. In this figure it becomes clear that the weights belonging to client 1 differs vastly from the weights of client 2 through 5. These results are interesting as one can easily identify the training data owned by client 1. Figure 54 shows similar weights for all clients within the *Supra Ventricular* class. As previously mentioned, client 1 was only trained using examples from the *Normal* class, while the other clients had only seen a few training examples from this class. This means that when the clients had finished training their local model and sent their updates to the central server to be aggregated, the update from client 1 would significantly differ from the updates of the four other clients. When the clients download the global model again, all clients, including client 1 would possess the same model weights. However, due to client 1 only having examples from the *Normal* class, this client would train its local model and become specialized again. The weights illustrated in Figure 53, were retrieved after the last round of the federated averaging process, but before the aggregation step. By extracting the weights at this time in the training process, one can easily observe that client 1 possess outlier data. In addition, the curve has a strong resemblance to the *Normal* beats class displayed in the 2D-histogram in Figure 27. Both the first graph in Figure 53 and the first graph in Figure 27 have relatively low values compared to the other classes. This observation makes it probable that client 1 had many examples of the *Normal* beats class.

In Section 2.2, federated learning is described as having the privacy benefit of not sharing training data between the clients and the server. Instead, it trains the model locally at each client and aggregates the client-updates. This means that federated learning should in theory protect its clients from data leakages. However, as explained in Section 2.6, client-updates can contain privacy-sensitive information. If an attacker were to perform an inference attack such as the model extraction attack simulated in this experiment, the attacker can infer information about the training data by looking at the weights before aggregation. The results obtained during the model extraction attack described in this section, support the fact that one is able to extract useful information from only the client-updates, and that this threatens the security of the clients potentially sensitive data.

When performing the model extraction experiment, we used a softmax regression model to train the model. This was done because the experiment was designed to make it possible to display the extracted weights. In practice, one would most likely never use a softmax regression model on this type of learning task as the model is too simple to train a well-performing model. However, in this experiment we found the softmax regression model to be useful as it only consist of one layer, making it easier to interpret the weights from the model. It is easier to interpret a softmax regression model compared to deep neural networks, because deep neural networks create a very high dimensional representations of the training data, thus reducing interpretability.

## 5.4 Differential Privacy

In Sections 5.1.2 and 5.3, we discussed two security issues in federated learning that both threaten the privacy of the participating clients. These privacy concerns are unfortunate, and in an attempt to increase the privacy when training models with federated learning, we conducted experiments in relation to both memorization and model extraction using differential privacy. Differential privacy provides privacy in the sense that it is not possible to determine if a client is present in a specific training round. This is accomplished by clipping the updates received from the clients, and by adding noise to the aggregated weights. First, we will observe how the model performance was affected by applying differential privacy in the federated learning process. Table 59 provides a section-overview of the experiments used in the discussion regarding differential privacy.

Overview of the Experiments	
Memorization:	Section 4.3.2
Model Extraction:	Section 4.3.3
Differential Privacy:	Section 4.4.2

Table 59: The results discussed in Section 5.4

### 5.4.1 Model performance

To assess how the model performance was affected by using differential privacy in federated learning, we conducted two experiments. The results of these experiments can be observed in Section 4.4.2. We wanted to compare the performance of the differentially-private FedAvg (DP-FedAvg) algorithm with the standard FedAvg algorithm. Therefore, we trained the ANN and the CNN model with DP-FedAvg. The differential privacy parameters for these experiments can be viewed in Tables 40 and 43. These privacy parameters were tuned according to how the validation accuracy changed when the noise multiplier and clipping norm were increased or decreased. Delta ( $\delta$ ) was defined as  $1/n$ , where  $n = 100000$  is the number of training points. After running the DP-FedAvg algorithms, we found that the test accuracy of the ANN model was 69.3%, and the test accuracy of the CNN model was 64.8%. These metrics can be viewed in Tables 41 and 44. Furthermore, the F1-scores illustrated Tables 42 and 45, along with the confusion matrices displayed in Figures 56 and 58, indicate that the models performed best on the *Unknown* class. This can be explained by the addition of noise, since the ECG recordings in this class do not follow a pattern. Furthermore, when using DP-FedAvg the client-updates are clipped according to the clipping norm. This can result in information about the data being lost, and can further explain the decrease in accuracy when training with DP-FedAvg. The training times for the experiment using DP-FedAvg is fairly similar to the training times when training with FedAvg. This makes sense as adding noise and clipping weights are considered cheap operations, as described in Section 2.6.

Training with DP-FedAvg requires the selection of how much noise to add to the aggregated updates, and how much the updates from the clients should be clipped. Therefore, a trade-off has to be made between model performance and privacy. The relationship between the amount of noise and model performance is inversely proportional, and the same is also true for how much the weight updates are regularized. The moments accountant displayed in Figures 57 and 59 give a cumulative privacy loss of  $\epsilon \approx 8.8$ , which is a good value for the privacy loss. To reiterate,  $\epsilon$  is a metric for measuring how strict the privacy is. The smaller  $\epsilon$  is, the better the privacy is. This is explained in greater detail in Section 2.6.1.

### 5.4.2 Memorization

In Section 5.1.2 the issue of memorization in machine learning models was discussed. This section revealed that memorization poses a significant threat in federated learning as well as in centralized learning. This was discovered through a series of experiments using both centralized and federated learning where the number of training examples within the *Unknown* class in the dataset had been reduced. The results of these experiments are shown in Section 4.3.2. Due to the high degree of memorization in federated learning, and the possible threat of inference attacks utilizing this memorization, we wanted to explore if differential privacy could decrease memorization in different models using federated learning. With respect to this, we conducted an experiment using the DP-FedAvg algorithm while forcing memorization in the model by reducing the number of training examples in the *Unknown* class. This experiment was performed on both the ANN model and the CNN model, and the training configuration and results can be observed in Section 4.4.2.

The results of this experiment show a significant reduction of memorization in both the ANN model and the CNN model when trained with federated averaging and differential privacy. Figure 60 illustrates the confusion matrix for the experiment performed on the ANN model, and it shows a complete reduction in memorization when training with DP-FedAvg as the value for the *Unknown* class is 0.00. Figure 52 illustrates the confusion matrix for the same experiment, but without differential privacy. The value for the *Unknown* class in this confusion matrix is 0.71. This means that by using DP-FedAvg instead of FedAvg, one has reduced memorization for the *Unknown* class by 0.71 in the ANN model. Moreover, Figure 61 describes the confusion matrix for the forced memorization experiment using DP-FedAvg on the CNN model. This figure shows a value of 0.24 for the *Unknown class* which is a reduction of 0.54 from the confusion matrix describing the same experiment, but without differential privacy. The confusion matrix for the forced memorization experiment without DP is illustrated in Figure 50.

The results obtained during this experiment, support the related work described in Section 2.6. In Section 2.6, it is explained that differential privacy tries to solve the problem of memorization of a particular client’s data. This is done by clipping the updates received from the clients, and by adding noise to the aggregated weights. Adding noise will change the values of the weights which are responsible for the memorization. Clipping the updates will cause the weights to become normalized, and this will make it difficult to extract outlier data from the model. This will lead to a decrease in memorization. However, in order to decrease memorization, one has to sacrifice model performance in the non-outlier data as the noise and clipping will negatively impact the pattern recognition in the model. When trying to decrease the issue of memorization, one has to adjust the noise multiplier and the clipping norm in order to find a balance between model performance and memorization.

### 5.4.3 Model Extraction

Section 5.3 discusses the experiments conducted in respect to the inference attack known as model extraction. In this experiment the data distribution was manipulated in such a way that one client had low variance within its own local data and high variance from the data of the other clients. Furthermore, the softmax regression model was trained and the weights of the 5 participating clients were extracted before aggregating the updates from the final round of local training. As discussed in Section 5.3, client updates can contain privacy-sensitive information, and by executing a model extraction attack one can infer such information which threatens the privacy of the participating clients. We wanted to explore how adding differential privacy to this situation could affect the interpretability of the extracted weights. Therefore, we executed the model extraction experiment with differential privacy. The results of this experiment can be observed in Section 4.4.3.

The results of the model extraction experiment with differential privacy consist of the extracted weights from all 5 clients, from both the *Normal* and the *Supra Ventricular* class. Figure 62 illustrates the weights for the *Normal* class. These weights consist of more noise compared to the extracted weights without differential privacy shown in Figure 53. The noise in Figure 62 makes the extracted weights less interpretable. However, one can still observe that client 1 stands out, but it is more difficult to see the presence of *Normal* beats from the weights. In Figure 63 one can observe the weights for the *Supra Ventricular* class. These weights also consist of more noise than the weights extracted in the experiment without differential privacy. From these results we know that differential privacy would help make the weights that represent the data, less interpretable. However, one has to increase the noise and clipping significantly to eliminate the chance of inferring privacy-sensitive information concerning the training data.

## 5.5 Federated Learning with Homomorphic Encryption

To further enhance privacy in federated learning, we conducted an experiment where we trained a softmax regressor using federated learning with homomorphic encryption (HE). The results of this experiment are presented in Section 4.4.4. Homomorphic encryption is an encryption scheme where one can perform operations on encrypted values, and receive mathematically correct results after decrypting the values. Thus, federated learning with homomorphic encryption is running the FedAvg algorithm with encrypted model parameters, and decrypting at the end of the whole federated learning process.

When performing federated learning with HE, we decided to use the smallest model in our model ensemble, the softmax regression model (see Section 3.2.4.1). Due to our discoveries concerning training time when doing federated learning with homomorphic encryption in Section 2.7, we did not run as many global epochs and client epochs in this experiment compared to the other experiments. In Section 2.7.3, it was mentioned that homomorphic encryption encrypts the weights of the model as large integers, making computations on encrypted values inefficient. This computational inefficiency was evident after conducting the experiment. Table 55 shows that the training time after 3 rounds of encrypted FedAvg was 796 seconds, even though we used a small model and only ran 3 global epochs. When we ran the FedAvg algorithm with 15 epochs on the CNN model without homomorphic encryption, we obtained a training time of 687 seconds. Furthermore, in Table 56, the size of an encrypted weight is described. This weight has the size

of 300 bytes, which was 12.5 times larger than the length of the unencrypted weight. Doing backpropagation with encrypted values of this size explains the long training time for performing federated learning with homomorphic encryption.

In the experiment regarding federated with learning HE, we used a softmax regression model. In Table 55, one can observe that we did not obtain a good test accuracy using this model because it was too simple. However, as explained in Section 4.4.4, the purpose of the experiment was not to train a well-performing model, but to demonstrate training with federated learning combined with homomorphic encryption. Figure 64 shows a reduction in validation loss with respect to the number of epochs. This result shows that we were able to optimize the model with federated learning combined with homomorphic encryption.

When doing federated learning with HE, the model parameters remain encrypted during the entire federated learning process. This means that from the moment the central server initializes the federated learning process, all mathematical operations will be done on encrypted values. This means that the clients will perform gradient descent on encrypted values, and that the aggregation will be done on encrypted values. Since all the model parameters are encrypted, it is more challenging to execute model extraction, which is explained in Section 5.3. This is because an adversary would not be able to see the actual model parameters, only the encrypted values, making it more difficult to infer the training data. Because of the poor training time of federated learning with homomorphic encryption, one could argue that homomorphic encryption is unnecessary if a secure protocol, such as TLS, is being used. However, even if the connection between the server and the clients is secure, the model would still be vulnerable. When using a secure connection, a client would still be able to read the weights of the model. This would not be ideal if the model were to leak information about the other clients to a malicious client. Homomorphic encryption remedies this problem by making the model unreadable to the clients or other adversaries. One potential disadvantage of doing federated learning with homomorphic encryption is that the server has to be trustworthy, because the server is the only entity which is able to encrypt and decrypt the weights.

## 5.6 Summary

As described in Section 2.2, federated learning is a relatively new machine learning approach which emerged as a result of an increased focus on privacy and security within machine learning. Federated learning offers an entirely new way to approach a machine learning task in the sense that it allows clients to keep their own data, and train a model locally which is later aggregated to a central server. In allowing clients to keep their data private, the approach greatly reduces the privacy concerns linked to how centralized learning stores data used in training. In centralized learning, data used in training is stored on a central server which means that the clients have to communicate their raw training data to server. In federated learning, the raw training data is never communicated. Instead, the clients train the downloaded global model on their local training data and only communicates the updated model weights which are deleted after aggregation. In doing this, federated learning reduces the amount of information that needs to be transmitted and stored, therefore increasing the privacy of its clients. However, the increase in privacy comes at the expense of a reduction in model performance.

As discussed in Section 5.1.1, the results of the federated learning and centralized learning experiments show a slight benefit to training the CNN model with centralized learning. This is can be a result of the aggregation process in federated learning which occurs after each global epoch. In the aggregation process the arithmetic mean is calculated from the clients updates, resulting in an updated model that could potentially have lost some information in regards to the updates. This can result in an slight decrease in model performance compared to centralized learning. In addition, the discussion in Section 5.1.1 reveals that federated learning has a longer training time than centralized learning. In an actual implementation of federated learning, the training times obtained during the experiments would have been far higher as the clients used in this report's experiments were initiated locally. The reason why one would experience a higher training time using federated learning, is mainly because centralized learning only has to perform  $E$  epochs, while federated learning has to perform  $T \cdot E$  epochs, where  $T$  is the number of global epochs. In light of this, one can observe that one has to be willing to sacrifice some performance in terms of efficiency and accuracy in order to reap the privacy benefits that federated learning offers.

Even though federated learning offers privacy benefits due to it not communicating raw training data, federated learning is still vulnerable and has its fair share of privacy issues. One of these issues was discussed in Section 5.1.2, and concerns the issue of memorization in both federated and centralized learning. From the experiments executed in regards to memorization, it was discovered that the issue of memorization was prominent in both federated learning and centralized learning. However, it was discovered that memorization occurred less in smaller models such as the ANN model compared to bigger models such as the CNN model. In addition, it was also observed that memorization was less present in the CNN model when trained with federated averaging which could be explained by the averaging of the weights executed during the aggregation step.

Another privacy issue in federated learning was discussed in Section 5.2, and it concerned model robustness. Federated averaging was not robust against corrupted clients, since the arithmetic mean tends to shift towards outliers. From the experiments conducted, one could observe that the federated averaging process broke due to corrupting the data of one client. This further resulted in extremely poor model performance. To remedy this problem, robust federated aggregation (RFA) was applied which uses the geometric median with calls to a secure average oracle instead of the arithmetic mean. After applying the RFA algorithm to aggregate the client updates, a relatively well-performing model with a similar training time to federated averaging, was obtained. Even though RFA greatly increases the robustness of the model, it can lead to poor model performance on non-malicious outliers as these clients are weighted less in the aggregation process.

In Section 5.3, the threat of model extraction attacks in federated learning was discussed. In the experiments conducted in regards to model extraction, it was found that it was possible to extract the weights of the participating clients before aggregation. From these weights one could also infer what kind of data a particular client had. This was a concerning result as an attacker that performs such an attack on federated learning, would be able to infer privacy-sensitive information about the participating client's data. This would pose a significant threat to the privacy of the clients.

In order to remedy the privacy issues of memorization and weight extraction attacks, one can utilize differential privacy. In Section 5.4, differential privacy in federated learning was discussed. It was discovered that differential privacy can help decrease the amount of memorization in both the CNN model and the ANN model using the DP-FedAvg algorithm. Furthermore, it was observed that applying differential privacy to the training process would complicate inferring information from the extracted weights. Differential privacy aims to make it impossible to determine whether a specific client has participated in a training round by adding noise and clipping the weights. In applying differential privacy, one will be able to enhance privacy in federated learning, but it will also compromise the training process and in turn the model performance. From the results discussed in Section 5.4.1, one can observe that the model performance decreases significantly when applying differential privacy to the FedAvg algorithm.

In Section 5.5, the use of homomorphic encryption in a federated learning environment was discussed. Homomorphic encryption allows operations between encrypted values, thus enabling machine learning on encrypted values. In the context of federated learning, we were able to perform the whole process using encrypted values, from model distribution to model aggregation. While applying homomorphic encryption to federated learning has the advantage of making the model updates secure, the computational complexity of doing so is high. We found that the training time of applying homomorphic encryption to federated learning was relatively high, despite only running few epochs. This is because homomorphic encryption operates with large integers, in addition to time spent encrypting the values. Moreover, if one were to use a secure connection such as TLS when communicating updates between the server and the clients, the advantage to using homomorphic encryption would decrease. However, using homomorphic encryption makes the model unreadable to all clients which is an added benefit as a client would not be able to infer any information about other clients. A potential disadvantage to utilizing homomorphic encryption is that the server has to be trustworthy as the server is the only entity which is able to encrypt and decrypt the weights.

## 6 Conclusion

The main focus of this report was to explore if federated learning could be a viable alternative to centralized learning in terms of privacy, security and model performance. In order to determine this, two research questions were formulated. The first research question presented in the report was:

*How does federated learning increase privacy when applied to decentralized data?*

In Chapter 5, the federated learning approach was discussed with respect to privacy. The discussion implied that federated learning increases privacy when applied to decentralized data because the approach allows data-owners to participate in training models without having to share their data with any third parties. The information that is being communicated between the server and the clients is limited to weight updates, meaning that no raw data is ever sent. Furthermore, when utilizing federated learning, the updates from the clients are ephemeral. This means that the server does not store any of the clients weight updates after aggregation. The limited data communication and data storage in federated learning makes for an approach which is less vulnerable to attacks, thus increasing the privacy of the decentralized data. This is a huge advantage compared to centralized learning where the raw data has to be imported from the various data-owners. If the data is highly privacy-sensitive, this can be unsafe because an adversary could intercept the raw training data. The nature of federated learning increases the privacy of the decentralized data because it limits the amount of information being communicated and stored.

By itself, federated learning provides an increase in privacy due to it limiting the information that is being communicated. However, the approach is not by itself resilient to all threats and attacks as discussed in Chapter 5. To explore how a more secure federated learning process could be achieved, a second research question was formulated:

*How can different methods enhance security in federated learning environments, and how do these methods affect model performance?*

Chapter 5 discusses three different statistical and cryptographic methods for enhancing security in federated learning. The first method explored was the robust federated aggregation algorithm. This method proved to enhance the security of the model in the sense that it made the model much more robust against corrupted clients. When testing the algorithm with data poisoning, the algorithm managed to train a well-performing model by weighting the corrupted client less than the other clients. The model performance obtained when using robust federated aggregation without any corrupted clients is slightly worse than with the federated averaging algorithm, but the algorithm provides a vast security benefit in the federated learning process. Second, differential privacy was explored. Differential privacy proved to be an effective method for reducing memorization in models and preventing adversaries from inferring useful information from weight updates. Differential privacy is a method that adds noise and clips weights in order to disguise which clients are participating in a training round of federated learning. When using differential privacy, one has the ability to adjust the clipping norm and the noise multiplier with respect to level of privacy one wishes to achieve. If the degree of privacy is large, the model performance suffers. However, with the correct values the method can provide a huge security benefit while still training a well-performing model. Finally, homomorphic encryption was tested. Federated learning with homomorphic encryption allowed training with an encrypted model, from the model distribution to the aggregation process. This method caused a severe increase in training time, but made the entire federated learning process secure against third-party attacks. The benefit of utilizing this method decreases if one already uses a secure connection, but can still prevent malicious clients from inferring information about other clients. This is because when using federated learning with homomorphic encryption the weights can only be decrypted by the server.

Federated learning can be a viable alternative to centralized learning in terms of privacy, security and model performance. The federated averaging algorithm applied to privacy-sensitive, decentralized data, provides a great privacy benefit while still training a well-performing model. In addition, one can utilize federated learning along with numerous different methods in order to further enhance security within the



federated learning process. However, there is always a trade-off that needs to be made between security and model performance. In conclusion, federated learning is especially well suited for environments that requires a great deal of privacy and security. In an environment such as the one Infiniwell works with, the algorithm appears to be suitable as they wish to provide patients with a more secure machine learning solution than one can achieve with centralized learning.

## 7 Future Work

Federated learning is an approach that emerged a few years ago, and the research done within the field is still limited. This report focused on exploring how the approach increases privacy, and how one could combine it with other methods to further enhance the security of the federated learning process. While the research regarding privacy and security within federated learning provided in this report is relatively thorough, there is still more that can be explored.

First, homomorphic encryption (HE) could have been tested more thoroughly. As mentioned in Section 2.7, fully homomorphic encryption (FHE) allows all operations on encrypted values. The research presented in this report, only covered partial homomorphic encryption (PHE). PHE only supports either addition and multiplication of encrypted values. Therefore, in order to get a more complete solution one could replace PHE with FHE, which would allow for testing of more complex models with homomorphic encryption. As seen in the results in Section 4.4.4, federated learning with HE did not perform well. This was because federated learning with HE had extensive training times, which meant that we were not able to train any well-performing models using HE. Therefore, it would also be wise to do more research on, and implement more efficient algorithms for performing homomorphic encryption in federated learning environments. An example is the *BatchCrypt* algorithm [42], which uses techniques such as quantization to reduce the encryption overhead caused by HE.

As mentioned in Section 2.4.2, the robust federated aggregation (RFA) algorithm is not resilient against byzantine attacks. This is because an adversary is able to modify the aggregation method. In April 2021, a new aggregation method called *FedCom* was introduced [44]. This method has proved to be robust against byzantine attacks. This method requires each client to submit a data commitment, and uses the Wasserstein distance among the data commitments to evaluate the behaviour of different local models. If the algorithm discovers unusual behaviour it will prevent the ongoing model poisoning attack. By implementing this aggregation algorithm, the federated learning process could become even more robust against potential attacks.

In Section 2.2.1.3, the secure aggregation algorithm was presented. This method is considered more secure against inference attacks as it allows the participating clients to act as parties in the SMC protocol and calculate the aggregated weights between each other before sending the aggregated weights to the server. This means that the server will only see the aggregated weights, and update the global model based on this value. It would be interesting to observe how this aggregation method would secure the federated learning process compared to other aggregation methods such as RFA and FedAvg.

The methods explored in this report, have been researched individually. Therefore, it would be fascinating to observe how the methods would work when combined. For example, the robust federated aggregation algorithm could be combined with differential privacy to obtain a federated learning process which both keeps the data of the clients private, and makes the model more robust against attacks. In examining different combinations of the security measurement methods explored, one could discover a federated learning process that would provide sufficient protection against potential threats and attacks while still producing a well-performing model. In addition, one could implement downstream and upstream compression in order to provide a more communication-efficient federated learning process.

Finally, the implementation of federated learning in this report used TensorFlow Federated (TFF) for most of the algorithms. The implementation provided is not a production-ready pipeline because the focus of the bachelor's thesis was to research federated learning and related methods, in a theoretical framework. In addition, TFF is not currently meant to be used in a production setting. Therefore, it would be beneficial for both the research and Infiniwell to attempt to implement a federated learning environment which could be deployed. This would allow for a study of the differences between local simulations and real-world implementations of federated learning. Moreover, this could have provided Infiniwell with a real-world federated learning solution that would be suitable for their work, making their current pipeline more secure.

## 8 Broader Impact

The research presented in this report aims to gain insight into how federated learning would work in an environment that requires a high level of data privacy. In addition, the report explores privacy vulnerabilities in federated learning, and presents suitable methods to remedy these. However, the research conducted and presented could potentially have an unintended impact. First, the experiments performed with respect to attacks in federated learning could potentially be utilized maliciously. In this report, the aim of performing such attacks was purely to establish what impact they would have on the federated learning process. Nevertheless, it is possible to utilize the attack methods presented in this report as a basis for performing attacks with malicious intent. Moreover, some of the privacy-preserving methods described in this report are extremely computational heavy, e.g. homomorphic encryption. This could potentially have an harmful impact on the environment because the methods require large amounts of power to be computed.

## Bibliography

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. <https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>, 2012. Pages 1-2. Accessed 19. March 2021.
- [2] J. Carreira and A. Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset? <https://arxiv.org/abs/1705.07750>, 2017. Accessed 13. April 2021.
- [3] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context? <https://arxiv.org/abs/1901.02860>, 2019. Accessed 13. April 2021.
- [4] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Aurelio Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large Scale Distributed Deep Networks. <https://papers.nips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf>, 2012. Pages 1-4. Accessed 19. March 2021.
- [5] T. Ben-Nun and T. Hoeffler. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. <https://arxiv.org/abs/1802.09941>, 2018. Pages 1-43. Accessed 19. March 2021.
- [6] T. Chilimbi, Y. Suzue, J. Apacible, K. Kalyanaraman, and Microsoft Research. Project Adam: Building an Efficient and Scalable Deep Learning Training System. <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-chilimbi.pdf>, 2014. Pages 571-582. Accessed 19. March 2021.
- [7] M. Li, D. G. Andersen, J. Woo Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su. Scaling Distributed Machine Learning with the Parameter Server. <http://web.eecs.umich.edu/~mosharaf/Readings/Parameter-Server.pdf>, 2014. Pages 1-15. Accessed 22. March 2021.
- [8] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. <https://arxiv.org/abs/1106.5730>, 2011. Pages 1-8. Accessed 22. March 2021.
- [9] Q. Ho, J. Cipar, H. Cui, J. Kyu Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. [http://www.cs.cmu.edu/~seunghak/SSPTable\\_NIPS2013.pdf](http://www.cs.cmu.edu/~seunghak/SSPTable_NIPS2013.pdf), 2013. Pages 1-8. Accessed 22. March 2021.
- [10] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya. GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent. <https://arxiv.org/abs/1803.05880>, 2018. Accessed 22. March 2021.
- [11] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. <https://arxiv.org/abs/1503.02531>, 2015. Accessed 22. March 2021.
- [12] J. Konecny, H. Brendan McMahan, and D. Ramage. Federated Optimization: Distributed Optimization Beyond the Datacenter. <https://arxiv.org/abs/1511.03575>, 2015. Accessed 19. March 2021.
- [13] C. Briggs, Z. Fan, and P. Andras. A Review of Privacy-preserving Federated Learning for the Internet-of-Things. <https://arxiv.org/abs/2004.11794>, 2020. Page 6. Accessed 19. March 2021.
- [14] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. <https://arxiv.org/abs/1602.05629v3>, 2016. Accessed 19. March 2021.
- [15] V. Felbab, P. Kiss, and T. Horváth. Optimization in Federated Learning. <http://ceur-ws.org/Vol-2473/paper13.pdf>, 2019. Page 2. Accessed 22. March 2021.

- [16] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting Gradients - How easy is it to break privacy in federated learning? <https://arxiv.org/abs/2003.14053>, 2020. Accessed 05. May 2021.
- [17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. Brendan McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. <https://dl.acm.org/doi/pdf/10.1145/3133956.3133982>, 2017. Pages 1175-1191. Accessed 26. March 2021.
- [18] Keith B Frikken. Secure Multiparty Computation (SMC). [https://doi.org/10.1007/978-1-4419-5906-5\\_766](https://doi.org/10.1007/978-1-4419-5906-5_766), 2011. Accessed 02. April 2021.
- [19] M. Asad, A. Moustafa, T. Ito, and M. Aslam. Evaluating the Communication Efficiency in Federated Learning Algorithms. <https://arxiv.org/abs/2004.02738>, 2020. Pages 1-2. Accessed 25. March 2021.
- [20] F. Sattler, S. Wiedemann, K. Müller, and W. Samek. Robust and Communication-Efficient Federated Learning from Non-IID Data. <https://arxiv.org/abs/1903.02891>, 2019. Page 3. Accessed 25. March 2021.
- [21] L. Lyu, H. Yu, and Q. Yang. Threats to Federated Learning: A Survey. <https://arxiv.org/abs/2003.02133>, 2020. Pages 1-4. Accessed 26. March 2021.
- [22] C. Wu, S. Zhu, X. Yang, and P. Mitra. Mitigating Backdoor Attacks in Federated Learning. <https://arxiv.org/abs/2011.01767>, 2020. Page 1. Accessed 26. March 2021.
- [23] V. Tolpegin, S. Truex, M. Emre Gursoy, and L. Liu. Data Poisoning Attacks Against Federated Learning Systems. <https://arxiv.org/abs/2007.08432>, 2020. Page 2. Accessed 26. March 2021.
- [24] L. Lyu, H. Yu, X. Ma, L. Sun, J. Zhao, Q. Yang, and P. S. Yu. Privacy and Robustness in Federated Learning: Attacks and Defenses. <https://arxiv.org/abs/2012.06337>, 2020. Accessed 2. April 2021.
- [25] D. Chen, Y. Zhang, N. Yu, and M. Fritz. GAN-Leaks: A Taxonomy of Membership Inference Attacks against Generative Models. <https://arxiv.org/abs/1909.03935>, 2019. Pages 2-3. Accessed 4. April 2021.
- [26] Jie Feng, Xueliang Feng, Jiantong Chen, Xianghai Cao, Xiangrong Zhang, Licheng Jiao, and Tao Yu. Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification. <https://www.mdpi.com/2072-4292/12/7/1149>, 2020. Accessed 4. April 2021.
- [27] K. Pillutla, S. M. Kakade, and Z. Harchaoui. Robust Aggregation for Federated Learning. <https://arxiv.org/abs/1912.13445>, 2019. Pages 5-8. Accessed 26. March 2021.
- [28] H. P. Lopuhaa and P. J. Rousseeuw. Breakdown Points of Affine Equivariant Estimators of Multivariate Location and Covariance Matrices. <https://projecteuclid.org/journals/annals-of-statistics/volume-19/issue-1/Breakdown-Points-of-Affine-Equivariant-Estimators-of-Multivariate-Location-and-10.1214/aos/1176347978.full>, 1991. Pages 229-248. Accessed 28. March 2021.
- [29] A. S. Nemirovsky and D. B. Yudin. Problem Complexity and Method Efficiency in Optimization. <https://doi.org/10.1137/1027074>, 1983. Accessed 28. March 2021.
- [30] Endre Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. Tohoku Mathematical Journal, 1937. Accessed 28. March 2021.
- [31] F. Bach, G. Maillard, and N. Brosse. Statistical machine learning and convex optimization - Lecture 2. <https://www.di.ens.fr/~fbach/orsay2016/lecture2.pdf>, 2016. Page 1. Accessed 29. March 2021.

- [32] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting Training Data from Large Language Models. <https://arxiv.org/abs/2012.07805>, 2020. Pages 1-4. Accessed 6. April 2021.
- [33] S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome. What Do Compressed Deep Neural Networks Forget? <https://arxiv.org/abs/1911.05248>, 2019. Accessed 06. April 2021.
- [34] O. Thakkar, S. Ramaswamy, R. Mathews, and F. Beaufays. Understanding Unintended Memorization in Federated Learning. <https://arxiv.org/abs/2006.07490>, 2020. Page 7. Accessed 7. April 2021.
- [35] H. Brendan McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning Differentially Private Recurrent Language Models. <https://arxiv.org/abs/1710.06963>, 2017. Pages 1-6. Accessed 30. March 2021.
- [36] C. Dwork and A. Smith. Differential Privacy for Statistics: What we Know and What we Want to Learn. <https://cs-people.bu.edu/ads22/pubs/2008/DworkSmith.pdf>, 2009. Pages 1-5. Accessed 31. March 2021.
- [37] M. Abadi, A. Chu, I. Goodfellow, H. Brendan McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep Learning with Differential Privacy. <https://arxiv.org/abs/1607.00133>, 2016. Pages 3-4. Accessed 31. March 2021.
- [38] G. Andrew, O. Thakkar, H. Brendan McMahan, and S. Ramaswamy. Differentially Private Learning with Adaptive Clipping. <https://arxiv.org/abs/1905.03871>, 2019. Accessed 01. April 2021.
- [39] A. Acar, H. Aksu, A. Selcuk Uluagac, and M. Conti. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. <https://arxiv.org/abs/1704.03578>, 2017. Pages 4-8. Accessed 23. March 2021.
- [40] John B. Fraleigh. A First Course in Abstract Algebra, 7th Edition. **ISBN-13:** 978-0201763904, 2002. Accessed 13. January 2021.
- [41] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. [https://link.springer.com/chapter/10.1007/3-540-48910-X\\_16](https://link.springer.com/chapter/10.1007/3-540-48910-X_16), 1999. Accessed 24. March 2021.
- [42] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. <https://www.usenix.org/system/files/atc20-zhang-chengliang.pdf>, 2020. Pages 496-499. Accessed 25. March 2021.
- [43] A. Wrålsen. Vitenskapelig forankring av bacheloroppgaven: Del 1 , 2017. Accessed 05. April 2021.
- [44] B. Zhao, P. Sun, L. Fang, T. Wang, and K. Jiang. FedCom: A Byzantine-Robust Local Model Aggregation Rule Using Data Commitment for Federated Learning. <https://arxiv.org/abs/2104.08020v1>, 2021. Accessed 29. April 2021.

## **Attachments**

**Attachment A** - Project Handbook

**Attachment B** - System Documentation

