Deep Leakage in Federated Learning: Understanding Privacy Vulnerabilities

Master's Thesis in Complex Adaptive Systems

CARL KRONQVIST
MALTE OLSSON

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis 2025

Deep Leakage in Federated Learning: Understanding Privacy Vulnerabilities

CARL KRONQVIST
MALTE OLSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Deep Leakage in Federated Learning: Understanding Privacy Vulnerabilities
CARL KRONQVIST
MALTE OLSSON

Supervisor: Håkan Warston
Examiner: Kristian Gustafsson, Department of Physics

Master's Thesis 2025
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Image restoriation by deep leakage from weights on a federated network.

Typeset in LaTeX
Printed/ Department of Physics
Gothenburg, Sweden 2025

Abstract

Deep Leakage attacks in Federated Learning have traditionally relied on FedSGD, yet real-world deployments commonly adopt FedAVG due to its reduced communication overhead. This study investigates the feasibility and limitations of executing DL attacks within a FedAVG setting. A custom FL framework was developed to support FedAVG and state-of-the-art DL techniques to operate on shared model weights instead of gradients. Experiments conducted using the CIFAR-10 dataset revealed that while DL attacks are possible under FedAVG, their success diminishes as local training (batch size and epochs) increases, due to degraded gradient approximations. Additionally, model initialization strategies, dataset size, and image resolution significantly impact reconstruction quality. These findings highlight critical trade-offs between privacy and performance in FL systems, emphasizing the need for cautious design choices in real-world applications.

Keywords: Federated Learning, FedAVG, Deep Leakage, Privacy Attack, Model Inversion, Gradient Approximation, Image Reconstruction, Robustness Analysis, Local Training, Data Leakage

Acknowledgements

Firstly, we want to express our deepest gratitude to our supervisors at Saab, Håkan Warston and Isaac Baglin, for their invaluable guidance and encouragement throughout the course of this thesis. Their expertise within the subject and willingness to help has had a great impact on the project outcomes. Moreover, we would like to thank Maria Stegberg and the X Innovation Lab at Saab Surveillance for supporting us in our everyday work.

Secondly, we wish to thank our examiner and supervisor at Chalmers, Kristian Gustavsson at the department of physics, for his feedback and faith in good music, as well as our opponent, Albin Steen, for helping us understand the difference between machine learning and artificial intelligence.

Finally, we wish to thank our families' for their unyielding support during our education at Chalmers, and lastly, H.M. the King.

Thank you!

Malte & Carl, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADAM	Adaptive Moment Estimation
CNN	Convolutional Neural Network
DL	Deep Leakage
DLG	Deep Leakage from Gradients
FedAVG	Federated Averaging
FedSGD	Federated Stochastic Gradient Descent
FL	Federated Learning
GD	Gradient Descent
IG	Inverting Gradients
IID	Identically and Independently Distributed
LPIPS	Learned Perceptual Image Patch Similarity
Lr	Learning Rate
ML	Machine Learning
MLP	Multi-Layered Perceptron
NN	Neural Network
Non-IID	Non-Identically and Independently Distributed
PSNR	Peak Signal-to-Noise Ratio
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SME	Surrogate Model Extension
SOTA	State-of-the-Art
SSIM	Structural Similarity Index Measure

Nomenclature

Below is the nomenclature of indices, sets, parameters, variables and functions that have been used throughout this thesis.

Indices

t	Index for time step
l	Index of layer in a neural network
i	Index for iterations (e.g., Surrogate Model Extension and Deep Leakage from Gradients updates)

Sets

$\mathcal{D}$	Dataset or collection of data points
$\mathcal{W}$	Set of weights in weight space

Parameters

E	Number of local training epochs per communication round
T	Total number of local gradient descent steps per round
B	Local batch size used by each client
N	Local dataset size (number of samples per client)
R	Number of communication rounds
C	Fraction of clients participating in each round
η	Learning rate
$\eta_{\hat{\boldsymbol{X}}}$	Learning rate for dummy data $\hat{\boldsymbol{X}}$ in Surrogate Model Extension
η_α	Learning rate for interpolation factor α in Surrogate Model Extension

λ	Regularization weight (e.g., for TV loss in Surrogate Model Extension and Inverting Gradients)
α	Interpolation parameter in weight averaging for Surrogate Model Extension

Variables

$\boldsymbol{x}$	Input vector
x	Scalar input
$\boldsymbol{X}$	Batch of input data
y	Ground truth label (single)
$\boldsymbol{y}$	Ground truth labels (batch)
$\hat{y}$	Predicted output (single)
$\hat{\boldsymbol{y}}$	Predicted output (batch)
w	Weight in a neuron
$\boldsymbol{w}$	Model weights (vector)
θ	Threshold (bias) in a neuron
$\boldsymbol{w}^0, \boldsymbol{w}^T$	Initial and final weights in a local training round
$\hat{\boldsymbol{X}}$	Dummy input data used for gradient inversion
$\hat{\boldsymbol{y}}$	Dummy label(s) used for gradient inversion

Functions

$\ell(\cdot)$	Local loss function
$f(\cdot)$	Mapping function represented by the model
$\mathcal{L}$	Reconstruction loss function in Deep Leakage
$\mathcal{L}_{SME}$	Reconstruction loss function in Surrogate Model Extension
$\mathcal{L}_{DLG}$	Reconstruction loss in Deep Leakage from Gradients
$\mathcal{L}_{IG}$	Reconstruction loss in Inverting Gradients
$\mathrm{TV}(\hat{\boldsymbol{x}})$	Total variation regularization for dummy input

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

The recent advancements in machine learning (ML) and artificial intelligence (AI) is driven by a dramatic increase in computational power from graphical processing units (GPUs), enabling larger models to be trained on larger datasets [9, 15]. This thesis is written in collaboration with Saab Surveillance. Saab Surveillance is a Swedish defence company that specializes in sensors and electronic warfare. The company wants to leverage the rapid technological progress of large and complex machine-learning models and computationally efficient hardware in its products.

A key factor of an ML model's performance is the amount of training data it sees. Generally, the more data, the better the performance [10, 11]. However, training only one instance of a model on a full corpus can lead to a number of issues; practically, storing all data in one place can be infeasible in many cases [17].

Consider an example of a model performing auto-completion for text writing in mobile phones utilizing only the different users' already written text for training. This would lead to all users' text being stored in a global corpus on a common storage unit. Not only would this require immense storage capacity, but it would also, more importantly, impose privacy and integrity risks. Furthermore, this becomes a bandwidth issue as the full corpus needs to be transferred from the local devices to the common storage unit [4].

Alternatively, each user could have its own instance of the model and only train it on its local data. This would mitigate the privacy and integrity risks but would give the users a much worse-performing model as the amount of data each model instance sees would be substantially smaller than for the global case. To address these issues, federated learning (FL) was proposed by McMahan et al. in 2017 [20]. FL is a privacy-preserving decentralized machine learning paradigm designed to collaboratively train models across multiple clients by exchanging gradients or weights to the server while private data is kept local. This scheme is illustrated in Figure 1.1 where the clients have their own local data partitions, performing local training and transmitting the resulting local updates. The server then aggregates these results and sends back global updates.

Nevertheless, research has revealed that the security of FL is compromised, as private training data can be recovered by an adversarial actor through a gradient inversion technique known as deep leakage (DL). In Figure 1.2, Zhu et al.[28] demonstrate how training data can be reconstructed using their attack deep leakage from gradients

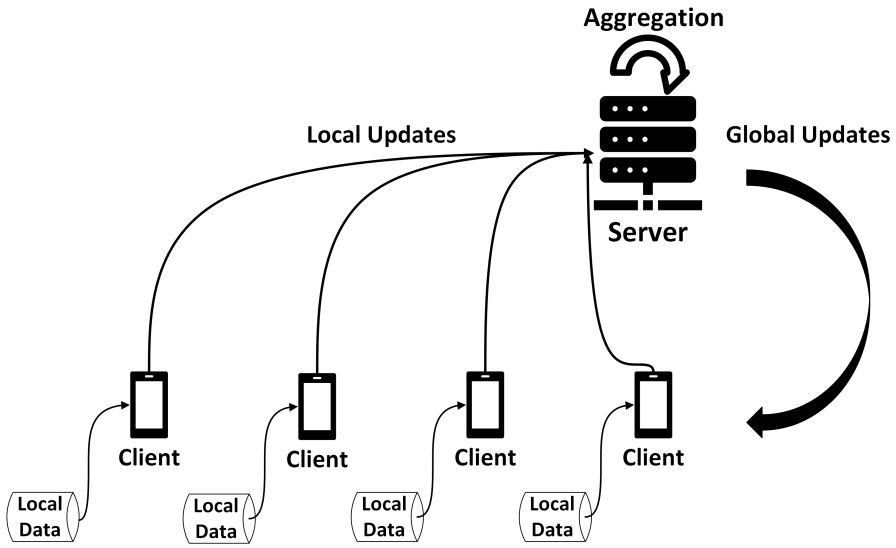

Figure 1.1: Instead of sending the local training data directly to the server, clients compute one or several training steps and sends the local updates in according to the FL scheme. For FedSGD, the local updates consists of gradients, and for FedAVG they instead consists of weights. After aggregation the server returns global updates which are identical for all clients.

(DLG). This vulnerability is significant as modern DL attacks can be successfully carried out on shared gradients trained on complex datasets with large batch sizes [7, 26]. With this, FL provides a false sense of security.

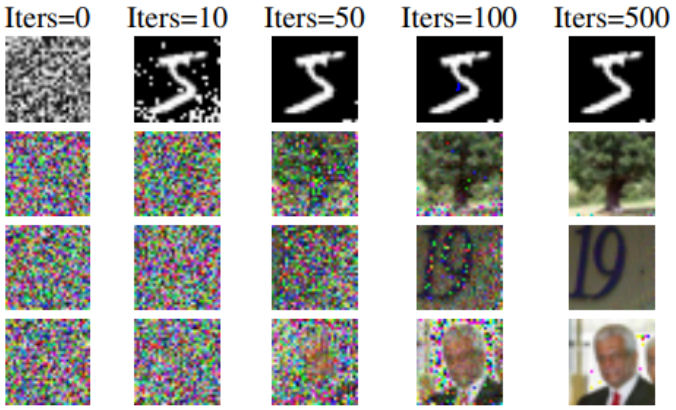

Figure 1.2: Iterative image reconstruction using Deep Leakage from Gradients (DLG), where the input is recovered from random noise by optimizing the difference between computed and stolen gradients(adapted from [28]).

In FL, federated stochastic gradient descent (FedSGD) and federated averaging (FedAVG) are two common approaches for aggregating the local updates from the clients. In FedSGD, clients compute and send gradients to the server for averaging, while in FedAVG, clients perform several local updates before sending their model

weights for averaging, reducing communication and improving efficiency. FedAVG and FedSGD are theoretically identical when only a single batch is used to train locally. Further details about FL, FedSGD, FedAVG and their intersection will be presented in Chapter 2.

1.1 Purpose & Objective

DL attacks tend to assume a FedSGD approach. However, in real-world applications of FL, FedAVG is preferred because of its low communication cost. The goal of this project is to design an FL codebase which performs DL attacks within a FedAVG context and to perform a robustness analysis to determine it's limitations. In the end, the purpose of the thesis is to answer the following questions:

1. Is it possible to perform a DL on a federated network using FedAVG?
2. If possible - what information about the model, parameters, hyper parameters and the dataset is needed for the DL to be successful?
3. If the attack is possible to conduct, can we identify the region in which it becomes ineffective?

To answer these questions, four objectives have been stated which upon successful completion carries the final result of the project.

1. Produce a FedAVG implementation of Federated Learning.
2. Adapt a State-Of-The-Art Deep Leakage attack so that they attack the shared model weights rather than the shared gradients.
3. Run the FedAVG DL attack within the produced FL Framework and quantify the success of the reconstruction.
4. Perform a robustness analysis to determine:
   4.1. the effectiveness of DL when increasing the amount of local training (batch size + local epochs)
   4.2. whether the training state of the FL model affects the quality of a DL attack within a FedAVG context.

1.2 Scope

The project is limited to not:

1. Investigate cyber security aspects such as cryptography.
2. Create or optimize large or novel machine learning models for distinct and isolated purposes.
3. Focus on high-performance code efficiency, machine learning operations (MLOps), pipelining or deployment of machine learning models.
4. Handle any company restricted (or higher) material. Thus, all subject specific knowledge gained through the project can be publicly disclosed in the thesis.

1.3 Thesis Outline

In Chapter 2 fundamental theory regarding ML, FL and DL will be introduced. Chapter 3 covers the methodology of the work including justification of model structures, dataset, training settings etc. The chapter is divided into two sections each covering FL and DL separately. Chapter 4 presents the results for the corresponding methodology. Chapters and 5 and 6 includes discussion along with respective conclusions.

2

Theory

This chapter presents the fundamental theories that form the basis of the project. It begins with an overview of machine learning with a focus on neural networks, including their structure, function, and training procedures. It then introduces federated learning, outlining its core principles and various training paradigms. Finally, the concept of deep leakage is explored, along with the underlying ideas that motivate its study in the context of privacy and model security.

2.1 Machine Learning with Neural Networks

AI is a broad term that refers to the simulation of intelligence in machines. ML is a subset of AI in which algorithms are trained without being explicitly programmed. In turn, a neural network (NN) is a type of machine learning model inspired the human brain and consists of coupled neurons [22]. In the following subsection, the fundamentals of ML with NNs are covered to give the reader an understanding of some if their tasks, structure and training dynamics. The theory we present is heavily based on the work of [22], and we refer to this work for a more comprehensive treatment of the subject.

2.1.1 Neural Networks

Perceptron A neuron $a_j^{(l)}$ in a perceptron takes one or several inputs $a_k^{(l-1)}$ from the previous layer $l-1$, multiplies these with weight w_{jk}, summarizes the products, subtracts with a threshold θ_j and runs the sum through an activation function g:

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} - \theta_j^{(l)} \tag{2.1}$$

$$a_j^{(l)} = g(z_j^{(l)}) \tag{2.2}$$

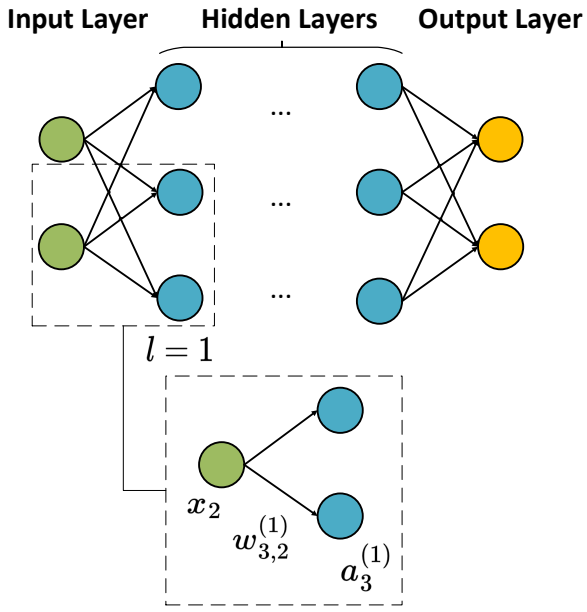

Figure 2.1: Visualization of an MLP with two inputs and outputs and multiple hidden layers. The hidden layers transforms the input to the output and consists of neurons coprising parameters which are trained trough backpropagation.

This is done for each neuron sequentially in the perceptron as $a_j^{(l)}$ will become an input to $a_j^{(l+1)}$. A perceptron is fully connected, meaning that each neuron in layer l is connected to each neuron in layer $l+1$ as shown in Figure 2.1. However, the multi layered perceptron (MLP) is the most basic structure a NN can take. There are other structures such as convolutional neural networks (CNNs), recurrent neural networks (RNN) and many more complex structures. From here on, we will refer to these structures as models. Thus, we additionally denote:

$\boldsymbol{x}$: The inputs to the model
$\boldsymbol{\theta}, \boldsymbol{w}$: The parameters of the model
$f(\cdot)$: The function which the model represents
$\hat{y} = f(\boldsymbol{x})$: The output of the model

The inputs $\boldsymbol{x}$ can be seen as the $l = 0$ layer of the perceptron:

$$x_k = a_k^{(0)}, \tag{2.3}$$

thereby obeying the same dynamics as described in Equation 2.1. With thresholds and non-linear activation functions, $f(\cdot)$ can represent complex and non-linear functions which other approaches might struggle to formulate. This is the very essence of NNs.

Training A model is trained by feeding samples trough it and continuously adjusting the parameters. The samples contain both inputs and corresponding labels y. When having fed the model with one $\boldsymbol{x}$ or a batch $\boldsymbol{X}$ of samples the output $\hat{y}$ or outputs $\hat{\boldsymbol{y}}$ of the model, also referred to as the prediction, is measured trough a loss function $\ell(y, \hat{y}; \boldsymbol{w})$ (for a single sample). The loss function varies depending on the task of the model but is typically formulated in such a way that a high loss indicates a big difference between the ground truth label and the prediction made by the model.

As the goal is to retrieve a model who's predictions align with the ground truth labels, we minimize the loss function w.r.t the model parameters. In the vast majority of cases, gradient based optimization techniques are used, more specifically versions of gradient descent (GD). The process of computing the gradients of the loss function with respect to the parameters is called backpropagation, which starts with computing the error at each neuron in layer l, given by:

$$\delta_j^{(l)} = \frac{\partial \ell}{\partial a_j^{(l)}} g'(z_j^{(l)}) \tag{2.4}$$

where $g'(z_j^{(l)})$ is the derivative of the activation function. The gradient of the loss function with respect to the weight is computed as:

$$\frac{\partial \ell}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)} \tag{2.5}$$

The error is propagated backwards using:

$$\delta_k^{(l-1)} = \sum_j w_{jk}^{(l)} \delta_j^{(l)} g'(z_k^{(l-1)}) \tag{2.6}$$

Finally, when having computed the gradients of the loss function with respect to the parameters, the weights and thresholds are updated according to the optimization method. For GD this becomes:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \eta \frac{\partial \ell}{\partial w_{jk}^{(l)}} \tag{2.7}$$

$$\theta_j^{(l)} \leftarrow \theta_j^{(l)} + \eta \frac{\partial \ell}{\partial \theta_j^{(l)}} \tag{2.8}$$

where η is the learning rate. This iterative process typically continues until the loss function converges to a minimum. Henceforth, in terms of notation, we shall consider $\boldsymbol{\theta}$ as a part of $\boldsymbol{w}$ which now resembles all the parameters of the model and for the entire model, using a batch of images, this can be expressed as:

$$\boldsymbol{w'} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \ell(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{w})) \tag{2.9}$$

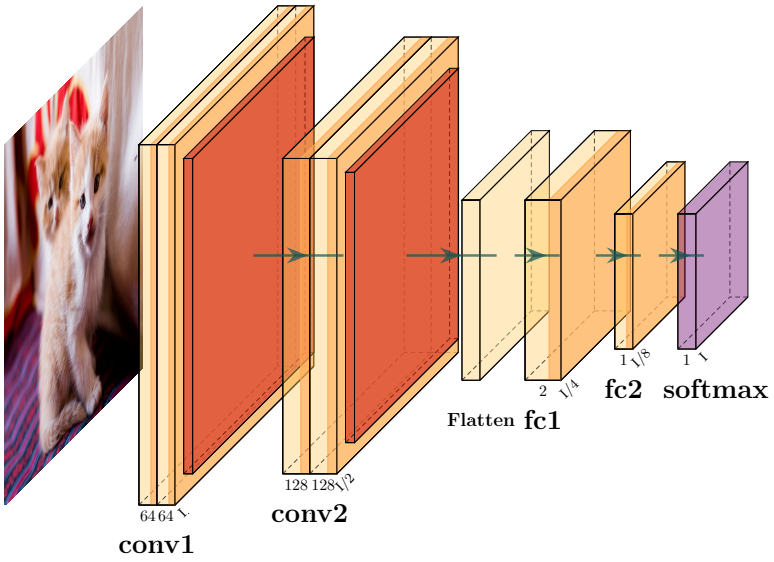

Figure 2.2: A simple CNN comprising two convolutional layers, two fully connected layers and and a softmax output.

2.1.2 Image Classification

Image classification is the task of assigning a label to an image based on its visual content. This is achieved by training a model to recognize patterns and features within images. The model learns to map an input image $\boldsymbol{x}$ to an output class label y.

Feature Extraction and Convolutional Layers Unlike traditional the previous introduced MLP, convolutional neural networks (CNNs) are commonly used for image classification. CNNs leverage convolutional layers that apply filters (kernels) to extract spatial features such as edges, textures, and shapes. The output of a convolutional layer is a feature map, which captures important local structures. In Figure 2.2 the structure of a simple CNN is illustrated.

The general operation of a convolution is defined as:

$$y_{i,j}^{(l)} = g\left(\sum_{m,n} w_{m,n}^{(l)} x_{i+m,j+n}^{(l-1)} - \theta^{(l)}\right) \tag{2.10}$$

where $w_{m,n}^{(l)}$ represents the filter weights.

Loss Function and Optimization For classification problems, the most commonly used loss function is categorical cross-entropy:

$$\ell(y, f(\boldsymbol{x}; \boldsymbol{w})) = \ell(y, \hat{y}; \boldsymbol{w}) = -\sum_{c} y_c \log(\hat{y}_c) \tag{2.11}$$

Each output neuron of the image classification model represents a class where y_c is the ground truth label (one-hot encoded) and $\hat{y}_c$ is the predicted probability for class c. The model parameters are updated using a gradient-based optimization techniques as discussed earlier and the class probabilities are computed using the softmax function:

$$\hat{y}_c = \frac{\exp(z_c)}{\sum j \exp(z_j)} \tag{2.12}$$

For a batch of B samples, the loss can be summed or averaged across all samples. For the averaging case, we denote the loss of a batch as:

$$\ell(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{w})) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{w})) \tag{2.13}$$

2.1.3 IID & non-IID

In ML, a common assumption is that training data is independently and identically distributed (IID). This means that each data sample $\boldsymbol{x}_i$ is drawn from the same underlying probability distribution $P(X)$ and is statistically independent of other samples:

$$P(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \prod_{i=1}^{n} P(\boldsymbol{x}_i). \tag{2.14}$$

This assumption simplifies analysis and optimization, as many machine learning algorithms rely on uniform data distributions for stable training and convergence.
However, in many real-world applications, data is often non-identically and/or non-independently distributed (non-IID). This can take various forms, such as:

- **Feature distribution skew**: Different subsets of data may have different feature distributions, i.e., $P(X)$ varies across datasets.
- **Label distribution skew**: Certain subsets may contain only a limited set of labels, meaning $P(Y|X)$ differs across datasets.
- **Concept shift**: The relationship between features and labels $P(Y|X)$ changes over time or across different data sources.

Mathematically, in a non-IID setting, the data distribution for two given subsets D_k and D_j may be expressed as:

$$P_k(\boldsymbol{x}, y) \neq P_j(\boldsymbol{x}, y) \quad \text{for } k \neq j. \tag{2.15}$$

Non-IID data presents challenges in model training, as optimization techniques that rely on uniform sampling may struggle with biased or imbalanced distributions. In decentralized settings, where different clients or nodes hold distinct data distributions, ensuring convergence and generalization requires specialized techniques.
Understanding the distinction between IID and non-IID data is crucial when designing learning algorithms for distributed and federated learning, where data heterogeneity directly impacts model performance and aggregation strategies.

2.2 Federated Learning

Typically, a NN is composed of a single model. In 2017 McMahan et al. [20] introduced FL which involves multiple clients training the same model through sharing model updates. Local clients train an instance of a global model on their local data without exchanging the actual data itself, but rather the gradients or weights. A central server then collects the shared gradients or weights for aggregation and distributes an updated version back to all clients. If data across clients is IID, standard optimization methods converge reliably. However, non-IID data can cause weight divergence, slower convergence, and poor generalization. FL addresses these challenges effectively. Its primary advantage lies in enhancing the privacy of each client's training data while also improving model performance. Additionally, FL enables private collaboration, making it highly applicable across various scenarios. For instance, in a hospital, where each patient's information is private, it can still be leveraged for global purposes, so that all clients can benefit from each other [1]. In addition, FL enables parallel training, which increases computational efficiency. A larger number of clients enable for more computational power, leading to a faster training process compared to a single model training on all the data independently. As there are many applicable areas of use, the main focus in this thesis is the FL models ability is to handle the integrity issue of private training data between clients.

In the following FL theory, we refer to the work of [20] where FL was initially proposed. FL aims to minimize a global objective function that aggregates local objectives from multiple K clients. These local objective functions are the loss from each client k's loss function computed on their private data.

Since data is partitioned across K clients, the global objective function, which is the average loss over all clients can be rewritten as:

$$\ell_g(\boldsymbol{w}) = \sum_{k=1}^{K} \frac{N_k}{N_g} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w})) \tag{2.16}$$

where $\frac{N_k}{N_g}$ is the weight assigned to client k, representing its contribution to the global objective. This weight ensures that clients with larger datasets have a proportionally higher influence on the overall optimization process. Thus, the global function $\ell_g(\boldsymbol{w})$ is a weighted sum of the local client objectives, where the contribution of each client depends on the fraction of total data it holds [21]. There are alternative ways so weight the contributions of each client. However, this is not something which will be covered in the thesis.

Training in FL is performed in a number of rounds R , where each round r consists of local training on the entire or partial subset of each clients data, followed by aggregation at the central server. The computational workload for each round is influenced by three main parameters:

C: Fraction of clients used each round

E: Number of training epochs each client performs locally per round

B: Size of the local batch used for client updates [21]

Generally there are two primary optimization methods in FL. These two are FedSGD and FedAVG, which stand for Federated Stochastic Gradient Descent and Federated Averaging. These methods define how clients update their local models and how the server aggregates these updates, each with different computational and communication characteristics.

2.2.1 FedSGD

FedSGD is a federated version of the traditional Stochastic Gradient Descent (SGD) algorithm, where each client computes the gradient on its local data and sends it to the central server for aggregation.

A client k computes the gradient over its dataset according to Equation 2.9 at a round r:

$$\nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}^{\boldsymbol{r}})) \tag{2.17}$$

Depending on the context, $\boldsymbol{y}$ and $\boldsymbol{X}$ might also be r-specific, meaning that the client sees different data each round. Within our context however, clients see the same data each round.

For each client, the gradients are then sent to the central server. The server averages the gradients from all participating clients based on their respective dataset sizes similar to Equation 2.16. The global model is then updated from the averaged gradients and distributed to all clients as follow:

$$\boldsymbol{w}^{r+1} = \boldsymbol{w}^r - \eta \sum_{k=1}^{K} \frac{N_k}{N_g} \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}^{\boldsymbol{r}})) \tag{2.18}$$

Algorithm 1 presents the protocol of FedSGD.

2.2.2 FedAVG

For FedSGD, all clients shared the same weights $\boldsymbol{w}$ for their respective gradient computation as no local training (stepping) was done. For FedAVG, weights will start to deviate within rounds, thus we need to add a subscript for $\boldsymbol{w}_k$ and $\boldsymbol{w}_g$ representing the weights for client k and the global aggregated model.

FedAVG extends the idea of FedSGD by introducing local iterations on the model before the aggregation step, making it a more computationally efficient algorithm. Instead of each client performing just one computation of gradient descent as in FedSGD, in FedAVG, each client is allowed to perform multiple local updates on its

Algorithm 1: Federated Stochastic Gradient Descent (FedSGD)

Data: K clients, learning rate η, number of global iterations T,
initial global model $\boldsymbol{w}_g^0$
Result: Updated global model $\boldsymbol{w}_g$
for $t = 0$ **to** $T - 1$ **do**
  for *each client* $k \in \{1, 2, \ldots, K\}$ **do**
    $\boldsymbol{w}_k^t \leftarrow \boldsymbol{w}_g^t$; `// Initialize local model`
    Compute local gradient: $\nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}_k^t))$;
    Send gradient to server;
  end
  Server Aggregation:
    Receive $\nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}_k^t))$ from all clients;
    $\boldsymbol{w}_g^{t+1} \leftarrow \boldsymbol{w}_g^t - \eta \sum_{k=1}^{K} \frac{N_k}{N_g} \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}_k^t))$;
  ClientUpdate:
    Send updated global model $\boldsymbol{w}_g^{t+1}$ to all clients;
end

private data before sending the updated model to the server for aggregation. We refer to these times steps as $t = 0, 1, 2, ..., T$.

Each client k performs T steps of GD locally, equivalent to the dynamics from Equation 2.9:

$$\boldsymbol{w}_k^{r,t+1} = \boldsymbol{w}_k^{r,t} - \eta \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}_k^{\boldsymbol{r,t}})) \tag{2.19}$$

After performing T local updates, the client sends its updated model $\boldsymbol{w}_k^{r,T}$ to the central server. The central server aggregates the clients models by averaging, typically w.r.t. number of data samples. The global model $\boldsymbol{w}_g^{r+1}$ is updated as the weighted average of the models sent by the clients:

$$\boldsymbol{w}_g^{r+1} = \sum_{k=1}^{K} \frac{N_k}{N_g} \boldsymbol{w}_k^{r,T} \tag{2.20}$$

Lastly, the global model is distributed to all clients. In Algorithm 2, each step is shown.

2.2.3 Intersection between FedSGD and FedAVG

FedSGD and FedAVG share the same underlying principle of federated optimization, where clients perform computations locally and a central server aggregates the updates. The key dynamical difference lies in the number of local updates performed before aggregation and the type of information sent to the central server. However, there is an intersection where both algorithms follow the same process.

Specifically, when setting B = full and $E = 1$ in FedAVG, each client performs

Algorithm 2: Federated Averaging (FedAVG)

Data: K clients, learning rate η, number of global iterations T, batch size B, initial global model $\boldsymbol{w}_g^0$
Result: Updated global model $\boldsymbol{w}_g$
for $t = 0$ **to** $T - 1$ **do**
  for *each client* $k \in \{1, 2, \ldots, K\}$ **do**
    $\boldsymbol{w}_k^t \leftarrow \boldsymbol{w}_g^t$; `// Initialize local model`
    Perform E local steps over minibatches of size B:;
    for *each local step* e **do**
      Sample mini-batch $(\boldsymbol{X}_{k,e}, \boldsymbol{y}_{k,e})$ from local data;
      $\boldsymbol{w}_k^t \leftarrow \boldsymbol{w}_k^t - \eta \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_{k,e}, f(\boldsymbol{X}_{k,e}; \boldsymbol{w}_k^t))$;
    end
    Send updated model $\boldsymbol{w}_k^{t+1} \leftarrow \boldsymbol{w}_k^t$ to server;
  end
  Server Aggregation:
    Receive $\boldsymbol{w}_k^{t+1}$ from all clients;
    $\boldsymbol{w}_g^{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N_g} \boldsymbol{w}_k^{t+1}$;
  ClientUpdate:
    Send updated global model $\boldsymbol{w}_g^{t+1}$ to all clients;
end

a single full-batch gradient step on its entire dataset before aggregation. This scenario coincides with the behavior of FedSGD, resulting in identical global updates. Each client k computes a full-batch gradient update using its entire local dataset:

$$\boldsymbol{w}_k^{r+1} = \boldsymbol{w}^r - \eta \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}^r)) \tag{2.21}$$

The server then aggregates these updated weights using a weighted average:

$$\boldsymbol{w}^{r+1} = \sum_{k=1}^{K} \frac{N_k}{N_g} \boldsymbol{w}_k^{r+1} \tag{2.22}$$

Substituting the local update rule into the aggregation yields:

$$\boldsymbol{w}^{r+1} = \sum_{k=1}^{K} \frac{N_k}{N_g} \left(\boldsymbol{w}^r - \eta \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}^r)) \right) \tag{2.23}$$

$$= \boldsymbol{w}^r - \eta \sum_{k=1}^{K} \frac{N_k}{N_g} \nabla_{\boldsymbol{w}} \ell_k(\boldsymbol{y}_k, f(\boldsymbol{X}_k; \boldsymbol{w}^r)) \tag{2.24}$$

The resulting update is equivalent to taking a gradient step using the full gradient of the global loss function:

$$\boldsymbol{w}^{r+1} = \boldsymbol{w}^r - \eta \nabla_{\boldsymbol{w}} \ell_g(\boldsymbol{w}^r) \tag{2.25}$$

where $\nabla_{\boldsymbol{w}} \ell_g(\boldsymbol{w}^r)$ denotes the gradient of the global objective as defined in Equation 2.16.

Thus, when $B = \text{full}$ and $E = 1$, FedAVG reduces exactly to FedSGD.

2.3 Deep Leakage

In 2019, Zhu et al. introduced the concept of Deep Leakage from Gradients (DLG) [28], demonstrating that an adversary with access to shared gradients and specific hyperparameters can reconstruct private training data through gradient matching. By iteratively optimizing dummy data to align with observed gradients, attackers can effectively extract sensitive information. While deep leakage attacks can target various model architectures [18, 28], most research focuses on image classification due to its straightforward evaluation of attack efficiency. Accordingly, this project adopts image classification to facilitate comparison with existing work.

In the context of FL, these attacks are particularly relevant. An honest-but-curious adversary in FL is assumed to have access to all necessary information, such as shared updates, model architecture, and specific hyperparameters, enabling them to perform DL attacks effectively. These adversaries adhere to the prescribed protocol but attempt to infer private data from shared updates without actively disrupting the training process. Unlike malicious actors who may engage in data poisoning [14], honest-but-curious participants operate as legitimate clients or servers while extracting sensitive data from gradients (FedSGD) or weights (FedAVG).

In this section, we describe the core idea behind gradient matching, the components required for it to be effective, and finally, the challenges involved in extending this approach from gradients to model weights. In Section 3.2, we further elaborate the context in which these components can be retrieved by the adversary and how this closely relates to FL.

2.3.1 Gradient Matching

The essential components required for an adversary to perform a gradient-based DL attack on a victim are as follows:

- **A stolen gradient:** The gradient computed during the victim's training process, which carries information about the input data and corresponding labels.
- **The victim's model:** The model architecture and parameters on which the stolen gradient was computed.
- **The loss function:** The specific loss function used during training, which is necessary to correctly simulate a gradient computation.

Importantly, having access to these components also grants the adversary indirect access to additional information, including:

- **The data format:** The architecture of the input and output layers reveals the shape and type of both the input $\boldsymbol{x}$ and the corresponding label y.
- **The task type:** From the loss function and output layer, the adversary can often infer the nature of the task (e.g., classification, regression, multi-label classification).

- **The number of classes or output targets:** For classification problems, the dimensionality of the output layer suggests the number of classes.

Zhu et al. focus on standard distributed training, where a client samples a mini-batch $(\boldsymbol{X}, \boldsymbol{y})$ from a dataset and computes the gradient of its loss function w.r.t. the model weights, $\boldsymbol{w}$. The idea behind gradient matching is to find an input, $\hat{\boldsymbol{X}}$ which generates a gradient that is as similar to the stolen gradient as possible. To perform the attack, the adversary randomly initializes dummy inputs $\hat{\boldsymbol{X}}$ and labels $\hat{\boldsymbol{y}}$ and feeds the inputs trough the model on which the victim has trained. The output generated by the model from the dummy inputs along with the dummy labels are then put into the loss function on which a dummy gradient is computed w.r.t the model weights.

In the following subsection, the theory does not rely on a federated learning setting. We therefor exclude subscripts for round r and client k. To simplify the notation, we introduce the following shorthands:

$$\nabla\ell(\boldsymbol{w}) := \nabla_{\boldsymbol{w}}\ell(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{w})) \tag{2.26}$$

$$\nabla\ell(\hat{\boldsymbol{w}}) := \nabla_{\boldsymbol{w}}\ell(\hat{\boldsymbol{y}}, f(\hat{\boldsymbol{X}}; \boldsymbol{w}))\Big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} \tag{2.27}$$

Where we refer to $\nabla\ell(\boldsymbol{w})$ as the stolen gradient and $\nabla\ell(\hat{\boldsymbol{w}})$ as the dummy gradient. Earlier, we used $\hat{\boldsymbol{y}}$ to denote the model's prediction. In this context, however, $\hat{\boldsymbol{y}}$ represents a *dummy label*. Although we place a hat on $\hat{\boldsymbol{w}}$, the weights are considered fixed; it is the inputs $\hat{\boldsymbol{X}}$ and labels $\hat{\boldsymbol{y}}$ that vary and induce variation in the resulting gradient.

So far, this process is identical (except for the fact that the training data is just dummy noise) to how one would typically train a NN, as was described in Section 2.1.1. But now, instead of using the dummy gradient to update the weights, it is compared with the stolen gradient.

If we formulate this comparison as the squared euclidean distance between the dummy gradient and the stolen gradient, we get what we refer to as a **reconstruction loss** which we consider as the objective function in an optimization problem.

$$\mathcal{L}_{DLG} =: ||\nabla\ell(\hat{\boldsymbol{w}}) - \nabla\ell(\boldsymbol{w})||^2 \tag{2.28}$$

In this optimization problem we seek to minimize the reconstruction loss w.r.t the dummy inputs $\hat{\boldsymbol{X}}$ and dummy labels $\hat{\boldsymbol{y}}$. Given the stolen gradient in the training process, the original training data is approximated by:

$$\hat{\boldsymbol{X}}^*, \hat{\boldsymbol{y}}^* = \arg\min_{\hat{\boldsymbol{X}}, \hat{\boldsymbol{y}}} ||\nabla\ell(\hat{\boldsymbol{w}}) - \nabla\ell(\boldsymbol{w})||^2 \tag{2.29}$$

where $\hat{\boldsymbol{X}}^*, \hat{\boldsymbol{y}}^*$ are the converged batch of images and labels respectively.

The formulation of the reconstruction loss $\mathcal{L}_{DLG}$, as described by Zhu et al. [28], ensures that the objective is at least twice differentiable with respect to the dummy inputs $\hat{\boldsymbol{X}}$ and labels $\hat{\boldsymbol{y}}$, allowing it to be optimized using standard gradient- and hessian-based methods if we assume that $f(\cdot)$ is twice differentiable—a reasonable assumption, as most modern machine learning models, including neural networks, satisfy this condition. Algorithm 3 shows how DLG can be performed when using GD as optimizer:

Algorithm 3: Deep Leakage from Gradients (DLG) via Gradient Matching

Data: Stolen gradient $\nabla\ell(\boldsymbol{w})$, model f, initial dummy inputs $\hat{\boldsymbol{X}}_0$, dummy labels $\hat{\boldsymbol{y}}_0$, loss function ℓ, number of iterations I, learning rate η
Result: Reconstructed data $\hat{\boldsymbol{X}}^*, \hat{\boldsymbol{y}}^*$
Initialize: $\hat{\boldsymbol{X}} \leftarrow \hat{\boldsymbol{X}}_0$, $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}}_0$;
for $i = 0$ **to** $I - 1$ **do**
  $\nabla\ell(\hat{\boldsymbol{w}}) \leftarrow \nabla_{\boldsymbol{w}}\ell(\hat{\boldsymbol{y}}, f(\hat{\boldsymbol{X}}; \boldsymbol{w}))$: Compute dummy gradient
  $\mathcal{L}_{DLG} \leftarrow \|\nabla\ell(\hat{\boldsymbol{w}}) - \nabla\ell(\boldsymbol{w})\|^2$: Compute reconstruction loss
  Update dummy inputs and labels: $\hat{\boldsymbol{X}} \leftarrow \hat{\boldsymbol{X}} - \eta\nabla_{\hat{\boldsymbol{X}}}\mathcal{L}_{DLG}$ $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} - \eta\nabla_{\hat{\boldsymbol{y}}}\mathcal{L}_{DLG}$
end
Output: $\hat{\boldsymbol{X}}^* \leftarrow \hat{\boldsymbol{X}}, \hat{\boldsymbol{y}}^* \leftarrow \hat{\boldsymbol{y}}$;

2.3.2 Transition to Weights

This subsection investigates the differences and challenges that arise when trying to extend a DL attack to target weights instead of gradients.

In terms of required components, the main difference, compared to the ones presented in the previous subsection, is that instead of the adversary having access to a stolen gradient $\nabla\ell(\boldsymbol{w})$, it has access to the victims weights before and after having taken t training steps from $\boldsymbol{w}^{r,0}$ to $\boldsymbol{w}^{r,0}$ within a training round r.

From Section 2.1.1, we know that NNs are typically trained iteratively, as described by Equation 2.9. Rewriting this expression to align with the notation used in our DL setting, the update rule can be formulated as:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta\nabla\ell(\boldsymbol{w}^t) \tag{2.30}$$

Where η denotes the learning rate, and $\nabla\ell(\boldsymbol{w}^t)$ the gradient of the loss function with respect to the weights at time step t.

Now, if the victim only trains for one time step ($T = 1$) which corresponds to full batch size ($B = full$) and one epoch ($E = 1$), the adversary can simply plug this into Equation 2.30 and derive the gradient:

$$\boldsymbol{w}^T = \boldsymbol{w}^0 - \eta\nabla\ell(\boldsymbol{w}^0) \equiv \nabla\ell(\boldsymbol{w}^0) = \frac{\boldsymbol{w}^0 - \boldsymbol{w}^T}{\eta} \tag{2.31}$$

This is an important edge case as it constitutes the transition from gradient- to weight-based DL attacks which occurs in the intersection between FedSGD and FedAVG. Furthermore, the adversary can use the stolen weights, given that it has access to η, to compute the gradient and utilize a gradient-based DL attack instead.

When increasing the number of times steps ($T > 1 \Leftrightarrow E > 1 \vee B \neq full$), a gradient must be approximated if the adversary want to use a gradient-based attack. The simplest approximation approach is simply to plug T into Equation 2.31:

$$\nabla \ell(\boldsymbol{w}^0) \approx \frac{\boldsymbol{w}^0 - \boldsymbol{w}^T}{T\eta} \tag{2.32}$$

However, the quality of this approximation worsens as the number of training steps increases. Additionally, the approximation assumes the usage of vanilla GD while in realistic scenarios, more sophisticated optimizers are typically employed such as Adam [12], further increasing the discrepancy.

3
Methodology

This chapter outlines the methodology employed in this project. It includes detailed descriptions and justifications for the selected datasets, machine learning models, training parameters, hyperparameters, as well as the FL and DL configurations. For both the FL and DL components, corresponding experiments are also presented, each accompanied by a description of their setup and a motivation for their design, including the specific research questions or hypotheses they aim to explore.

3.1 Federated Learning with FedAVG

A central part of the project is the setting in which the Federated Learning is run. This section describes and justifies many of the methods and design decisions that are being used throughout the project on which DL is heavily dependent. As it is yet to discover the feasibility of performing a DL attack on a FL network using FedAVG, the general attitude in the project is to strive for simplicity rather than for realism and then eventually append layers of complexity when image reconstruction can be observed successfully. In other words, data distributions, model structures, parameters and hyper parameters are kept simple initially.

3.1.1 Data distributions

To align the project with previous work, the popular CIFAR10 dataset [13] is used. The dataset is widely used for training machine learning models, particularly for image classification tasks.

The dataset consists of 60 000 32x32 color images images divided into 50 000 training samples and 10 000 test samples with uniformly distributed classes (6000 samples of each in total and 5000 samples of each in the training set). The 10 classes include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck with only one class assigned to one sample. In Figure 3.1, 16 randomly sampled images are shown. The samples display the low resolution of the images, which one typically doesn't see in modern day photographs. However, the low resolution renders a relatively low computational cost speeding up cumbersome experiments throughout the project. Additionally, we hypothesize that images with lower resolution will be easier to reconstruct with DL in later experiments.

The training samples has been split into a validation set (20%) and a training set

Figure 3.1: Samples from the CIFAR10 dataset [13].

(80%) from which the data has been further partitioned to be distributed to the clients. The partitioning can be done in several ways. The simplest way is to partition the data uniformly with an IID-partitioner as seen in the top plot in Figure 3.2. This way, all clients train on the same identical but independent distributions (IID). As 10 clients are used throughout the experiments, which we'll discuss later, using 10 partitions gives a partition size of 5000 training samples.

Another alternative is to use a non-IID partitioner, which distributes data unevenly among clients, reflecting the variability seen in real-world scenarios. Unlike IID partitioning, where each client receives a balanced sample of the entire dataset, non-IID partitioning can lead to significant data heterogeneity, with some clients having data concentrated in specific classes while others lack data from those classes entirely.

A common method to implement non-IID partitioning is to use a Dirichlet distribution. The bottom plot in Figure 3.2 illustrates how the CIFAR-10 dataset is partitioned using a Dirichlet distribution with $\alpha = 1$. Here, the parameter α controls the degree of non-IID-ness. Lower values of α result in more skewed distributions, where each client primarily receives data from a small subset of classes, while higher values of α produce distributions closer to IID. Setting $\alpha = 1$ strikes a balance, creating moderate data imbalance that can effectively simulate real-world federated learning scenarios where clients have diverse and unevenly distributed datasets. For the complete equation and formulation of the Dirichlet distribution, see Appendix A.1.

3.1.2 Models & Hyperparameters

Throughout the course of this thesis, it is important to note that optimizing the performance of the federated learning setup, through model architecture tuning or hyperparameter optimization, is considered out of scope. This project is intended as a proof of concept, with the primary focus on the relative insights gained from

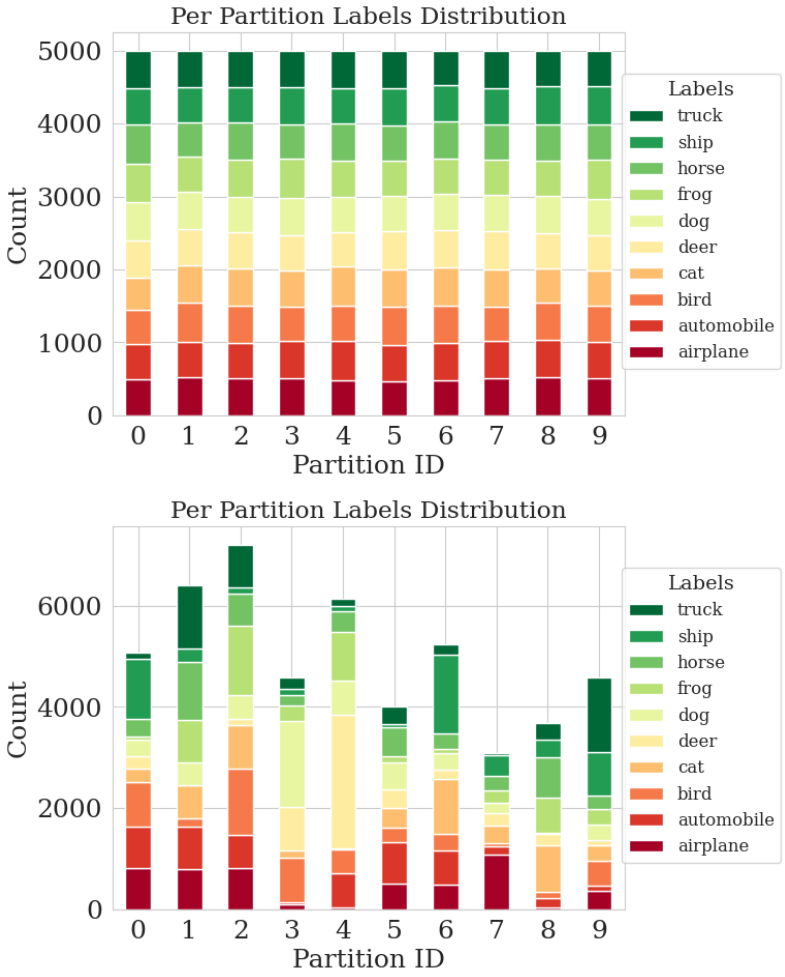

Figure 3.2: CIFAR10 data partitioning with uniform IID partitioner and Dirchlet non-IID partitioner with $\alpha = 1$

various experimental configurations. Accordingly, local dataset size (N), model architecture, and optimizer settings may vary depending on the purpose of each experiment.

LeNet-5 LeNet-5 is a well-established convolutional neural network originally proposed by LeCun et al. [16]. It was designed for handwritten digit recognition and has since become a common baseline in deep learning research. In this thesis, LeNet-5 is primarily used to study the training dynamics of federated learning in a controlled setting due to its simplicity and efficiency.

CNNCifar CNNCifar is a lightweight convolutional neural network frequently used in the context of federated learning and deep leakage attacks [27]. Its structure is straightforward, making it suitable for exploratory experiments where reproducibility and computational efficiency are essential. It is designed to handle 32×32 RGB images such as those in the CIFAR-10 dataset and consists of two convolutional layers followed by average pooling operations and two fully connected layers:

```
Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
ReLU
AvgPool2d(kernel_size=2, stride=2)
Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
ReLU
AvgPool2d(kernel_size=2, stride=2)
Linear(8192, 200)
ReLU
Linear(200, num_classes)
```

The benefit of using the CNNCifar model on the CIFAR-10 dataset is that it allows us to leverage well-established hyperparameter recommendations [5]. This enables us to focus our efforts on more Deep Leakage evaluation.

Optimizer and Local Loss Function Both Adam and stochastic gradient descent (SGD) are used in various experiments throughout this thesis. Adam[12] is an adaptive optimization algorithm widely recognized for its efficiency in handling large datasets and complex models by adjusting the learning rate individually for each parameter. SGD, on the other hand, is a more traditional optimization method that updates parameters using the gradient of the loss function and is often favored for its simplicity and strong convergence properties, especially when combined with techniques like momentum. The choice of optimizer depends on the specific goals of each experiment. The loss function used is cross-entropy loss, a standard choice for classification tasks as it effectively quantifies the difference between predicted probabilities and true labels. Both optimizers, along with cross-entropy loss, are widely supported in the literature and are known for their reliability and effectiveness across a range of deep learning scenarios.

3.1.3 Federated Learning setting

In this subsection the remaning settings related to the Federated Learning is presented. Specifically the choice of number of clients and strategy are justified.

Number of clients What would be a realistic scenario regarding the number of clients is impossible to determine as FL can be used and is used in various contexts. In our experiments, the number of clients needs to be sufficiently large to capture the essential dynamics of the FL yet as small as possible to speed up computation.

Strategy FedAVG is used throughout all experiments in the project as it is the core of our research. Within the strategy, the fraction of clients, C, needs to be specified. This will be constantly set to $C = 1$ for all experiments.

Number of rounds, epochs & batch size The three remaning settings are the number of rounds, epochs and the batch size. The experiments investigate the

number of epochs E and batch size B as variables and run the training for as many rounds as needed to obtain convergence and qualitative behaviour.

3.1.4 Implementation

The first objective of this thesis is to implement the FedAVG algorithm using the Flower [3] library, a widely used framework for federated learning applications. Flower provides a flexible and scalable environment for developing and evaluating federated learning models, making it well suited for this study. With Flower's built-in functionalities, a simulation program is developed to run experiments. This simulation enables efficient testing of different parameters, allowing for systematic evaluation of their impact on model performance. By taking advantage of Flower's flexibility, the implementation makes it easier to simulate different federated learning scenarios while keeping the process reproducible and scalable. For further details, we refer to our GitHub repository [19].

3.1.5 Federated Learning Experiments

In this subsection, the experiments for FL performance are specified along with their corresponding settings. Four experiments have been conducted:

1. Experiment 1.1 Intersection Between FedSGD & FedAVG
2. Experiment 1.2 Decreasing Batch Size
3. Experiment 1.3 Increasing Epochs
4. Experiment 1.4 Optimal training

The general purpose of these four experiments is to investigate how the performance of a federated network depends on its training state.

3.1.5.1 General Configuration

Experimental setup parameters, which are consistent for all the FL experiments in the section:

Parameter	**Value**
Model	LeNet
Optimizer	Adam
Loss Function	Cross-Entropy
Learning Rate	0.001
Number of Clients	10
Local Data Size N	5000
Strategy	FedAVG
Distribution	IID
Rounds	200

We refer to Appendix A.2.1 for the full configuration.

3.1.5.2 Intersection Between FedSGD & FedAVG

Description This experiment focuses on the point where FedSGD and FedAVG intersect. Specifically, this is achieved by using a full batch size meaning that the entire local dataset is treated as a single minibatch ($B = N$). Consequently, each client performs a gradient update using all of its local data at once. By combining this full batch size with a single epoch per communication round, where each client completes only one pass over its local dataset before sending updates to the central server, the FedSGD setup becomes equivalent to FedAVG under these constraints:

Experiment 1.1:	**B**	**E**	**T**
	Full	1	1

Motivation & Hypothesis The motivation behind this experiment is to set a baseline benchmark for subsequent experiments. We hypothesize that FedSGD performs rather poorly as the number of updates $T = E\frac{N}{B}$ is small, leading to the need for many communication rounds to obtain a somewhat *good* reslut.

3.1.5.3 Decreasing Batch Size

Description In this experiment, the number of epochs are kept to 1. Several simulations are run with decreasing batch sizes from full (same simulation as for Experiment 1.1) to 1 where local step is take for every single image in the local dataset. We now move on from the intersection point and continue study FedAVG.

Experiment 1.2:	**B**	**E**	**T**
	[Full, 64, 32, 16, 8, 1]	1	[125, 250, 500, 1000, 8000]

Motivation & Hypothesis The motivation behind this experiment is to isolate and investigate the impact of batch size on training performance. We hypothesize that reduced batch sizes may lead to improved convergence compared to Experiment 1.1, particularly in terms of convergence speed, as smaller batches result in a higher number of training steps per communication round. Furthermore, we expect that the optimal performance will not be achieved with either a batch size of 1 or a full batch. Instead, we aim to identify an intermediate batch size that provides the best trade-off between convergence speed and final performance.

3.1.5.4 Increasing Epochs

Description In this experiment, we investigate training performance with respect to the number of local epochs, rather than the batch size. Using both full batch size and batch size 1, we run simulations with 1, 5, and 10 local epochs per communication round. As batch size 1 renders N times more local updates than full batch size, the simulations are run for 25 and 200 rounds respectively. We divide these into two sub-experiments, Experiment 1.3a and Experiment 1.3b, where they investigate full batch size and batch size 1 respectively.

Experiment 1.3a:	B	E	T
	Full	[1, 5, 10]	[1, 5, 10]

Experiment 1.3b:	B	E	T
	1	[1, 5, 10]	[8000, 40 000, 80 000]

Motivation & Hypothesis For the full batch size simulations, the effect of varying the number of local epochs is isolated, as there is only one gradient update per epoch. This allows us to study how repeated passes over the local dataset influence convergence and performance without the added complexity of minibatching. We hypothesize that increasing the number of epochs will lead to faster convergence in early rounds, as clients perform more work locally. However, we also expect diminishing returns beyond a certain point, as the benefits of additional local computation may be outweighed by increased model divergence across clients. Our goal is to identify whether an optimal number of epochs exists under both full batch size and batch size 1 settings.

3.1.5.5 Optimal Training

Description This final FL experiment investigates the *optimal* number of local epochs given the best-performing batch size from Experiment 1.2, which was found to be $B = 16$.

Experiment 1.4:	B	E	T
	16	[1, 5, 10]	[500, 2500, 5000]

Motivation & Hypothesis The goal of this final federated learning experiment is to identify a configuration (B, E) that yields strong overall performance. Determining such a configuration under realistic constraints provides valuable insight into practical parameter settings. These findings can then serve as a reference point for designing and tuning deep learning experiments later in this thesis.
We hypothesize that increasing the number of epochs with the fixed batch size $B = 16$ will improve convergence up to a certain point, beyond which the benefits may taper off due to increased local overfitting or divergence between clients.

3.2 Deep Leakage on Model Weights

In this project, we consider a setting in which an honest-but-curious adversary performs DL attacks on its neighbors within the federated network. In Section 2.3, we introduced the necessary components for performing a DL attack on gradients and weights, respectively. Since the adversary is an active participant in the network, it already has access to many of these components, listed in Table 3.1, through its own training process. The only requirement is to intercept a gradient or a weight update when it is communicated from the victim client to the global server.

Table 3.1: Overview of components used or inferred in gradient- and weight-based DL attacks. For a gradient-based DL attack, a gradient is stolen as the federated network has been trained with FedSGD whereas in the FedAVG scenario, the outgoing weight is intercepted.

Component	**Description**	**Access Method**
Stolen Gradient: $\nabla\ell(\boldsymbol{w})$ (FedSGD)	Gradient computed during the victim's training	Intercepted
Stolen Weight: $\boldsymbol{w}_T$ (FedAVG)	Updated weights after the victim's training	Intercepted
Initial Weight: $\boldsymbol{w}_0$	Weight which the victim's gradient computed on or training is initialized with	Shared globally
Victim's Model: $F(\cdot)$	Architecture and parameters used by the victim during training	Shared globally
Loss Function: $\ell(\cdot)$	The loss function which the victim has used during training	Shared globally
Data Format	Input/output layer shapes reveal the structure of $\mathbf{x}$ and $\mathbf{y}$	Inferred from model architecture
Task Type	Indicates classification, regression, etc., based on loss function and output structure	Inferred from model and loss function
Number of Classes	Output layer size suggests the number of classes in classification tasks	Inferred from output layer dimensionality

In this section, we describe two prior attack methods—DLG and IG—which both target gradients—as well as, to the best of our knowledge, the current state-of-the-art attack targeting model weights; SME [27].

3.2.1 Deep Leakage from Gradients & Inverting Gradients

As presented in Section 2.3, DL as a phenomenon was introduced firstly as Deep Leakage from Gradients (DLG) [28] and revolves around minimizing a distance between a stolen gradient $\nabla\ell_k(\boldsymbol{w}^r)$ from client k with ground truth data $(\boldsymbol{X}_k, \boldsymbol{y}_k)$ at round r and a dummy gradient $\nabla\ell(\hat{\boldsymbol{w}})$ with respect to the dummy image $\hat{\boldsymbol{X}}$ and dummy label $\hat{\boldsymbol{y}}$:

$$\hat{\boldsymbol{X}}^*, \hat{\boldsymbol{y}}^* = \arg\min_{\hat{\boldsymbol{X}}, \hat{\boldsymbol{y}}} \underbrace{||\nabla \ell(\hat{\boldsymbol{w}}) - \nabla \ell(\boldsymbol{w})||^2}_{\mathcal{L}_{DLG}} \tag{3.1}$$

Building on this foundation, Inverting Gradients (IG) [6] was introduced to enhance the effectiveness of DLG. While DLG successfully demonstrated that gradient-based attacks can reconstruct private data, its optimization process often led to reconstructions with significant noise and artifacts. IG addressed these shortcomings by introducing additional regularization techniques, notably Total Variation (TV) regularization, which encourages spatial smoothness in the reconstructed images. Furthermore, instead of using mean squared error (MSE) as reconstruction loss, cosine similarity is used:

$$\hat{\boldsymbol{X}}^*, \hat{\boldsymbol{y}}^* = \arg\min_{\hat{\boldsymbol{X}}, \hat{\boldsymbol{y}}} \underbrace{1 - \frac{\langle \nabla \ell(\boldsymbol{w}), \nabla \ell(\hat{\boldsymbol{w}}) \rangle}{\|\nabla \ell(\boldsymbol{w})\| \|\nabla \ell(\hat{\boldsymbol{w}})\|} + \lambda \mathrm{TV}(\hat{\boldsymbol{X}})}_{\mathcal{L}_{IG}} \tag{3.2}$$

where the TV regularization term is defined as:

$$\mathrm{TV}(\hat{\boldsymbol{x}}) = \sum_{i,j} \sqrt{(\hat{x}_{i+1,j} - \hat{x}_{i,j})^2 + (\hat{x}_{i,j+1} - \hat{x}_{i,j})^2}. \tag{3.3}$$

This term reduces high-frequency noise in the reconstructed image(s) by penalizing sharp variations in pixel intensities, leading to more realistic and structured reconstructions.

3.2.2 Label Restoration

IG and DLG both reconstruct the images and labels simultaneously by minimizing their reconstruction loss functions w.r.t. $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{y}}$. This method struggles to reconstruct larger batch sizes, especially when there are multiple images of the same class in a batch. To address the challenges posed by larger batch sizes and repeated labels within a batch, zero-shot label restoration was introduced in *Towards General Deep Leakage in Federated Learning* [8]. Unlike DLG and IG that jointly reconstruct images and labels, this approach separately infers labels by analyzing the gradients of the fully connected layer, leveraging the fact that the gradient of the cross-entropy loss with respect to the logits is directly influenced by the one-hot encoded labels.

Given the final logits $\boldsymbol{z}$ before softmax and their corresponding probabilities $\boldsymbol{p}$, the gradient of the loss function w.r.t. logit z_n is given by:

$$\frac{\partial \ell}{\partial z_n} = p_n - y_n \tag{3.4}$$

By aggregating these gradients over a batch, the method infers the label distribution without relying on the reconstructed images. This enables accurate label restoration even when multiple samples share the same label within a batch. Additionally, this approach is robust to variations in batch size and label distribution, making it more scalable compared to prior techniques.

3.2.3 Permutation Ambiguity & Linear Sum Assignment

When reconstructing a batch of training samples from stolen gradients or weights, one commonly encounters permutation ambiguity, i.e., a mismatch between the order of reconstructed samples and the ground truth samples. This phenomenon arises due to the symmetric nature of the loss function with respect to input permutations.

Solving this issue is obviously neither possible nor needed in a realistic scenario, as the adversary typically would be interested in the content of the data rather than its ordering. However, to quantify reconstruction quality w.r.t. ground truth images, it is vital that these align.

Consider a batch of N ground truth samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ with corresponding labels $\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}$. The gradient of a loss function $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{w})$ with respect to model parameters $\boldsymbol{w}$ is for a batch computed as we earlier presented in Equation 2.13:

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}_i; \boldsymbol{w}) \tag{3.5}$$

Importantly, this sum is invariant to permutations of the sample indices:

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{w}) = \ell(\boldsymbol{y}', \hat{\boldsymbol{y}}'; \boldsymbol{w}) = \ell(\boldsymbol{y}', f(\boldsymbol{X}'; \boldsymbol{w})) \tag{3.6}$$

for any permutation π of $\{1, \ldots, n\}$ such that $\boldsymbol{X}' = \{\boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{x}_{\pi(n)}\}$ and similarly for $\boldsymbol{y}'$. This means that the optimization process has no incentive to preserve the original order of the samples, as any permutation yields the same gradient.

As a result, the recovered samples $\hat{\boldsymbol{X}}$ and labels $\hat{\boldsymbol{y}}$ from gradient inversion may be arbitrarily permuted versions of the true data. This makes it nontrivial to evaluate the quality of reconstruction in a sample-wise manner.

To address this issue, the linear sum assignment problem (LSAP), also known as the Hungarian algorithm, can be used to optimally match reconstructed images to their corresponding ground truth samples based on a predefined similarity metric. The LSAP seeks to minimize the total cost of assigning each reconstructed image to a ground truth image such that each pair is matched only once. Formally, given a cost matrix $C \in \mathbb{R}^{n \times n}$, where $C_{i,j}$ represents the dissimilarity between reconstructed image i and ground truth image j, the goal is to find a one-to-one assignment that minimizes the total cost:

$$\min_{\pi} \sum_{i=1}^{n} C_{i,\pi(i)}$$

where π is a permutation over $\{1, \ldots, n\}$.

3.2.4 Surrogate Model Extension

As stated in the report Surrogate Model Extension (SME): A Fast and Accurate Weight Update Attack on Federated Learning, Junyi Zhu et al. propose the current state-of-the-art (SOTA) method for attacking model weights rather than gradients to reconstruct training images [27]. The idea of inverting the weights begins with a surrogate model, denoted as $\hat{\boldsymbol{w}}$. For the SME, $\hat{\boldsymbol{w}}$ constitutes model weights which may differ from $\boldsymbol{w}$, as oppose to earlier notation.

The general objective function of the SME can be formulated as:

$$\arg\min_{\hat{\boldsymbol{X}} \in D} \min_{\hat{\boldsymbol{w}} \in \mathcal{W}} \mathcal{L}_{\text{sim}}(\boldsymbol{w}^0 - \boldsymbol{w}^T, \nabla \ell(\hat{\boldsymbol{w}})) \tag{3.7}$$

Where we define:

$$\nabla \ell(\hat{\boldsymbol{w}}) := \nabla_{\hat{\boldsymbol{w}}} \ell(\hat{\boldsymbol{y}}, f(\hat{\boldsymbol{X}}; \hat{\boldsymbol{w}})) \tag{3.8}$$

Here, $\hat{\boldsymbol{X}}$ as usual is the dummy data, which serves as an approximation of the actual training dataset. $\boldsymbol{w}^0$ represent the weights of the global model before the victim conducted any training and $\boldsymbol{w}^T$ is the stolen weights of this victim after a round r of training. Importantly, as reviewed in subsection 2.3.2, the client may perform several steps t steps of training within a round.

The approximated $\hat{\boldsymbol{w}}$ in SME is a linear combination of the initial weights $\boldsymbol{w}^0$ and the trained weights w^T after T local training steps. Instead of only using $\boldsymbol{w}^0$ as our $\hat{\boldsymbol{w}}$, which we saw lead to an approximation discrepancy in subsection 2.3.2, letting the algorithm find a $\hat{\boldsymbol{w}} \in \mathcal{W}$, using a mixing coefficient α, where $\mathcal{W}$ is the linear combination, makes finding a hidden gradient easier:

$$\hat{\boldsymbol{w}} = \alpha \boldsymbol{w}^0 + (1 - \alpha) \boldsymbol{w}^T \tag{3.9}$$

where $\hat{\boldsymbol{w}}$ represents the interpolated weights, defining a model state between the initial and final training stages. The parameter α regulates the proportion of $\boldsymbol{w}^0$ and $\boldsymbol{w}^T$ in the interpolation. Initially, $\alpha_0 = 0.5$, indicating equal weighting of $\boldsymbol{w}^0$ and $\boldsymbol{w}^T$. In Chapter 5 we further discuss why this is reasonable.

SME assumes that the best-reconstructed data $\hat{\boldsymbol{X}}$ should minimize the reconstruction loss function $\mathcal{L}$ when used with the interpolated weights $\hat{\boldsymbol{w}}^*$. The loss function for data reconstruction is defined as:

$$\mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) = \underbrace{1 - \frac{\langle \boldsymbol{w}^0 - \boldsymbol{w}^T, \nabla \ell(\hat{\boldsymbol{w}}) \rangle}{\|\boldsymbol{w}^0 - \boldsymbol{w}^T\| \|\ell(\hat{\boldsymbol{w}})\|}}_{\mathcal{L}_{\text{sim}}} + \lambda \underbrace{\text{TV}(\hat{\boldsymbol{X}})}_{\mathcal{L}_{\text{prior}}} \tag{3.10}$$

where $\mathcal{L}_{\text{sim}}$ denotes the cosine similarity between the true and estimated weight updates, and $\mathcal{L}_{\text{prior}}$ applies total variation (TV) regularization to the dummy data $\hat{\boldsymbol{X}}$. The similarity measure quantifies how closely the true weight change $\boldsymbol{w}^0 - \boldsymbol{w}^T$ aligns with the gradient of the loss function $\ell(\hat{\boldsymbol{w}})$, with cosine similarity ensuring alignment in the direction of weight updates.

The calculated loss with respect to the interpolated weights $\hat{\boldsymbol{w}}$ is then used for the gradient update rule for the dummy data $\hat{\boldsymbol{X}}$:

$$\hat{\boldsymbol{X}}_{i+1} = \hat{\boldsymbol{X}}_i - \eta_{\hat{\boldsymbol{X}}} \nabla_{\hat{\boldsymbol{X}}} \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) \tag{3.11}$$

where $\eta_{\hat{\boldsymbol{X}}}$ is the learning rate for updating the dummy data and $\nabla_{\hat{\boldsymbol{X}}} \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}})$ is the gradient of the loss function with respect to $\hat{\boldsymbol{X}}$. The index i represents the iteration step, where the dummy data $\hat{\boldsymbol{X}}$ is updated iteratively over several steps. This update rule ensures that the generated data $\hat{\boldsymbol{X}}$ gradually approximates the original training data, in expectation, improving over iterations.

The loss also adjusts α to find the best interpolation between the initial and final model weights $\boldsymbol{w^0}$ and $\boldsymbol{w^T}$. The gradient update for α_k is:

$$\alpha_{i+1} = \alpha_i - \eta_\alpha \nabla_\alpha \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) \tag{3.12}$$

where η_α is the learning rate for adjusting α and $\nabla_\alpha \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}})$ is the gradient of the loss function with respect to α. Since the reconstruction loss $\mathcal{L}$ depends on α only through $\hat{\boldsymbol{w}}$, and given Equation 3.9, we can express the gradient as:

$$\nabla_\alpha \mathcal{L}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{w}}} \cdot \frac{\partial \hat{\boldsymbol{w}}}{\partial \alpha}. \tag{3.13}$$

With Equation 3.9, we compute:

$$\frac{\partial \hat{\boldsymbol{w}}}{\partial \alpha} = \boldsymbol{w}^0 - \boldsymbol{w}^T. \tag{3.14}$$

Substituting this into the previous expression, we obtain:

$$\nabla_\alpha \mathcal{L}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{w}}} \cdot (\boldsymbol{w}^0 - \boldsymbol{w}^T). \tag{3.15}$$

In Algorithm 4, each step of SME is clearly presented.

Algorithm 4: Surrogate Model Extension (SME)

Data: Stolen weights $\boldsymbol{w}^0, \boldsymbol{w}^T$, number of iterations K, learning rates $\eta_{\hat{\boldsymbol{X}}}, \eta_\alpha$, loss function $\mathcal{L}_{\text{SME}}$
Result: Reconstructed dummy data $\hat{\boldsymbol{X}}_K$
Initialize: $\hat{\boldsymbol{X}}_0$, $\alpha_0 \leftarrow 0.5$;
for $i = 0$ **to** $K - 1$ **do**
  $\hat{\boldsymbol{w}} \leftarrow \alpha_i \boldsymbol{w}^0 + (1 - \alpha_i)\boldsymbol{w}^T$; `// Interpolate weights`
  $\hat{\boldsymbol{X}}_{i+1} \leftarrow \hat{\boldsymbol{X}}_i - \eta_{\hat{\boldsymbol{X}}} \nabla_{\hat{\boldsymbol{X}}} \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}_i, \hat{\boldsymbol{w}})$; `// Update dummy data`
  $\alpha_{i+1} \leftarrow \alpha_i - \eta_\alpha \nabla_\alpha \mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}_i, \hat{\boldsymbol{w}})$; `// Update interpolation coefficient`
end
Output: Reconstructed data $\hat{\boldsymbol{X}}_K$;

3.2.5 Implementation

The integrated attack targets our Federated Learning network, implemented in Flower, by stealing the weights of each client and the global server at every round. The attack is executed simultaneously across all clients and the stolen weights is saved to restore the training data. Using these stolen weights, the SME attack can then be performed to restore the training data.

By leveraging PyTorch's Autograd functionality [24], we can efficiently compute the gradient of the reconstruction loss $\mathcal{L}$, which is essential for performing DL attacks. Autograd records the operations performed on tensors that have `requires_grad=True` and builds a dynamic computational graph. During the backward pass, it traverses this graph in reverse to compute the gradients of each tensor with respect to the loss using backpropagation. For further details, we refer to our GitHub repository [19].

3.2.6 Deep Leakage Experiments

In this subsection, the experiments for DL are specified along with their corresponding settings. Six experiments have been conducted:

1. Experiment 2.1 Decreasing Batch Size
2. Experiment 2.2 Increasing Epochs
3. Experiment 2.3 Maximal Local Training
4. Experiment 2.4 Local Data Size
5. Experiment 2.5 Training State
6. Experiment 2.6 Image Resolution

Additionally, in some experiments, we apply the following federated learning strategies for comparison:

- **Full FedAVG:** Clients perform multiple local training steps before sending their model updates to the server. A relatively high number of epochs is used, along with a full batch size.
- **FedAVG = FedSGD:** Clients do not perform any local training. Instead, they compute their updates in a single local step and send them directly to the server. This represents the intersection point discussed in Subsection 2.3.2, where the transition to weight-based updates makes FedAVG equivalent to FedSGD.
- **Realistic FedAVG:** A more practical version of FedAVG that we propose, where clients perform a limited number of local training steps. It is intended to better reflect the constraints of real-world federated systems, where clients may have limited computational resources or time. Naturally, this is a simplification, as the specifics of any learning scheme depend on the task and context. What we mean in practice is that a *moderate* level of training is used—less than Full FedAVG but more than FedSGD.

3.2.6.1 General Configurations

In Section 2.2, some settings were used which induces challenges for DL. Especially, three major changes have been made for the FL setup:

1. A transition from the LeNet5 model to a CNNCifar model has been made.
2. Instead of using Adam, vanilla SGD is employed.
3. The local data size N varies but is typically much lower in the following experiments compared to the ones in Section 2.2 for FL.

Model The model used in the following DL experiments is a CNN tailored for the CIFAR-10 dataset, referred to as CNNCifar. Its architecture is described in Subsection 3.1.2. Junyi Zhu et al. [27] demonstrate that the performance of the SME attack varies significantly across different model architectures, emphasizing the importance of proper parameter tuning. To mitigate the need for extensive and computationally expensive hyperparameter tuning, this project adopts CNNCifar for all DL experiments. Moreover, using the same model facilitates comparison with the results reported in [27].

Optimizer The federated network in the following experiment uses SGD as its optimizer, with a learning rate of 0.004, replacing the previously used Adam optimizer. This switch is motivated by the fact that while Adam is a more sophisticated optimizer often leading to improved training performance, it also complicates DL attacks both theoretically and practically. Importantly, the SME attack is specifically designed for SGD, and its effectiveness with Adam has not been explored. Furthermore, Junyi Zhu et al. [27] base several of their theorems and proofs on the assumption of using SGD. The chosen learning rate follows recommendations from the LEAF benchmark, which suggests 0.004 as a suitable value for achieving effective training with this network in federated learning settings [5]. The loss function used is cross-entropy, which remains unchanged despite the switch in optimizer.

Number of Clients and Local Data Size The performance of a federated network is highly influenced by the amount of data it is exposed to during training. In the FL context, this data availability is primarily governed by the number of clients and their respective local dataset sizes. In previous FL experiments, strong performance was achieved by using a relatively large local dataset size ($N = 5000$) across a small number of clients (10 in total). However, this setup poses challenges for DL attacks, as reconstructing from $N = 5000$ is unrealistic.

To address this, one could theoretically reduce the local data size to around $N \approx 32$, while maintaining the total dataset size by increasing the number of clients to approximately 1500. Although this configuration is reflective of real-world scenarios, simulating such a large number of clients is computationally expensive and infeasible within the constraints of this project.

As a compromise, we restrict our experiments to 10 clients with small local datasets,

accepting the resulting drop in federated performance. This setup can still be considered realistic, as an adversary could plausibly launch DL attacks on a small subset of neighboring clients within a larger network. Nonetheless, this limitation prevents us from fully exploring the trade-off between DL attack feasibility and FL performance under large-scale conditions.

Limit to Image Reconstruction In this project, the primary focus of the DL attack is image reconstruction, while label reconstruction is deliberately excluded from the scope and treated as a separate challenge. This separation allows us to focus exclusively on the complexity of reconstructing input images.

To define the constraints of our setup, we make the key assumption that the adversary has access to the ground truth labels, denoted by y. Rather than assuming the labels were intercepted directly, we posit that a successful label restoration attack has been performed beforehand. While this simplification sidesteps the inherent difficulty of reconstructing images without known labels, it allows us to isolate and study the image reconstruction process in depth.

As discussed in Subsection 3.2.3, knowledge of the label distribution can, in practice, be just as informative as knowing the labels themselves. In gradient-based DL attacks, such distributions may be inferred through techniques like the one illustrated in Equation 3.4. Although no analogous method currently exists for weight-based attacks, we hypothetically assume that such a method could be developed and has already yielded the true labels in our scenario.

The following parameters remain fixed throughout all experiments in the federated network:

Parameter	**Value**
Model	CNNCifar
Optimizer	SGD
Loss Function	Cross-Entropy
Learning Rate	0.004
Number of Clients	10
Strategy	FedAVG
Data Distribution	IID

We refer to Appendix A.2.2 for the full configuration.

3.2.6.2 Evaluation

Evaluating the quality of reconstructed images is challenging. To address this, we draw inspiration from the FEDLAD framework (Federated Evaluation of Deep Leakage Attacks and Defenses) by Isaac Baglin et al., which offers a diverse range of evaluation metrics across multiple datasets.[2]

FEDLAD offers a wide range of metrics, but in this report the main focuses will be analyzing the following metrics:

- **PSNR** (Peak Signal-to-Noise Ratio)
- **LPIPS** (Learned Perceptual Image Patch Similarity)
- **SSIM** (Structural Similarity Index Measure)

PSNR Peak Signal-to-Noise Ratio evaluates how much noise is present in a reconstructed image by comparing the original signal to the distortion introduced during reconstruction. Higher PSNR values indicate that the restored image is closer to the original with less visible noise.

LPIPS Learned Perceptual Image Patch Similarity (LPIPS) is designed to measure perceptual similarity between two images by comparing their deep feature representations rather than raw pixel values. This approach aims to mimic human visual perception providing a "perceptual distance" that reflects image similarity in a way that aligns with human judgment[25]. A lower value indicates better recovery, as the reconstructed image appears more similar to the original.

SSIM Structural Similarity Index Measure evaluates the similarity between two images by comparing their luminance, contrast, and structural patterns, offering a more complete view of image quality. Values closer to 1 indicate that the reconstructed image is highly similar to the original in terms of structure, luminance, and contrast.

Averaging All experiments below are conducted on each of the 10 clients in the federated network, and their corresponding metrics are averaged. Since some experiments involve only a small number of images, this averaging approach improves the robustness of our evaluation by reducing the impact of sample anomalies.

3.2.6.3 Decreasing Batch Size

Description This experiment investigates how reducing the local batch size, while keeping the number of epochs fixed at $E = 1$, influences the quality of reconstructed images. Lowering the batch size increases the number of local weight updates performed by each client during a single communication round, as the model processes smaller subsets of the local dataset in each step. This setup makes it possible to observe how batch size alone affects how much information about the original training data is preserved in the model's weights, which in turn impacts the quality of the reconstructed images.

Experiment 2.1:	**B**	**E**	**Round**	**N**	**T**
	[8(full), 4, 2, 1]	1	1	8	[1, 2, 4, 8]

Motivation & Hypothesis The purpose of this experiment is to evaluate how batch size influences the attack's ability to retain information necessary for image reconstruction. Smaller batch sizes result in more frequent weight updates, making it harder for the attack to recover details in the images. We hypothesize that decreasing the batch size will lead to a decline in reconstruction quality, as more frequent updates make it harder for the attack to recover the original inputs.

3.2.6.4 Increasing Epochs

Description This experiment investigates the effect of varying the number of local training epochs on the ability to reconstruct original images. By adjusting the number of epochs, we observe how increasing local training impacts the model's weights and, subsequently, the success of the reconstruction attack. The batch size is fixed at 8 (full), and we evaluate how different epochs influence restoration quality by performing the attack after one round.

Experiment 2.2:	**B**	**E**	**Round**	**N**	**T**
	8(full)	[1, 5, 10, 20]	1	8	[1, 5, 10, 20]

Motivation & Hypothesis The reason why this experiment is conducted is to examine how varying the number of local epochs affects the ability to recover original inputs from stolen model weights. It is hypothesized that increasing the number of epochs leads to more weight updates, which could reduce the success of the reconstruction attack by update the model further from its initial state.

3.2.6.5 Maximal Local Training

Description This experiment is designed to evaluate the effect of intensified local training on the ability to reconstruct original images from stolen model weights. By combining a minimal batch size with an increased number of local epochs, the setup results in the highest number of local weight updates observed in this study.

Experiment 2.3:	**B**	**E**	**Round**	**N**	**T**
	1	20	1	8	160

Motivation & Hypothesis The purpose of this experiment is to examine how increased local training affects the model's vulnerability to reconstruction attacks. We hypothesize that the combination of a small batch size and a high number of local epochs will weaken the attack's reconstruction ability, as the model parameters are pushed further away from their original state, making it increasingly difficult to recover the input image.

3.2.6.6 Local Data Size

Description In this experiment, we investigate the relationship between the amount of local training data N and the restoration quality of the images. Each of the ten

clients is assigned an equal amount of local data, and the test is repeated with varying data sizes. By gradually increasing the data size, we assess how larger local datasets impact the ability to reconstruct images from the model updates. The data sizes N tested in this experiment are: [8, 16, 32, 64, 96, 128, 160] and all attacks are conducted after the first round of training.

Experiment 2.4:	**B**	**E**	**T**
Full FedAVG	1	20	[160, 320, 640, 1280, 1920, 3200]
FedAVG = FedSGD	Full	1	1
Realistic FedAVG	[2, 4, 8, 16, 24, 32, 40]	5	20

Motivation & Hypothesis The purpose of this experiment is to explore how varying the amount of local training data influences the attack's vulnerability to reconstruction images. We hypothesize that as the amount of local data increases, the model weights become more generalized, making it harder for the attack to recover the original images. We expect that larger local datasets will result in a reduced ability to reconstruct the images from the stolen model weights.

3.2.6.7 Training State

Description In this experiment, we examine how the number of communication rounds affects the quality of image restoration. We run the attack over several rounds, while keeping other parameters constant. Ten clients are used in each test, and the results are averaged across them. The goal is to examine if the attack is affected by the round in which it is performed and how this impacts the restoration quality. The experiment runs for 50 rounds, with an attack occurring at each round.

Experiment 2.5:	**B**	**E**	**Rounds**	**N**	**T**
	2	5	[1:1:50]	8	20

Motivation & Hypothesis The purpose of this experiment is to examine how the number of communication rounds impacts the attack's vulnerability to reconstruct images. We hypothesize that the timing of the attack within the communication rounds will influence the restoration quality. Specifically, we expect that performing the attack in later rounds, when the model has undergone more updates, will result in a lower quality of image restoration compared to attacks performed in earlier rounds.

3.2.6.8 Image Resolution

Description To understand how input resolution affects model performance, we conduct a very simple experiment using three different image sizes:128×128, $256 \times$

256 and 1024×1024. The goal is to evaluate how resolution influences the quality of the restored image. In this experiment, the evaluation is performed on a single selected image from each dataset in order to isolate the effect of resolution on restoration quality. For these experiments, we used three publicly available image datasets at varying resolutions: `imagenet-1k-128x128`, `celeba-hq-256x256`, and `oilpaint_1024x1024`.[1]

Motivation & Hypothesis The purpose of this experiment is to examine how different input resolutions affect the ability to restore images from stolen model weights. We hypothesize that higher resolution images will make the model's task more challenging, as they contain more detailed information. As a result, we expect that the quality of the restored images will decrease as the resolution increases, due to the increased complexity of the model's parameters.

[1]Datasets available at:
`https://huggingface.co/datasets/benjamin-paine/imagenet-1k-128x128`,
`https://huggingface.co/datasets/korexyz/celeba-hq-256x256`,
`https://huggingface.co/datasets/arsalanaa/oilpaint_1024x1024`

4
Results

This chapter presents the results corresponding to the methodology outlined in Chapter 3, with focus on the experiments described in Section 4.2. Brief comments and interpretations are provided to highlight key observations, but a more thorough analysis and discussion of the results, along with their underlying explanations, is deferred to Chapter 5.

4.1 Federated Learning Experiments

In Section 4.2, we proposed four experiments to evaluate the performance of FL under different training configurations:

1. Experiment 1.1: Intersection Between FedSGD & FedAVG
2. Experiment 1.2: Decreasing Batch Size
3. Experiment 1.3: Increasing Epochs
4. Experiment 1.4: Optimal Training

The primary objective of these experiments is to investigate how the training state of a federated network affects its overall performance.
The following table summarizes the experimental setup parameters, which remain consistent across all FL experiments in this section:

Parameter	**Value**
Model	LeNet
Optimizer	Adam
Loss Function	Cross-Entropy
Learning Rate	0.001
Number of Clients	10
Local Data Size N	5000
Strategy	FedAVG
Distribution	IID
Rounds	200

Table 4.1: Federated learning experiment setup parameters

We refer to Appendix A.2.1 for the complete configuration details.

4.1.1 Intersection Between FedSGD & FedAVG

Experiment 1.1:	**B**	**E**	**T**
	Full	1	1

In Figure 4.1, three especially interesting phenomenons can be observed:
1. Remarkably similar performance on training and validation data.
2. Oscillating behavior.
3. Slow convergence in terms of rounds.

Firstly, the similarity between the training and validation data is interesting from an ML perspective as this type of behavior is rarely seen. Often instead, there is a discrepancy between the performances. We believe there are two possible reasons to why this is the case; the IID and the large batch size. The IID ensures that the validation set is sampled from the same distribution as the training set. If not, we would see significantly different performances. But more importantly, the large batch size inherent from using FedSGD further neglects sample-specific differences between the validation and training set. Additionally, a larger batch size implies fewer steps in between validation, which describes why train and validations accuracy is tightly bound.

Secondly, the large oscillations in accuracy suggest that the model struggles to converge smoothly. This could stem from high variance in gradient updates, sensitivity to the learning rate, or the lack of mini-batch stochasticity as a result of the minimal local training, which typically helps stabilize training dynamics. We suspect this behavior is a consequence of oscillating around a local optimum.

Finally, as we will see when comparing with more local training, this learning scheme demands several communication rounds rendering slow convergence.

4.1.2 Decreasing Batch Size

Experiment 1.2:	B	E	T
	[Full, 64, 32, 16, 8, 1]	1	[125, 250, 500, 1000, 8000]

In Figure 4.2, we observe how the batch size influences the validation accuracy. Notably, the `Batch full` setting corresponds to the same configuration as in Figure 4.1, and it performs the worst when compared to the smaller batch sizes. These results indicate that the stochasticity introduced by smaller batches mitigates the oscillatory behavior previously observed and leads to faster convergence.
Most importantly, the results suggest that a *moderate* batch size is preferable over the extremes (i.e., batch size 1 and full batch). Specifically, batch size 8 achieves better performance up until round 40, after which it is overtaken by batch size 16, which reaches the highest overall performance at round 63. Interestingly, as training continues beyond round 78, batch size 64 begins to outperform the others.

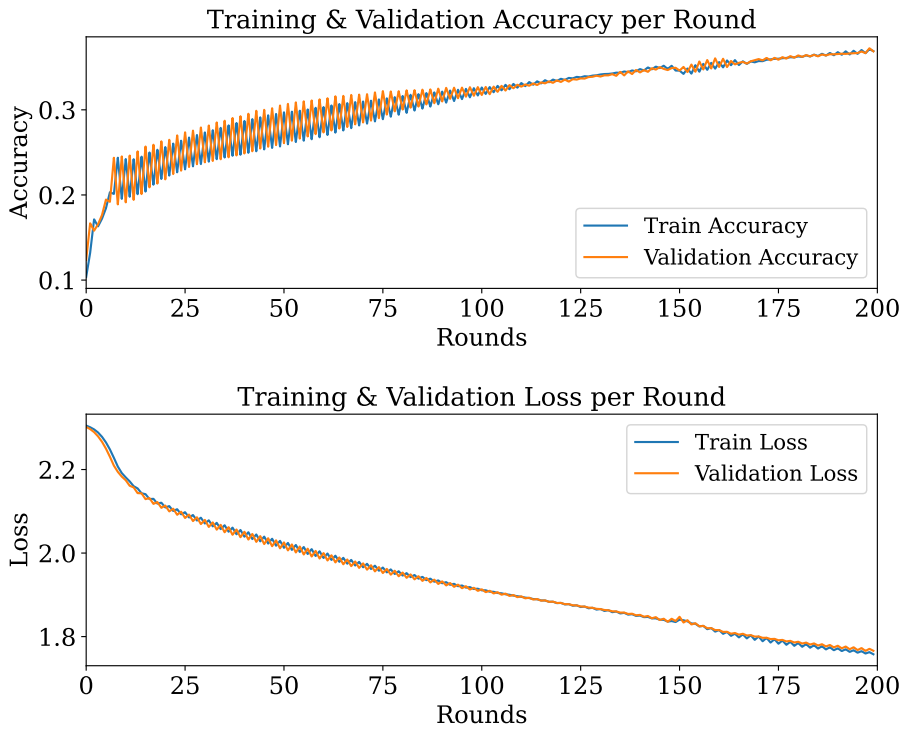

Figure 4.1: Training and validation performance of the LeNet5 model over 200 communication rounds in a FL setup with 10 clients, using a full batch per round and 1 epoch per client with $N = 5000$.

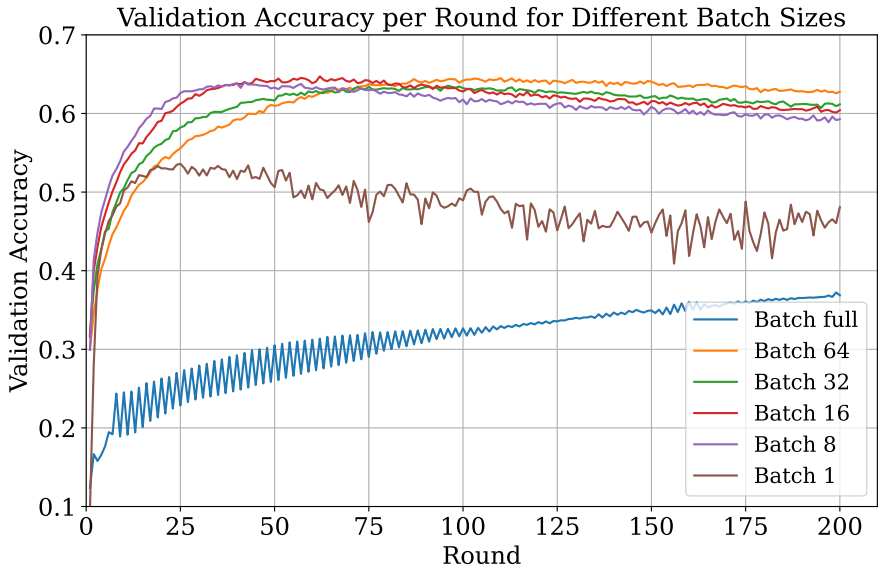

Figure 4.2: Validation accuracy across $B = [full, 64, 32, 16, 8, 1]$ of the LeNet5 model over 200 communication rounds in a FL setup with 10 clients, using 1 epoch per client with $N = 5000$.

4.1.3 Increasing Epochs

Experiment 1.3a:	B	E	T
	Full	[1, 5, 10]	[1, 5, 10]

Figure 4.3 shows the results for a batch size set to 'full' with varying numbers of epochs. The results clearly demonstrate that increasing the number of epochs leads to higher accuracy when the batch size remains fixed. This is expected, as more epochs allow for additional local training before global aggregation, enabling the model to generalize better.

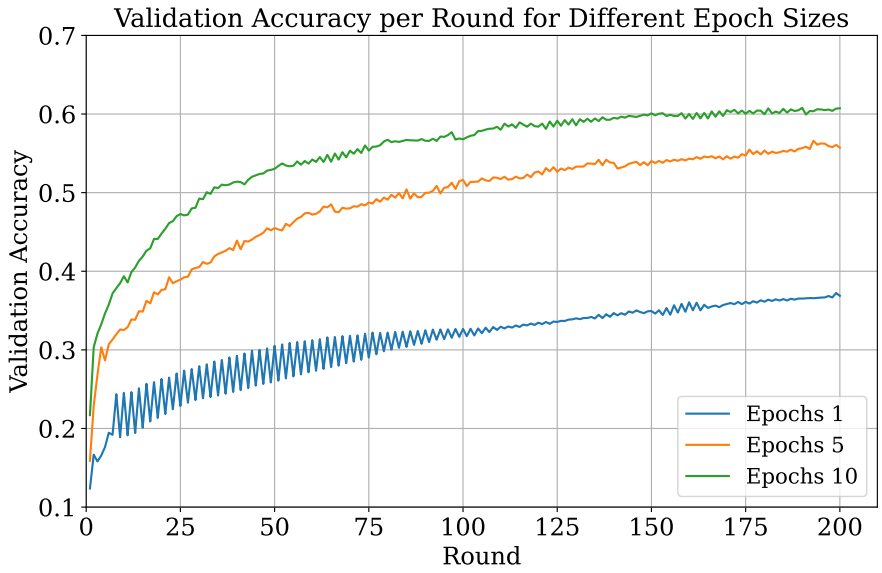

Figure 4.3: Validation accuracy across $E = [1, 5, 10]$ over 200 communication rounds in a FL setup with 10 clients, using full batch size with $N = 5000$.

Experiment 1.3b:	B	E	T
	1	[1, 5, 10]	[8000, 40 000, 80 000]

In Figure 4.4, the results illustrate how the accuracy remains around 0.5 across all the different epochs. Since the model is trained with a batch size of 1, increasing the number of epochs leads to rapid overfitting in all cases. The plot further highlights that using 5 epochs results in the best performance, while increasing to 10 epochs does not provide any noticeable improvement and instead leads to overfitting. This suggests that, in this scenario, a higher number of epochs is not necessarily preferable, as it can cause the model to overfit rather than generalize effectively.

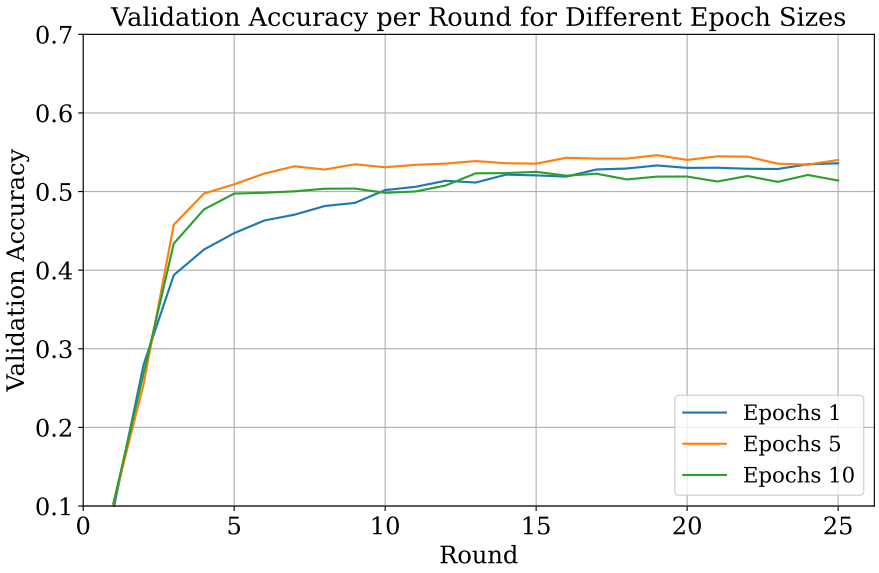

Figure 4.4: Validation accuracy across $E = [1, 5, 10]$ over 25 communication rounds in a FL setup with 10 clients, using batch size 1 with $N = 5000$.

4.1.4 Optimal Training

Experiment 1.4:	**B**	**E**	**T**
	16	[1, 5, 10]	[500, 2500, 5000]

This experiment was conducted using a batch size of 16, as Experiment 1.2 concluded that this configuration yielded the highest accuracy. Figure 4.5 illustrates how the validation accuracy evolves with an increased number of local epochs. As expected, the accuracy peaks rapidly due to the combination of a small batch size and multiple training iterations. However, for both 5 and 10 local epochs, the model begins to overfit relatively early.

Despite this tendency to overfit, increasing the number of local epochs can still be a convenient strategy—particularly in privacy-sensitive scenarios, as it reduces the frequency of model updates while maintaining a relatively high level of accuracy.

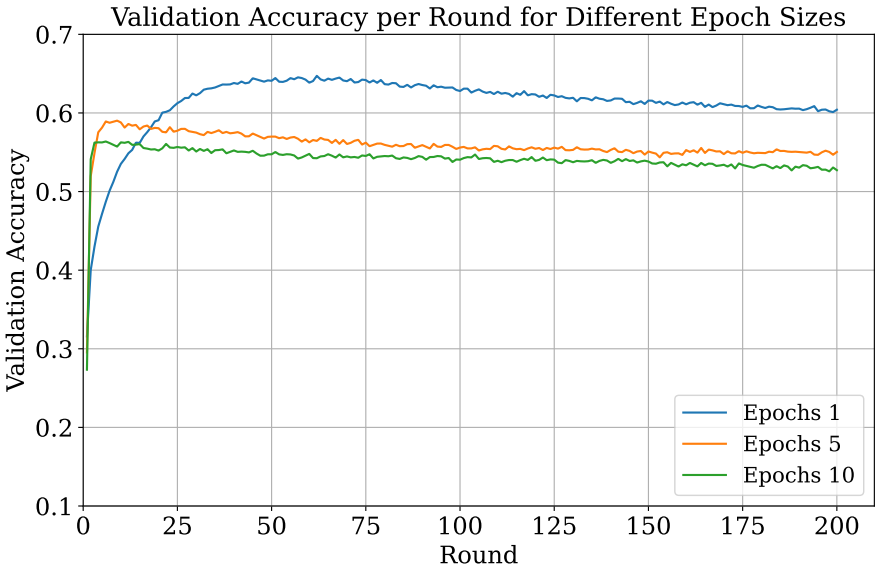

Figure 4.5: Validation accuracy for batch size 16 with different numbers of local epochs (1, 5, and 10) over 200 rounds.

4.2 Deep Leakage Experiments

In Section 4.2 , we proposed six experiments to investigate the performance of DL within our constructed framework for different training configurations:

1. Experiment 2.1 Decreasing Batch Size
2. Experiment 2.2 Increasing Epochs
3. Experiment 2.3 Maximal Local Training
4. Experiment 2.4 Local Data Size
5. Experiment 2.5 Training State
6. Experiment 2.6 Image Resolution

The purpose of these experiments is to investigate how the training state, data itself, and general FL setting affects the adversary's overall performance in terms of quality of reconstructing images.

Experimental setup parameters, which are consistent for all the DL experiments in this section:

Parameter	**Value**
Model	CNN-CIFAR
Optimizer	SGD
Loss Function	Cross-Entropy
Learning Rate	0.004
Number of Clients	10
Strategy	FedAVG
Data Distribution	IID

We refer to Appendix A.2.2 for the full configuration.

All experiments below are conducted on each of the 10 clients in the federated network, and their corresponding metrics are averaged. Since some experiments involve only a small number of images, this averaging approach improves the robustness of our evaluation by reducing the impact of sample anomalies.

4.2.1 Decreasing Batch Size

Experiment 2.1:	**B**	**E**	**Round**	**N**	**T**
	[8(full), 4, 2, 1]	1	1	8	[1, 2, 4, 8]

To understand the relationship between local training dynamics and image reconstruction quality, this experiment explores the effect of decreasing the local batch size while keeping other parameters constant. Reducing the batch size forces the model to perform more frequent weight updates on smaller subsets of data during each local training round. This setup allows us to isolate and examine how batch size alone influences the success of reconstructing original images from the model's stolen weights.

Table 4.2 presents the results for varying the local batch size while keeping the number of epochs fixed to $E = 1$. The first row represents the scenario where FedAVG behaves equivalently to FedSGD, meaning the entire local dataset is used as a single batch and only one epoch is performed. As the batch size decreases, the local client performs more updates per communication round, and the results indicate a clear but small reduction in restoration quality according to the applied evaluation metrics. This suggests that increased local training, driven by smaller batch sizes, leads to weight updates that make it harder to recover the original inputs.

Table 4.2: Evaluation metrics (PSNR, SSIM, LPIPS) for restored images at different local batch sizes B, after one epoch of training. Results are averaged over 10 clients in a federated learning setting on the CIFAR-10 dataset.

Batch size	**Average PSNR ↑**	**Average SSIM ↑**	**Average LPIPS ↓**
8 (Full)	20.721	0.744	0.065
4	20.546	0.732	0.071
2	20.395	0.716	0.076
1	19.945	0.690	0.084

4.2.2 Increasing Epochs

Experiment 2.2:	B	E	Round	N	T
	8(full)	[1, 5, 10, 20]	1	8	[1, 5, 10, 20]

This experiment examines how the number of local training epochs affects the ability to reconstruct original images from stolen model weights. Increasing the number of epochs results in more weight updates before aggregation, which can influence restoration quality. The evaluation is based on a fixed set of $N = 8$ images, providing a consistent comparison across configurations.

Table 4.3 presents the results for the experiment evaluating the impact of the number of local epochs. Similar to the batch size experiment, the first row represents a scenario where FedAVG is equivalent to FedSGD, with only one local update per round. Compared to the results observed in the batch size experiment, the decrease in PSNR and SSIM, along with the increase in LPIPS, appears to be less pronounced when adjusting the number of local epochs. Interestingly, the PSNR even increase in the second row, where 5 local epochs are used. This could be due to the fact that PSNR focuses solely on pixel-level differences, which may not fully reflect perceptual quality. These results suggest that changes in batch size may lead to a greater divergence than the variations caused by increasing the number of local epochs.

Table 4.3: Evaluation metrics (PSNR, SSIM, LPIPS) for reconstructed images at different local epoch sizes (E), using batch size $B = 8$. Results are averaged over 10 clients in a federated learning setting on the CIFAR-10 dataset.

Epoch size	Average PSNR ↑	Average SSIM ↑	Average LPIPS ↓
1	20.721	0.744	0.065
5	20.728	0.742	0.066
10	20.541	0.729	0.069
20	20.168	0.710	0.074

4.2.3 Maximal Local Training

Experiment 2.3:	B	E	Round	N	T
	1	20	1	8	160

This experiment explores the combined effect of batch size and number of epochs in order to maximize local training within a single communication round. The goal is to evaluate how extreme local training conditions influence the model's susceptibility to input reconstruction attacks. The evaluation is again performed using a fixed set of 8 images to allow for a consistent comparison across configurations.

The final experiment involving epochs and batch size was designed to maximize

local training effort. As shown in Table 4.4, this was achieved by setting the number of local epochs to $E = 20$ and using a batch size of $B = 1$, which results in the highest number of local weight updates before aggregation in this investigation. Under these conditions, the evaluation metrics indicate a significant decline in reconstruction quality, with noticeably lower PSNR and SSIM values, and a substantial increase in LPIPS. These results further support the observation that increased local training reduces the SME attack's capacity to restore images.

Table 4.4: Evaluation metrics (PSNR, SSIM, LPIPS) for reconstructed images using local training with $E = 20$ epochs and batch size $B = 1$ on the CIFAR-10 dataset. Results are averaged over 10 clients in a federated learning setting.

Epochs	**Batch Size**	**Average PSNR ↑**	**Average SSIM ↑**	**Average LPIPS ↓**
20	1	16.569	0.456	0.172

To illustrate the visual differences, Figure 4.6 presents reconstructed images under two different local training settings. The top row shows reconstructions produced when using the intersection point where FedAVG behaves equivalently to FedSGD ($B = \text{full}$, $E = 1$). The middle row shows reconstructions obtained after extended local training using the full FedAVG setup ($B = 1$, $E = 20$). The bottom row displays the corresponding original ground truth images.

As seen in Figure 4.6, the reconstructions in the top row retain more structural details and visual similarity to the original images. In contrast, the middle row which represents the case with more extensive local training shows less defined structures and contours, indicating a reduction in the model's ability to retain detailed input features as local training increases.

Figure 4.6: Comparison of reconstructed images under different settings. The top row contains the ground truth images. The middle row shows reconstructions using the intersection of FedSGD and FedAVG with $B = \text{full}, E = 1$. The bottom row presents reconstructions from full FedAVG with $B = 1, E = 20$.

4.2.4 Local Data Size

Experiment 2.4:	B	E	T
Full FedAVG	1	20	[160, 320, 640, 1280, 1920, 3200]
FedAVG = FedSGD	Full	1	1
Realistic FedAVG	[2, 4, 8, 16, 24, 32, 40]	5	20

This experiment was designed to investigate how the amount of local data N per client influences the quality of reconstructed images. Specifically, the focus was on evaluating how increasing the local dataset size affects the ability of the SME attack to recover original inputs from the model's stolen weights.

The results are presented in Figure 4.7. Across all tested configurations, including the intersection point where FedAVG equals FedSGD ($E = 1$, $B =$ full), the full FedAVG case ($E = 20$, $B = 1$), and a more realistic setting ($E = 5$, $B = N/4$), the reconstruction quality consistently declines as the local data size N increases. This trend suggests that when each client trains on a larger local dataset, the weights become less reflective of the individual samples, reducing the SME attack's ability to reconstruct the original images.

The Full FedAVG setting shows a consistent drop in PSNR as the local data size increases. The trend follows a similar shape to the FedAVG = FedSGD and the realistic FedAVG configurations, but the PSNR values for Full FedAVG remain lower by a few points across all tested data sizes. In addition to the overall decline, the gap between the different configurations remains stable across the tested data sizes. For all cases, smaller local datasets result in higher PSNR values, while increasing the number of samples per client leads to a steady reduction in reconstruction quality. This consistent pattern highlights the influence of both local training settings and data volume on the outcome of image restoration.

4.2.5 Training State

Experiment 2.5:	B	E	Rounds	N	T
	2	5	[1:1:50]	8	20

This experiment aimed to investigate how the number of communication rounds influences the quality of reconstructed images. The focus was on evaluating whether the round at which the SME attack is performed impacts its ability to recover original inputs from the model's stolen weights. The experiment was conducted over 50 rounds, with an attack executed at each round.

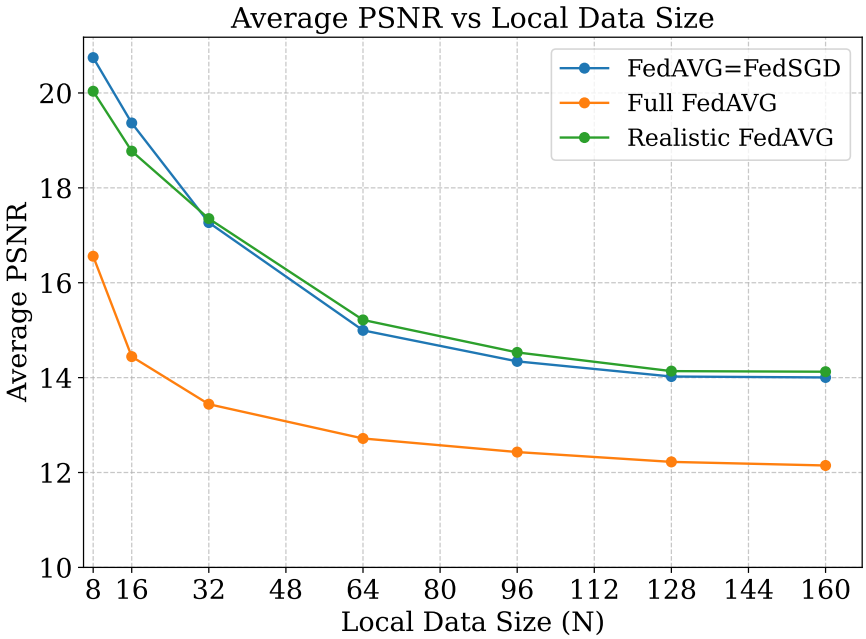

Figure 4.7: Average reconstructed image quality (PSNR) vs. local data size N on CIFAR-10, averaged over 10 clients with three settings, Intersection point for FedAVG and FedSGD:($E = 1, B = \text{full}$), Full FedAVG: ($E = 20, B = 1$) and a realistic setting for FedAVG: ($E = 5, B = N/4$). The data sizes used for each setting $N \in \{8, 16, 32, 64, 96, 128, 160\}$.

The results of this experiment are presented in Figure 4.8. Under a realistic FedAVG setting, the test was conducted using 10 clients, each holding a local dataset of $N = 8$ images. During each communication round, clients performed local training with a batch size of $B = 2$ and $E = 5$ local epochs before model aggregation. The evaluation shows a consistent decline in reconstruction quality as the number of training rounds increases. This illustrates the effect of performing the attack at different stages of the federated learning process, where attacks conducted in earlier rounds result in higher reconstruction quality compared to those executed after several rounds.

The PSNR values reflect this trend, starting at around 20 in the initial rounds and gradually dropping below 17 as the number of rounds approaches 50. The decrease in reconstruction quality is steady during the initial stages of training, with no significant fluctuations between rounds. However, this decline plateaus after approximately 25 communication rounds, indicating that further updates no longer result in notable degradation or improvement in reconstruction performance.

These findings suggest that the model begins to overfit after around 25 communication rounds, which can be seen in Figure 4.9. While the training accuracy continues to increase and eventually reaches a high level of confidence, the validation accuracy plateaus. This is likely due to the small local dataset size, with only

8 images per client, which limits the model's ability to generalize. As a result, even with continued training, the validation performance does not improve.

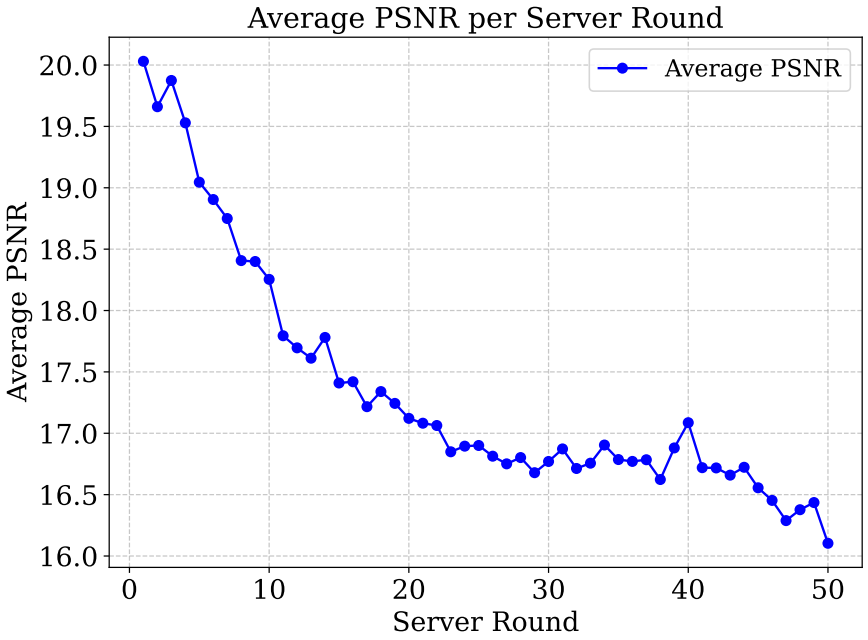

Figure 4.8: Average reconstructed image quality (PSNR) over communication rounds for $N = 8$ local images per client on CIFAR-10, averaged across 10 clients. Each client performs local training with batch size $B = 2$ and $E = 5$ local epochs per round.

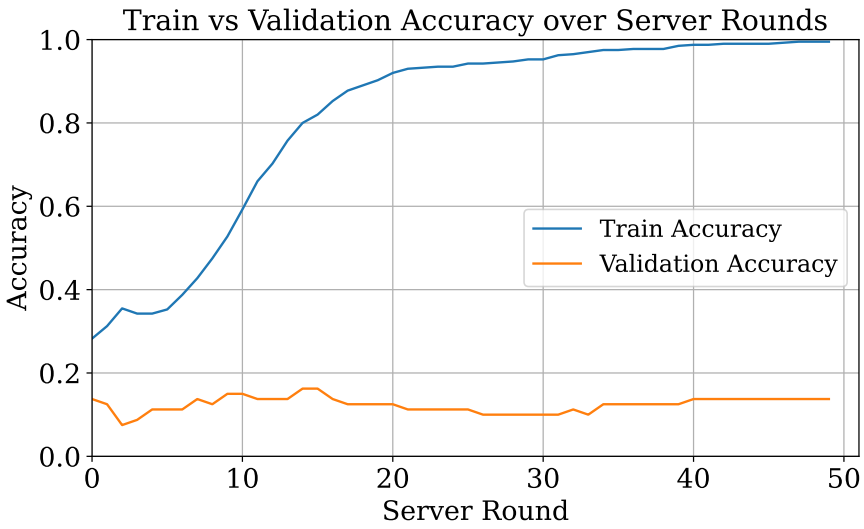

Figure 4.9: Training and validation accuracy over communication rounds for $N = 8$ local images per client on CIFAR-10, averaged across 10 clients. Each client performs local training with batch size $B = 2$ and $E = 5$ local epochs per round. Accuracy is reported for both training and validation sets across every round.

4.2.6 Image Resolution

To understand how input resolution affects model performance, we conducted experiments using three different image sizes: 128×128, 256×256, and 1024×1024. The aim was to evaluate how resolution influences the quality of the restored images under the SME attack method.

The results, illustrated in Figure 4.10, show that both 128×128 and 256×256 images yield reasonably good restoration quality. However, at 256×256, some pixel-level artifacts begin to appear, including slight color distortion and blurriness. When the resolution increases to 1024×1024, the attack is no longer able to restore a coherent image. Although the restored outputs still exhibit some structural resemblance to the original images, the overall quality is noticeably poor.

Figure 4.10: Impact of input resolution on model performance across three datasets:imagenet-1k-128x128, celeba-hq-256x256, and oilpaint_1024x1024. The results illustrate how varying image sizes (128×128, 256×256, and 1024×1024) affect learning quality, convergence speed, and generalization.

5
Discussion

This chapter presents and analyzes the experimental findings concerning the application of FL and the implementation of DL attacks within an FL framework. Additionally, future work is outlined, and finally, ethical considerations related to this study are discussed.

5.1 Federated Learning with FedAVG

The Federated Learning experiments on the CIFAR-10 dataset yielded several interesting insights. In the initial experiment, we investigated the point at which the performance of FedSGD and FedAvg intersect. This setup, which involved minimal local training on each client (one gradient update per round), required a large number of communication rounds and slow convergence. This highlights the inefficiency of relying only on global updates without sufficient local learning.

As we increased the amount of local training performed by each client before aggregation, the convergence time in a global perspective with respect to rounds, decreased heavily. This trend suggests that allowing clients to perform more local updates enables the global model to benefit more effectively from each client's unique data, accelerating convergence. It shows the collaborative learning power in FL systems when clients are allowed to contribute more meaningful updates. However, this benefit has limits. When the number of local training epochs became too high, the model began to overfit to local data, leading to a decline in generalization performance. This trade-off between local learning and global model generalization is crucial for designing effective FL networks. These findings are well illustrated in Figure 4.2, where we can clearly observe the effect of different local training regimes on convergence speed and final accuracy.

5.2 Deep Leakage on Model Weights

Success of DL decrease as more training is induced From the results of Experiments 2.1, 2.2 and 2.3 in Subsection 4.2 we recognized a clear trend in accordance with our hypothesis in Subsection 3.2.6 along with the theory regarding transition to weights in Subsection 2.3.2; more local training worsens quality of reconstructed images from DL attacks across all metrics (PSNR, LPIPS & SSIM).

We can try to understand this by looking at the minimization of $\mathcal{L}_{SME}$ as an arbitrary gradient-based DL attack but with a sophisticated gradient approximation:

$$\mathcal{L}_{\text{SME}}(\hat{\boldsymbol{X}}, \hat{\boldsymbol{w}}) = \underbrace{1 - \frac{\langle \boldsymbol{w}^0 - \boldsymbol{w}^T, \nabla \ell(\hat{\boldsymbol{w}}) \rangle}{\|\boldsymbol{w}^0 - \boldsymbol{w}^T\| \|\ell(\hat{\boldsymbol{w}})\|}}_{\mathcal{L}_{\text{sim}}} + \lambda \underbrace{\text{TV}(\hat{\boldsymbol{X}})}_{\mathcal{L}_{\text{prior}}}$$

Where $\nabla \ell(\hat{\boldsymbol{w}})$ is the sophisticated gradient approximation.

Then, naturally, we can shift our focus to try to understand how this sophisticated approximation becomes increasingly more difficult to make, as more local training is added in between weight observations in a round from $\boldsymbol{w}^0$ to $\boldsymbol{w}^T$. Equation 2.31 In Subsection 2.3.2 declared the simplest case:

$$\nabla \ell(\boldsymbol{w}^0) \approx \frac{\boldsymbol{w}^0 - \boldsymbol{w}^T}{T\eta}$$

which held exactly if and only if $T = E(B/N) = 1$ and the learning rate was non-adaptive.

We believe that the decreasing restoration quality from the SME DL attack, as more training is added, is a direct consequence of this decaying approximation. However, there are more nuances to this. The SME attack aims to find a $\hat{\boldsymbol{w}}$ in $\mathcal{W}$ defined as the linear combination of $\boldsymbol{w}^0$ and $\boldsymbol{w}^T$:

$$\hat{\boldsymbol{w}} = \alpha \boldsymbol{w}^{\boldsymbol{0}} + (1 - \alpha) \boldsymbol{w}^{\boldsymbol{T}}$$

And to understand what really happens under the hood, imagine a victim's model taking a path in weight space from $\boldsymbol{w}^0$ to $\boldsymbol{w}^T$ with several hidden time steps t. If we find a gradient with corresponding $\hat{\boldsymbol{w}}$ which is parallel with $\boldsymbol{w}^0 - \boldsymbol{w}^T$, we find a minimizer to $\mathcal{L}_{sim}$. We believe that finding such a $\hat{\boldsymbol{w}}$ is easier when the path from $\boldsymbol{w}^0$ to $\boldsymbol{w}^T$ is close to the the linear combination of $\boldsymbol{w}^0$ and $\boldsymbol{w}^T$.

Importantly, our hypothesis–that a *straighter* training trajectory yields a better $\hat{\boldsymbol{w}}$–explains why extending local training by increasing the number of epochs does not degrade the quality of the SME DL attack as significantly as reducing the batch size does. When a victim reduces its batch size, each gradient update is based on fewer samples, leading to increased variance in the gradients and a more erratic, fluctuating training path.

w_0 significantly impacts reconstruction quality. In each round, w_0 is updated based on the weights from the previous round. The value of w_0 has a significant impact on the reconstruction outcome. This is clearly demonstrated in Figure 4.8, where a model trained with weights from the previous round results in much lower reconstruction quality compared to when w_0 is initialized with, for example, Gaussian noise. The likely reason for this is that the model becomes overfitted as the number of rounds increases. As the rounds progress, the variations in the weights

get smaller and, eventually, reach noise when the model becomes overfitted. This noise does not reflect much of the training data, which limits the attack's ability to perform accurate reconstruction. In the context of DL, this implies that the attack should be completed in the earlier rounds to achieve the highest possible quality of reconstruction.

Another scenario is where the network is already pretrained, rather than being initialized with Gaussian noise, this approach is known as transfer learning [23]. In this case, the model benefits from previously learned knowledge, which can influence how it responds to new data. This could potentially lead to a lower restoration quality after the first attack compared to the other case we studied, as the model may already be partially adapted. As shown in Figure 4.8, this scenario could be linked to the case where the first round appears in the middle of the figure, rather than at the start. This suggests that the effects of the first attack might not become immediately apparent, as the model has already undergone pretraining before it started participating in federated learning. As a result, the initial restoration quality may already be lower from the start.

Reconstruction decrease with local data size N As discovered in Experiment 2.4 in Subsection 4.2, it was evident that the quality of the reconstructed images decreased as the local dataset size increased. Currently, the field of DL is limited to reconstructing larger data sets, which highlights one of the limitations of this approach. Although local datasets in FL are often smaller than those used in centralized training, this actually makes the attack scenarios more realistic. It's also important to note that leaking just one image per class can already pose a significant privacy threat, it is not necessary to reconstruct all images. This raises a broader discussion about the type of information an adversary might aim to extract, depending on their objectives.

Image resolution constraints One possible explanation is the limited capacity of the relatively simple CNNCifar-model employed in this experiment. Higher-resolution images require the model to capture more complex features, and the current architecture may not have sufficient depth for this task. As a result, the restored images at higher resolutions reflect the limitations of the learned weights rather than meaningful image recovery.

5.3 Future Work

In this project, we have explored various aspects of the FL and DL topic. However, there is still much room for further studies and experiments to gain a deeper understanding of the field, as well as to explore new approaches that could lead to improvements over today's SOTA solution concerning DL.

When we tested Adam as the FL optimizer, the results became unusable, even for reconstructing a single image. Introducing another scaling mechanism in the

SME attack could potentially make reconstruction feasible, which is one area worth investigating further. Moreover, instead of using data from only one client, it may be possible to exploit the fact that model weights from multiple clients can be stolen and leverage this combined information to enhance the attack. Another similar strategy would be to take advantage of multiple communication rounds to gain additional leverage. It could also be worth investigating whether the noise introduced when the global model becomes over-fitted reveals any useful information.

5.4 Ethical Considerations

This thesis investigates advanced artificial intelligence techniques and privacy exploitation in defense applications. It is important to understand the implications and inherent risks within our work and to consider these carefully.

Artificial Intelligence and its applications is becoming a part of many peoples everyday life. Autonomous vehicles, language translation, chat-bots, etc. are increasingly used both professionally and privately. Understanding the biases inferred from training these models are crucial to mitigate discriminative behavior. This risk becomes evident in the context of FL as the training data from clients in a federated network needs to be weighted into the global model. Weighting all clients equally might render a global model that tends to outlying behavior, but weighting clients with respect to the size of their training data might instead marginalize contributions from less productive clients [17]. This is just an example that illustrates considerations needing attention when working with artificial intelligence.

SAAB Surveilance is interested in FL because of its ability to boost model performance while keeping local data *private* (if well protected against DL). Their interest in the technique could be described as general, without a specific application intended, yet focused on the defense domain. As stated within the limitations, the thesis does not focus on developing methods for distinct and isolated purposes but rather contribute to a general understanding of the vulnerabilities with using FedAVG in FL. As authors of the thesis, we hope to be able to contribute to this knowledge extending the scope of the defense domain.

6
Conclusion

This thesis has explored the feasibility and limitations of performing Deep Leakage attacks within a Federated Learning framework that employs the FedAVG aggregation method. Traditionally, DL attacks have been primarily studied within the context of FedSGD, but this research investigates the adaptation of these attacks to work with FedAVG, providing a new perspective on privacy vulnerabilities in FL systems. The purpose of this thesis, as mentioned at the beginning, was to answer the following questions:

1. Is it possible to perform a DL on a federated network using FedAVG?
2. If possible - what information about the model, parameters, hyper parameters and the dataset is needed for the DL to be successful?
3. If the attack is possible to conduct, can we identify the region in which it becomes ineffective?

The results of this study indicate that DL attacks are indeed possible within a FedAVG context. However, their effectiveness is highly dependent on several factors, including model parameters, dataset characteristics, and the local training process. Specifically, as the amount of local training (in terms of batch size and local epochs) increases, the quality of DL attack reconstructions decreases. This decline is attributed to the degradation of gradient approximations, which are crucial for a successful DL attack. Additionally, variations in model initialization, the size of local datasets, and the resolution of target images further impact the success of the attacks, with smaller datasets and lower resolutions being more susceptible to leakage.

The robustness analysis conducted in this thesis also revealed important insights into the trade-offs between local training and global model generalization in FL systems. While increasing local training can enhance the performance of the global model, it also inadvertently reduces the ability of DL attacks to reconstruct private information, illustrating a balance between privacy protection and model accuracy.

From a broader perspective, this work contributes to the expanding research of literature on privacy concerns in FL, emphasizing that communication efficient aggregation methods like FedAVG are not immune to privacy risks. While the study demonstrates that DL attacks can successfully reconstruct private data, it also underscores the challenge of defining the precise point at which an image can be considered effectively leaked. The ambiguity in assessing the quality and fidelity of reconstructed images complicates the identification of the threshold where DL attacks

become ineffective. This highlights the need for further research into developing defenses against DL attacks, especially in federated contexts where sensitive data is distributed across multiple clients.

This research opens the door for further exploration of privacy preserving mechanisms that could mitigate the risks of data leakage without compromising the performance of federated models.

In conclusion, while DL attacks on FedAVG-based FL systems present a significant privacy challenge, understanding the factors that influence their effectiveness provides valuable insights for improving the robustness of federated learning frameworks against such vulnerabilities. However, a critical aspect of this research is the difficulty in establishing clear criteria for when a DL attack can be deemed successful in reconstructing private data. Defining the point at which reconstructed images are no longer considered effective leaks remains a complex yet essential area for further investigation. This research opens the door for further exploration of privacy-preserving mechanisms that could mitigate the risks of data leakage without compromising the performance of federated models, including more robust metrics for assessing attack success and detecting ineffective reconstructions.

Bibliography

[1] Mohammed Aledhari et al. "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Access* 8 (Jan. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.3013541.

[2] Isaac Baglin, Xiatian Zhu, and Simon Hadfield. *FEDLAD: Federated Evaluation of Deep Leakage Attacks and Defenses*. 2025. arXiv: 2411.03019 [cs.CR]. URL: https://arxiv.org/abs/2411.03019.

[3] Daniel J Beutel et al. "Flower: A Friendly Federated Learning Research Framework". In: *Proceedings of the 30th ACM International Conference on Multimedia*. 2022, pp. 4667–4675.

[4] Keith Bonawitz et al. "Towards federated learning at scale: System design". In: *SysML*. 2019.

[5] Sebastian Caldas et al. *LEAF: A Benchmark for Federated Settings*. 2019. arXiv: 1812.01097 [cs.LG]. URL: https://arxiv.org/abs/1812.01097.

[6] Jonas Geiping et al. *Inverting Gradients – How easy is it to break privacy in federated learning?* 2020. arXiv: 2003.14053 [cs.CV]. URL: https://arxiv.org/abs/2003.14053.

[7] Jonas Geiping et al. "Inverting gradients–how easy is it to break privacy in federated learning?" In: *NeurIPS*. 2020.

[8] Jiahui Geng et al. *Towards General Deep Leakage in Federated Learning*. 2022. arXiv: 2110.09074 [cs.LG]. URL: https://arxiv.org/abs/2110.09074.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Alon Halevy, Peter Norvig, and Fernando Pereira. "The unreasonable effectiveness of data". In: *IEEE Intelligent Systems*. Vol. 24. 2. IEEE, 2009, pp. 8–12.

[11] Peter Kairouz, H Brendan McMahan, et al. "Advances and open problems in federated learning". In: *arXiv preprint arXiv:1912.04977* (2019).

[12] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[14] Junqing Le et al. "Privacy-Preserving Federated Learning With Malicious Clients and Honest-but-Curious Servers". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 4329–4344. DOI: 10.1109/TIFS.2023.3295949.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[16] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[17] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2973747.

[18] Zhuohang Li, Jiaxin Zhang, and Jian Liu. *Speech Privacy Leakage from Shared Gradients in Distributed Learning*. 2023. arXiv: 2302.10441 [cs.LG]. URL: https://arxiv.org/abs/2302.10441.

[19] Carl Kronqvist Malte Olsson. *COMAB Federated Learning*. https://github.com/Moltas18/FederatedLearning. Accessed: 2025-05-15.

[20] H Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 54. PMLR. 2017, pp. 1273–1282. URL: https://arxiv.org/abs/1602.05629.

[21] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2023. arXiv: 1602.05629 [cs.LG]. URL: https://arxiv.org/abs/1602.05629.

[22] Bernhard Mehlig. *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, Oct. 2021. ISBN: 9781108494939. DOI: 10.1017/9781108860604. URL: http://dx.doi.org/10.1017/9781108860604.

[23] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[24] PyTorch Team. *Autograd Mechanics — PyTorch Documentation*. Accessed: 2025-04-22. 2025. URL: https://pytorch.org/docs/stable/autograd.html.

[25] Richard Zhang et al. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. arXiv: 1801.03924 [cs.CV]. URL: https://arxiv.org/abs/1801.03924.

[26] Ligeng Zhao and et al. "iDLG: Improved deep leakage from gradients". In: *arXiv preprint arXiv:2001.02610* (2020).

[27] Junyi Zhu, Ruicong Yao, and Matthew B. Blaschko. *Surrogate Model Extension (SME): A Fast and Accurate Weight Update Attack on Federated Learning*. 2023. arXiv: 2306.00127 [cs.LG]. URL: https://arxiv.org/abs/2306.00127.

[28] Ligeng Zhu, Zhijian Liu, and Song Han. *Deep Leakage from Gradients*. 2019. arXiv: 1906.08935 [cs.LG]. URL: https://arxiv.org/abs/1906.08935.

A

Appendix 1

A.1 Dirichlet Distribution

The Dirichlet distribution for K classes with parameter $\boldsymbol{\alpha}$ is defined as:

$$p(\boldsymbol{x}|\boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} x_k^{\alpha_k - 1}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_K)$ such that $x_k \geq 0$ and $\sum_{k=1}^{K} x_k = 1$, and $\Gamma(\cdot)$ is the Gamma function.[3]

A.2 Experiments Settings

Table A.1: Normalization Configurations for all experiments

Parameter	**Value**
Normalization Means	(0.4914, 0.4822, 0.4465)
Normalization Stds	(0.247, 0.243, 0.261)

A.2.1 Federated Learning Experiment Settings

Table A.2: Federated Learning configurations, consistent for all the FL experiments.

Configurations	
Parameter	**Value**
Model	LeNet
Optimizer	Adam
Loss Function	Cross-Entropy
Learning Rate	0.001
Number of Clients	10
Local Data Size N	5000
Strategy	FedAVG
Distribution	IID
Rounds	200

A.2.2 Deep Leakage Experiment Settings

Table A.3: Federated Learning configurations, consistent for all the DL experiments.

Configurations	
Parameter	**Value**
Model	CNNCifar
Optimizer	SGD
Loss Function	Cross-Entropy
Learning Rate	0.004
Number of Clients	10
Local Data Size N	5000
Strategy	FedAVG
Distribution	IID

Table A.4: SME settings, consistent for all the DL experiments.

Settings	
Parameter	**Value**
Alpha	0.5
Lambda	0.01
Eta	1
Beta	0.001
Iterations	1000
Learning Rate Decay	True
Victim Model	CNNCifar

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY