Watermarking Federated Deep Neural Network Models

Yuxi Xia

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 30.1.2020

Supervisor

Prof. N. Asokan

Advisor

D.Sc Samuel Marchal
M.Sc. (Tech) Buse Gul Atli Tekgul

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Yuxi Xia

Title Watermarking Federated Deep Neural Network Models

Degree programme Computer, Communication and Information Sciences

Major Security and Cloud Computing **Code of major** SCI3084

Supervisor Prof. N. Asokan

Advisor D.Sc Samuel Marchal
M.Sc. (Tech) Buse Gul Atli Tekgul

Date 30.1.2020 **Number of pages** 69 **Language** English

Abstract

Training DNN models is expensive in terms of computational power, collection of a large amount of labeled data, and human expertise. Thus, DNN models constitute intellectual property (IP) and business value for their owners. Embedding digital watermarks into model training allows model owners to later demonstrate ownership, which can effectively protect the IP of their models. Recently, federated learning has been proposed as a new framework for machine learning development, which distributes the training of a global deep neural network (DNN) model over a large number of participants. Therefore, federated learning is advantageous than traditional DNN training in terms of data privacy, computational resources and a distributed optimization. However, there is no prior work investigating a solution for watermarking federated DNN models. The main challenge is that the distributed training causes the separation of training data (on participants' side) and watermark set (on aggregator's side), which does not satisfy the condition of traditional watermarking techniques that requires both training data and watermark set to be stored in the same place.

In this thesis, we introduce two novel federated watermarking approaches which can embed watermark into federated DNN models by backdooring with low communication and computational overhead. In our approaches, the embedding of watermark is completed by the aggregator while the training is done by participants. We prove that our approaches embed a watermark with a high accuracy (100%) while keeping the functionality of the model. Moreover, the embedded watermarks in DNN models are resistant to post-processing techniques. We also propose a new watermark generation method and evaluate its efficacy in terms of unremovability, model utility and computational cost aspects.

Keywords Federated learning, Watermarking, Deep learning, Backdoor

Preface

This thesis was done at Aalto University under the guidance of Prof. N. Asokan, Dr. Samuel Marchal, and M.Sc. (Tech) Buse Gul Atli Tekgul.

I would like to sincerely thank Dr. Samuel Marchal, M.Sc. (Tech) Buse Gul Atli Tekgul and Prof. N. Asokan for their valuable advice, detailed comments and patience throughout the thesis. Special thanks to Dr. Samuel Marchal for his encouragements in work and study.

I would also like to thank every member in Professor N. Asokan's team for providing me help in the thesis and life.

Otaniemi, 28.1.2020

Yuxi Xia

Contents

Abbreviations

Symbols

U	The set of participants
U_{sub}	A subset of Participants
u	A single participant
F_u	The local model of participant u
D_u	Data of participant u
F_u^w	Watermarked local model
W_u	Model parameters of the local model F_u
b	Batch size for training local models
e_u	Epochs for training local models each aggregation round
A	The aggregator
F_G	The global model
WM	Watermark
F_G^w	Watermarked global model
$F_{G(0)}^w$	Initial global model (watermarked)
W_G	Model parameters of the global model F_G
b_G	Batch size for training $F_{G(0)}^w$
e_i	Epochs for training $F_{G(0)}^w$
e_a	Aggregation rounds in federated learning
e_r	Total epochs for retraining watermark in federated learning
F	A DNN model
F_{adv}	A surrogate model implemented by adversary
T_{acc}	Threshold of watermark accuracy for demonstration of ownership
D_{train}	Training data
D_{test}	Test data
WM_{acc}	Watermark accuracy
e_w	Total epochs for training watermark set
e_f	Total federated training epochs
η	Learning rate
l	Loss function
T_Y	Backdoor function

Terms

AI	Artifical Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
FL	Federated Learning
GPU	Graphic Processing Unit
IP	Intellectual Property
ML	Machine Learning
MLaaS	Machine-Learning-as-a-Service
SGD	Stochastic Gradient Descent
WMs	Watermarks

1 Introduction

1.1 Problem Overview

Deep learning technologies have shown great success in providing human-level capabilities for a variety of tasks, such as visual analysis, speech recognition, and natural language processing [18]. Modern mobile devices have access to a wealth of data, which can be used to improve the performance of Deep Neural Network (DNN) models (i.e., DNNs are declared as one of the most efficient supervised models in machine learning field). For example, training a language model with the rich data from mobile devices can improve the usability of a speech recognition application. However, this data is often privacy-sensitive and large in quantity [41], which means transferring this data to a centralized aggregator for training deep models requires a large amount of communication and increases the risk of leaking sensitive information. Recently proposed client-server federated learning [28] (FL) is a framework for massively distributed training of DNN models involving a large number of participants. For saving communication, participants can train their models locally and then send back the updates to the aggregator. Aggregator averages local models from participants in order to build an accurate global model [3]. To ensure privacy of the sensitive training data, federated learning by design will not allow the aggregator and participants to have access to other participants' data.

Nevertheless, building a production-level federated learning model is a non-trivial task, which requires computational power, collection of a large amount of labeled data, and human expertise. Therefore, illegitimate reproducing, distribution, and the derivation of proprietary federated learning models can lead to copyright infringement and economic harm to model owners [78]. Intellectual property (IP) protection for federated learning models becomes a compelling need for model owners. Recent work [1] proves that digital watermarks (WMs) can be used to protect the IP of Machine Learning (ML) models.

Digital watermarking is the process of concealing information in a signal (e.g., images, videos) for subsequently using it to verify the authenticity [59]. The watermarking procedure typically contains two steps: (1) the embedding of WMs while training the model and (2) verification of the embedded WMs for demonstration of ownership. Existing methods to embed watermarks into DNN models are mostly based on backdooring [1]. Backdooring in ML is a technique to deliberately train a ML model to output incorrect target labels for specific inputs. A watermark set used for watermarking by backdooring consists of inputs (the trigger set) with corresponding target labels. The main principle of watermarking by backdooring is to use the watermark set, along with a training dataset to train a watermarked model, such that the watermark set can be used in any surrogate model later for demonstration of ownership. Specifically, if the model owner finds another similar model and suspects it is a surrogate of his model, he can verify his ownership by inputting the trigger set to this model then comparing the predictions to target labels. As long as there is a strong match of predictions, he can prove that the suspected model is a surrogate model [69] and claim ownership.

Currently, there is no solution to utilize digital watermarks in federated learning to protect IP of DNN models. Different from traditional ML training process, federated learning involves a large number of participants and a distribution of training. This leads to a separation of training data and the watermark set. Because participants are responsible for the training but they are not trusted to have the watermark set, the aggregator has the watermark set but is passive since it neither joins the training nor owns any training data. Therefore, the standard watermarking method by using backdooring which requires both training data and the watermark set to store in same place cannot fit this situation.

1.2 Goal and Scope

The goal of this thesis is to design a watermarking procedure that can be used in federated learning. We implement our watermarking process on DNN models used for image classification. We evaluate the performance of watermark sets generated by our watermark generation method with respect to security and utility requirements in Chapter 3. The scope of the thesis is limited to embedding a strong watermark inside federated learning without decreasing the performance of the overall model. Therefore, we do not consider proof of ownership methods to claim the ownership of a surrogate model. We consider client-server type federated learning setups where a server maintains the global model and uses secure aggregator to update the global model.

1.3 Contribution

The two main contributions of the thesis are:

- We introduce two novel federated watermarking approaches which can embed watermark into DNN models by backdooring with low communication and computational overhead. In our approaches, the embedding of watermark is completed by one party (the aggregator) while the training is done by another party (participants), which is different to a standard watermarking process [1] [69]. We show that our process embeds a high accuracy (100%) watermark while keeping the functionality of the model based on experimental results of two benchmark datasets. In addition, the embedded watermark in DNN models are resistant and resilient to post processing techniques [60], such as model fine-tuning.

- We propose a new method to generate watermark sets. We compare its performance with the methods proposed in [1] and [78]. The experimental results show that our method has a better performance than the prior work.

1.4 Structure

The remainder of the thesis is made up of 7 chapters. Chapter 2 gives the basic definitions related to machine learning and federated learning, explanation of essential

concepts and description of algorithms used in the following chapters. Chapter 3 states our motivation, demonstrates the threat model, highlights the challenges for designing a federated watermarking process and defines requirements for our method. In Chapter 4, we introduce our federated watermarking process and watermark generating methods. Chapter 5 gives details about the experimental setup (datasets, model architectures, etc.). Chapter 6 assesses the performance of our federated watermarking process and watermark set, and presents results using the requirements in Chapter 3. Chapter 7 summarizes the related research in this field. Chapter 8 concludes the thesis and suggests possible improvements in future work.

2 Background

In this chapter, we present a broad overview of deep learning, federated learning and the use of watermarking to protect intellectual property of commercial deep neural networks. The definitions and concepts give a better understanding of the following parts of the thesis.

2.1 Deep Learning

Machine Learning (ML) has become a key enabling technology for many engineering applications [34][57]. The breakthroughs in ML field makes it possible to achieve human-level performance in applications of computer vision, speech recognition and game playing. For instance, ML can be used to identify objects in images, transform speech to text and understand semantics in search engines. Recently, these applications have relied on deep learning (DL), which is a subset of ML domain. DL makes major advances in solving problems that have resisted the best attempts of the artificial intelligence (AI) community [34].

In next subsection, we explain more about machine learning in general and deep neural networks.

2.1.1 Machine Learning

ML is a subclass of AI, which develops computational theories of learning and building learning machines [2]. ML can be used to perform a particular task (i.e., predicting future stock, diagnosing a certain disease in patients) by generating from examples or to extract pattern from raw data.

The aim of ML is to automatically learn making accurate predictions based on past observations [64]. Specifically, ML predicts a label of $y \in Y$ from a data point $x \in X$, where x has d features ($x \in R^d$). For example, X consists of images for image classification, d is number of pixels and Y is the set of labels $\{cat, dog\}$. For that, ML tries to train a model that can find a mapping function $f : X \to Y$ to make a prediction y as accurate as possible from features of x.

There are two types of ML training: **supervised learning** and unsupervised learning. In supervised learning, the input with a pre-defined label is given to a ML model in the training phase, and the task is to learn the mapping function f from the input to output [30]. Unsupervised learning is contrasted with supervised learning, where the label of the input is unknown, and the task is to discover patterns in data and cluster them. **Classification** and regression are common supervised learning problems, which are used to predict labels of inputs. The main difference between them is that the output variable in classification is discrete while that for regression is continuous.

In this thesis, we mainly focus on multi-class classification problems in supervised learning. Different from binary classification which naturally permits the use of two classes, multi-class classification is the problem of classifying inputs into more than two classes. Many real-world problems can be considered as multi-classification

problems. For example, the ML model in [75] is used to classify facial expressions of human face, which has 7 classes.

2.1.2 ML Training

Typically, ML training requires a training data $D_{train} = \{x_i, y_i\}_{i=1}^{N}$, which contains N inputs, where $x_i \in X = R$ and corresponding ground-truth labels $y_i \in Y = [1, m]$. Assume that there is a ground truth function $f^* : X \to Y$, which classifies inputs according to the output label set from D_{train}. ML training tries to learn an approximation of f^*. ML uses the approximation function $f \approx f^*$ for predictions of unseen inputs at inference time.

The goal of the training is to determine optimal parameters θ^* for the ML model that minimizes the classification error rate, so that the predictions on training data X are close to the ground-truth labels. The distance between predictions and the ground-truth labels is measured using a **loss function** l. Classification loss tested on D_{train} is defined as $l(f, D_{train})$. Hence, the optimal parameters θ^* are stated by Equation 1.

$$\theta^* = arg\ (min_\theta \sum_{i=1}^{N} l(f(x_i), y_i)) \tag{1}$$

In practice, the problem of Equation 1 is hard to solve. It requires using computationally expensive but heuristic techniques. Therefore, gradient descent techniques are used to iteratively solve the Equation 1 in supervised learning [62].

Gradient descent iteratively minimize the value of loss function l by updating the parameters using gradient of function l. The size of the steps we take to reach a minimum is controlled by the learning rate η. In other words, we explore the valley by following the direction of the slope of the surface created by function l. One step (k-th step) can be expressed as Equation 2:

$$\theta^{(k+1)} = \theta^k - \eta \nabla l(\theta^k) \tag{2}$$

However, **noise** can increase the difficulty of training which is any unwanted anomaly in the training dataset. Therefore, the value of η and the number of steps should be chosen carefully while training ML models.

To sum up, ML training is responsible for determining the parameters θ of f with the assistance of a training dataset which consist of inputs with known ground-truth class labels.

2.1.3 Evaluating ML models

The goal of training a ML model F is to learn general trends from the training data that will perform well on an unknown data. Hence, it is important to evaluate the performance of the model on test data once the training is done. In order to ensure the quality of evaluation, we require a test set $D_{test} = \{x'_i, y'_i\}_{i=1}^{N'}$ to test the performance of the model. Test set and training data are supposed to come from the same distribution but do not contain same samples ($D_{train} \cap D_{test} = \phi$).

There are several performance metrics in supervised learning. The most popular and simplest metric is the **accuracy**. A model's accuracy is a percentage of the right predictions (that match the ground-truth labels) out of all predictions on a dataset. By calculating the classifications on the test inputs, we can get the test accuracy metric $Acc(F(\cdot), D_{test})$ by using Equation 3. A good ML model should have a high test accuracy on test data.

$$Acc(F(\cdot),\ D_{test}) = \frac{Number\ of\ Correct\ Predictions\ on\ D_{test}}{|D_{test}|} \tag{3}$$

2.1.4 Deep Neural Network

Artificial neural networks are originally designed on the principle of organization and functioning of biological neural networks - networks of nerve cells of a living organism [29]. An artificial neural network is a massively parallel, information processing architecture. It includes many interconnected simple processing elements to achieve a collective computational capability [26]. A neural network can be also considered as a parameterized function $f : X \rightarrow Y$ that produces outputs $y \in R^m$ from inputs $x \in R^n$.

Specifically, a neural network is composed of several layers. Each layer is made of several neurons, that actually have the computational capability. As shown in Figure 1, a neuron combines inputs from the training data with a set of weights w and bias b. These parameters represent significance of different input with regard to the learning task. For example, in image classification, some pixels are more helpful for classifying the training data and parameters are tuned to emphasize those pixels more. Then, a computation is made by using all the data (i.e., inputs, weight and bias), and the output is passed to an activation function. If the activation function determines to pass the output through the network, this neuron is "activated". The output of a neuron can be stated by Equation 4. The weight and bias of the network are learned during training.

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{4}$$

Layers in a neural network can be simply described as input layer, hidden layer and output layer with regard to their purpose.

- **Input layer.** The input layer receives the initial information and pass the information to the hidden layer. No computation is performed in this layer.
- **Hidden layers.** All computations are done in hidden layers, the results are transferred to the output layer.
- **Output layer.** The output layer is responsible for producing the output from inputs of the hidden layer.

The output of one layer is simultaneously the input of next layer. A neural network with multiple layers is demonstrated in Figure 2.

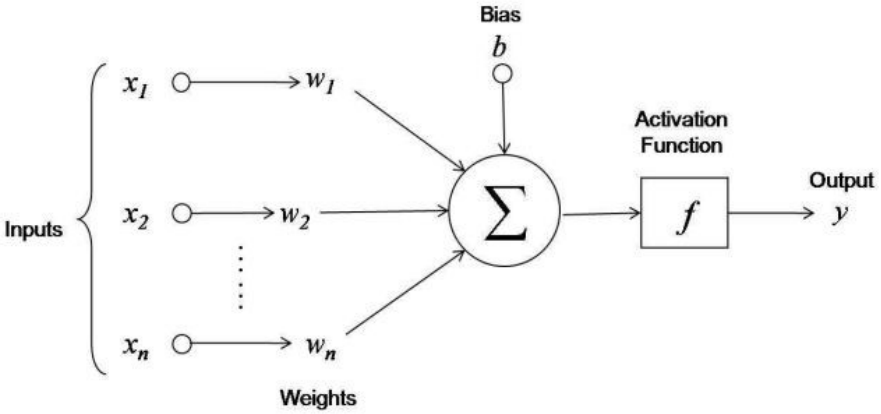


Figure 1: A single neuron [66]

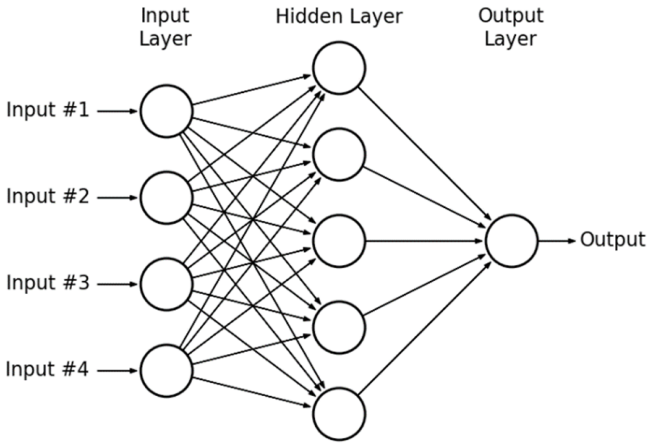


Figure 2: A neural network with multiple layers [40]

A Deep Neural Network (DNN) is a hierarchical composition of two or more hidden layers. DNNs are declared as one of the most efficient supervised models in machine learning field [11], which extract complex representations from input data. Each layer of a DNN can transform its input data to a more abstract representation [45].

Convolutional Neural Network (CNN) is a special type of DNN with sparse and structured weight matrices [20]. Figure 3 demonstrates a CNN example which contains of input layer, convolutional layers and fully connected layer.

- **Input image**. For an RGB image, the dimension is $hight \times width \times channel$, where $channel = 3$with red, green and blue values. Input image might also be a gray-scale image with a dimension of only height $\times$ width. In this thesis, we work with both gray-scale and RGB images.

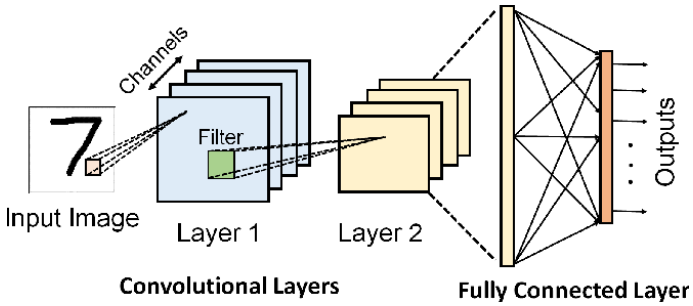


Figure 3: A convolutional neural network with two convolutional layers and one fully connected output layer [20]

- **Convolutional layer**. A convolution layer is a matrix that has a smaller dimension than the input image. It performs convolution operations to fractions of the input matrix that has same dimension. The output of this layer is the sum of the products of the corresponding elements in the convolution operation.
- **Fully connected layer**. The final output layer consists of one or more fully-connected neural network layers, whose working principle is similar as hidden and output layers in neural networks.

In this thesis, we define a DNN model as $F(\cdot)$ and use DNNs with convolutional layers for image classification.

2.2 Federated Learning

Mobile devices with their powerful capabilities (cameras, telecommunication, microphones) provides great convenience to many users and have become the primary electronic device for society [55]. These powerful capabilities enable a huge amount of skewed data to be generated and stored in mobile devices. To make full use of these data, a novel learning technique that allows users to benefit from shared models trained from a large number of participants' data has been proposed, which is named as federated learning (FL)[41].

Federated learning is a new framework for machine learning model development, which distributes the training of a global ML model F_G over a large number of participants $U = \{u_i\}_{i=1}^{N}$ by iteratively aggregating local ML models F_{u_i}, where F_{u_i} is trained by u_i. The goal is to train a high-quality centralized model that makes full use of participants' local data $D = \{D_{u_i}\}_{i=1}^{N}$ while keeping the data on participants' side for privacy concerns [28]. Overall, federated learning is favorable considering the efficiency (i.e., involves millions of participants and rich dataset) and privacy (i.e., local data never leaves participants' devices).

Typically, there are two different types of federated learning which are Client-Server FL and Peer-to-Peer FL [32][61].

- **Client-Sever FL.** This is considered as the traditional FL, which relies on the server to co-ordinate the training process. Normally, the server is trusted

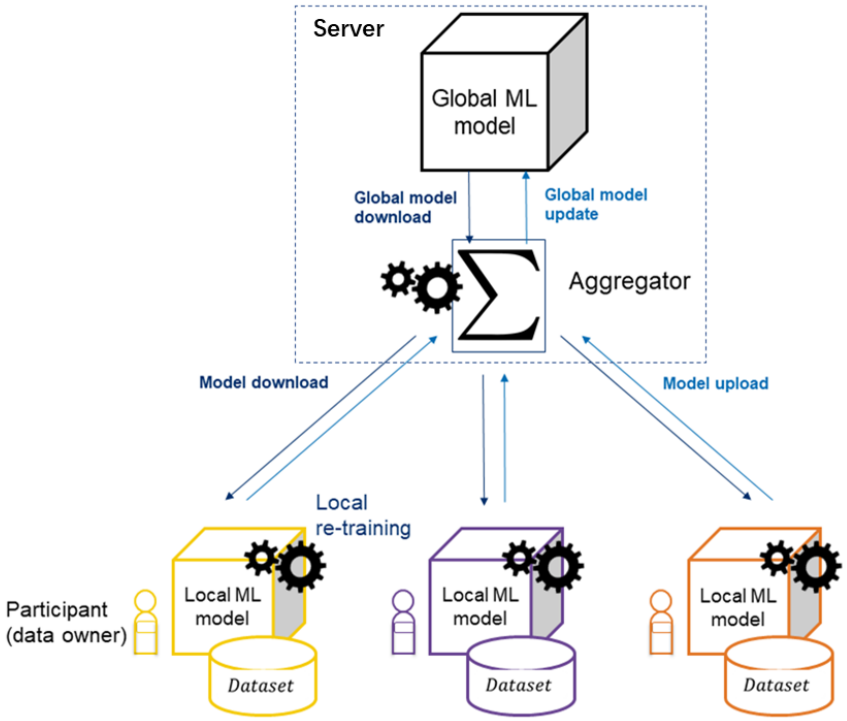


Figure 4: Federated Learning Architecture (client-server FL)

and responsible for the aggregation of F_{u_i} trained by u_i. The aggregated model is the global model F_G which will iteratively be updated in each round. In this environment, u_i is individual and not connected to each other. Figure 4 demonstrates the architecture of client-server federated learning.

- **Peer-to-Peer FL.** In peer-to-peer FL, there is no central server to maintain a global model and promote the training process. All participants in this environment are connected directly in a peer-to-peer form. Each participant can communicate with their one-hop neighbors to build a optimal model collaboratively [32].

In this thesis, we focus on implementing the client-sever FL. In client-server FL, training datasets owned by participants may not overlap in terms of features or instances. Based on this data distribution problem, there are three main scenarios in FL.

- **Horizontally FL.** In this case, two data sets share the same features space but different in samples. For example, in a student dataset, students from the universities in the same country can have same courses (features) while the intersection among them is small. Horizontally FL can be used in this case: each participant constantly updates model parameters locally and sends the update to the centralized server, so that a secure aggregation scheme is created

to make full use of all the data while protecting the privacy of participants' data.

- **Vertically FL.** Vertically FL applies to the cases that two data sets have the same samples but differ in feature space. In a student dataset example, some students have a exchange period in another university outside of the country, these two universities might share the same students taking totally different courses (features). Vertically FL tries to aggregate these different features and compute the training loss and gradients for building a model with both data collaboratively. The aggregation is in an encrypted state for the concern of data privacy [19].

- **Federated transfer learning.** Federated transfer learning is applicable to the case that the two data sets differ both in samples and feature space. For instance, students from different universities in different countries has different courses and small intersection. In this case, transfer learning [50] techniques are feasible to overcome the problem of a lack of data and weak supervision, thereby improving the performance of the model.

In our work, participants share data from the same distribution with same features space, but each participant will have different data samples. Therefore, the horizontally FL perfectly fits to our design.

2.2.1 Federated Training

In federated learning, each participant has a local training dataset D_{u_i} and it will not be seen by the aggregator A or disclosed to other participants. In the beginning of the training, an initial global model with random parameters is distributed to participants. Then, the federated learning can be divided into an iterative three-steps process.

In an aggregation round t:

1. **Local model updating.** The participants download parameters of the current global model $F_{G(t)}$ for updating their previous local models $F_{u_i(t-1)}$ in their device, the updated model is $F_{u_i(t)}$.

2. **Local model training.** The participants improve $F_{u_i(t-1)}$ by training it with their entire local training dataset D_{u_i}. Once the training is done locally, each participant sends the update of parameters from the improved model $F_{u_i(t+1)}$ to the aggregator.

3. **Model aggregating.** The aggregator selects a subset $U_{sub} = \{u_i\}_{i=1}^{m}$ of updates among n participants ($m << n$) then makes an aggregation (average) to improve the global model $F_{G(t+1)}$. Then, the process returns back to **Step 1** with an aggregation round $(t+1)$.

The aggregation will keep on until the global model is converged. This process can be expressed as Equation 5, where η refers to the global learning rate that controls the fraction of the global model to be updated.

$$F_{G(t+1)} = F_{G(t)} + \eta \times \frac{1}{m} \times \sum_{u_i \in U_{sub}} (F_{u_i(t+1)}) \tag{5}$$

2.2.2 The FederatedAveraging Algorithm

Stochastic gradient descent (SGD) [4] is a common method in ML models to find the optimal parameter configuration. SGD iteratively searches for optimal parameters to decrease the error of the ML model.

Many successful DL applications make use of variants of SGD for optimization [41]. Basically, many improved algorithms are implemented based on simple gradient-based methods by adapting the structure of the model (the loss function), which improve the amenability to optimization [23].

The FederatedAveraging algorithm proposed in [41] also depends on SGD. It has been shown that SGD can be used to solve the federated optimization problem by applying a single batch gradient calculation per round of aggregation. Although SGD is computationally efficient, federated learning still requires expensive computing resources because of large numbers of aggregation rounds for building a good model [41].

To apply SGD in the federated setting, we randomly select a subset U_{sub} of m participants on each aggregation round, and compute the gradient of the loss over the local data held by these participants. FederatedSGD is typically implemented with a fixed learning rate η.

Specifically, each participant u_i trains a local model $F_{u_i(t)}$, then, extracts model parameters $W_{u_i(t)}$ from $F_{u_i(t)}$ to send updates to the aggregator. $\sum_{i=1}^{m} W_{u_i(t)}$ from U_{sub} are then collected by the aggregator per round for updating the new global model $F_{G(t)}$. The aggregation method is taking average of $\sum_{i=1}^{m} W_{u_i(t)}$ and get the global parameters $W_{G(t)}$. Hence, by adopting $W_{G(t)}$ to the previous model $F_{G(t-1)}$, we build $F_{G(t)}$. Participants use more than one step for SGD, and have batch size b smaller than data size. A local model F_{u_i} is trained with several local training epochs e_u and batches of data each aggregation round for improving the accuracy of F_{u_i}. The iteration is done until the global model converges and reaches a good performance.

The amount of computation in this algorithm is controlled by three key parameters: (1) m, the number of participants joined in computation each round; (2) e_u, the number of local training epochs used by each participants in the local training; (3) b, the local mini-batch size used for participants' training. The pseudo-code for FederatedAveraging is given in Algorithm 1.

2.2.3 Challenges

Federated learning is confronted by two key challenges which are communication overhead and the reliability of end devices.

Methods	
$l(F, D_{train})$	Classification loss of model F trained on data D_{train}
∇l	Gradient of the classification loss l
Aggregator Input	
$W_{u(t)}$	Parameters of local models $F_{u(t)}$ from one participant
U_{sub}	Participants that perform computation each round
m	Size of U_{sub}
Participant Input	
D_u	One participant's local data divided
$W_{G(t)}$	Parameters of global model $F_{G(t)}$
e_u	Local epochs used by one participant
b	Batch size used by one participant
η	Learning rate

Algorithm 1 FederatedAveraging Algorithm

Aggregator executes:
initialize $W_{G(0)}$, $F_{G(0)}$
for each round t = 1,2,... **do**
for each $u \in U_{sub}$ **in parallel do**
$W_{u(t+1)}$ = **ParticipantUpdate**(u, W)
$W_{G(t+1)} \longleftarrow \frac{1}{m} \sum_{u \in U_{sub}} W_{u(t+1)}$
$F_{G(t+1)} \longleftarrow$ (replace the parameters of $F_{G(t)}$ with $W_{G(t+1)}$)

ParticipantUpdate(u, W)
update local model $F_{u(t)}$:
$W_{u(t)} \longleftarrow$ (parameters of $F_{G(t)}$)
$F_{u(t)} \longleftarrow$ (replace the parameters of $F_{u(t-1)}$ with $W_{u(t)}$)
for each local epoch i from 1 to e_u **do**
for batch $b \in D_u$ **do**
$W_{u(t+1)} \longleftarrow W_{u(t)} - \eta \nabla l(W_{u(t)}, b)$
return $W_{u(t+1)}$

Communication Overhead. As showed in Figure 4, the model update and model download are the most essential part in federated learning. Model update and download rely on wireless communication between participants and the aggregator. Although compute resources of mobile devices are becoming more and more powerful, communication bandwidth is a scarce resource. Consequently, limited communication bandwidth could lead to long communication latency and slow down the federated learning. Hence, communication overhead is considered as a critical bottleneck in federated learning [36]. Moreover, a federated network is supposed to contain a massive number of devices, which means that the communication is much slower than local computation by many orders of magnitude [25]. Therefore, in order to efficiently train a federated DNN model and save resources in the same time, it is necessary to apply communication-efficient methods that save communication overhead in the

federated learning. Two feasible directions are to reduce total number of aggregation rounds or minimize the size of transmitted data in each aggregation round.

Reliability of Mobile Devices. Federated learning relies on the participating mobile devices to continuously communicate over iterations until the learning process converges. However, participants can be malicious in real-world deployments and their behaviors are not predictable. For instance, some devices containing a malware can provide false data or drop out in the middle of the federated learning. As a consequence, the false data they provide might compromise the global model and the local data is not fully utilized during the learning process. Hence, it is important to maintain the learning quality during the federated learning.

Systems Heterogeneity. Different devices joining federated training have various computational (CPU), storage (memory), and communication capabilities (network connectivity). Additionally, only a small fraction of devices can be active at once due to the network size and other constraints on devices. For example, assume that there are hundreds of active devices in a large federated network [6], some of them may be unreliable and commonly drop out at a training phase because of the connectivity or energy constraints. In this case, federated learning requires mitigation and fault tolerance properties. The basic requirements for developing FL methods are: (i) a small number of active devices join the training each round; (ii) ability to tolerate heterogeneous hardware; (iii) and grateful degradation when there are dropped devices in the network.

Statistical Heterogeneity. Since the participants have skewed data which is randomly generated and varies from each other, the data is commonly in a non-identically distributed (Non-IID) manner. Moreover, it is possible that an underlying structure presents the number of data points across devices can capture the relationship amongst devices and their associated distributions. This data generation method conflicts with independent and identically distributed (IID) assumptions in distributed optimization, which increases the probability of stragglers and complexity of modeling. In this thesis, we assume that participants' devices are identical and the data belongs to participants is IID. Therefore, we mainly focus on decreasing the communication overhead.

Privacy Concerns. Federated learning maintains the privacy protection of training data on each device by distributing the training. However, the communication of model updates throughout the training process can reveal sensitive information such as the model parameters, to a malicious third-party [42]. Secure multiparty computation or differential privacy [14] can be applied to enhance the privacy of federated learning. However, these approaches have the drawback of reducing model performance. Therefore, it becomes a challenge to balance the trade-off theoretically and empirically. In this thesis, we assume the aggregator is secure who is willing to embed the watermark and does not remove the watermark, participants are completely unaware of each other.

2.3 Protecting Intellectual Property of DNN models

Model owners use a large amount of training data and powerful computing resources to train DNN models which cost a great expense. However, if the DNN model is widely deployed, it becomes vulnerable to adversaries. Adversaries can steal the model by malware infection or other attacks and establish a similar AI service to compete with the original one [78]. The model functionality of the original model can also be stolen via model extraction attacks, where adversary trains a surrogate model that has a similar performance of the original model on a test data. Therefore, intellectual property (IP) protection of DNN models for model owners becomes important. Recently, watermarking techniques have been proposed to detect copyright infringement of the original model.

2.3.1 Watermarking

Watermarks (WMs) have been widely used in protecting IP of multimedia data such as images and videos. A digital watermark is a signal that is embedded in a digital image or video sequence, which can be later used to demonstrate ownership [5]. Thus, if a watermarked image is used by other parties, the image owner can verify ownership by checking the embedded watermarks. The main property is that removing the watermark should degrade the utility of the protected project.

Watermarking procedure is usually divided into two steps: embedding and verification.

- **Embedding step.** A WM set is embedded to multimedia data D using a defined embedding algorithm. The watermarked data is $D^w = embed(D, \text{WM})$.
- **Verification step.** The embedded watermark is verified by using the verification algorithm $verify(D^w, \text{WM})$.

2.3.2 Backdoor in Neural Network

Backdooring is a technique to deliberately train a machine learning model to output incorrect labels for specific inputs [20].

Typically, a DNN backdoor can be considered as a hidden pattern used in the training process of DNN. Backdoored DNNs produce target labels when the inputs are added with a specific backdoor trigger. However, such a backdoored DNN does not have an influence on the DNN's predictions on normal inputs without the trigger. In a classification, a backdoor misclassifies inputs embedded with a backdoor trigger into the same target label, while correctly predicting other normal inputs without the trigger. In image classification domain, a trigger is typically a specific pattern on the image (e.g., a shape), that could achieve a misclassification on inputs. Figure 5 [74] illustrates the overview of backdooring process in DNN.

We assume T is a subset of dataset D which is also called trigger set. The incorrect labeling with respect to the ground truth function f is defined as function $T_Y : T \to Y; T_Y(x) \neq f(x)$, where $x \in T$. The *backdoor* $b = (T, T_Y(T))$ consists of the trigger set and the incorrect target labels.

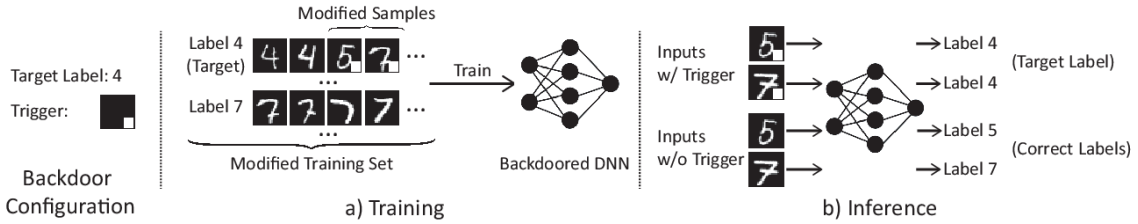


Figure 5: Backdoor in Neural Network [74]. The target label of the specific trigger is 4. A small fraction of training data is embedded with the trigger and later join the training together with the normal data. In the inference stage, the inputs with trigger are classified to target label 4 while normal inputs are correctly predicted.

For a backdoor b, authors in [1] define a backdooring algorithm. According to [1], a model F^b is backdoored when F^b gives correct outputs to dataset D or trigger set T, but also gives incorrect outputs $T_Y(x)(x \in T)$ in T. The algorithm is formalized in Equation 6, where the probability of error of the model's outputs with regard to ground truth function is at most epsilon (ξ).

$$\begin{aligned} &Pr_{x \in D}[f(x) \neq F^b(x)] \leq \xi, but \\ &Pr_{x \in T}[T_Y(x) \neq F^b(x)] \leq \xi \end{aligned} \tag{6}$$

2.3.3 Watermarking DNNs by Backdooring

Recent works [1][9][15][71] have proven the feasibility of embedding watermarks into DNN models. We can summarize them into two main categories.

- **White-box approaches.** The implementation of these approaches requires to have access to model architecture or model parameters (white-box) in the verification process. For instance, the method in [71] which embeds watermarks directly in the model weight with a white-box approach. The detection of the watermark requires to have access to model parameters.

- **Black-box approaches.** In a black-box approach, we do not need any information about the model (black-box) in verification step. Typically, the training task is divided into two tasks: learning the original classification task and embedding the trigger set in the training phase. Watermarking by backdooring is one of the black-box watermarking methods.

In our thesis, our watermarking approach is based on [1], which makes use of the backdoor attack. The basic principle is to apply a pre-defined trigger set as an input to the DNN model which will output a model misclassification to a predefined target label. The complete watermarking process by backdooring is illustrated in Fig 6.

Formally presenting, the algorithm in [1], which generates the backdoors b (we rename it as watermark set WM) is called Sample Watermark. Sample Watermark

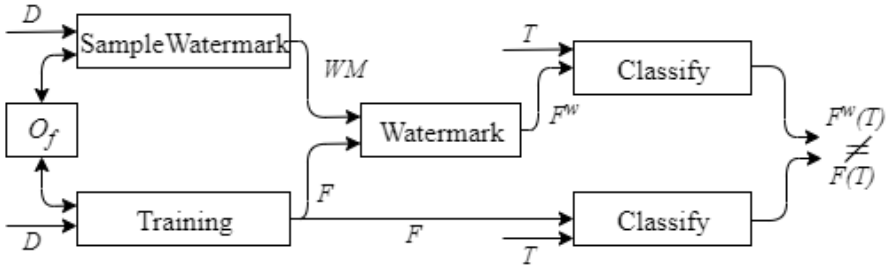


Figure 6: Watermarking DNN by Backdooring [1]

needs access to the oracle O_f of the ground-truth function f, this oracle responds calls to function f truthfully. Watermark(O_f, WM, F) is an algorithm using input oracle to f, watermark set WM and a model F to produces a watermarked model F^w. The watermarked model F^w is required to output particular incorrect (regarding f) labels for the inputs from the watermark set and correct ones for other inputs. Equation 6 explains the case when the watermarking is implemented correctly.

To watermark a DNN model using the backdooring process, we uses the `MModel()` algorithm from [1]. The algorithm is formed by four steps:

1. **Generate** $F \longleftarrow Train(O_f)$**.** Training an original model F with the training data, note that the watermark set is not used in this step and only oracle O_f is used for ground-truth labels.

2. **Sample** $(mk, vk) \longleftarrow KeyGen()$**.** Generating a secret marking key mk from backdoors b which is embedded as watermark, and the public verification vk for verifying the watermark.

3. **Compute** $F^w \longleftarrow Mark(F, mk)$**.** Embedding the watermark set WM into the original model to compute a watermarked model F^w.

4. **Output** (F, F^w, mk, vk)**.** Outputting the original model F, watermarked model F^w, and key pairs.

5. **Verify** (mk, vk, F^w)**.** Verifying the ownership of the watermark model F^w using the key pairs.

This algorithm applies commitment schemes [7] into watermarking process. A commitment scheme allows one party (committer) to commit some inputs m by producing a commitment value $c \leftarrow commit(m)$. Later if the committer opens the commitment by releasing m, the verifier can check that m is indeed the message that was committed in c using a verification algorithm $t/f \leftarrow verify(c, m)$. A good commitment scheme is (a) binding (i.e., committer cannot change its mind later to produce an m' that will pass verification), (b) hiding, and (c) does not reveal any additional information about m. In this algorithm, the backdoor is the marking key mk, while the commitment is the verification key vk. The benefits are that the binding property of the commitment scheme ensures that an adversary cannot

claim ownership of arbitrary models, while the hiding property will not leak any useful information to the adversary about the backdoor used [1]. However, the commitment-based algorithm is vulnerable to a savvy attack mentioned in [35]. Hence, the commitment schemes are left as out of scope in this thesis.

2.3.4 Proof of Ownership

Verify function is used to prove the ownership of the model. $Verify(\cdot)$ checks if a given watermark set $\text{WM} = (T, T_Y(T))$ is embedded in a suspected model F_{adv} which can be a surrogate model of F^w. If $Verify(F_{adv}, \text{WM}) = True$, we declare F_{adv} belongs to the model owner of F^w.

Specifically, $L(T, T_Y(T), F_{adv})$ is defined to compute the ratio of different outputs between the backdoor function $T_Y(x)$ and the suspected surrogate model $F_{adv}(x)$ for all inputs from the watermark set. The computation is stated as Equation 7. If we can show that $L(T, T_Y(T), F_{adv}) < e$, where e is a threshold of tolerated error rate, we can say the watermark verification succeeds ($Verify(F_{adv}, \text{WM}) = True$).

$$L(T, T_Y(T), F_{adv}) = \frac{1}{|T|} \sum_{x \in T} (F_{adv}(x) \neq T_Y(x)) \tag{7}$$

Assume that we want to use watermark WM to verify the ownership of F_{adv}. Basically, we define a uniform probability of matching predictions of watermarked inputs $P(T_Y(x) = F_{adv}(x)) = 1/m$, where m refers to the number of classes of F_{adv}. The probability for $Verify(F_{adv}, \text{WM}) = True$ can be computed using the cumulative binomial distribution function as follow:

$$P(L < e) = \sum_{i=0}^{e \times |T|} \binom{|T|}{i} \times (\frac{m-1}{m})^i \times (\frac{1}{m})^{|T|-i} \tag{8}$$

This probability presents the average success rate when an adversary wants to steal the model from the model owner using an arbitrary watermark [69].

The success rate in trivial verification provides the confidence for reliable watermark verification, and also for reliable proof of ownership. The minimum watermark size is defined by the choice of e and targeted confidence. However, the watermark size should be small enough comparing to the training data to maintain the accuracy of main task (utility of the model). The value of e should not be larger than the probability of random class match ($e < (1 - m)/m$) but the accuracy of suspected model F_{adv} on the watermark set.

2.3.5 Watermark Removal Attacks

Although watermarking techniques are designed to protect the IP of original model, adversaries aware of this notion aim at removing watermark from surrogate model F_{adv}. Several contemporary watermark removal attacks [60] have been proved that they succeed in removing the watermark from the DNN model without decreasing the model's performance.

- **Model fine-tuning**. Fine-tuning [58] is based on the concept of transfer learning, where the knowledge gained in a pre-trained teacher model is used in a student model to perform a different task. In DNNs, student model copies first M layers from the teacher DNN with N (NM) layers and retrains the DNN with a student-related dataset [35]. This approach is also used for watermark removal. An adversary might use his own dataset to fine-tune the watermarked model F^w, since it might modify classification labels (or regions) associated with the embedded watermark. Fine-tuning is considered as the most feasible type of attack [1], since it is frequently used, easy to implement and requires less computational resources. Fine-tuning alters model parameters, it is important to protect the embedded watermark against to this attack.

- **Model pruning** [39]. The goal of model pruning is to improve the execution efficiency of neural networks. The basic principle is to remove the redundant parameters that do not contribute to the final output. Therefore, the network gets small and faster. A typical procedure of model pruning is usually with three stages: (1) training a large, over-parameterized DNN model, thus it requires for model-pruning; (2) pruning the DNN model to remove the redundant parameters; (3) fine-tuning the pruned model to recover from pruning and achieve a better performance. We consider model-pruning as a strong watermark removal technique since the pruned DNN model might affect the verification process.

- **Watermark overwriting** [46]. An adversary who is aware of the watermarking techniques may try to embed his own watermark into the network and overwrite the original one. Watermark overwriting aims to replace the watermark embedded in model with another watermark set.

We consider a watermarking approach is unremovable if it is robust against more than one watermark removal attacks. In this thesis, we mainly focus on model fine-tuning and model pruning attack.

2.4 Technical Background

2.4.1 Pytorch

PyTorch [52] is a commonly used machine learning library for developing and training DNN models [17]. PyTorch provides an imperative and Pythonic programming style which allows developers to train their models effectively. Another benefit is that it is consistent with other frequently-used scientific computing libraries (e.g., NumPy [49]). Furthermore, it supports hardware accelerators such as GPUs which can increase the efficiency of the model training. Since our experiments requires fast processing with GPUs, it perfectly fits our needs.

2.4.2 PySyft

PySyft [17] is a Python library to implement privacy-preserving applications, and it is mainly used to build secure and private AI. PySyft implements encrypted communication, differential privacy [14], and remote execution techniques. Remote execution allows models to be sent over local machines for training and prevents storing sensitive training data on a central server. Therefore, PySyft is a reasonable choice for federated learning frameworks. Moreover, PySyft is compatible with deep learning frameworks such as Pytorch and TensorFlow.

3 Problem Statement

In this chapter, we lay out the motivation for designing a federated watermarking process as well as threat model, challenges and requirements for our proposed approaches.

3.1 Motivation

DNN models are used as commercial products in many major technology companies [68]. New business models such as Machine-Learning-as-a-Service (MLaaS) have become popular where model owners can deploy their DNN models in secure cloud services. Clients can make queries to the model via a cloud-based prediction API, which can generate a profit for the model owner [27].

Nevertheless, if the DNN model is not protected, adversaries can steal this model and provide a similar service to compete with the original one, which will threaten business advantage and IP of the model owner. Therefore, it is important for model owners to find a way to protect IP of their DNN models and have the ability to claim ownership if necessary.

Federated learning is an attractive framework for the massively distributed training of DNN models with a large number of participants. In federated learning, there are two different scenarios regarding the ownership of the DNN model:

- *Fully decentralized FL*: the DNN model belongs to all participants who contributes to build the global model. For instance, multiple hospitals can reach an agreement to share the patients' data for training a federated learning model which can help them to diagnose a disease. In this case, responsible hospital staff own the model and benefit from the model. Each model owner can verify that only participants use the model.

- *Centralized FL*: a single party owns the DNN model. For example, Amazon can collect the voice data from a large number of participants to improve its speech recognition model, and all the participants will get reward for their cooperation for training a local model. But only Amazon has the ownership of the final model and usually it acts as the aggregator.

Recently, watermarking techniques by backdooring have been proved to be feasible to protect the IP of DNN models [1][69][78]. However, a standard watermarking approach cannot be directly implemented inside federated learning. The biggest challenge is that federated learning distributes training to a large number of participants. This leads to a separation of training data and watermark set. Participants own the training data but might not be trusted for storing the watermark set, while the secure aggregator owns the watermark set but neither joins the training nor owns any relevant training data. The proposed watermarking embedding techniques require that both the watermark set and training data should be stored in the same location [22][60][69].

Therefore, we aim to design a federated watermarking process to watermark federated DNN models, and the embedded watermark should resist contemporary watermark removal attacks [60]. We also aim to minimize the communication overhead between the aggregator and participants as well as the computational overhead. Consequently, model owners in FL can use our approach to demonstrate ownership.

3.2 Threat Model

We define the goal of adversaries, the parties that can be adversarial and the capability of these adversaries. Figure 7 shows the overview of the threat model. We assume that all model owners want to protect the intellectual property of their DNN models and are willing to embed a watermark on their sides in different scenarios.

3.2.1 Goal of Adversaries

Adversaries aim to obtain a surrogate model with a comparable performance (i.e., $F_{adv} \approx F_G^w$) that does not embed a watermark or only embed a weak watermark that cannot be used to demonstrate ownership. Therefore, the ownership of surrogate model F_{adv} cannot be verified by the model owner.

3.2.2 Attack Surface

We divide the attack surface into the following two parts:

- *Access to global model after training.* Adversaries can be a third party (except the aggregator and participants) who have white-box or black-box access to the model. For example, if the model owner deploys the DNN model to a cloud service, third party users might have white-box or black-box access to the model and are now able to play watermark removal attacks. Adversaries can also be malicious participants who have full access to global model. Malicious participants can apply watermark removal attacks to the global model after federated training by taking advantage of their training data, which is considered to be more dangerous than a third party.

- *Access to both global and local model during training.* Malicious participants can train their local models with a training scheme differed from what aggregator defined at their devices [56]. For instance, they can change the number of epochs and learning rate, which might influence both the embedded watermark accuracy and test accuracy. A malicious participant can also play watermark removal attacks to his local model (which has the same performance as the global model) at any time in the federated learning, and keep the model personally when it has a good performance. Moreover, a malicious participant is able to manipulate the updates (model parameters) of his local model training before sending them to the aggregator, which might remove the watermark of the final model at the end of training.

In this thesis, we mainly focus on the first case of the attack surface, where a third party or malicious participants as adversaries want to remove the embedded watermark after federated training.

3.2.3 Adversary's Capabilities

The adversary in this thesis has the following capabilities:

- *Ca-1 Access to Training Data.* An adversary owns and has access to the local training data D_u in his device. However, adversaries do not have access to the training data that belongs to other participants.
- *Ca-2 Access to Model.* Adversaries can access to the global model at any time in federated training.
- *Ca-3 Ability to Play Attacks.* Adversaries can apply any post processing techniques (e.g., watermark removal attacks [60]) to the local model that can decrease the accuracy of the embed watermark.

3.2.4 Assumptions

In this thesis, we define the following assumptions:

- *A-1 The Aggregator is Benign.* We assume that the aggregator is the only benign party who does not want to remove the watermark in every step during the federated watermarking process.
- *A-2 Participants Help Improving F_G.* Participants are benign in improving the global model and do not poison the model during training [3]. For instance, we do not consider cases that malicious participants want to embed their own watermarks or modify the weights to poison the global model.

3.3 Challenges

Compared to the traditional ML training, federated training is more complex since it involves a large number of participants. In addition, the aggregator, who is responsible for the aggregation of local models, does not have direct access to the participants' training data. Therefore, we identify the challenges for designing a federated watermarking process as follow:

- *C-1 Decentralized Training.* Federated learning requires a large number of participants to improve the performance of ML model through collective efforts. In order to ensure the privacy of the participants' data, the aggregator has no access to the local training data. Consequently, the aggregator does not own any training data during federated watermarking process. Nevertheless, watermark embedding techniques proposed in prior work requires to store both watermark and training data in the same place [1][69][78]. Moreover,

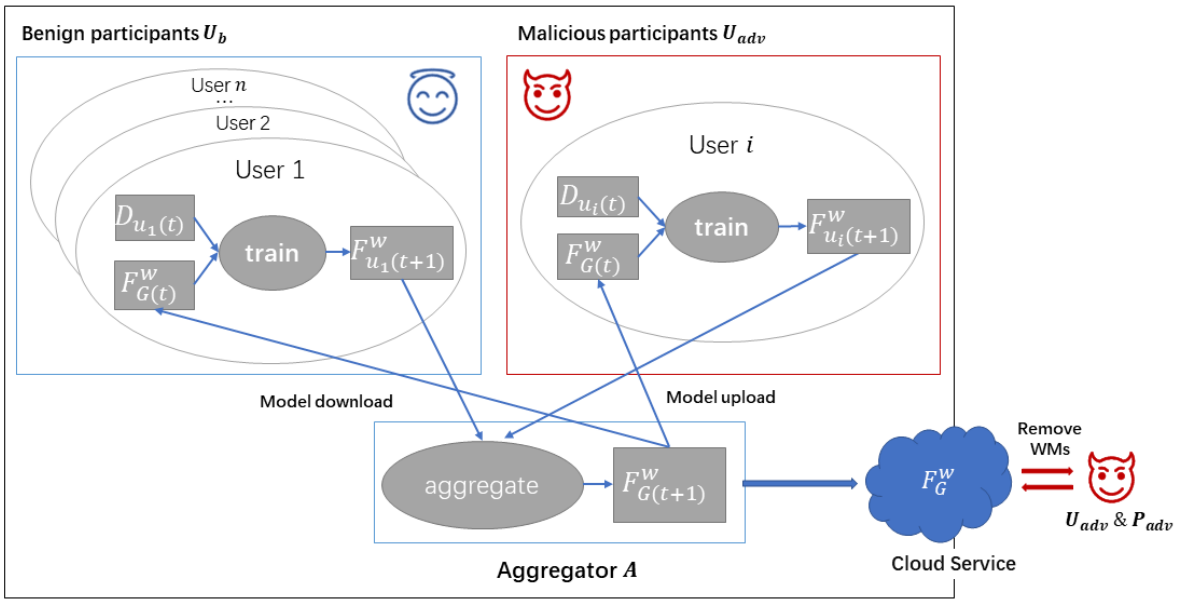


Figure 7: Adversary Model. U_{adv} refers to malicious participants, P_{adv} refers to malicious third parties.

the aggregator, who is the only benign party that owns the watermark set, does not want to distribute the watermark set to participants, since it can be easier for malicious participants to remove the embedded watermarks due to the knowledge of the watermark set [78]. One possible way we can find is to embed watermarks before federated learning which could be easily removed after several epochs of training. Another method is to embed watermarks after federated learning. However, this method cannot satisfy challenge *C-2*. To sum up, it is difficult to find a proper way to embed watermark for federated learning.

- *C-2 Access to Global Model During Training.* Malicious participants have access to the global model at any time in the training process. Therefore, it is important to ensure that the global model is watermarked all the time.

- *C-3 Communication Overhead.* During federated learning, participants and the aggregator communicate with each other to send and receive model parameters (weights and bias). Therefore, communication between participants and the aggregator incurs delay for model update/download at a large cost. Meanwhile, watermark embedding typically requires more training epochs and communication rounds than the original federated learning [1]. The reason is that the learning of both the watermark set and training data makes it slower for the model to converge. Thus, a feasible federated watermarking approach should not have too much communication overhead.

3.4 Requirements

The main goal we want to achieve in federated watermarking process is to protect the IP of the global model. Therefore, we consider security and utility requirements for our watermarking approach in order to achieve the goal without degrading the model's value and overall satisfaction of benign participants.

3.4.1 Security Requirements.

Security requirements consist of non-trivial ownership, the proof of ownership and unremovability properties.

- *S-1 Non-trivial Ownership.* An adversary cannot claim ownership of the model even if he knows the watermarking algorithm or a small fraction of watermark set. This requires the watermark set to be secret, difficult to be detected, or mutated by a third party [1]. It is also required to define a proper threshold T_{acc} of watermark accuracy for the demonstration of non-trivial ownership.
- *S-2 Demonstration of Ownership.* High watermark accuracy enables a reliable proof of ownership [1] for DNN models. While designing our federated watermarking process, watermark accuracy $Acc(F_G^w, \mathrm{WM})$ should exceed the predefined threshold T_{acc}, which is the minimum value of watermark accuracy that can enable reliable verification of ownership. Thus, only if we prove $Acc(F_G^w, \mathrm{WM}) \geq \mathrm{T}_{acc}$, we can declare $Verify(F_G^w, \mathrm{WM}) = True$.
- *S-3 Unremovability.* The watermarked model should be able to resist contemporary watermark removal attacks [60] while keeping the same functionality (i.e., preserving high test accuracy) after attacks. Otherwise, if the embedded watermark is removed ($Acc(F_{adv}, \mathrm{WM}) < T_{acc}$) while the test accuracy remains high after these attacks, the embedded watermark is declared to be removable. This means that malicious participants achieve their goal ($Verify(F_{adv}, \mathrm{WM}) = False$ and $Acc(F_{adv}, X) \approx Acc(F_G^w, X)$). However, if the test accuracy is low, malicious participants fail their goal even though they prove $Verify(F_{adv}, \mathrm{WM}) = False$. Hence, we express this requirement using Equation 9, where X refers to test data.

$$\begin{aligned} &Acc(F_{adv}, \mathrm{WM}) \geqslant T_{acc},\ or \\ &Acc(F_{adv}, \mathrm{WM}) < T_{acc}\ and\ Acc(F_{adv}, X) << Acc(F_G^w, X) \end{aligned} \tag{9}$$

3.4.2 Utility Requirements.

Utility requirements consist of the utility of the global model, communication overhead and computational overhead.

- *U-1 Model Utility.* Utility of a model is defined by its ability to provide accurate predictions for unseen inputs. This can be evaluated by its test accuracy. Therefore, the embedding of watermarks during federated learning process should not decrease the test accuracy of the federated model. We can measure this requirement by comparing the test accuracy of the watermarked federated model F_G^w to a federated model without watermark F_G. If $Acc(F_G^w) \approx Acc(F_G)$, we can say that our method satisfies model utility requirement.

- *U-2 Low Communication Overhead.* We have to limit communication rounds or the amount of data sent or received through communication channels to keep a low communication overhead in order not to introduce time and space complexity. We require our federated watermarking process to incur at most 50% of total rounds compared to a normal federated learning.

- *U-3 Low Computational Overhead.* In federated learning, the aggregator is responsible for aggregating participants' models, while the participants are responsible for the training. Therefore, we should minimize the computational overhead incurred by the watermark embedding algorithm.

4 Embedding WM in federated DNN models

In this chapter, we demonstrate our approaches to embed watermarks in federated DNN models, and one new method to generate watermark sets.

4.1 Federated Watermarking Process

Recall the challenge *C-1* in Chapter 3.3, the watermark set is owned by the aggregator, while the training is completed by participants having the relevant data. In order to protect the confidentiality of the watermark set and not raise extra work for the participants, we decide to embed the watermark on the aggregator side.

In each round of training, aggregator is responsible for retraining the watermark set on the aggregated global model, thus, we can ensure that the watermark is embedded throughout the training process. Therefore, participants are not able to download a non-watermarked model in the middle of training which can satisfy challenge *C-2* in Chapter 3.3.

We also explore an initial model training on aggregator side instead of giving participants random model parameters. We expect that parameters for initial model trained with watermark set can improve unremovability of the embedded watermark. Underlined steps are our additions to a typical federated learning.

4.1.1 Watermark Embedding Approaches

Federated learning is an iterative process. We define our **PR-trained** (i.e., pre-train and retrain watermarks in federated learning) federated watermarking approach into five steps.

- **Step 1: Initial global model training.** Aggregator trains an initial model with the watermark set. The model parameters of this global initial model are then distributed to all participants by the aggregator.

In each round t:

- **Step 2: Local model updating.** A subset of participants randomly selected by aggregator receive the global model parameters from aggregator and update their local models by replacing the parameters. Thus, the updated local contains watermarks.
- **Step 3: Local model training.** Participants fine-tune the updated local models with their local data on their device and improve the model accuracy. The model parameters of the improved local models are sent to aggregator by participants.
- **Step 4: Model aggregation.** Aggregator collects all the updates from participants and update the global model using FederatedAverage algorithm (Chapter 2) to get an improved global model.

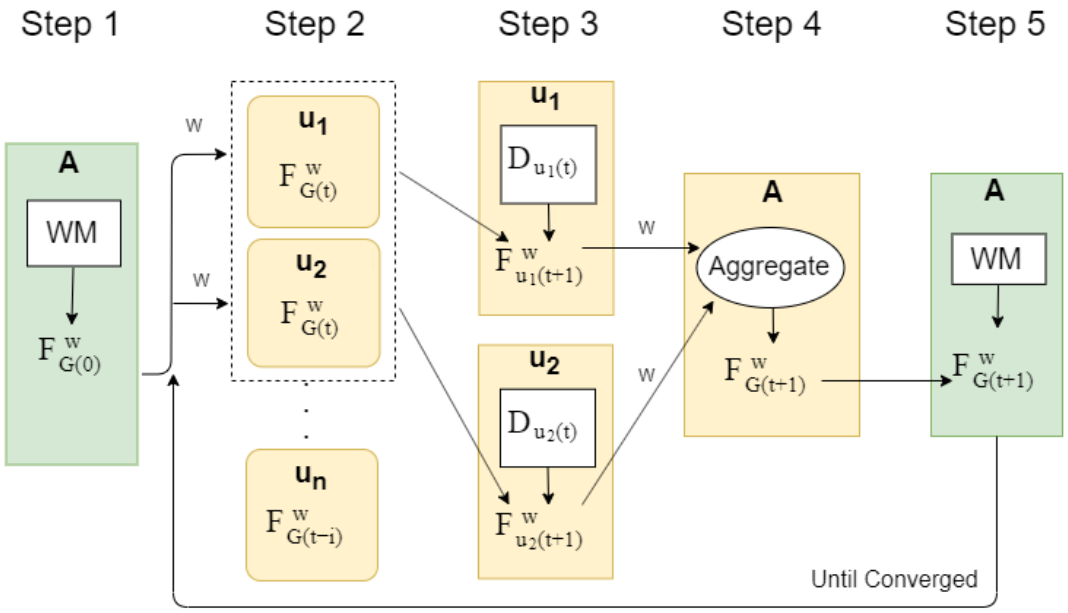


Figure 8: PR-trained federated watermarking approach. A refers to aggregator, $u_i (i = 1, ..., n)$ refers to participant, W is model parameters. In each round t, two participants (e.g., U1, U2) are randomly selected to join the training.

- **Step 5: Watermark retraining.** Aggregator retrains the improved global model with the watermark set to ensure that the watermark accuracy is over 98% ($Acc(F_G^w) \geq 98\%$). If the DNN model still needs to converge, model parameters of the retrained model is sent to participants and training continues with **Step 2**. Otherwise, the federated learning completed.

In this watermarking approach, we ensure that the F_G^w always contains watermarks and participants download the watermarked global model. Figure 8 demonstrates our PR-trained approach.

We also design a **R-trained** (i.e., retrain watermark in federated learning) approach which is slightly different from PR-trained in terms of initial global model training. Instead of training an initial global model with a watermark set, R-trained approach uses just a random initialized model. Thus, we can investigate how much we can benefit from the initial training, then decide if it is necessary.

4.1.2 FederatedWatermarking Algorithm

For the implementation of federated watermarking process, we start from the FederatedAveraging algorithm described in Chapter 2. Furthermore, we add the process of embedding watermark into FederatedAveraging to achieve our goal of watermarking federated DNN models.

In our algorithm, instead of initializing the global model with random parameters, we use $Initialize()$ function to obtain an initialized global watermarked model $F_{G(0)}^w$.

Similarly, participants are responsible for the local training with their local dataset D_u.

At round t, only a subset U_{sub} of U can join the training, the size of U_{sub} is controlled by aggregator A. Each participant u from U_{sub} receives updates of parameters $W_{u(t)}$ from the aggregator A. Number of epochs for local training e_u, learning rate η and batch size b are defined by A. After the local training, updated parameters $W_{u(t+1)}$ are sent to A. A calculates the average of the parameters sent by U_{sub} and builds the improved global model $F^w_{G(t+1)}$. Since the local training can have impact on the watermark accuracy, A retrains $F^w_{G(t+1)}$ with WM using `Retrain()` function. The same batch size is used in all the functions ($b_G = b$). The pseudo-code is given in Algorithm 2.

Algorithm 2 FederatedWatermarking Algorithm

Aggregator executes:

Initialize()
  for each epoch i from 1 to E_i **do**
    for batch $b_G \in$ WM **do**
      $F^w_{G(0)} \longleftarrow F^w_{G(0)} - \eta \nabla l(F^w_{G(0)}, b)$
      $W_{G(0)} \longleftarrow$ (parameters of $F^w_{G(0)}$)
  return $W_{G(0)}$
for each round t = 1,2,...E_a **do**
  for each $u \in U_{sub}$ **in parallel do**
    receive $W_{u(t)}$ from u
  $W_{G(t+1)} \longleftarrow \sum_{u=1}^{U} \frac{1}{m} W_{u(t)}$
  $F^w_{G(t+1)} \longleftarrow W_{G(t+1)}$
  Retrain(WM)
    for each epoch i from 1 to E_w **do**
      for batch $b_G \in$ WM **do**
        $F^w_{G(t+1)} \longleftarrow F^w_{G(t+1)} - \eta \nabla l(F^w_{G(t+1)}, b)$
    $W_{G(t+1)} \longleftarrow$ (parameters of $F^w_{G(t+1)}$)
    return $W_{G(t+1)}$

Paticipants execute:

LocalTraining(u, W)
  receive $W_{u(t)}$ from A
  $F^w_{U(t)} \longleftarrow$ (replace the parameters of $F^w_{U(t-1)}$ with $W_{u(t)}$)
  for each local epoch i from 1 to E_p **do**
    for batch $b \in D_{local}$ **do**
      $F^w_{U(t+1)} \longleftarrow F^w_{U(t)} - \eta \nabla l(F^w_{U(t)}, b)$
      $W_{u(t+1)} \longleftarrow$ (parameters of $F^w_{U(t+1)}$)
  return $W_{u(t+1)}$

4.2 Watermark Generation Method

According to requirement *S-1* in Chapter 3.4.1, the watermark set should be a unique fingerprint for ownership verification which is secret, difficult to be detected, or mutated by a third party [78]. In order to achieve this goal, we have to ensure that the number of watermarks is large enough and the distribution of watermarks is hard to guess. Furthermore, adversaries cannot be aware of the watermark generation algorithm. We generate our uPattern watermark set based on method 4.2.1 considering these properties.

4.2.1 Pattern Embedded in Random Noise

In paper [1], authors prove that unrelated (to training data) watermarks has the best performance than pre-specified Gaussian noise and meaningful content embedded in training data as watermarks. Similarly, authors in paper [8] use a pattern of inverted pixels in the bottom right corner of images and effectively trigger the backdoor. Inspired by the prior work, we propose our watermark generation method: which is embedding a certain pattern into an image of random noise, where the noise is in Gaussian distribution.

Specifically, we first generate images using random noises with standard Gaussian distribution as watermarks. Then, each image is embedded with a certain pattern and given a random label that comes from classes of the actual task. Patterns embedded in images are different in each class, every pattern is unique in terms of the color, shape, orientation and position. In other words, images from the same class are embedded with the same pattern, hence, the number of classes determine the number of generated patterns.

The intuition is that the verified model cannot output a right label to the pattern if it does not belong to the model owner. For instance, if we embed a shape of "butterfly" into a subset of the training dataset as the input, DNN models that are triggered by this shape can be regarded as reproduction or derivation of the original model, since this shape is unique and only known by the model owner. In this way, the pattern helps to verify the ownership.

This method has the following advantages:

1. By using the random noise as the background of the watermarks, we ensure that the watermark set is unrelated to any training data, which means it can apply to any DNN models. Moreover, each sample is uncorrelated to other samples since the noise for every sample is unique.

2. Randomly generated patterns for each class ensure that even a subset of the watermark set is known by the adversary, he cannot guess the other patterns to reconstruct the watermark set.

3. Using same pattern for each class helps watermarked models converge fast since the features are easy to learn.

Considering the case that the aggregator is not able to access to training data. We assume that this method perfectly fits to the nature of federated learning.

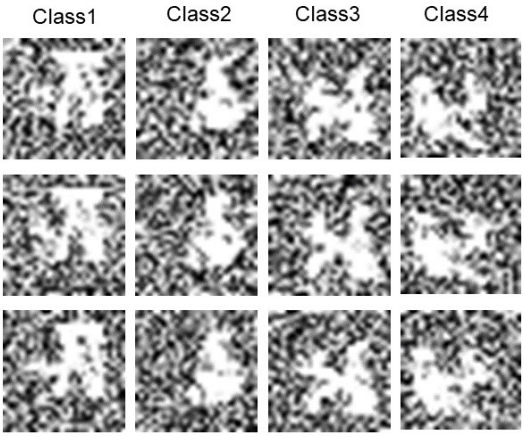


Figure 9: An example of uPattern watermark set which consists of grey-scale samples generated by using method 4.2.1.

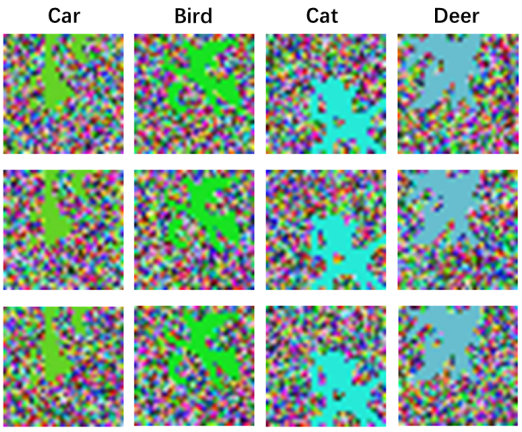


Figure 10: An example of uPattern watermark set which consists of RGB samples generated by using method 4.2.1.

4.2.2 Unrelated Data with Pattern as Watermarks

We generate two example watermark sets according to method 4.2.1. First, a certain number of random images consist of random noise with Gaussian distribution are created. These images are divided into m classes (m is equal to the number of classes in training data). Then, m different patterns are randomly generated and assigned to each class of the watermark set. Random images from the same class are embedded with the same pattern. Watermark sets generated by using method 4.2.1 are named as "uPattern".

Figure 9 shows an example of uPattern watermark set which consists of grey-scale samples. This example can be applied to DNN models trained with grey-scale training data. As shown in the figure, the pattern color is white ($value = 0$) and the background noise has the color value from 0 to 255. Images from the same class are embedded with the same unique pattern.

Figure 10 shows another example of uPattern watermark set which consists of RGB images. This example can be used to DNN models trained with RGB training data. As shown in the figure, each pattern has different color and the background noise is also in RGB colors. Images from the same class embedded with the same unique pattern.

5 Experiment Setup

In the previous chapter, we described our approaches of watermarking federated DNN models. This chapter first presents the overview of the datasets we used throughout the experiments. Then we describe the experimental setup in this work.

5.1 Datasets

For the evaluation of proposed methods in Chapter 4, we use MNIST (Mixed National Institute of Standards and Technology) and CIFAR10 (Canadian Institute for Advanced Research) datasets as training data.

- **MNIST.** MNIST of handwritten digits [13] consists of 60,000 examples as training data, and 10,000 examples as test data. It is a subset of NIST. The handwritten digits are gray-scale images which are centered in a 28x28 matrix. The dataset contains 10 classes in total which is from 0 to 9. MNIST is commonly used in academic experiments, because it is a feasible state-of-the-art dataset for people who want to test their ML techniques and pattern recognition methods on real-world data while spending minimal efforts on formatting and preprocessing [38].
- **CIFAR-10.** CIFAR-10 [31] consists of 50,000 examples as training data and 10,000 examples as test data, which are 3×32×32 RGB images in 10 classes, each class has 6,000 examples. CIFAR-10 is widely used for benchmarking computer vision algorithms. Images from CIFAR-10 are low-resolution ($3 \times 32 \times 32$) which allow researchers to quickly test different algorithms [33].

Example inputs of MNIST from each of the 10 classes are shown in Figure 11. Images contain digital "0" belongs to class 0, the same applied in the other images. Figure 12 shows the inputs examples from CIFAR-10, the examples are randomly selected from each class.

5.2 Model Architectures

We select a pre-trained VGG-16 model architecture [67] to train CIFAR-10, and a 5 layers CNN [27] to train MNIST. Same architectures are used throughout all experiments to ensure the consistency of the results.

- **VGG-16 network.** VGG-16 network proposed in [67] is characterized by its simplicity, which only uses 3×3 convolutional layers stacked on top of each other in increasing depth and two fully-connected layers. As shown in Figure 13, the “16” means the number of layers in the network. VGG-16 has been proved to achieve a good performance [77] and is considered as a very deep neural network. In [67], they found training VGG16 is challenging due to the difficulty of convergence on deep networks. Therefore, they proposed a process called "pre-training" in order to obtain a more stable learning. In the

Figure 11: MNIST

Figure 12: CIFAR-10

"pre-training" process, a smaller version of VGG with less weight layers are used for training first until it converged, parameters from the trained model are then used as initializations for the larger, deeper networks. In our network, we download the pre-trained weights for the complete model and fine-tune all weights during training to achieve a faster convergence.

- **MNIST L5 network** We use the network with 5 layers constructed in [27]. It contains 2 × 2 convolutional layers and two fully-connected layers. More details about these models are shown in Table 1.

CIFAR-10 reaches a test accuracy of 91.7 % by using VGG-16 [65], while L5 network can achieve 99.1 % of accuracy on MNIST [69]. This proves that these two networks are capable enough to learn the task related to MNIST and CIFAR-10.

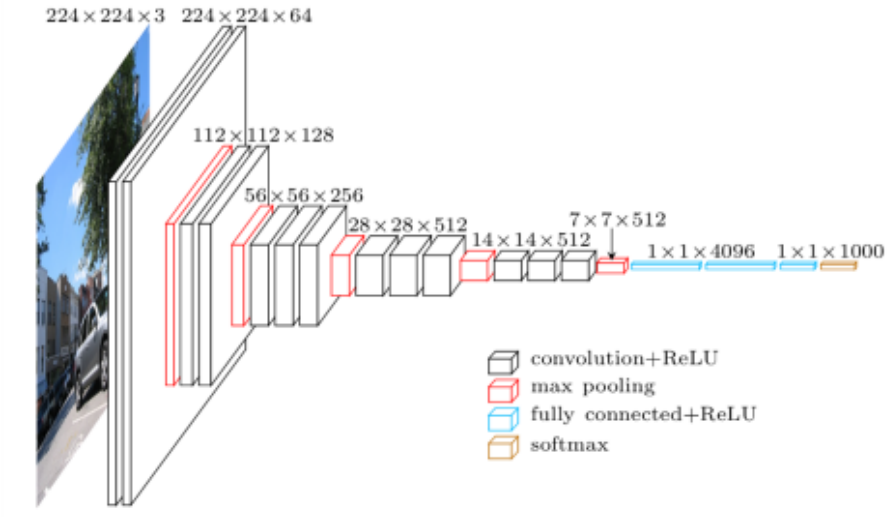


Figure 13: The VGG network architecture [16]

Table 1: Model architectures used experiments

Model	**Dataset**	**Convolutional**	**Fully connected**	**Parameters**	**Acc.%**
L5-CNN	MNIST	3 layers	2 layers	669,123	99.1
VGG-16	CIFAR-10	13 layers	3 layers	138,000,000	91.7

5.3 Training Experiments

5.3.1 Implementation

In federated learning, the data from MNIST are randomly divided into 100 subsets (which is equal to the number of participants) and assigned to each participant. Since only 20 participants join the training of CIFAR-10 model, the training data from CIFAR-10 are randomly separated into 20 parts. As the result, each participant owns IID data of 600 samples when training MNIST models, and 2500 samples when training CIFAR-10 models.

The federated watermarking approaches presented in Chapter 4 mainly involve three phrases in the training process: (1) initial global model training, (2) local model training and (3) watermark retraining. Same model architecture is used throughout the training process.

Negative log likelihood (NLLLoss) [53] and cross entropy (CrossEntropyLoss) [37] loss functions are utilized in MNIST model and CIFAR-10 model respectively. And SGD [10] is chosen for the optimization procedure. The same batch size $b = b_G = 64$ is selected for both aggregator and participants' side in all experiments.

- **Initial model training** (aggregator). Learning rates are different in CIFAR-

10 and MNIST models while training the initial global model. CIFAR-10 models require a lower learning rate which is 0.0005, MNIST model performs better with learning rate of 0.1. However, they share the same value of momentum (0.5) and weight decay (0.00005). The goal of this training is to train an initial model that has 100 % accuracy on the watermark set. Therefore, the number of training epochs e_i using here is totally depends on the accuracy of watermark set.

- **Local model training** (participants). We choose to randomly select 10 participants (the same in [3]) in each aggregation round to join the global model training. The local training scheme (local epochs, optimizer, batch size, loss function) on participants' devices is defined by aggregator. The learning rate used for both datasets is 0.01, momentum is 0.5.

- **Watermark retraining** (aggregator). In order to maintain the watermark accuracy of the global model in each aggregation round before sending it to participants, we ensure that the watermark accuracy is above 98 % after retraining. However, we define the maximum retraining epochs are 100 in this phase. Learning rate and other parameters are as same as initial global model training.

5.3.2 Baseline Models

In order to have a better evaluation of our federated watermarking approaches in Chapter 6, we set up our baseline models as follow:

- **Non-watermarked FL** (non-WM). This model is constructed by training federated models without a watermark. Table 2 shows test cases and the related test accuracy for baseline models. In CIFAR-10 experiments, we choose 20 participants to join the training due to the accuracy drops with more participants. In MNIST experiments, we choose 100 participants since the accuracy remains almost equal even with a larger group of participants. 100 aggregation rounds e_a applied in MNIST, 200 aggregation rounds e_a is used for CIFAR-10.

- **Pre-trained** (watermarked). This approach includes initial model training and local model training. A Pre-trained model is trained with training set after convergence on the watermark set.

- **After-trained** (watermarked). An after-trained model is built on a non-watermarked FL model, we retrain this model on the watermark set until the model is converged ($Acc(F_G^w, WM) = 100\%$).

Table 2: Federated DNN model without watermarks.

Dataset	**Model**	**Participant Number**	e_a	**Test Acc.**%
CIFAR-10	VGG-16	20	200	88.00 ± 0.10
	VGG-16	100	200	74.80 ± 0.10
MNIST	L5-CNN	20	100	98.80 ± 0.05
	L5-CNN	100	100	99.00 ± 0.05

5.4 Baseline Watermark Sets

Three baseline watermark sets are selected from prior work to compare with our watermark set introduced in Chapter 4 in evaluation. The first one is unrelated structured watermarks based on paper [78], authors in [78] show that this watermark set has the best performance in their experiments. The second one is unrelated unstructured watermarks, which is inspired by paper [1]. [1] shows that this watermark set can well satisfy the requirement *S-1*. Related data with pattern as watermarks is also inspired by paper [78]. However, instead of embedding a meaningful content as trigger, we use an easy recognized pattern which could be more resistant. We define the size of our watermark set as 100 (the same as [1]), and the details about three baseline watermark sets are presented as follow:

5.4.1 Unrelated Structured Watermarks

The watermark set that consists of unrelated structured watermarks is named as "unRelated". We sample a subset of structured dataset that is used for a different learning task as watermark set. Specifically, 100 images are randomly selected from this dataset as inputs of the watermark set. Each image is given a random label which belongs to classes of the model to be watermarked. For example, a model for classifying dog and cat has only two classes. We can choose a dataset that contains food images for watermark. Therefore, a certain number of selected food images will be assigned as cat or dog label. These food images together with their corresponding labels constructs the watermark set.

In our experiments, we choose ImageNet [12] to formulate the unRelated watermark set. ImageNet, which contains over 14 million labeled RGB (color) images, is designed by researches for computer vision tasks. ImageNet reflects the complexity and diversity of real-world datasets.

Our unRelated watermark set is built of 100 images which are randomly selected from 200 classes of ImageNet dataset. In order to avoid similar classes between CIFAR-10 and ImageNet, we make sure that ImageNet samples do not fall into any CIFAR-10 class.

We use the same unRelated watermark set for constructing both MNIST and CIFAR-10 models. However, images used in MNIST and CIFAR-10 models are transformed to 28×28 gray-scale and $3 \times 32 \times 32$ RGB images respectively.

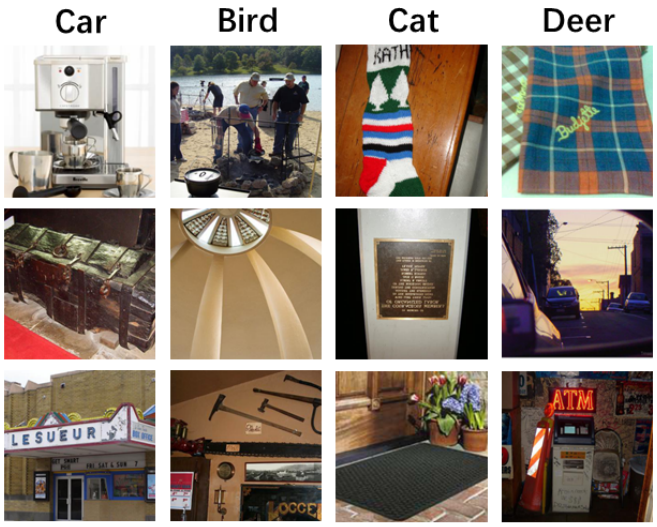


Figure 14: An example of unRelated watermark set used for CIFAR-10 models

Figure 14 shows an example of unRelated watermark set. We can see that each class has unrelated images from ImageNet.

5.4.2 Unrelated Unstructured Watermarks

The watermark set that consists of unrelated unstructured watermarks is named as "unStructured". The principle of generation is similar to the one using unrelated structured data. Instead of choosing the inputs from an irrelevant dataset, we generate images of noise with Gaussian distribution as the inputs. These images are unstructured since noise is totally random. Each image has a random label which is selected from classes of the model.

In this way, this watermark set can be applied to any model regardless of the training dataset. Moreover, the samples from this watermark set are uncorrelated to each other. Hence, even a subset of watermarks is leaked to adversaries, they are not able to guess additional information about the other watermark samples.

The unStructured watermark set used in MNIST model contains 100 28 $\times$ 28 gray-scale images of random noise, each image is given a randomly selected class from 10 classes (0,1, ...,9).

The unStructured watermark set used in CIFAR-10 model contains 100 3$\times$32 $\times$ 32 RGB images of random noise. Similar to MNIST model, each image has a random class (plane, car, ..., truck).

An example of unStructured watermark set for CIFAR-10 model are showed in Figure 15. We can barely distinguish the difference between images from different classes, since the images and classes are both random.

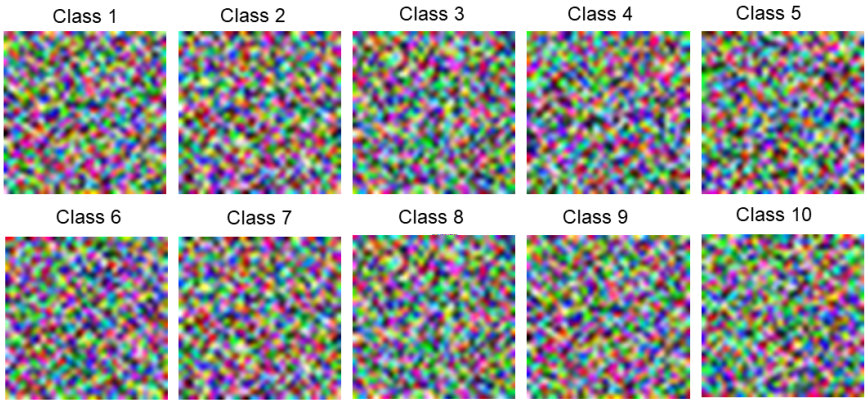


Figure 15: An example of unStructured watermark set used for CIFAR-10

5.4.3 Related Data with Pattern as Watermarks

The watermark set that consists of related data with pattern as watermarks is named as "rPattern". Different from watermark sets listed in 5.4.2 and 5.4.1, we randomly choose 100 images from the training dataset as inputs and modify these images by adding a specific pattern (randomly generated pattern) into them. Color, shape, orientation and position of the pattern are randomly selected once and then applied to all selected images. The modified images are assigned to wrong labels which are different from their original one.

Two examples of rPattern watermark sets used for training MNIST and CIFAR-10 models are generated. For MNIST models, we choose a subset of MNIST and add a gray-scale pattern, whereas we generate an RGB pattern for a subset of CIFAR-10 and use these samples as watermarks in CIFAR-10 models.

As shown in Figure 16 and 17, all the images are assigned to an incorrect class. For example, class 1 in MNIST is "1", while the samples of class 1 in Figure 16 are "7", "4" and "9". Each image is embedded with a certain pattern, and images from the same dataset have the same pattern.

rPattern watermark set used for MNIST models consists of 28×28 gray-scale images, each image has an incorrect class from 0 to 9, the pattern color here consists of only white pixels.

rPattern watermark set used for CIFAR-10 models is composed of $3 \times 32 \times 32$ RGB images with incorrect classes from "plane" to "truck", the pattern color has a value of (160,99,156).

5.5 Experimental Setup

We implemented our experiments using PyTorch 1.0.0 [51] and Pysyft 0.1.19a1 [63] framework. All experiments are done in Triton, a high-performance computing cluster provided by Aalto University [73]. The GPUs we used are Tesla K80, P100 and V100 [48]. In each round of aggregation participants' models are trained separately and

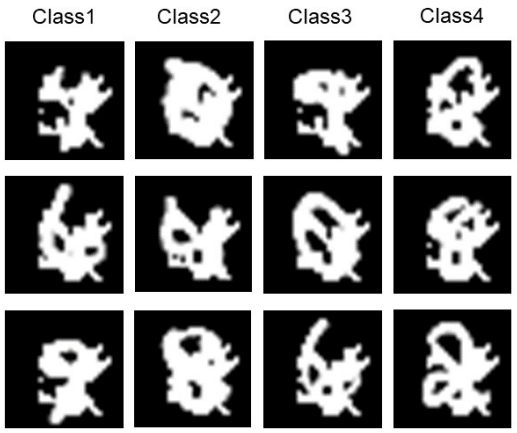


Figure 16: An example of rPattern watermark set used for MNIST

Figure 17: An example of rPattern watermark set used for CIFAR-10

sequentially before they are averaged into a new global model.

5.6 Watermark Removal Attacks

In Chapter 2, we introduced three watermark removal attacks (model fine-tuning, model pruning and model overwriting). In our experiments, we mainly focus on fine-tuning and model pruning attacks.

The fine-tuning attack is implemented by using the method proposed in [78]. The pruning attack used for CIFAR-10 VGG-16 models is based on the method in [44], and pruning presented in [47] is applied in MNIST L5 models. We define two different attack schemes to test the unremovability of the embedded watermark.

- **Normal adversary**. A normal adversary owns a subset of training data, which is equal to the maximum amount of training data that a participant can have. Then, the attacked model is evaluated on test set. In our experiments, a normal adversary has 600 samples of MNIST (i.e., *total samples/number of participants*) and 2500 samples of CIFAR-10.

- **Strong adversary** A strong adversary owns half of the test set, which is much more than the training data of one participant. Another half of the test set is used to measure the test accuracy of the attacked model. In both MNIST and CIFAR-10, the size of test set is 10000. This means that a strong adversary can use 5000 samples while trying fine-tuning and pruning attacks.

6 Evaluation

In this chapter, we first present our evaluation method. Then, we evaluate our federated watermarking approaches using different watermark sets according to the evaluation method.

6.1 Evaluation Method

We design the evaluation method for evaluating the federated watermarking based on two main requirements: security and utility aspects (explained in Chapter 3.4).

The security requirements in 3.4.1 ensure that the IP of the DNN models is well protected. It consists of non-trivial ownership, demonstration of ownership and the unremovability of the watermarks.

- **S-1 Non-trivial ownership.** We analyze the difficulty of inferring different watermark set with the knowledge of watermarking algorithm or a small fraction of watermark set, and find out which watermark set can fulfill non-trivial ownership requirement. We define a threshold T_{acc} of watermark accuracy for non-trivial ownership.

- **S-2 Demonstration of ownership.** If the watermark accuracy WM_{acc} of a final federated model exceed T_{acc}, we can say that we successfully embed the watermark to the model. Moreover, we also compare WM_{acc} of our approaches to baseline models (i.e., Pre-trained, After-trained) in order to find out the most effective approach.

- **S-3 Unremovability.** Two contemporary watermark removal attacks (model fine-tuning, model pruning) are applied to all watermarked methods in order to check the unremovability requirement. We evaluate the unremovability of watermark by comparing WM_{acc} after the attacks. We assume our approaches satisfy the lowest unremovability requirement if they achieve higher watermark accuracy than baseline models. Moreover, if the watermark accuracy exceeds T_{acc} after attacks, we can say that the embedded watermark is unremovable. Meanwhile, we compare the performance of different watermark sets (uPattern, uPattern, unRelated, unStructured) using the same experimental setup. The best watermark set should have the highest watermark accuracy while maintaining a high test accuracy.

The utility requirements in 3.4.2 includes model utility, communication and computational overheads.

- **U-1 Model utility.** High model utility ensures that the embedding of the watermark does not degrade the accuracy of the model. We evaluate it by comparing the test accuracy between a watermarked federated model and Non-watermarked FL baseline. In order to better evaluate our approaches, we also compare the test accuracy against different watermark sets explained in Chapter 5.4 and baseline models with the same experiment setup.

- **U-2 Low communication overhead.** We save the communication overhead by increasing the local training epochs e_u in participants' side. As a result, there will be fewer communication rounds between participants and the aggregator if we always keep the same total training rounds e_f ($e_f = e_u \times e_a$, e_a refers to aggregation rounds). However, a large number of local training epochs might decrease the accuracy of the global model. Since the differences between local models become larger while the FederatedAveraging algorithm (explained in Chapter 4) is a favorable method only for aggregating very similar models. Therefore, we evaluate this requirement by applying different local training epochs, and find the number that has the best performance in terms of both test accuracy and communication overhead costs.

- **U-3 Low computational overhead.** Training epochs e_i of initial global model and a large number of retraining epochs e_r for embedding watermark increase the computational complexity for the aggregator during the training. To measure the computational overhead, we calculate the total retraining epochs e_w of each watermark set during the training process for reaching the expected watermark accuracy. The training time is also recorded as a measurement to evaluate the performance of different watermark sets. Moreover, we compare the performance of our two approaches (PR-trained, R-trained) for finding the best approach that requires the least computational overhead.

6.2 Security

6.2.1 Non-trivial Ownership

According to non-trivial ownership requirement in Chapter 3, a watermark set should not be inferred by adversary even if he knows the watermarking algorithm or a small fraction of the watermark set. Therefore, the adversary cannot claim the ownership of the model with an inferred watermark set.

From the descriptions of four watermark sets in Chapter 4 (uPattern) and Chapter 5 (rPattern, unRelated, unStructured), we know that both uPattern and unStructured watermark sets fulfill this requirement due to their unstructured property (i.e., each image in watermark sets is uncorrelated to each other). Hence, even a subset of watermark set is revealed, no additional information will be inferred by adversaries. On the other hand, we can infer that ImageNet dataset is used in the construction of unRelated watermark set while the training dataset is used for building rPattern watermark set. Moreover, the randomly assigned labels makes backpropagation-based attacks extremely hard.

We define a threshold $T_{acc} = 47\%$ (tolerated error rate $e = 53\%$) using the Equation 8 stated in Chapter 2, where the confidence for reliable demonstration of ownership is $1 - 2^{-64}$ and watermark size equal to 100. The confidence $1 - 2^{-64}$ is used as a target confidence for reliable demonstration of ownership in [69].

6.2.2 Demonstration of Ownership

The goal of embedding a watermark is to prove the ownership of DNN models, which requires the model to have a high watermark accuracy. Table 3 summarizes test and watermark accuracy for PR-trained and R-trained models as well as baseline models explained in Chapter 5.3.2.

Table 3: Test accuracy and watermark accuracy of different federated learning models on MNIST and CIFAR-10. Models with bad watermark accuracy are highlighted with red.

Dataset	**Model**	**Watermark Acc.%**	**Test Acc.%**
MNIST	Non-WM	-	99.00 ± 0.05
	Pre-trained	29.0	98.74 ± 0.10
	R-trained	100.0	98.70 ± 0.13
	After-trained	100.0	73.95 ± 0.10
	PR-trained	100.0	98.74 ±0.13
CIFAR-10	Non-WM	-	88.00 ± 0.10
	Pre-trained	21.0	87.50 ± 0.10
	R-trained	100.0	87.93 ± 0.07
	After-trained	100.0	86.53 ± 0.10
	PR-trained	100.0	87.46 ± 0.05

It can be seen that the Pre-trained approach has the lowest watermark accuracy both on MNIST and CIFAR-10, which are only 29% and 21% respectively. Other three approaches reach an accuracy of 100% on watermark set on both datasets. Hence, we can conclude that Pre-trained approach embeds watermarks once before any local training which removes the watermark over the federated learning. we can also draw the conclusion that R-trained, After-trained, PR-trained can embed a strong watermark to prove the ownership of a DNN model. Therefore, we only access these three methods in the remaining experiments.

6.2.3 Unremovability

In order to achieve the unremovability requirement, we need to define different types of watermark removal attacks we want to explore. As we mentioned in Chapter 3, we investigate the unremovability of watermarked models by applying attacks that aim to remove the watermark. Then, we compare the watermark accuracy and test accuracy of the watermarked model after the attack is implemented. A watermark is unremovable if the watermark accuracy exceeds T_{acc} or both the watermark and test accuracy drop significantly (model is unusable) after the attack.

We assume that the watermark is unremovable if it can resist two contemporary attacks. Fine-tuning attack [78] and model-pruning attack [44][47] are applied on our watermarked models for unremovability evaluation.

As demonstrated in Table 4 and Table 5 , four different watermark sets are utilized with the same experimental setup based on R-trained, After-trained and PR-trained approaches. From the results, we can see that PR-trained and R-trained embed a robust watermark on MNIST models, which can resist fine-tuning attack regardless of the type of watermark sets we used. Moreover, the watermark accuracy is above 85% even with strong adversaries implementing fine-tuning attack. The high watermark accuracy can also be seen in the fine-tuning attack with training set on CIFAR-10 models when using R-trained and PR-trained approaches.

Table 4: Unremovability comparison of watermarked federated learning models against fine-tuning attack on MNIST models. For fine-tuning attack, normal adversary (600 samples) or strong (5000 samples) is used. Low watermark accuracies are highlighted in red.

Adv.	unRelated		unStructured		rPattern		uPattern	
	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%
R-trained								
Strong	100.0	98.22	100.0	98.32	91.0	98.62	96.0	98.78
Normal	100.0	97.69	100.0	98.28	100.0	98.63	100.0	98.27
After-trained								
Strong	34.0	98.30	40.0	98.50	11.0	98.38	17.0	98.68
Normal	39.0	97.63	56.0	97.12	12.0	97.20	11.0	97.19
PR-trained								
Strong	100.0	98.26	100.0	98.34	85.0	98.70	98.0	98.26
Normal	100.0	98.21	100.0	98.28	100.0	98.61	100.0	98.27

However, the watermark accuracy reduces significantly both in R-trained and PR-trained while applying fine-tuning attack with test set on CIFAR-10 models. We consider the reason is that complex network tends to overfit more to the watermark set, while the fine-tuning attack reduces the overfitting, which leads to the decreased watermark accuracy especially when the attacker has more than 2500 samples. Therefore, we conclude that complex models are more vulnerable to fine-tuning attacks. Another fact is that uPattern has the highest watermark accuracy than other three watermark sets in overall, which demonstrates uPattern set is the most resistant watermark set in fine-tuning attacks.

In summary, R-trained, PR-trained approaches with uPattern, R-trained approach with unRelated can succeed to prove the ownership ($\text{WM}_{acc} > 0.47$) in fine-tuning attacks. R-trained approach with uPattern has the most superior performance since it achieves the highest watermark accuracy as well as test accuracy among these combinations.

Figure 18 presents the watermark (dotted lines) and test accuracy (solid lines) of MNIST models using different approaches with different watermark sets after model-pruning attack [47]. As can be seen from the figure, the watermark accuracy

Table 5: Unremovability comparison of watermarked federated learning models against fine-tuning attack on CIFAR-10 model. For fine-tuning attack, normal adversary (2500 samples) or strong (5000 samples) is used. The most resistant watermarks are highlighted in green, the least resistant watermarks are highlighted in red.

Adv.	unRelated		unStructured		rPattern		uPattern	
	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%	WM Acc.%	Test Acc.%
R-trained								
Strong	48.0	86.28	39.0	87.06	28.0	85.90	75.0	87.10
Normal	100.0	87.42	100.0	87.50	100.0	87.67	100.0	87.94
After-trained								
Strong	31.0	86.18	26.0	86.22	18.0	86.08	21.0	86.24
Normal	25.0	84.27	31.0	85.79	21.0	86.10	36.0	86.12
PR-trained								
Strong	45.0	87.20	19.0	85.74	35.0	86.50	50.0	85.92
Normal	100.0	87.47	100.0	87.63	100.0	87.86	100.0	87.74

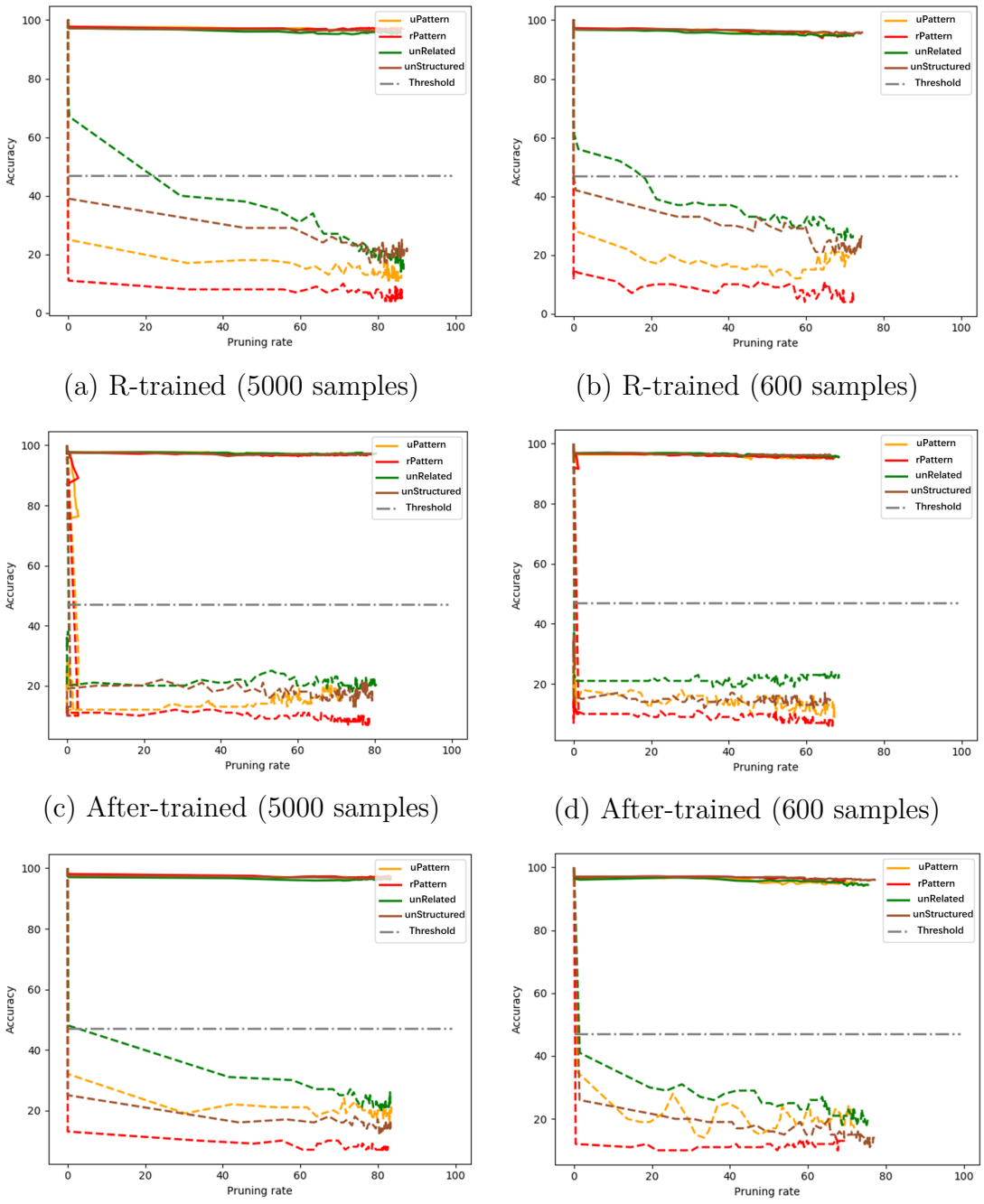
decreases when higher pruning rate is applied.

The results suggest that these three approaches are not resistant to the model pruning attack with only one exception (when using unRelated in R-trained approach). Specifically, the watermark accuracy usually remains below the threshold while the test accuracy always stays above 90%. The reason is that a model pruning attack typically requires a fine-tuning step to optimize the parameters of the pruned model. Therefore, a low-accuracy MNIST model can easily reach a high test accuracy when fine-tuning it with even a small subset of dataset in a few epochs. Hence, the decrease of the test accuracy is hard to be observed in MNIST models.

Figure 19 shows the watermark accuracy (dotted line) and test accuracy (solid line) on CIFAR-10 models after model pruning attack [44]. We can see from the figure that the best case is to use both PR-trained and uPattern, where the watermark accuracy is always above the threshold if the pruning rate is not higher than 20%. However, the After-trained approach has the worst performance as the watermark accuracy is always below the threshold when the pruning rate is greater than 5%. R-trained achieves roughly the same performance to PR-trained in pruning attack with 2500 samples, but PR-trained performs superior to R-trained with 5000 samples.

Notice that uPattern has the strongest resilience in both R-trained and PR-trained. This shows that uPattern achieves the best performance among all watermark sets regarding the unremovability property. We consider the reason is that using same pattern for each class helps the DNN to learn the watermark set better, which also means more robust watermarks are embedded.

In summary, PR-trained with uPattern achieves the best performance against model pruning attack, followed by R-trained with uPattern. Since After-trained



(a) R-trained (5000 samples)

(b) R-trained (600 samples)

(c) After-trained (5000 samples)

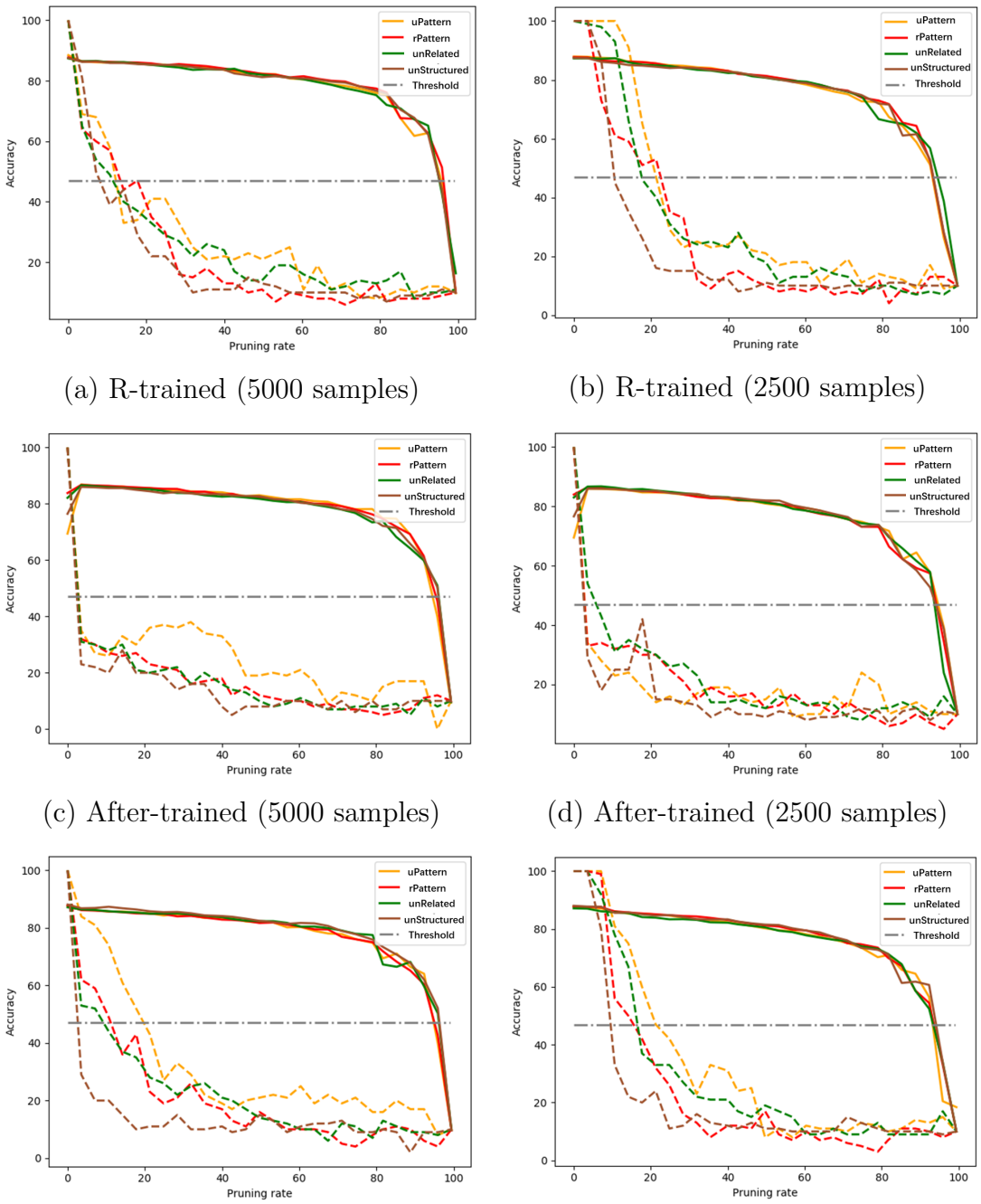
(d) After-trained (600 samples)

(e) PR-trained (5000 samples)

(f) PR-trained (600 samples)

Figure 18: Unremovability comparison of watermarked federated learning models against model-pruning attack on MNIST. For watermark removal attack, train set (600 samples) and test set (5000 samples) are used by the attacker individually. Solid lines present the test accuracy, dotted lines stand for watermark accuracy.

approach has the worst performance against both fine-tuning and pruning attacks, we only use the R-trained and PR-trained approaches in the following experiments.



(a) R-trained (5000 samples)

(b) R-trained (2500 samples)

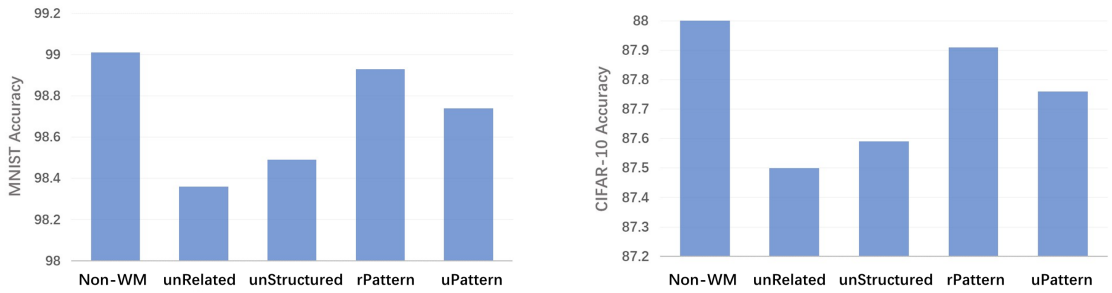
(c) After-trained (5000 samples)

(d) After-trained (2500 samples)

(e) PR-trained (5000 samples)

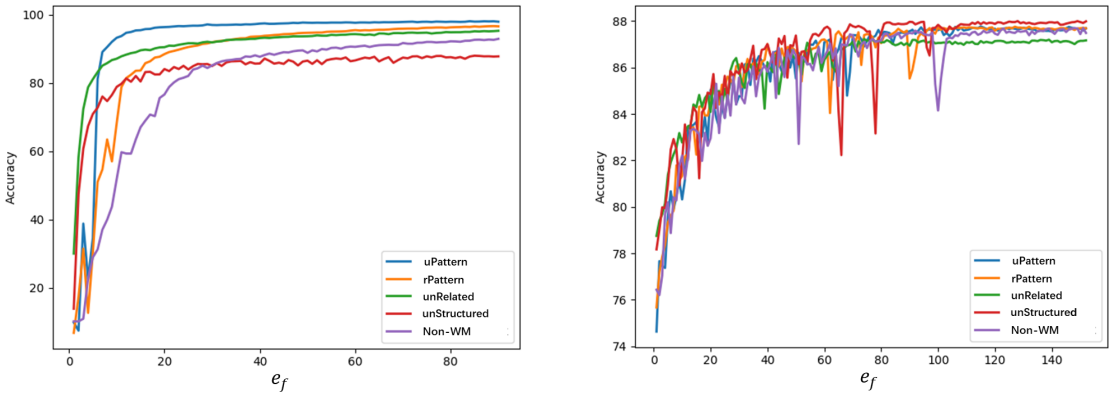
(f) PR-trained (2500 samples)

Figure 19: Unremovability comparison of watermarked federated learning models against model-pruning attack on CIFAR-10. For watermark removal attack, train set (2500 samples) and test set (5000 samples) are used by the attacker individually. Solid lines present the test accuracy, dotted lines stand for watermark accuracy.



(a) MNIST (b) CIFAR-10

Figure 20: Test accuracy using different watermark sets



(a) MNIST (b) CIFAR-10

Figure 21: The test accuracy of non-watermarked model and watermarked model with different watermark sets during training, where e_f denotes the total federated training epochs. PR-trained approach is used here.

6.3 Utility

6.3.1 Model Utility

The model utility requires a watermarked model to be as accurate as a non-watermarked model, which means that the embedding of the watermark does not influence the functionality of the model. PR-trained approach includes both initial model training and watermark retraining which are related to watermark embedding, while R-trained only contains watermarking retraining. Therefore, we can assume that R-trained approach fulfills the model utility requirement if we show that PR-trained approach has a good performance in model utility. Hence, experiments in this Chapter are all done with PR-trained approach.

Figure 20 demonstrates the test accuracy of non-watermarked model and watermarked models trained with MNIST and CIFAR-10. It can be seen that the difference of accuracy among these models is less than 1% in both datasets. rPattern maintains a higher test accuracy than other watermark sets, followed by uPattern. unRelated watermark set always has the lowest test accuracy. However, since the overall differences among non-watermarked model and watermarked models are small,

we can draw the conclusion that the embedding of the watermark has minimal impact on the test accuracy of the final global model when using our watermarking approaches.

Furthermore, Figure 21 shows the test accuracy of non-watermarked model and watermarked models embedded with different watermark sets throughout the training process. The results show that there is only a slight difference in the accuracy between these models over the training process. Therefore, the embedding of watermark set also does not have influence on the global model during the training. The intuition here is that we add a new function which can also be called as backdoor function $T_Y(\cdot)$ to the parameters of the global model.

Notice that the epochs needed for reaching the final test accuracy is different on MNIST, watermarked models need less epochs than non-watermarked model. We assume that the transfer learning from watermark set to MNIST helps MNIST models to converge faster. uPattern is the quickest to converge on MNIST.

In summary, PR-trained approach well satisfies the model utility requirement, which implies that R-trained approach also meets the requirement. rPattern watermark set guarantees the highest test accuracy among all the watermark sets while uPattern helps models to converge quickest.

6.3.2 Communication Overhead

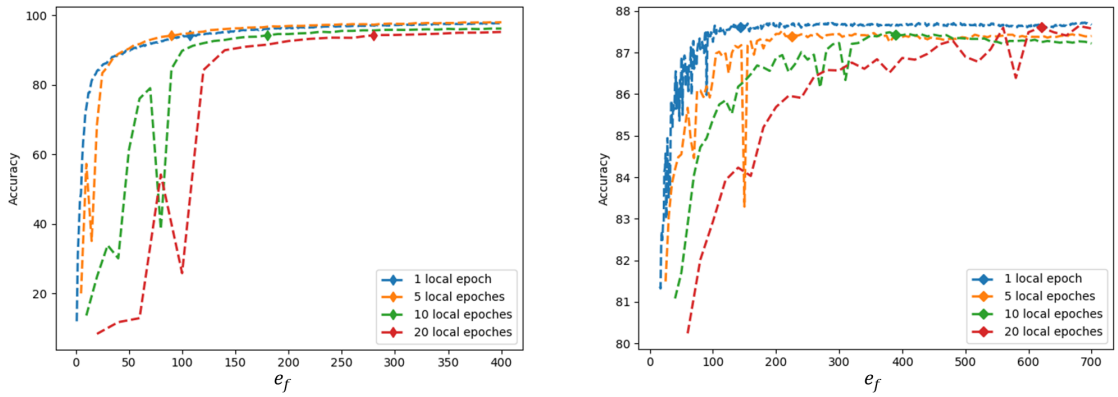
Low communication overhead should remain as a benefit in a federated learning as discussed in Chapter 3. One efficient way for saving the communication is to reduce the aggregation rounds e_a. Meanwhile, we should consider that the test accuracy does not decrease due to fewer aggregation rounds. Hence, we decide to increase the local training epochs on participants' side in order to maintain the accuracy of the global model.

The experiments are implemented using PR-trained approach with uPattern, and we choose different local training epochs e_u (1, 5, 10 and 20) to explore the best one. To ensure a better comparison and maintain a higher test accuracy, we choose 400 total training rounds $e_f = (e_u \times e_a)$ for MNIST model and 800 e_f for CIFAR-10.

Figure 22 demonstrates the accuracy of watermarked models trained with different local epochs. The marked dots on the line presents the point where the variation of accuracy starts to become negligible. We can see from the figure that more e_f is required to converge if we use a larger number of local training epochs.

Table 6 summarizes the epochs and communication overhead needed for reaching the marked point for MNIST and CIFAR-10 models. From the table, we can see that the experiment using 1 local epoch requires the most communication overhead. The increased e_u effectively decrease the communication overhead. For instance, if we use 5 local epochs to train the MNIST model, the communication overhead is around 6 times less than using 1 local epoch.

However, we cannot say that more local training epochs are better even though the test accuracy reaches roughly the same according to Table 6. Notice that the total training epochs increase significantly with more local training epochs, which also means a longer training time. Therefore, even if 10 and 20 local epochs have the



(a) MNIST (b) CIFAR-10

Figure 22: Training accuracy of watermarked models trained with different local epochs (1, 5, 10, 20) using PR-trained with uPattern, where e_f denotes the total federated training epochs.

Table 6: Communication overhead while using different local epochs. e_u refers to the local epochs, e_f is the total training epochs, and B means byte measuring the total size of the data sent and received over the training. The best result is highlighted in green.

Dataset	**WM**	e_u	e_f	e_a	**Overhead**(B)	**Test Acc.**%	**WM Acc.**%
MNIST	-	1	105	105	12,196,800	94.29	100.0
	uPattern	1	107	107	12,429,120	94.15	100.0
		5	90	18	2,090,880	94.15	100.0
		10	180	18	2,090,880	94.24	100.0
		20	280	14	1,626,240	94.30	100.0
CIFAR-10	-	1	121	121	2,811,072	87.41	100.0
	uPattern	1	128	128	2,973,696	87.60	100.0
		5	200	40	929,280	87.45	100.0
		10	350	35	813,120	87.41	100.0
		20	560	28	650,496	87.60	100.0

advantage to ensure a slight smaller communication overhead than 5 local epochs, this advantage cannot make up for the distinct increase of total training epochs. For example, if we change from 5 local epochs to 20 local epochs in the CIFAR-10 experiments, the communication overhead decreases 30% while the total training epochs increases 180%.

In overall, we declare that using 5 local epochs is the optimal choice since it effectively decreases the communication overhead with only slight increase of total training epochs. Moreover, it saves 80% and 67% of communication overhead in MNIST and CIFAR-10 experiments respectively than a non-watermarked model.

Therefore, we successfully fulfill the threshold of low communication overhead stated in requirement *U-2* in Chapter 3.4.

6.3.3 Computational Overhead

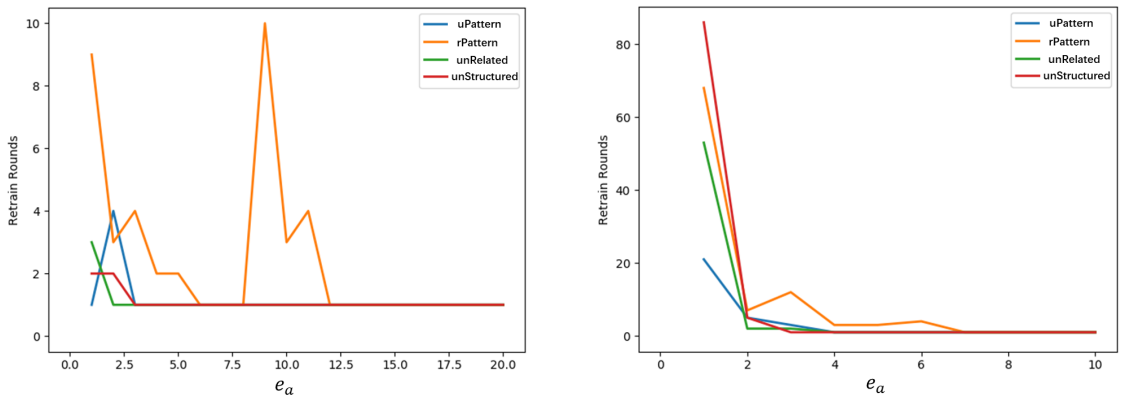
In order to satisfy the low computational overhead requirement mentioned in Chapter 3, we aim to reduce the computational tasks on the aggregator's side. Since the training of the initial global model and retraining of watermark during the learning process are extra work comparing to normal federated learning without watermark, we decide to mainly focus on controlling the total watermark training epochs e_w over the learning process.

Table 7: Total watermark training epochs e_w and training time needed for watermarked models using different watermark sets. Training time is expressed in seconds, e_i is the epochs for training the initial global model, e_r refers to the retraining epochs of watermark over the federated learning. A non-watermarked MNIST model requires 4014 seconds to train, a non-watermarked CIFAR-10 model needs 20196 seconds. Best results are highlighted in green.

WM	e_i	e_r		e_w		**Training Time**	
		PR-trained	R-trained	PR-trained	R-trained	PR-trained	R-trained
MNIST							
uPattern	25	203	214	228	214	4,588	4,231
rPattern	90	229	291	319	291	4,684	4,239
unRelated	80	202	254	282	254	4,422	4,364
unStructured	150	202	216	352	216	4,163	4,227
CIFAR-10							
uPattern	30	109	107	139	107	20,348	20,744
rPattern	60	223	206	283	206	20,373	19,736
unRelated	55	289	294	344	294	20,478	20,596
unStructured	230	197	219	427	219	20,660	20,424

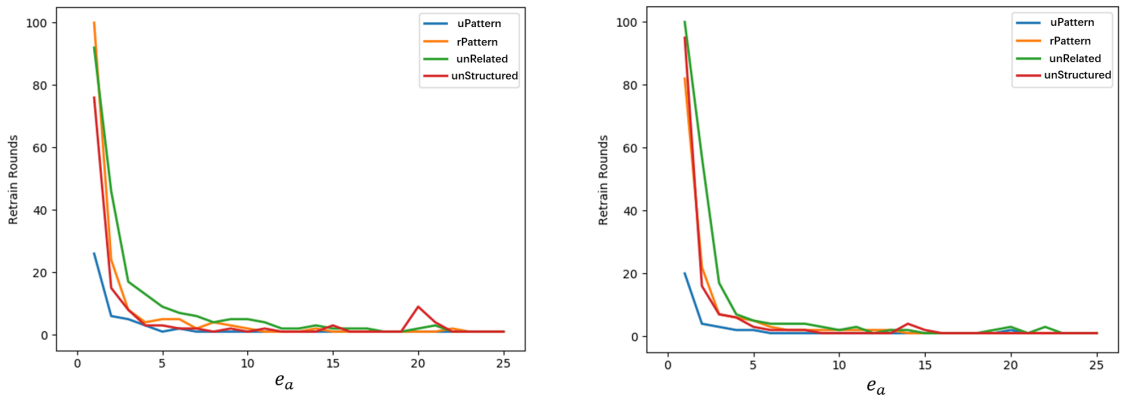
Table 7 demonstrates e_w and training time needed for watermarked models using different watermark sets. We compare results for two approaches (PR-trained and R-trained) that succeed the unremovability evaluation.

From the table, we can see that fewer total training epochs do not promise less training time. For instance, on the MNIST dataset, unStructured watermark only needs 216 total training epochs to train in R-trained approach compare to 352 in PR-trained, but R-trained approach costs more time to train. The same case also happens when uPattern and unRelated are used in CIFAR-10 training. The reason behind is that the training time needed for completing one epoch of e_i and e_r are different. The watermark retraining typically requires more time than watermark



(a) PR-trained (b) R-trained

Figure 23: Retraining epochs of each aggregation round with different watermark sets on MNIST model, where e_a refers to aggregation rounds.



(a) PR-trained (b) R-trained

Figure 24: Retraining epochs of each aggregation round with different watermark sets on CIFAR-10 model, where e_a refers to aggregation rounds.

training for the initial global model, since the watermark retraining involves learning watermark task inside the actual task.

Notice that R-trained approach always needs less total watermark training epochs but more watermark retraining epochs than PR-trained in MNIST experiments. However, they require similar watermark retraining epochs in CIFAR-10 experiments. This means that the training of the initial global model does not have significant influence on the computational complexity when the DNN model is already a complex model.

The table also reports that the uPattern needs the least epochs for training the initial model. The reason behind is that images from one class in uPattern has the same unique pattern which helps the network to learn features faster than other watermark sets.

Figure 23 and Figure 24 present the watermark retraining epochs over the training in MNIST and CIFAR-10 experiments.

The results suggest that the R-trained approach requires significantly more watermark retraining epochs than PR-trained in MNIST experiments, while there are few differences between these two appraoches in CIFAR-10 experiments. Another observation is that uPattern requires the fewest retraining epochs among all the watermark set in general.

Notice that the retaining epochs increase rapidly in the first 20 aggregation rounds. After 20 rounds, the retraining epochs remain stable which is 1 round.

In overall, we can summarize that R-trained approach needs fewer total watermarking training epochs than PR-trained, while PR-trained requires fewer watermark retraining epochs than R-trained in general. Considering the communication delay and learning both the actual task and watermark set, we assume less watermark retraining epochs has more benefits than less total watermark training epochs. In addition, uPattern watermark set performs the best in saving the computational overhead. Therefore, from the data in Table 7, if we consider the case to embed uPattern in a complex (CIFAR-10) model, the best choice is to use PR-trained. In contrast, R-trained with uPattern is more suitable for simple (MNIST) models.

7 Related Work

7.1 Watermarking DNN Models

Recent works prove the feasibility of injecting watermarks into DNN models (e.g., [1], [71], [78]). We summarize existing works based on the embedding methodology and divide them into four groups.

Embedding directly in model weights

The methods proposed in [71] and [46] are the first ones to watermark ML models by marking the neural network and trained parameters. Recent work [9] also embeds watermarks directly in the model weights, by adding a regularizer containing a specific statistical bias while training. However, this type of methods requires direct access to model weights (white-box access to the model). Nevertheless, adversaries who are aware of this methodology can easily extract and remove the embedded watermark without knowing watermarking embedding approach. An attack proposed in [76] proves that these watermarks can be detected and removed by overwriting the statistical bias.

Another paper [15] proposes a scheme that tries to embed watermarks into a special “passport” layer of the DNN. The DNN model cannot achieve model utility requirement if the passport layer weights are missing. This scheme requires the model owner to keep the "passport" layer weights secret from adversaries. Even though the experimental results show that this method is able to against model-pruning attack with a high pruning rate (80%), adversaries can reverse engineer a set of effective, forged "passport" layer weights which are enough to maintain the utility of the model.

Adversarial examples as watermark

Adversarial examples can also be used as watermarks. For example, authors in [43] propose a zero-bit watermarking algorithm to embed zero-bit watermarks into remote models. This method relies on communications with models through the remote API. Specifically, the watermarking process is achieved by making use of inputs of API which can convey a value to embed identification information into the model. The identification information is extracted from the remote model later for ownership verification. The basic principle is that the embedded watermark makes a slight modification on the original model’s decision boundaries around a set of specific inputs that form the hidden key. Predictions of the watermarked remote model to these inputs are compared to those of the original model. A strong match indicates the presence of the watermark in the remote model with a high probability.

Although the paper shows this approach is feasible, it heavily depends on adversarial examples and their transferability property across different models. We cannot generalize that adversarial examples can be transferred to any models in any condition [24].

Embedding strings into outputs of the layers

DeepSigns [60] is an end-to-end framework that embeds watermark (an arbitrary N-bit string) into the probability density function [54] of the abstract training data representations acquired in different layers of the model. The watermark information can only be triggered by passing specific inputs (keys) to the watermarked model.

This method has no visible impacts on the static properties (e.g., weight matrices) of the DNN model. The experimental results show that this method is robust to model overwriting attack which is not evaluated in our thesis.

Embedding in model classification results

This approach which is applied in our work embeds watermarks in the output of the model classification. Similar to the methods used in [1][78] , we also embed watermarks by using the backdoor attack, which causes a model misclassification to a specific target label [3].

7.2 Generating Trigger Sets

Watermarking by backdooring requires an effective trigger set. Several works [1] [78] generate different trigger sets for improving the performance of the embedded watermark.

- **Random noise.** In [78], authors used Gaussian noise as watermarks which is difficult to differentiate from pure noise. As a result, the images with the Gaussian noise is recognised as the target class (incorrect class).
- **Unrelated images.** In both [1] and [78], unrelated images are chosen as trigger set. The trigger set used in [78] is randomly selected from a dataset which does not suit to the original training task. For example, a CIFAR-10 DNN model can use the trigger set from MNIST dataset. Different from [78], paper [1] constructs a trigger set consisting of 100 randomly generated abstract images, and each image is randomly assigned to a target class. This approach also ensures that the samples from the trigger set are unrelated to each other.
- **Training data.** The most common trigger set used currently is to choose a subset of inputs from the training data and randomly label them with incorrect classes [1][3][21][60]. However, if an adversary obtains trigger set samples, he can guess the trigger set. Hence, this approach can not satisfy non-trivial ownership requirement in Chapter 3.4.1.
- **Training data with meaningful content.** Authors in [78] generate a trigger set by embedding meaningful content to selected images from training data. The meaningful content used in this paper is a text 'Test'.
- **Training data with pattern.** [8] and [74] use a pattern of inverted pixels in the bottom right corner of the images as the backdoor trigger, each poisonous image is mislabeled according to certain rules (i.e., each image is assigned to $(m + 1)$ class, where m is the ground-truth label).

Paper [78] proves that using unrelated images as watermarks has the best performance among the first four kinds of trigger sets.

7.3 Backdoor Defences in ML

Our watermarking approaches are based on backdoor attack. Several recent works propose different defense methods against backdoor in ML.

The method in [70] analyses the learned representation of classes (spectral signatures) by utilizing tools from robust statistics. The main idea is that two significant sub-populations are showed if training data is poisoned by backdoor attack. The two sub-populations are a large number of clean, correctly labelled inputs and a fraction of poisoned, mislabelled inputs. These can be separated by the tools of robust statistics and singular value decomposition. This method requires to have access to the poisoned dataset. [72] proposes another technique (NEO) that does not need to access the poisoned dataset. NEO analyses the inputs of the model and determines if the model is backdoored.

Model pruning techniques in [44][47] removes redundant neurons to improve the training efficiency, which also effectively defenses the backdoor attack. However, paper [74] shows that pruning attack can easily cause a significant loss in performance for some models. Fine-tuning is another frequently used method for removing the backdoor in ML models [1][3][78] , but it is also proved that this attack has less impact than model pruning attack. Notice that both fine-tuning and model pruning do not offer detection capabilities to identify backdoored images.

NeuralCleanse proposed in [74] tries to reverse engineer the backdoor trigger by formulating the problem as an optimisation problem. However, the proposed mitigation technique requires computationally expensive retraining.

8 Conclusion

In this thesis, we propose a practical analysis of approaches that can be used to watermark a federated DNN models by backdooring. We apply two watermark removal attacks and showed the unremovability of the embedded watermark. In this Chapter, we draw the conclusion of the work we have done, and present possible future work.

8.1 Summary

In this thesis, we design two federated watermarking approaches (PR-trained, R-trained) and propose a new method to generate watermarks. We perform a comprehensive evaluation with our watermarking approaches and watermark generation method on two benchmark datasets according to the evaluation method in Chapter 6.1, which is based on the requirements we stated in Chapter 3.4. To meet the *S-1 Non-trivial ownership* requirement, uPattern watermark sets generated by using our method are designed to be unrelated to the training data, and each sample of uPattern is uncorrelated to others, such that the adversary is not able to claim ownership even if he knows the watermarking algorithm or a small fraction of watermark set. Moreover, we define a threshold of watermark accuracy based on prior work [69] for the demonstration of non-trivial ownership. We evaluate PR-trained and R-trained approach, and show that they both satisfy the *S-2 Demonstration of ownership* requirement, which can embed a watermark with an accuracy exceeds the threshold. Therefore, the embedded watermark can be used for declaring the ownership. In the evaluation for *S-3 Unremovability* requirement, we compare performance of PR-trained and R-trained approaches with the baseline model (After-trained), and demonstrate our approaches perform better than Afer-trained. The watermark embedded by our approaches are resistant to fine-tuning attack and model pruning attack (when pruning rate $\leq 20\%$). Furthermore, uPattern has the best performance comparing to watermark sets proposed in prior work.

We evaluate the test accuracy between watermarked models and non-watermarked models. The results show that the DNN models watermarked by our approaches achieve a similar test accuracy as non-watermarked models. This means that our approaches successfully meet the *U-1 Model Utility* requirement. Besides, uPattern achieves the second highest test accuracy among all the watermark sets. We also stated that *U-2 Low communication overhead* requires to save at least 50% of communication overhead compare to a normal federated learning. Therefore, we measure the communication overhead costed by applying different local training epochs, and find out that using 5 local epochs in our approaches can effectively save 80% and 67% of communication overhead in MNIST and CIFAR-10 experiments respectively while maintaining the test accuracy and watermark accuracy. Furthermore, experimental results show that uPattern needs the least watermark retraining rounds, which means a lowest computation overhead to satisfy *U-3 Low computational overhead* requirement. The results also illustrate that R-trained approach is the best choice for embedding uPattern in simple models in terms of computational overhead. In

contrast, RR-trained approach with uPattern is more suitable for complex models.

In overall, we show that our federated watermarking approaches are feasible to watermark federated DNN models, which achieves the goal of the thesis stated in Chapter 1.2. Our approaches allow us to protect DNN models for both white-box and black-box settings. We overcome the three challenges stated in Chapter 3.3: *C-1 Decentralized Training*, *C-2 Access to Global model During Training* and *C-3 Communication Overhead*, demonstrate that our approaches can perform as good as other state-of-art watermarking techniques and is robust to two watermark removal attacks. In addition, uPattern watermark sets are the best in overall after comparing with different types of watermarks.

8.2 Future Work

The evaluation for the unremovability of embedded watermark using our approaches in Chapter 6 indicates that more work can be done against model pruning attack. Therefore, focusing on improving maximum tolerated pruning rate of embedded watermark in model pruning attack might be a possible future work. In addition, more watermark removal attacks (e.g., model overwriting) can be applied to evaluate the federated watermarking technique.

In Chapter 3.2.2, we analyze the case that malicious participants can apply any watermark removal attack in the middle of the federated learning. In this thesis, we only implement the experiments to apply attacks after training. Thus, the future work can explore the unremovability of the embedded watermark by applying attacks during training.

References

[1] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th Security Symposium (Security 18)*, pages 1615–1631, 2018.

[2] E. Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[3] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[4] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[5] L. Boney, A. H. Tewfik, and K. N. Hamdy. Digital watermarks for audio signals. In *Proceedings of the third IEEE international conference on multimedia computing and systems*, pages 473–480. IEEE, 1996.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[7] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[8] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[9] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 105–113. ACM, 2019.

[10] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[11] K. J. Cios. Deep neural networks—a brief history. In *Advances in Data Analysis with Computational Intelligence Methods*, pages 183–200. Springer, 2018.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[14] C. Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pages 338–340, 2011.

[15] L. Fan, K. W. Ng, and C. S. Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. *arXiv preprint arXiv:1909.07830*, 2019.

[16] D. Frossard. Vgg in tensorflow. *VGG in TensorFlow· Davi Frossard*, 2017.

[17] github users. A library for encrypted, privacy preserving machine learning. https://github.com/OpenMined/PySyft, 2019. Online; accessed 29 January 2020.

[18] Y. Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[19] W. A. Group. Federated learning white paper v1.0. https://www.fedai.org/static/flwp-en.pdf, 2018. Online; accessed 29 January 2020.

[20] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[21] J. Guo and M. Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[22] J. Guo and M. Potkonjak. Evolutionary trigger set generation for dnn black-box watermarking. *arXiv preprint arXiv:1906.04411*, 2019.

[23] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran. Blocking transferability of adversarial examples in black-box learning systems. *arXiv preprint arXiv:1703.04318*, 2017.

[25] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An in-depth study of lte: effect of network protocol and application behavior on performance. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 363–374. ACM, 2013.

[26] W. T. Illingworth. Beginner's guide to neural networks. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, pages 1138–1144. IEEE, 1989.

[27] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[28] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[29] J. E. Korteling, A.-M. Brouwer, and A. Toet. A neural network framework for cognitive bias. *Frontiers in psychology*, 9, 2018.

[30] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[31] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. http://www.cs.toronto.edu/kriz/cifar.html, 2014.

[32] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173*, 2019.

[33] S. B. Laxmi and M. Vijaya. A weighted mean square error technique to train deep belief networks for imbalanced data. *International Journal of Simulation–Systems, Science Technology*, 19(6), 2018.

[34] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[35] H. Li, E. Willson, H. Zheng, and B. Y. Zhao. Persistent and unforgeable watermarks for deep neural networks. *arXiv preprint arXiv:1910.01226*, 2019.

[36] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019.

[37] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[38] C.-L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern recognition*, 36(10):2271–2285, 2003.

[39] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[40] N. Manzini. Single hidden layer neural network. https://www.nicolamanzini.com/single-hidden-layer-neural-network/, 2017. Online; accessed 29 January 2020.

[41] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[42] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[43] E. L. Merrer, P. Perez, and G. Trédan. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894*, 2017.

[44] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 3, 2016.

[45] G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[46] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(1):3–16, 2018.

[47] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[48] J. Nickolls and W. J. Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[49] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[50] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[53] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[54] S. B. Pope. Pdf methods for turbulent reactive flows. *Progress in energy and combustion science*, 11(2):119–192, 1985.

[55] J. Poushter et al. Smartphone ownership and internet usage continues to climb in emerging economies. *Pew Research Center*, 22:1–44, 2016.

[56] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor. Chained anomaly detection models for federated learning: an intrusion detection case study. *Applied Sciences*, 8(12):2663, 2018.

[57] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[58] A. K. Reyes, J. C. Caicedo, and J. E. Camargo. Fine-tuning deep convolutional networks for plant recognition. *CLEF (Working Notes)*, 1391, 2015.

[59] L. Robert and T. Shanmugapriya. A study on digital watermarking techniques. *International journal of Recent trends in Engineering*, 1(2):223, 2009.

[60] B. D. Rouhani, H. Chen, and F. Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 2018.

[61] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.

[62] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[63] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[64] R. Schapire. Machine learning algorithms for classification. *Princeton University*, 10, 2015.

[65] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13, 2019.

[66] S. Siddiqui. How would we find a better activation function than relu? https://medium.com/shallow-thoughts-about-deep-learning/how-would-we-find-a-better-activation-function-than-relu-4409df217a5c, 2019. Online; accessed 29 January 2020.

[67] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[68] F. C. staff. How amazon, google, microsoft, and ibm sell ai as a service. https://www.fastcompany.com/40474593/how-amazon-google-microsoft-and-ibm-sell-ai-as-a-service, 2017. Online; accessed 29 January 2020.

[69] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan. Dawn: Dynamic adversarial watermarking of neural networks. *arXiv preprint arXiv:1906.00830*, 2019.

[70] B. Tran, J. Li, and A. Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, pages 8000–8010, 2018.

[71] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277. ACM, 2017.

[72] S. Udeshi, S. Peng, G. Woo, L. Loh, L. Rawshan, and S. Chattopadhyay. Model agnostic defence against backdoor attacks in machine learning. *arXiv preprint arXiv:1908.02203*, 2019.

[73] A. University. Triton user guide. https://scicomp.aalto.fi/triton/, 2019. Online; accessed 29 January 2020.

[74] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, page 0, 2019.

[75] H. Wang et al. Facial expression decomposition. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 958–965. IEEE, 2003.

[76] T. Wang and F. Kerschbaum. Attacks on digital watermarks for deep neural networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2622–2626. IEEE, 2019.

[77] W. Yu, K. Yang, Y. Bai, T. Xiao, H. Yao, and Y. Rui. Visualizing and comparing alexnet and vgg using deconvolutional layers. In *Proceedings of the 33 rd International Conference on Machine Learning*, 2016.

[78] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 159–172. ACM, 2018.