TU WIEN Informatics

Black-box Model Watermarking in Federated Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Data Science

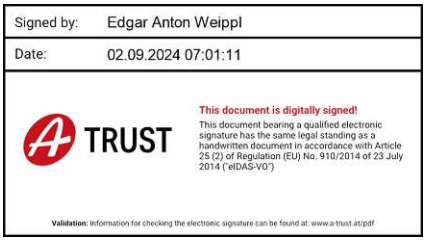
by

Yana Sakhnovych, BSc
Registration Number 11777748

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl. Ing. Dr. techn. Edgar Weippl
Assistance: Univ. Lektor Mag. Dipl. Ing. Rudolf Mayer



Vienna, September 1, 2024

Yana Sakhnovych

Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Yana Sakhnovych, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. September 2024

Yana Sakhnovych

Acknowledgements

I would like to thank my parents for their encouragement and support throughout my studies and all other aspects of life. My sister deserves special thanks for always telling me about cool ML stuff, long before I even considered studying Computer Science.

Christian, my incredible partner, helped me maintain my sanity throughout the writing of this thesis.

I am deeply grateful to Rudolf Mayer for all the discussions, the insightful feedback, and continuous inspiration. I would also like to thank Prof. Weippl for supervising this thesis.

Lastly, my awesome friends have been a source of great support and (mostly) wanted distractions throughout this journey.

Abstract

Federated learning allows multiple parties to collaboratively train a model, without needing individual participants to directly reveal their private data. However, sharing the model at various stages of training poses risks for the model owners, particularly from insider threats such as malicious clients who may steal the model. To counter these threats, embedding a watermark in the model allows owners to prove ownership and protect against unauthorized use.

This thesis aims to evaluate the effectiveness, fidelity, robustness, and efficiency of state-of-the-art federated black-box watermarking approaches. A key focus is on intermediate models, specifically assessing how well these models are protected during the training process, and exploring how they can be exploited in watermark removal attacks. Additionally, this work proposes modifications to existing watermarking methods in federated learning to address the identified vulnerabilities.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

Federated Learning (FL) is a paradigm in machine learning where a model is trained collaboratively on multiple distributed devices (participants), each with their own local dataset. In the parallel federated learning setting, the participants receive a global model from the server and further train it on their local dataset. Afterward, the updated model, or the gradients of each client, are shared with the server, which in turn aggregates these to obtain a new version of the global model. This process can be repeated for multiple communication rounds.

FL allows to collaboratively train a model, without needing individual participants to directly reveal their private or otherwise sensitive data. While this process is advantageous from a privacy perspective, it still means that the participants and the server gain access to the model in various stages throughout training.

In a cross-silo setting, where few organizations, each with large amounts of data, train a model collaboratively, likely all participants have at least partial ownership over the resulting model. Even in this setting, the rights of the individual participants to distribute the model might be limited. In a cross-device setting with possibly thousands of clients that have only little data, for example, when training a model for a mobile application, the clients are likely to have a more restricted claim to the training result.

Thus, measures need to be taken to ensure that the model is protected not only against external attackers, but also against malicious clients wanting to sell or otherwise misuse the obtained model.

Embedding a watermark into the model allows the owner(s) to later prove ownership of a model suspected to be used against the agreed terms, based either on an inspection of the model's internals (in white-box watermarking), or the output of the model (black-box

watermarking). Black-box watermarking is of particular interest, as it can be used in diverse settings, as access to the model via an API is sufficient to prove ownership. In black-box watermarking schemes, usually, the model (e.g., a deep neural network, DNN) is trained on selected inputs, called triggers, to make predictions that are statistically unlikely given a watermark-free model. These triggers can then be used to prove model ownership.

The general requirements for a watermarking scheme include [LMR23]:

- **Effectiveness**
  It should be possible to use the watermark as proof of ownership at any time.
- **Fidelity**
  The performance (effectiveness) of the model on the primary task should not degrade due to the watermarking process.
- **Robustness**
  Modifications of the model, including malicious removal attacks and benign processes (e.g., model aggregation in FL), should not degrade the watermark.
- **Efficiency**
  The computational overhead caused by the watermarking should be limited.

In addition, in federated learning an additional requirement arises: the protection of intermediate models [TXMA21]. The model is shared multiple times at various convergence states throughout the training process. The attacker should not be able to obtain any well-performing iteration of the model that is not or not sufficiently watermarked.

Multiple black-box watermarking methods that are specifically geared towards federated learning have been proposed. These schemes fall into one of two categories:

- **Server Embedding**
  The aggregator is a generally trusted entity and is responsible for the embedding of the watermark. The model is trained on the trigger set before each communication round without access to regular training data [TXMA21, LWL22, SYG+24].
- **Client Embedding**
  One or more clients are embedding the watermark by adding the trigger set to their local dataset. In this setting, no trust needs to be placed in the aggregator [LFG+22, LSY+21, YSY+22].

No comprehensive comparison between these schemes exists, and the individual results from [TXMA21, SYG+24, LFG+22, LWL22, LSY+21, YSY+22] were obtained using different hyperparameters and in some cases different types of trigger, and models [LBK+23],

with only a direct comparison between FedTracker[SYG+24] and Waffle[TXMA21] existing. In addition, the robustness of intermediate models as well as their possible usage in novel attacks have not been explored so far. This thesis aims to close these gaps by i) comparing existing black-box watermarking approaches in FL, and ii) exploring the attack vector posed by the sharing of intermediate models.

1.2 Research Questions

1. **To what extent do state-of-the-art federated black-box watermarking approaches differ in terms of fidelity, effectiveness, robustness, and efficiency?**
   This question aims to assess the differences between existing federated black-box watermarking schemes in combination with different types of trigger images to gain insights into their strengths and weaknesses. Fidelity can be measured by comparing the performance of a watermarked and a non-watermarked model trained under otherwise identical settings. Effectiveness can be determined by the accuracy of the trigger set. Finally, efficiency can be determined by either the number of additional rounds of communication, the increase of epochs of watermarking, or both, depending on the scheme. The final model can be considered robust to selected attacks if it is not possible to decrease the effectiveness of the watermark to a level where the ownership cannot be proven with a high amount of confidence, without compromising the model's performance on the original task at the same time.

2. **How can attackers leverage intermediate models?**
   In federated learning, the intermediate models are an additional resource available to an insider attacker, e.g., a federated learning participant. This question aims to develop ways in which intermediate models could be maliciously used to obtain a watermark-free model. The success of this attack can be measured by (i) the performance that models from which the watermark was removed still achieve on their original task, and (ii) by comparing the effectiveness of the attack to existing attacks on the final model.

   a) To what extent do the schemes succeed in protecting intermediate models?
   The level of protection of intermediate models can vary depending on the watermarking scheme. While in some schemes each intermediate model is fully watermarked, in others, the watermark is embedded incrementally over the course of many communication rounds. Especially in the latter schemes, the question from which point on in the training process a model is watermarked robustly, and which performance the model reaches before that, i.e., which *value* the model already has, arises.

   b) How can intermediate models be used by an attacker to remove the watermark?
   This question aims to investigate how in different watermarking techniques

intermediate models, which are partially or fully trained on the trigger set, can be used by an attacker to remove the watermark from the final model. Thus, novel attacks, tailored to specific schemes, will be developed, and their effectiveness evaluated.

3. **What strategies can be used to enhance the reliability of the watermark embedding and the robustness to removal attacks that utilize intermediate models?**
   This question seeks to both improve the reliability of the watermark embedding and the robustness against removal attacks, focusing on the novel attacks developed for Item 2. The defenses will aim at achieving a balance between the models' performance on the primary task and its accuracy on the trigger set (i.e., the watermark effectiveness) throughout the training process. The goal is to protect high-performing intermediate models while mitigating the risk associated with low-performing models that exhibit high accuracy on the trigger set, as they can be exploited by attackers. The success will be evaluated by comparing the level of protection of intermediate models and measuring the change in the effectiveness of the attacks when the novel defenses are employed.

1.3 Structure

This thesis is organized as follows: Chapter 2 explains the theoretical background of this work, including machine learning basics, federated learning and adversarial machine learning. In Chapter 3 existing schemes for model watermarking in the federated setting are described. Then, in Chapter 4,the threat model is discussed and novel watermark removal attacks that use intermediate models are proposed. Next, Chapter 5 describes the experiment design, for both federated training and watermark removal attacks. Chapter 6 contains the evaluation and analysis of the results of the experiments from the prior chapter. Chapter 7 describes changes to the federated watermarking schemes aimed to increase robustness and/or more reliable embedding of the watermark. In addition, this chapter evaluates the effects of these changes. Finally, in Chapter 8, the contributions of this work are described and the findings summarized.

CHAPTER 2

Background

This chapter introduces the background required for this work, including the basics of image classification using machine learning, federated learning and adversarial machine learning.

2.1 Supervised Machine Learning

In this section, fundamentals of supervised machine learning are discussed. In supervised machine learning, a model is trained to learn the relationship between data samples and corresponding target variables (also called labels), to be able to make predictions on unlabeled data.

2.1.1 Dataset

In supervised machine learning, a model is trained, using a dataset D that consists of multiple samples $X_1 \dots X_m$. Each sample X_i has one or more features $x_{i1} \dots x_{in}$ and one label y_i. This can be represented as:

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & y_1 \\ x_{21} & x_{22} & \dots & x_{2n} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} & y_m \end{bmatrix}$$

To evaluate supervised machine learning models for their performance (effectiveness), datasets are split into multiple parts

- Train(ing) Set
  The train set is, as the name suggest, used for training the model. During training,

a model could overfit to the training set, meaning it could learn to perform well on the training set by memorizing irrelevant details about the data. For this reason, the train set performance is not representative of the model's capabilities. However, it is useful to detect underfitting, meaning that the model does not manage to learn much from the training data, indicating that the model or training process is unsuitable for the task.

- Test(ing) Set
  The test set is used to evaluate the performance of the model on unseen data. As this data is not passed to the model during training, overfitting will negatively affect the performance of the model on the test set.

- Validation Set (optional)
  The validation set is somewhat similar to the test set, as it contains unseen data, i.e., data the model was not trained on. However, it serves a different purpose, e.g., optimizing some aspect of training. As such, it can be used to choose the best training / model configuration during a search for optimal hyperparameters, the best feature set, or similar. Using the test set for this can lead to data leakage, where the model indirectly learns information about the test set, compromising its validity for unbiased evaluation.

K-fold cross-validation is an alternative evaluation configuration. The dataset is split into k folds. Then, each fold serves as the test set once, while the remaining $k-1$ folds are used for training. The evaluation results from the different iterations can then be aggregated, e.g., by computing mean and standard deviations of the achieve performance metric. By using cross-validation, all samples are utilized for both training and evaluation without leaking information into the test set. The performance scores obtained are generally more reliable and robust than a single train-test split. The downside is the computational cost of repeating the training k times.

Variables in a dataset are often categorized into numerical-, categorical-, time series- or text-data. In the datasets used in this work, features x_{ij} consist of numerical data (RGB values of pixels), while the label y_i is a categorical variable.

2.1.2 Data Preprocessing for Image Classification

This section gives a brief summary of commonly applied pre-processing techniques. The datasets used in this work contain images that are being represented in the RGB color space, meaning that there are three color channels with 256 possible values ranging from 0 to 255. The pixels $p_1 \dots p_n \in P$ in an image are therefore represented by three such values.

- Encoding of Target Variable
  Categorical variables, including the target variable in classification tasks, need to be

encoded into a numerical format. The most common method is one-hot encoding, where the target class is represented as a binary vector v of length n, where n is the number of possible classes. For a given class C represented by index i, the vector v is defined as:

$$v_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

- Standardization
  Larger values lead to larger SGD gradients for the respective weights, while smaller values have the opposite effect, which in turn causes undesirable optimization behavior. In order to address this issue, pixel values are often standardized to have a mean of 0 and a standard deviation of 1. The formula is $\frac{P-\mu}{\sigma}$, where μ is the pixel mean and σ the standard deviation. We use standardization in our experiments.

In case of RGB images, these preprocessing steps are performed for each color channel.

2.1.3 Classification

In classification tasks, the target variable y is categorical, meaning that the goal is to divide samples into a number of predefined classes. These tasks can further be categorized by what form the y can take on:

- Binary classification
  Describes a task where the target variable consists of only two possible classes.
- Multi-Class
  Describes a task where the target variable can be any one of a number of distinct possible classes.
- Multi-Label
  Describes a task where the target variable can consist of multiple distinct classes.

When considering two-class (binary) classification problems, one class is referred to as the positive class and the other the negative class. In such a setting, there are four options for the outcome of a single prediction, as can be seen in Table 2.1 and is described below.

- True Positive (TP)
  The sample belongs to the positive class and was correctly classified by the model.
- True Negative (TN)
  The sample belongs to the negative class and was correctly classified by the model.

		Predicted	
		Positive	Negative
Class	Positive	TP	FN
	Negative	FP	TN

Table 2.1: Matrix displaying the possible outcomes of a prediction in the binary classification setting

- False Positive (FP)
  The sample belongs to the negative class, but was incorrectly classified as positive

- False Negative (FN)
  The sample belongs to the positive class, but was incorrectly classified as negative.

Many common classification metrics are derived from the ratios of these four outcomes in the dataset.

- Accuracy $= \frac{\mathrm{TP} + \mathrm{TN}}{n}$
  Accuracy is the ratio of correctly classified samples to the overall number of samples. As such, this metric does not take into consideration the class distribution of the underlying dataset. This means, that if the data is highly unbalanced with most samples belonging to a particular class, a dummy (zero-rule) classifier that always predicts the majority class achieves very high accuracy, in spite of not having learned how to properly differentiate between the classes.

- Recall $= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$
  Recall measures how many of the positive samples are assigned the positive class by the classifier. A dummy classifier that always predicts the positive class reaches a perfect recall score.

- Precision $= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$
  Precision captures how many of the samples, classified as positive, actually belong to the positive class. There is a tradeoff between recall and precision: By varying the threshold for classifying a prediction as positive, one can increase recall at the expense of precision, and vice versa.

- F1 Score $= \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$
  F-Score is the harmonic mean of precision and recall. It is a suitable metric for unbalanced datasets.

For a multi-class problem with K classes, all of these metrics can be applied in different ways:

- Compute Class-Wise Metrics
  Apply the calculation of scores for each class individually by considering that class the positive class and any other class the negative class. Report the results for each class.

- Micro-Averaging
  Count the true and false positives and the false negatives individually for each class, as in the previous point. Build the sums from the class-wise counts and compute the metrics based on these combined counts. For example, micro-averaged precision is computed as follows: $\frac{\sum_{i=1}^{K} \mathrm{TP}_i}{\sum_{i=1}^{K} (\mathrm{TP}_i + \mathrm{FP}_i)}$

- Macro-Averaging
  Computes the metric separately for each class, as in the first point, and aggregate the class-metrics by building their average. For precision, the formula is: $\frac{1}{K} \sum_{i=1}^{K} \mathrm{Precision}_i$

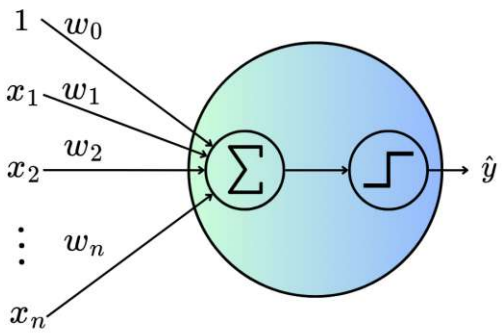


Figure 2.1: Perceptron

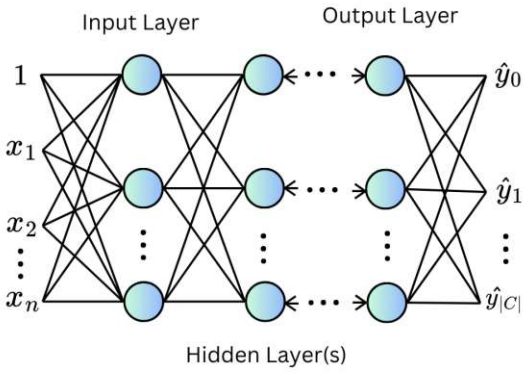


Figure 2.2: Fully connected feed-forward network

2.1.4 Neural Networks

The perceptron is a computational unit, inspired by the workings of neurons in the human brain. The concept was first proposed by McCulloch and Pitts [MP43] and later extended by Rosenblatt [Ros58]. A perceptron receives n numeric inputs, computes their weighted sum, and passes that to a so-called activation function, which produces the output of the perceptron. In addition to sample features $x_1 \dots x_n$, the perceptron also receives the constant 1 as an input x_0, which – multiplied with a weight, as any other input – serves as a bias term. The structure is illustrated in Figure 2.1. The activation function usually used for binary classification with a perceptron is a step function:

$$f(x) = \begin{cases} 1, & \text{if } x \geq \text{threshold} \\ 0, & \text{if } x < \text{threshold} \end{cases} \tag{2.1}$$

The weights of the perceptron are not fixed, but learned trough optimization.

After seeming promising at first, limitations of the perceptron were discovered: the inability to learn non-linearly separable functions, such as the XOR function. Later, multi-layer-perceptrons(MLP) rose to prominence – architectures of consisting of many perceptrons, connected through their in/outputs. They were shown to overcome the issues of single perceptron units. The simplest form of an MLP is a fully connected feed-forward network. The perceptrons (neurons) are arranged in layers, so that all neurons from one layer pass their output to each neuron in the subsequent layer (for the input layer and intermediate layers) or produce the output (for the output layer). The middle layers, that neither receive the original data as input, nor produce the model output, are called the hidden layers. The structure is shown in Figure 2.2.

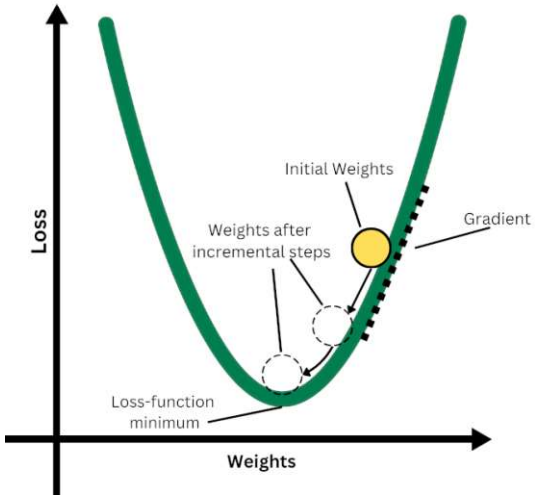


Figure 2.3: Gradient descent

The weights of MLPs are most commonly updated through gradient-decent based methods. The idea behind gradient descent is to iteratively move the weights in the direction that minimizes a so-called loss function, which quantifies the quality of the model output. This is illustrated in Figure 2.3. This direction corresponds to the gradient of the loss function.

A common loss function for multi-class classification problems is cross-entropy:

$$CE(y, \hat{y}) = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i), \tag{2.2}$$

, where y_i represents the actual probability distribution of y_i, and y_i denotes the predicted probability distribution by the model for class i.

The weights θ are then adjusted as follows:

$$v = -\alpha \nabla L(\theta_j)\theta_{j+1} = \theta_j + v \tag{2.3}$$

Here, α stands for the learning rate – a factor that determines how much the model learns per optimization step. A low learning rate can lead to a very slow training, and a too high value can cause an unstable training process and suboptimal weights.

In classical gradient descent, for one update of the weights, the gradients with respect to the whole dataset need to be computed, making this algorithm computationally expensive. In *stochastic gradient descent*, each update is performed based on a single sample. This can lead to oscillations and potentially many updates in a suboptimal direction, due to the noisiness of using a single sample for updates. Mini-batch gradient descent serves as a compromise, as the samples are divided in n batches, with each batch being used in one update. This is generally the most common method for training neural networks. To train adjust the weights of hidden layers, backpropagation is used, starting from the last to the first layer. The gradient is computed as the partial derivative of the loss function with respect to each individual weight in the network via the chain rule, and the weights are then updated as before.

2.1.5 Improvements to Neural Networks

Improvements that were proposed to neural networks, and that are relevant for this thesis, include the following:

- The use of more complex activation function to introduce non-linearity. The most common ones are:
  - Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ Sigmoid is also mostly used in the last layer of a neural network. It maps an input to a number between 0 and 1, which can be used for binary classification.
  - Softmax: $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ Softmax is also most commonly used in the last layer of a neural network, but for multi-class classification problems. It outputs a probability distribution over the possible classes.
  - TanH: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ The TanH function can be used of internal layers of a neural network.
  - Rectified Linear Unit: $\text{ReLU}(x) = \max(0, x)$ It is commonly used in the internal layers of neural network, as it leads to efficient computations, fast convergence and helps prevent the vanishing gradient problem.

  Their shapes are shown in Figure 2.4.
- Batch normalization is widely used to improve training efficiency [IS15]. The advantages of standardizing the input of a neural network were already discussed in Section 2.1.2. The same reasoning also applies to inputs to the hidden layers and the output layer. Batch normalization layers standardize, shift and scale the input batches they receive to increase overall stability in the neural network and to reduce the "internal covariate shift" – meaning changes in magnitudes of inputs

to hidden layers and / or the output layer based on adjusted weights in previous layers. The standardization parameters are derived from the samples in the batch, while parameters for scaling and shifting are learned during training.

- Modifications to the SGD algorithm aim to improve the convergence of the optimization process by different ways [SCZZ19].

  – During training with SGD, the direction of the gradient can oscillate, because no memory of previous gradients is kept, slowing down the convergence speed. **Momentum** [Pol64] addresses this issue by smoothing the update with past gradients. When using momentum, a velocity v is computed before each weight update. Initially, v is initialized with zeros. At iteration i the velocity is computed as:

  $$v_i = m \cdot v_{i-1} - \alpha \nabla L(\theta_j) \tag{2.4}$$

  Here, m is the factor that controls how strong the momentum is. The velocity is then added to the weights:

  $$\theta_i = \theta_{i-1} + v_i \tag{2.5}$$

  – **Nesterov's accelerated gradient** [Nes83] further improves the momentum approach. In Nesterov's accelerated gradient, the gradient is not computed at the current position, but at the future position, after the 'momentum step'. This can lead to a more stable convergence behavior [SMDH13]. The velocity is computed as follows:

  $$v_i = m \cdot v_{i-1} - \alpha \nabla L(\theta_i + m \cdot v_{i-1}) \tag{2.6}$$

  – Other optimization algorithms implement a mechanism for adjusting the learning rate for individual parameters, based on how much they were changed in previous optimization steps. **RMSProp** (Root Mean Square Propagation) [HSS12] does so by keeping a type of memory of previous gradients v_i and dividing the new gradient by the square root of this memory vector.

  $$\begin{aligned} v_i &= \beta \cdot v_{i-1} + (1-\beta) \cdot (\nabla L(\theta_i))^2 \\ \theta_i &= \theta_{i-1} - \frac{\alpha}{\sqrt{v_i + \epsilon}} \cdot \nabla L(\theta_i) \end{aligned} \tag{2.7}$$

  β is the decay parameter controlling the influence of the memory until that point, and ϵ is a small number added to the denominator to prevent division by numbers close to zero.

  – **Adam** [Kin14] combines momentum and an adaptive learning rate and updates the weights in the following manner:

$$
\begin{aligned}
m_i &= \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot \nabla L(\theta_i) \\
v_i &= \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot (\nabla L(\theta_i))^2 \\
\hat{m}_i &= \frac{m_i}{1 - \beta_1^i} \\
\hat{v}_i &= \frac{v_i}{1 - \beta_2^i} \\
\theta_i &= \theta_{i-1} - \frac{\alpha}{\sqrt{\hat{v}_i} + \epsilon} \cdot \hat{m}_i
\end{aligned}
\tag{2.8}
$$

where m_t has the same function as momentum, while the role of v_t is similar to adapting the learning rate based on previous gradients in RMSProp.

- Regularization methods can be used to mitigate overfitting to the training data. Simple regularization methods, like L1-regularization and L2-regularization achieve this by adding a penalty term, based on the magnitude of model weights, to the loss function of a model during training. Weight decay, is a form of regularization commonly used in neural networks. It is closely related to L2-regularization, but it is directly integrated in the weight update, instead of in the loss function:

$$\theta_{i+1} = \theta_i - \eta \left(\nabla_\theta L(\theta_i) + \lambda \theta_i \right) \tag{2.9}$$

The parameter λ controls the amount of regularization.

2.1.6 Convolutional Neural Networks

An important step towards the success of neural networks, specifically in the image domain, was the introduction of convolutional neural networks (CNN). CNNs have one or more convolutional layers, which are essentially feature detectors for inputs with a spatial relation, such as images or also audio. In such a convolutional layer, so-called filters are trained. Filters are $n \times n$ matrices, where n is typically much smaller than the size of the image. A sliding window (of the same size as the filter) is applied to the image and at each position, the dot product between the pixels in the window and the filter is computed as follows:

$$(I * K)(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x + i, y + j) \cdot K(i, j), \tag{2.10}$$

where I represents the input (e.g. a 2D image), K represents the convolutional kernel (filter). $(I * K)(x, y)$ represents the result of convolution at position (x, y) in the output feature map.

Convolutions are useful for pattern recognition. For example, at an early convolutional layer, low-level features, like edges of an image, could be captured, while at a later layer, the features get more complex. Important parameters for defining a convolutional layer are the size of the filter and the stride (the step size of the window in pixels).

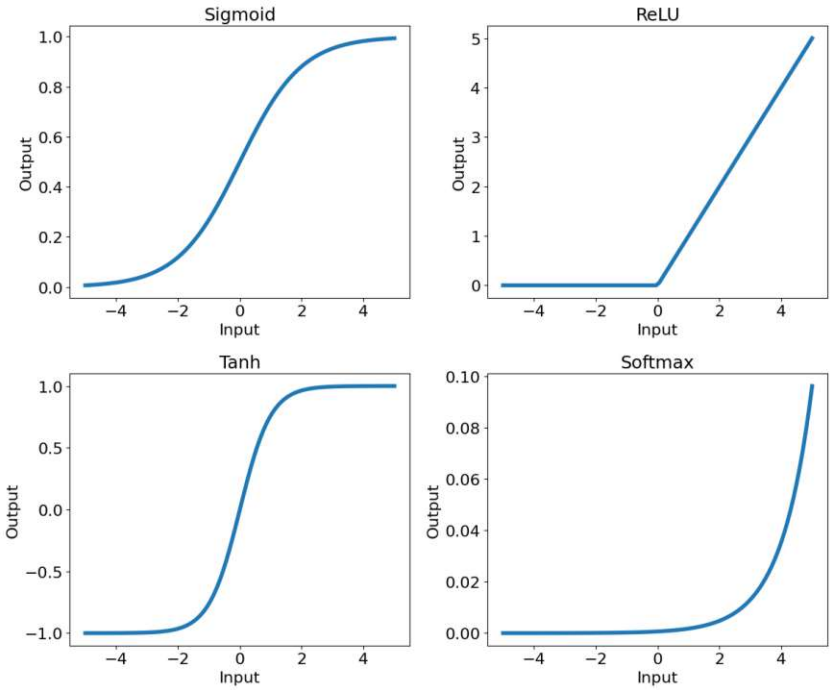


Figure 2.4: Activation functions

In CNNs, max-pooling is frequently performed after convolutional layers. Like in the convolutional layers, there is a window that slides over the input, but instead of a dot-product, the maximum value of the current window is computed. Pooling brings a number of advantages: Firstly, the size of the input is reduced, which makes further computations less expensive. Secondly, the discarding of values that are not a maximum in the respective cell reduces overfitting, and, lastly, since it does not matter where in a window the maximum was, some spatial invariance is introduced. Besides max-pooling, median pooling or other approaches are sometimes used. The final layers of CNNs are usually fully connected layers that map the features extracted by the convolutional layers to the output, e.g., classes in a classification task. An example of an established CNN architecture, VGG16 [SZ14], is described in Section 5.3.1.

2.2 Federated Learning

The term federated learning was first introduced in [MMR+17], referring to the "loose federation of participating devices" that are used to train a model without the need for central storage of the training data. Instead, each participant c computes one or multiple training steps on a local model with their respective private dataset D_c and subsequently sends a model update – usually, in the form of gradients or model weights – to a coordinator (also called a central server or central aggregator). This coordinator then aggregates the received model updates and adjusts the parameters of a global model.

With this, a so-called communication round is finished. The aggregator can initiate a further round by sending the new global model to the clients once again. This process may be repeated multiple times. The advantage of this approach compared to centralized learning is the reduced risk to privacy, as the participants never have to share their dataset.

The most common aggregation method is federated averaging (FedAvg), where the mean model weights, potentially weighted by the local dataset size or the quality of the local model, are computed. The algorithm is described in Algorithm 2.1. Instead of sending the new model weights, clients can also send their aggregated gradient. This version is shown in Algorithm 2.2.

Algorithm 2.1: Federated averaging (FedAvg)

$\theta \leftarrow \theta_0$;
for *each communication round* $r = 1, 2, \ldots$ **do**
  Select a subset of clients C_r for participation;
  for *each client* $c \in C_r$ ***in parallel*** **do**
    $\theta_c \leftarrow \theta$;
    for *each epoch* **do**
      for *each mini-batch* $(x_i, y_i) \in D_c$ **do**
        $\theta_c \leftarrow \theta_c - \eta \nabla L_{\text{local}}(\theta_c, x_i, y_i)$;
      end
    end
  end
  $\theta \leftarrow \sum_{c \in C_r} \frac{|D_c|}{\sum_{c' \in C_r} |D_{c'}|} \cdot \theta_c$;
end
return θ

There are some issues that can arise from this approach:

- **Data Distribution**
  In FL, clients use their own dataset to train the model locally. Depending on how the individual datasets are generated, they could have large differences. For example, let us consider a model that learns to predict illnesses based on medical records from different hospitals. Some of these hospitals could be highly specialized, leading to a disproportionate amount of patients with a specific type of disease. They could also differ in size, affecting the number of records they can use. Thus, the individual datasets are often not independent and identically distributed (IID) and are unbalanced in terms of their size. This can negatively affect the convergence behavior, as was shown in [LSZ+20]. In addition, the question arises of how and whether to handle conflicting properties of different clients' datasets, as improved performance on some clients' data could inflict decreased performance on others.

Algorithm 2.2: Federated averaging with gradients (FedAvg-G)

$\theta \leftarrow \theta_0$;
for *each communication round* $r = 1, 2, \ldots$ **do**
  Select a subset of clients C_r for participation;
  $\hat{g} \leftarrow 0$;
  for *each client* $c \in C_r$ ***in parallel*** **do**
    $\theta_c \leftarrow \theta$;
    $g_c \leftarrow 0$;
    for *each epoch* **do**
      for *each mini-batch* $(x_i, y_i) \in D_c$ **do**
        $g_c \leftarrow g_c + \nabla L_{\text{local}}(\theta_c, x_i, y_i)$;
      end
    end
    $\hat{g} \leftarrow \hat{g} + \frac{|D_c|}{\sum_{c' \in C_r} |D_{c'}|} \cdot g_c$;
  end
  $\theta \leftarrow \theta - \eta \cdot \hat{g}$;
end
return θ

- **Risks to Privacy**
  In spite of privacy-preserving aspects being a major draw of FL, just using FL by itself does not guarantee that the clients and their information are completely protected. The local models/gradients or even the global model can be targeted by attacks that try to reconstruct the training data or gather sensitive information about it[LWW+21].

To protect the data of participating clients, additional measures can be taken. Two frequently used measures are Differential Privacy and Homomorphic Encryption; they are described below.

- **Differential Privacy**
  The idea behind differential privacy is that each person who contributes their data to a dataset keeps the same amount of privacy as they would have, had they not contributed at all [DR+14]. In the federated setting, instead of single records, the local datasets of the clients can be considered for differential privacy calculation. Usually, such guarantees can be met by adding noise to the inputs of local models or to the model weights/gradients clients send to the server.

- **Homomorphic Encryption** In homomorphic encryption, instead of performing a mathematical operation on the plaintext representations of some data, one can perform that operation on the encrypted version; when decrypting the output of

the computation, one obtains the same result as if performed on the plaintext. That is, in a homomorphic encryption scheme, for 2 plaintext numbers x and y and some operation $\circ$:

$$\text{enc}(x \circ y) = (\text{enc}(x) \circ \text{enc}(y)$$

holds. In partially homomorphic schemes, only some operations (generally only addition or multiplication) are homomorphic.

In a federated setting, homomorphic encryption can be used to prevent the server from having access to the client's models, while still allowing for model aggregation on the server. However, it is important to note that homomorphic encryption only protects the intermediate models, information can still be leaked from the aggregated model that is decrypted by some (or all) clients.

Since the introduction of FedAvg, many modifications and alternative aggregation methods have been proposed for FL, that target different aspects like privacy concerns, convergence speed, handling of non-IID data, personalization of the local models and robustness to poisoning attacks.

2.3 Adversarial Machine Learning

The CIA triad comprises three fundamental principles of information security:

- **C**onfidentiality ensures that data is protected from unauthorized access, protecting privacy.
- **I**ntegrity ensures that data is not modified by unauthorized parties, meaning it remains trustworthy.
- **A**vailibility ensures reliable access to the system and data, ensuring the service is not disrupted.

The field of adversarial machine learning explores how a malicious party (an adversary) can compromise the confidentiality, integrity, or availability of machine learning systems – and how to defend against such attacks.

In [BNS+06] attacks on ML models are categorized based on the following criteria

- the CIA principle(s) they target
- the influence of the attack – differentiating between causative attack, where the training process is manipulated through modifications of the training data or the model directly and exploratory attacks that are performed after the training process is finished.

- the specificity of the attack, meaning whether a specific subset of instances is targeted or if the overall behavior of the model is affected.

Furthermore, attacks can also be classified based on the level of knowledge of the attacker. [BR18] differentiate between:

- White-box attacks, that require full knowledge about the training process and the data and assume full unrestricted access to the model.
- Gray-box attacks, where the attacker has partial knowledge. For example, the training algorithm could be known, but the model parameters and training data are not.
- Black-box attacks, that do not require knowledge beyond general knowledge about the task, that any user would have. In black-box attacks, the attacker is often assumed to not have direct access to the model but can query it and receive the model's output, like the labels or confidence scores.

Attacks and defenses relevant to this thesis are discussed below.

2.3.1 Adversarial Examples

Neural networks, but also several other forms of machine learning algorithms, are very sensitive to slight perturbations of inputs. This property can be used to purposefully mislead a neural network to produce the wrong output. For example, images belonging to a specific class can be altered in a way that is either difficult to perceive or not suspicious, but leads to misclassifications. Such altered instances are called adversarial samples [SZS+13]. One prominent example of adversarial samples, in the physical world, is a traffic sign classifier that misclassifies a stop sign for a speed limit sign due to a small modification in the form of a sticker placed on the sign itself [EEF+18]. This attack has also been demonstrated to work on Tesla cars in real-world scenarios [PT20].

An early approach to generating adversarial examples is the Limited-Memory-BFGS-Algorithmus (L-BFGS) attack [SZS+13]. It formalizes the generation of an adversarial example sample as a constrained optimization problem and finds exact solution for convex loss functions and close approximations, otherwise. With this algorithm, the search for optimal values is very computationally expensive. As such, less exact and more efficient methods are commonly used. One such method is the fast gradient sign method (FGSM) [GSS14]. The idea behind FGSM is to compute the gradient of the loss functions, just like during the training of the model, but to move in the opposite direction – the direction of the steepest ascend. The sign of the gradient of the loss function, serves as a linear approximation of the same, allowing to limit the size of the perturbation (in terms of the pixel-wise difference) by scaling it by a small factor ϵ. The perturbation η, can be computed as:

$$\eta = \epsilon \cdot \text{sign}(\nabla L(\theta)) \tag{2.11}$$

2.3.2 Model Stealing

Training ML models requires know-how, computational resources, and a large amount of training data. As such, it is an expensive process – one that a malicious party might try to bypass by stealing a model from its original owner. Research on model-stealing focuses on black-box attacks (where the attacker can only access the model trough queries), since in a white-box setting, stealing a model is as trivial as, e.g., copying its weights. Possible objectives in model stealing are not limited to exact model weights. The architecture, training hyperparameters, and model behavior (without replicating exact weights) can also be the target [OMR23]. In the black-box setting, the attacker can theoretically perform enough queries to precisely reconstruct the weights. For example, equation-solving attacks were proposed to steal the parameters of logistic regression models and of MLPs. They work by constructing and solving a system of equation based on the in- and outputs of the respective model. For this, between 1 and 4 queries are needed per parameter[TZJ+16]. However, for deep neural networks with many parameters, this becomes infeasible. Instead, a common approach in model stealing of deep neural networks is the training of a substitute model based on training data labeled through querying the original model.

2.3.3 Model Inversion

Model inversion attacks aim to recover (recreate) the training data from a model. Fredrikson et al. demonstrated the effectiveness of model inversion attacks on linear models [FLJ+14] and later on decision trees and neural networks [FJR15]. Their attack against neural networks focuses on facial recognition models, where the label represents the individual depicted in the image. The attacker selects a label and, starting from an image initialization, e.g., random noise, employs SGD to create an image that the model associates highly with that label, as outlined in Algorithm 2.3. $\tilde{f}_i$ is a function that takes an in image as input, returns the i-th output of the targeted classifier – a value between 0 and 1. Process is a function that can optionally encompass image post-processing steps to enhance image quality, like denoising.

This approach works best for simple models. For neural networks with hidden layers, the simple optimization approach is not as successful: gradient descent-based methods are likely to get stuck in a local minimum and the optimization over a high-dimensional, continuous data space might lead to unrealistic reconstructed samples that do not look like the actual training images [ZJP+20]. More recent approaches focus on utilizing generative adversarial networks (GANs) to reconstruct training images. Generative Adversarial Networks (GANs) [GPAM+14] are composed of two sub-networks: the generator and the discriminator. The task of the generator is to create realistic data samples, while the discriminator aims to distinguish between real samples from the original dataset and the synthetic data produced by the generator. The generator learns to create realistic samples by trying to fool the discriminator into classifying its output as real.

In [ZJP+20], the GAN is first trained on a public dataset of the same domain (e.g., images

Algorithm 2.3: Model inversion algorithm [FJR15]

Input: learning rate λ, maximum iterations α, window size for early stopping β, threshold for early stopping γ
$c(x) \coloneqq 1 - \tilde{f}_{\text{label}}(x)$
$x_0 \leftarrow 0$
for $i \leftarrow 1$ *to* α **do**
  $x_i \leftarrow \text{Process}(x_{i-1} - \lambda \cdot \nabla c(x_{i-1}))$
  if $c(x_i) \geq \max(c(x_{i-1}), \ldots, c(x_{i-\beta}))$ **then**
    break
  end
  if $c(x_i) \leq \gamma$ **then**
    break
  end
end
return $[\arg\min_{x_i}(c(x_i)), \min_{x_i}(c(x_i))]$

of faces of people not included in the training data) to learn domain-specific features (like general face structure). During optimization, both the discriminator loss and the loss of the original model are used. [WFL+21] also use a GAN, trained on auxiliary data, for model inversion, but they focus on increasing the diversity of the generated images by modifying the training objective.

2.3.4 Backdoor Attacks

Backdoor attacks also aim to maliciously cause misclassifications at will. As opposed to adversarial samples, they do not aim to find specific samples that cause certain model behavior, but to modify model behavior, given certain samples. This means, backdoor attacks are usually causative. The most common type of backdoor attack, is model poisoning. In backdoor attacks based on data poisoning, the model is not only trained in its primary task, but overtly also on the secondary task of recognizing triggers provided by the attacker. For example, an attacker could poison the training data by adding a small pattern to a subset of the training images and either labeling them as a specific class for a targeted attack, or any other class for a non-targeted attack. After training on the poisoned set, the model is supposed to perform well on benign samples, but misclassify images with a superimposed backdoor trigger. However, the backdoor could also be installed by i) adding a poisoned module to the model ii) using transfer-learning with a model that was poisoned during pre-training iii) directly modifying model parameters [LJLX22]. However, for this thesis, data poisoning attacks are most relevant. In deep neural networks, data poisoning attacks can be performed with little impact on the fidelity (the performance of the model on the primary task) because the over-parametrization of the models means that there is spare capacity to learn the additional, malicious task [LDGG18]. The stealthiness of the attack has been

extensively studied. Chen et al. looked at how the effectiveness is affected by reducing the fraction of poisoned samples and proposed blending the trigger pattern with the benign samples to generate poisoned images, where the pattern is difficult to detect [CLL+17]. Designing in-perceivable patterns was also the focus in many works, e.g., [LZS+18, LXZ+20, NT21, LMBL20, DLZL21]. Another focus has been crafting optimized triggers instead of using manually selected, arbitrary shapes or patches of pixel[LJLX22]. This was studied in [DLZL21, LMA+17, LZS+18, LXZ+20]. Many of these approaches are based on adversarial perturbations (adversarial examples). Backdoors can also be utilized in a non-malicious way in the form of model watermarking (see Section 2.3.5). Defenses against backdoors (which can also be used as attacks against model watermarks) are discussed in Section 2.3.6

2.3.5 Model Watermarking

Model watermarking is a technique for ownership verification. To this end, information on its owner is hidden (embedded) within the model. If a malicious party obtains the model and tries to claim ownership of it, this information can be extracted and used to disprove the claim. As such, model watermarking is a reactive defense against model stealing, meaning it does not prevent the attack but allows for legal recourse [LMR23].

There are white-box and black-box watermarking schemes. In the context of watermarking, these terms refer to the level of access needed to prove ownership, rather than the attackers' knowledge, as in the prior definition. In white-box watermarking, the watermark is hidden in the model parameters or neuron activations. In contrast, black-box watermarking works most commonly through intentional model backdoors. Thus, the watermark is based on regular model output given specific input samples.

White-box watermarks can be embedded by modifying model parameters post-training, or during training by adjusting the loss term to include the additional objective. Black-box watermarking schemes work by installing a backdoor into the model and demonstrating that the behavior (output) of the model, given trigger images, is known only to the model owner (given that the output would be highly unlikely for a watermark-free model). Different types of triggers and trigger-generation methods have been proposed. In [ABC+18], abstract images that are unrelated to the classification task and which are assigned with random labels are used. Zhang et al. compare the effect of using in-domain images with added patterns or noise to out-of-domain images [ZGJ+18]. Merrer et al. construct adversarial examples and use those as trigger images, to shift the decision boundary of the model [LMPT20]. Guo et al. argue that being able to add a small trigger based on the owners' signature to any image allows a clear association between the author's identity and the watermark and makes it more difficult for an attacker to create a forged proof of ownership by generating trigger images the model was not actually trained on. They propose using clean and watermarked versions of training images to make the triggers background-agnostic [GP18]. [DRCK19] proposes a black-box scheme that randomly generates potential trigger images but uses only those where the neuron

activations in the last layer are dissimilar to the training data. This shall increase the integrity of the watermark.

2.3.6 Backdoor Defenses

Backdoor removal methods can be used as a defense against backdoor attacks or as an attack against backdoor-based watermarks.

Li et al. identified three paradigms for backdoor removal [LJLX22]:

- Input Preprocessing
  Approaches based on input preprocessing do not actually eliminate the backdoor but aim to render backdoor triggers ineffective by applying image pre-processing to all input images passed through API queries. The modifications change the images enough so that the modified images no longer trigger the backdoor, while backdoor-free inputs remain unaffected.
- Backdoor Prevention and Backdoor Removal
  Approaches focused on backdoor elimination try to either prevent the backdoor from being embedded, or to cleanse a poisoned model.
- Trigger Detection
  Similarly to the input-preprocessing, the idea behind this paradigm is to prevent the backdoor from triggering. This is achieved by identifying suspicious inputs and refusing the respective queries.

In the survey, Liu et al. referred to these paradigms as trigger-backdoor mismatch, backdoor elimination and trigger elimination, respectively.

For this thesis, methods that remove an already embedded backdoor – meaning a subset of attacks belonging to the second paradigm – are of particular interest. They include the following approaches:

- Finetuning
  Fine-tuning is a technique in machine learning where a pre-trained model is further trained on a specific dataset to adapt its learned parameters to a new task or domain. In the context of backdoor removal, it refers to further training a backdoored model on clean data, with the aim to remove the backdoor [ABC+18]. Before fine-tuning, optionally, some layers of the model can be frozen, and/or the final layer can be re-initialized to increase the effectiveness of the removal. Fine-tuning by itself tends to not be particularly effective but is commonly combined with other attacks to keep the primary task performance of the model high while removing the backdoor.
- Pruning
  Pruning refers to a method used to reduce the model size by identifying and

eliminating redundant or less important weights in the neural network, while aiming to maintain its performance. Since backdoors exploit the over-parameterization of DNNs, this can be used for backdoor removal [LDGG18]. Different pruning-based backdoor removal methods vary in how they select the weights to be removed. In [HPTD15], the L1-norm and L2-norm are compared as criteria for identifying unimportant neurons, which are then removed to reduce the number of model parameters. The L1-norm approach was used in watermark removal attacks in [UNSS17], [SYG+24], and [TXMA21]. Another approach is, to prune based on neuron activation, instead of the magnitude of the model weights. [LDGG18] removes neurons that have low activations during training, while in [WYS+19], neurons are ranked based on differences in activations between normal training data and reverse-engineered trigger images, generated through model inversion, which is further detailed in the next bullet point. In global pruning, the parameters across the entire network are ranked, while in local pruning, the parameters are ranked on a layer-by-layer basis.

- Trigger Inversion
  Trigger-inversion based approaches aim to reconstruct the triggers using model inversion techniques, and then uses the results to remove the backdoor in a targeted way [WYS+19, GWX+19, CFZK19, HLR+21, LLZ+23]. For example, in Neural Cleanse [WYS+19] backdoors are removed in one of two ways: either through "unlearning" – where the reconstructed triggers are assigned new labels and the model is trained on this set to forget the old backdoor – or through pruning of neurons with low activations on normal training data and high activations on the reconstructed triggers, as mentioned previously.

CHAPTER 3

Watermarking in Federated Learning

This chapter describes existing schemes for model watermarking in the federated setting. It begins with approaches where the watermark is embedded by the server, followed by those in which one or multiple clients act as the watermarking parties.

3.1 Server-Side Watermarking Schemes

In this section, server-side watermarking schemes are discussed. In these schemes, the server is assumed to have access to the unencrypted model weights to be able to perform the watermarking.

3.1.1 Waffle

Waffle [TXMA21] is one of the first proposed watermarking schemes in the federated setting. In this scheme, the assumption is that the aggregator is a trusted party that is responsible for the watermarking. After initialization, the aggregator pre-trains the global model on the trigger set until it overfits. Then, the FL process proceeds as usual, but after each round, the aggregation further trains on the trigger set if the watermark accuracy has declined until a maximum number of epochs or a watermark accuracy threshold is reached. The watermark accuracy threshold chosen by the authors is quite high (98%), as their stated goal is for every intermediate model throughout training to be watermarked. The authors also address the choice of trigger image. In the targeted scenario, the aggregator has no access to training images, so out-of-distribution images unrelated to the primary predictive task are required. The authors argue that the trigger set should be easy to learn so that little retraining needs to be performed and the efficiency of the training process is not negatively impacted. They propose to use

their "WafflePattern": each target class is assigned one unique pattern that is added to randomly generated images. The hyperparameters that need to be set in this scheme are optimization parameters (learning rate, weight decay, momentum) both for the initial training on the trigger set and for the retraining in between FL rounds. In addition, the two thresholds that are used as stopping conditions during retraining need to be set (watermark accuracy threshold and maximum number of retraining epochs per round).

3.1.2 FedTracker

FedTracker [SYG+24] introduces two improvements to Waffle. Firstly, it addresses the issue of catastrophic forgetting, which is a problem in neural networks that are first trained on one task and subsequently on a different one. The performance on the first task quickly declines, as weights that are important for the initial task might be irrelevant for the other [KPR+17]. Since Waffle includes two very distinct, alternating tasks – watermarking on the server and normal training on the clients – catastrophic forgetting might affect the performance. FedTracker proposes an approach inspired by continual learning [LPR17], which restricts model updates from watermarking that would counteract the training on the primary task. To achieve this, a "global memory" m is stored on the server, which consists of the sum of aggregated gradients from all previous FL rounds. This global memory represents the descent direction of the primary task loss. One can then measure whether there is a conflict between the optimization for the primary task and the watermarking by considering the angle between the global memory m and the watermarking gradient, g. Thus, the authors formulate the following optimization problem:

$$\begin{aligned} \min_{x} \quad & \|g - x\|_2 \\ \text{s.t.} \quad & \langle (g - x), m \rangle \geq 0 \end{aligned} \tag{3.1}$$

This means that the watermarking gradient is projected to a point at which the gradient and the memory are, at worst, orthogonal. To find that point, quadratic optimization is used. After the optimization, the model is updated as follows at the end of round i:

$$m_i = m_{i-1} + g_i - x_i \tag{3.2}$$

Furthermore, a way to handle models with batch normalization layers [IS15] (BN) (cf. Section 2.1.5 is introduced. In batch normalization, the distribution of the layer inputs is learned. If the trigger set and the training data come from two different distributions, the BN layers would not function as intended. In FedTracker, those layers are frozen during watermarking.

Besides these two changes, Fedtracker also embeds client fingerprints into batch normalization layers to enable traitor tracing. This means that, given white-box access to a stolen model, one can identify the client that leaked it. The hyperparameters for this scheme are the same as in Waffle.

3.1.3 Merke-Sign

The authors of Merkle-Sign [LWL22] propose a general scheme for model watermarking in FL, which works with white-box and black-box watermarks. It tackles a slightly different setting than the other server-side schemes, as an additional goal is to enable all clients to prove ownership of the final model. However, the client watermarks are embedded only after the FL process is finished, so the training itself remains similar. Before each round of FL, the server embeds the watermark in a copy of the model. Furthermore, before sending the watermarked copy to a client, an additional watermark that is specific to that client is embedded. This watermark can later be used to identify which client leaked the model, if the need arises. After the round, the aggregated gradients of the clients are used to update the clean version of the model.

3.2 Client-Side Watermarking Schemes

In client-side watermarking schemes, the watermark is embedded by clients during local training.

3.2.1 Liu et al. and Yang et al.

Liu et al. [LSY+21] argue that in many settings and FL architectures, the central node is not the model owner and is not necessarily trusted. Thus, it might be desirable to deny the aggregator access to the model weighs by using, e.g., additive-homomorphic encryption. However, this means that the server cannot embed the watermark into the model. In the scheme they propose, one or more clients add the trigger set to their own data and embed it as part of their local update. As the watermarking clients (called initiators) might not be selected at each FL round (cf. Algorithm 2.1), to boost their contribution, their aggregated gradients are multiplied by a factor λ. The recommended value for λ is $\frac{N}{n}$, where N is the total number of clients and n is the number of clients selected per round. The model update of the initiator is computed as follows:

$$U = \lambda(L_i - G_{i-1}) \tag{3.3}$$

L_i stands for the local model and G_i for the global model at round i. Randomly generated images of Gaussian noise are used as the trigger set.

Yang et al. [YSY+22] employ a similar general scheme but propose a different trigger generation method. The motivation behind the change, is that multi-class trigger sets based on random noise are difficult to learn, potentially leading to overfitting. To overcome this issue, they use the position of noise as a more easily learned property. They divide the images into a grid and assign one cell to each class. For a trigger image of a particular class, random noise is superimposed on a training image at the position within the grid that corresponds to the cell associated with that class.

3.2.2 FedIPR

FedIPR [LFG+22] allows clients to embed black-box- and/or white-box watermarks into the model. As opposed to the approach by Liu et al.[LSY+21], the clients have their own individual trigger sets instead of a single global one. FedIPR does not include boosting updates of watermarking clients. As the trigger set for the black-box watermarks, adversarial samples are used. The authors found that participation on 25% of rounds is sufficient to embed the watermark, though it was not addressed how the time of participation affects the embedding or what the watermark accuracy of the black-box watermark is throughout training. Furthermore, in FedIPR, the white-box watermarks are used to detect so-called free-riders, i.e., clients that do not actually contribute to the training, but e.g., construct dummy-models to send to the aggregator. The server verifies which white-box watermarks are present in the global model. Watermarks from clients that did not perform any training are absent, while those from honest clients appear within 30 rounds.

CHAPTER 4

Watermark Removal Attacks in Federated Learning

This chapter introduces novel watermark removal attacks specifically designed for federated watermarking schemes. These approaches leverage intermediate models shared with clients throughout the training process to gain information on the watermark, that is otherwise unavailable.

4.1 Threat Modeling

In this section, we discuss the goals of the model owner, along with threat modeling aspects outlined in [BR18], namely the attacker's goals, knowledge, capabilities.

4.1.1 Model Owner

The model owner wants to train a model using FL. They do not trust all participating clients, so they want to protect their IP from them by embedding a watermark.

Their goals can be summarized as:

- High Fidelity
  The accuracy of the model should be as high as possible and thus not be negatively affected by the watermark embedding process.
- Effective and Robust Watermark
  The watermark should reliably be used to prove ownership of a model. This means it should be reliably embedded, and robust to possible attacks.
- Little Overhead
  The computational and communication overhead should be within reasonable limits.

4.1.2 Attacker

The motivation of the attacker is to obtain a model without the computational, organizational, or financial burden of training it himself, and without having to collect training data or convince clients to participate. In addition, the attacker wants to use the model freely, including the option of selling it or offering API access. Because of this, the attacker wants a watermark-free model to avoid being exposed. Their goals can be summarized as:

- Obtaining High-Performance model
  The attacker aims to obtain a model with high effectiveness on the original task, and claim it as their own. Ideally, the model should have the same accuracy (or another relevant metric value) as the final state of the original model, though a small decline in accuracy is acceptable in several settings.
- Preventing Proof of Ownership
  The watermark must be removed, or at least weakened to become insufficiently clear to prove the IP theft.
- Cost Tradeoff
  The cost of removing of the watermark should not outweigh the overall cost from gathering and labeling data and training the model from scratch.

We can differentiate between an insider attacker, who has access to the intermediate results of the training, and an outsider attacker, who only has access to the final model. We assume that an insider attacker has access to intermediate models of any round they participate in and has access to a local dataset of the same size and data distribution as the other clients. The attack vector of the insider attacker manipulating the training process, e.g., by crafting malicious model updates that negatively affect the watermarking, instead of following the protocol and training on their local dataset, are not considered in this thesis. In addition, the attacker has no access to the true trigger set. If they had, the watermark removal process would be as simple as training on that set with flipped labels. Following Kerckhoffs' principle [Pet11], security mechanisms should be robust even if the general process (here: the watermarking process) is known. However, certain aspects are kept secret. For example, in the crypto system RSA, the encryption method is publicly known, but private keys needed to decrypt messages need to remain secret [RSA78]. We assume that while the attacker may know which watermarking scheme is being used, they do not have access to the specific trigger image generation method or the trigger images themselves. To achieve their goal, both insider and outsider attackers can perform common backdoor removal methods on the final model. We discussed such methods, like pruning and model inversion with unlearning, in Section 2.3.6. Both of these methods, and also other methods from literature, can be performed both by insider and outsider attackers, because they only make use of the final model state. To our knowledge, there are no methods that address the insider attacker scenario and utilize intermediate model states. We propose multiple such attacks in the next section.

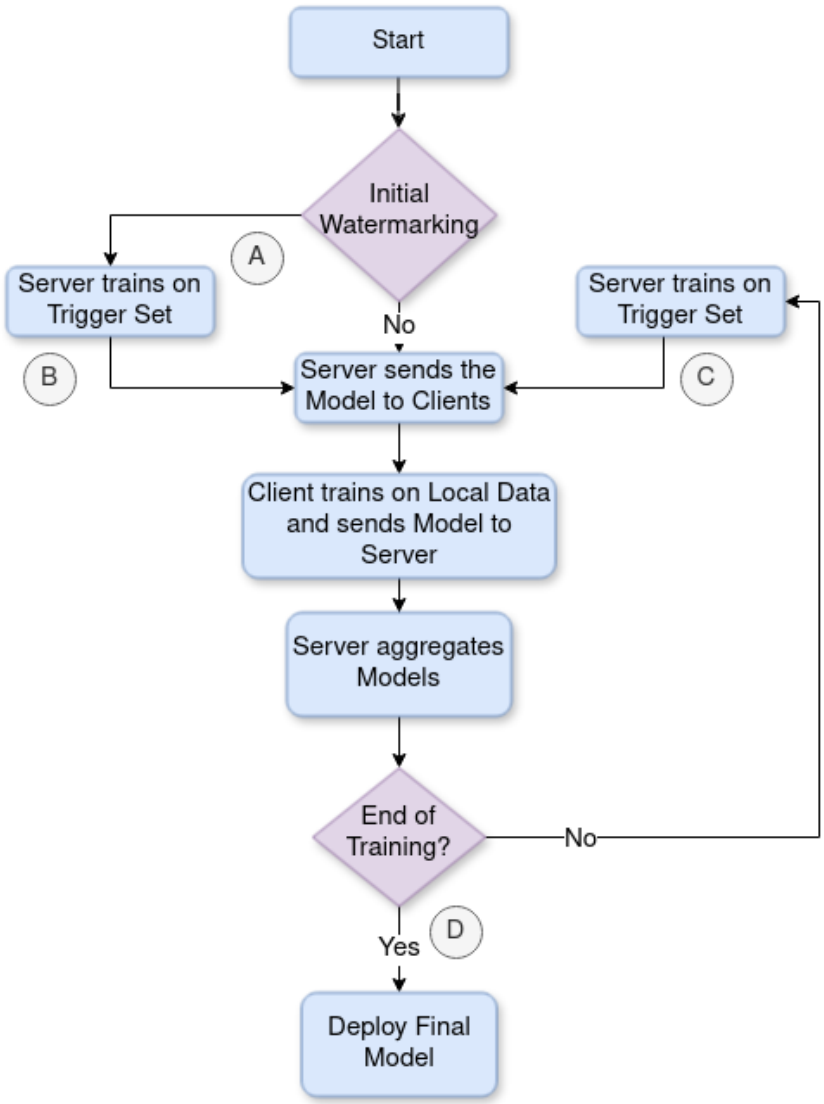


Figure 4.1: Server-side watermarking

4.2 Insider Attacks

In this section, we propose novel watermark removal attacks that make use of intermediate model states shared with clients throughout the FL process.

4.2.1 WM Diff

The idea behind this attack is to utilize the changes in weights that come from the initial watermark embedding in server-side schemes. As in these schemes a large part of the watermark embedding happens at once, the changes in weights caused by watermarking the model are not mixed together with changes from training the model for the regular task.

A malicious client that has access to the weights of a model before and immediately after watermarking could use this knowledge to compute the weight difference between the watermarked and the clean model. The watermark removal is achieved by subtracting the scaled model difference from the target model's parameters:

$$\theta'_n = \theta_n - \alpha(\theta_w - \theta_{w-1}) \tag{4.1}$$

, where θ_n are the weights of the target model, θ_{w-1} the weights immediately before the watermarking and θ_w the weights immediately after. α is a scaling factor. A high scaling factor α leads to a more effective attack, but also to a larger impact on the accuracy of the model on the primary task.

Figure 4.1 shows a flow chart of the watermarking process using server-side watermarking as proposed in [TXMA21]. There, the model is fully watermarked before the first round of FL, making it the ideal scenario for the attack: WM Diff can use the model weights prior to the initial watermarking (stage A in the figure), the weights after the watermarking but prior to training on the primary task (stage B) and the final model (stage C). In this setting, θ_{w-1} represents the weights at stage A, θ_w represents the weights at stage B.

The attacker could get access to the pre-watermarking weights in different ways:

- **Transfer Learning**
  One situation in which the initial weights might be easy to acquire is if transfer learning is used to train the model. Transfer learning is a method in which a model that was trained on a task a is then reused for a different task b. This is achieved by further training the model on new data specific to the task b. This approach increases efficiency, because the model for the task b does not need to be trained from scratch, and it allows using large ML models for tasks in which data is scarce, as the amount of data required to fine-tune the model is less than the amount if the model needs to be trained from scratch. Pre-trained weights for a wide range of different models are publicly available and frequently used. A client can easily

compare the model shared by the server with pre-trained weights that the client has access to, and determine if they match.

- **Delayed Watermarking**
  If the watermarking is not performed before the first round (between stages A and B), but at a later round in the FL process (stage C), a client participating in the round before and the round after would also be able to perform the attack. In this case, for the success of this attack, the malicious client would need to determine when the watermarking has taken place. For this, the client can track irregularities in the model performance on their data, as the watermarking is likely to cause a noticeable drop in performance immediately after the watermarking. The change between these two consecutive model states is caused by the watermarking but also by one regular round of FL. However, since it is only a single round, the effect of that is likely to be negligible.

This attack assumes that weights that are important for recognizing the trigger images during the initial watermarking process will i) still be important for that purpose during later iterations, and ii) will not become highly relevant for the primary task. Whether the first assumption is true will likely depend on how much retraining happens throughout the FL process.

4.2.2 Model Inversion

Model inversion was already described in Section 2.3.3. Like in the attack on the training data discussed there, the idea behind this attack is to reconstruct the trigger set (or something similar to it), and to then retrain the model on this set so that it outputs labels not associated with the watermark. This process is referred to as "unlearning" [WYS+19].

In the federated setting, an additional resource might be available to an attacker: intermediate models, fully trained on the trigger set but with different levels of performance on the primary task, spanning from no training on the primary task to fully converged. While it directly applies to WAFFLE, FedTracker, and Merkle-Sign, it might also be effective in client-side schemes that embed the watermark over time – if the watermark performance converges early during the training while the primary task performance is still low. There are different ways in which an attacker could try to utilize these models to remove the watermark. A common problem in model inversion for backdoor removal is not being able to constrain the solution [LLZ+23]. The resulting image could be in some way similar to the training data, to the trigger set, a mix of both, or a random artifact that has no meaning.

- **Single-State Inversion** Performing inversion on an early model state (from stage C), rather than the final one (stage D), while still using the final model for unlearning, might enable an attacker to avoid inverting images that contain

similarities to the training data. This approach could potentially reduce the negative impact of unlearning on the accuracy of the model. However, the inverted images might not be suitable for the attacker's purposes, because the early and final model states can be significantly dissimilar. Models from different rounds might focus on different aspects when classifying trigger images, and thus the inverted images from one model might not be useful for unlearning on the other.

- **Output-Based Multi-State Inversion** An alternative strategy is to increase the variety of the inverted images by performing the model inversion multiple times on different states of the model. This might lead to a higher variety than using the same model state for inversion multiple times and starting with a different initialization each time. This approach could increase the likelihood that relevant aspects of the trigger images will be included in the inverted images, but has the same drawbacks as the inversion of individual model state.
- **Loss-Based Multi-State Inversion:** Another approach involves performing the inversion on multiple models simultaneously, by using the sum of their losses. Later models will likely have a minimum in the loss landscape where the image is similar to the training data and another where the image is likely to be similar to the watermark, while early models are not expected to have a minimum related to the primary task. Adding the losses of an early and a late model might increase the likelihood of finding the common minimum related to the watermark.

CHAPTER 5

Experiment Design

This chapter describes the design and execution of the experiments in this thesis. First, the datasets and models are described, followed by discussing the general setting for the federated learning process and the different model watermarking schemes. Finally, the implementation and the hyperparameters of the attacks are described.

Table 5.1: Image classification datasets used in works on backdoor watermarking in federated learning

Paper	CIFAR-10	CIFAR-100	MNIST	Fashion-MNIST
Waffle [TXMA21]	✓		✓	
FedTracker [SYG+24]	✓	✓	✓	
Merkle-Sign [LWL22]	✓	✓	✓	✓
Liu et al. [LSY+21]	✓		✓	
Yang et al. [YSY+22]	✓		✓	
FedIPR [LFG+22]	✓	✓		

5.1 Datasets

For the selection of datasets, we focus on those commonly used in other works on backdoor watermarking in FL, to allow comparing our results. Table 5.1 shows the datasets employed in the most relevant related works. Notably, CIFAR-10 emerges as the most frequently utilized dataset, featured in all examined works, while MNIST is used in all but one.

5.1.1 MNIST

MNIST [LCB10] is a benchmark dataset consisting of labeled images of handwritten digits. Some examples are shown in Figure 5.1. With a training set of 60,000 samples

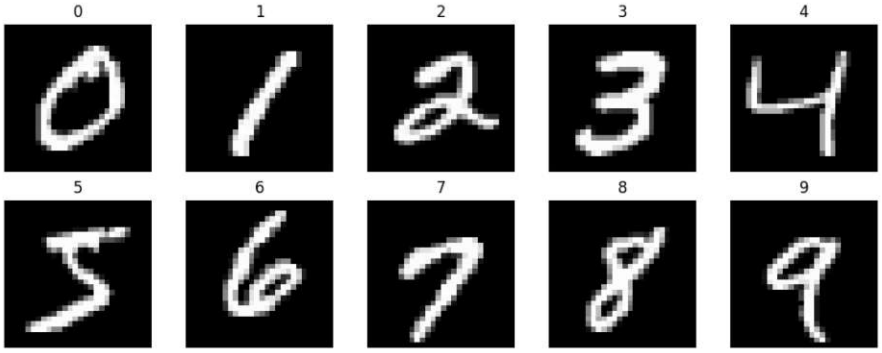


Figure 5.1: Example images of each class in MNIST

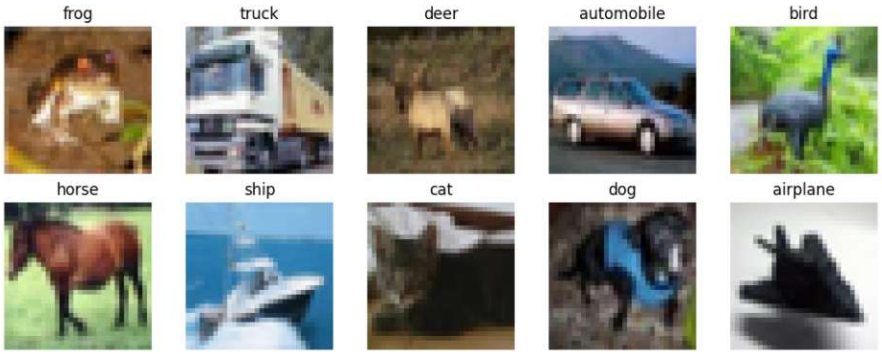


Figure 5.2: Example images of each class in CIFAR-10

and a test set of 10,000 samples, MNIST is a small dataset that allows fast training and has become the most frequently used image classification dataset for this reason [XRV17]. It has a balanced class distribution and contains grayscale images of size 28×28 pixel. Digit recognition on MNIST is not a challenging task anymore, as accuracies of $> 99.7\%$ have been reported [WZZ+13, KHB+18] on this dataset.

5.1.2 CIFAR-10

CIFAR-10[KH+09] consists of labeled images of different animals and objects with 10 classes; examples from each class are shown in Figure 5.2. It has a training set with 50,000 samples and a test set consisting of 10,000 samples, with each class containing an equal number of images in both sets. The size of the images is 32×32 pixels and all of them are RGB images. CIFAR-10 is a popular evaluation dataset. It is more challenging than MNIST, as the images are more diverse, while still having a small size that allows fast model training.

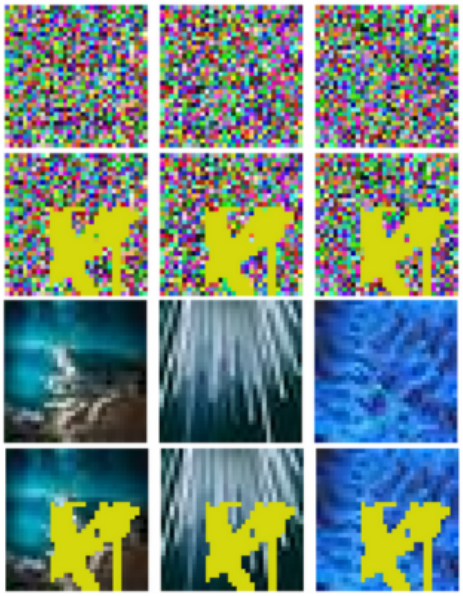


Figure 5.3: Trigger sets - from top to bottom: Random, RandomPattern, Abstract, AbstractPattern

5.2 Trigger Sets

Since we consider a setting in which the watermarking entity does not have access to any training data, the trigger sets used in our experiments consist of different types of out-of-distribution images, i.e. they are not related to the primary predictive task. In [TXMA21], the effect of different trigger images on the federated training process is studied, with a focus on how easy the trigger set is to learn. We want to perform a similar comparison but also look into whether an easy-to-learn watermark might also be easier to reverse engineer through model inversion. For this, we choose a similar selection of trigger set types.

- **Random (R)**: This type of trigger set is also called "unstruct". The images consist of random pixels drawn from a normal distribution. This type of trigger set is based on [DRCK19] and was used in [TXMA21, LSY+21, LFG+22].
- **RandomPattern (RP)**: This trigger type was introduced as "WafflePattern" in [TXMA21]. For each dataset, 10 "patterns", which consist of one or multiple contiguous shapes with a size of 100 pixels each, are generated following the code from the authors [1]. These patterns are superimposed on images with random pixel values sampled from a normal distribution. The position of the pattern is randomly chosen, but is consistent across all trigger images with that pattern. For

[1] Code available at `https://github.com/ssg-research/WAFFLE/blob/main/src/experiment.py`

Table 5.2: Model architectures for image classification used in works on backdoor watermarking in FL. Red means the model was trained on CIFAR-10, green means it was trained on MNIST, and black a different dataset.

Paper	VGG	LeNet	AlexNet	ResNet	Other
Waffle [TXMA21]	✓				CNN5
FedTracker [SYG+24]	✓	✓	✓	✓	
Merkle-Sign [LWL22]				✓✓	
Liu et al. [LSY+21]	✓✓	✓✓			
Yang et al. [YSY+22]	✓✓	✓✓			
FedIPR [LFG+22]			✓	✓	

color images, the same holds for the color of the pattern; for grayscale images, all patterns are white. Each pattern corresponds to one specific label.

- **Abstract (A)**: Images from the dataset of abstract images introduced in [ABC+18] are used as trigger images for this set. This dataset was initially proposed with the reasoning that the individual images should not be related so that when part of the trigger set is leaked, the rest is not negatively affected. These images are available on GitHub [2]. They are downsized to match the dimensions of the images in the respective classification dataset using a Lanczos resizing filter[3].
- **AbstractPattern (AP)**
  This trigger set is constructed similarly to RandomPattern, but instead of random pixels for the background (i.e., the area not covered by the pattern), an image from the abstract dataset is used.

Examples of the trigger images of one class are shown in Figure 5.3.

Each trigger set consists of 100 images, 10 per target label. The differences between our selection and the one from Waffle are that (i) instead of abstract images, the authors randomly sampled images from a subset of classes of ImageNet and (ii) for the set of non-random images with added patterns, they used a subset of the training data (with modified labels). In the case of (i), the change allows us to study the effect of the pattern, regardless of background, and to avoid using trigger images that one of the models we use was pre-trained on. For (ii), the reason for this change was the possibility of a data-less model owner, for whom a trigger set based on training images is of little use.

5.3 Models

Table 5.2 lists the DNN models that were used in the most relevant related works. The model listed as 'other' is not a well-known architecture like, e.g, VGG16, but a small

[2] https://github.com/adiyoss/WatermarkNN/tree/master/data/trigger_set/pics
[3] https://pillow.readthedocs.io/en/stable/handbook/concepts.html#PIL.Image.Resampling.LANCZOS

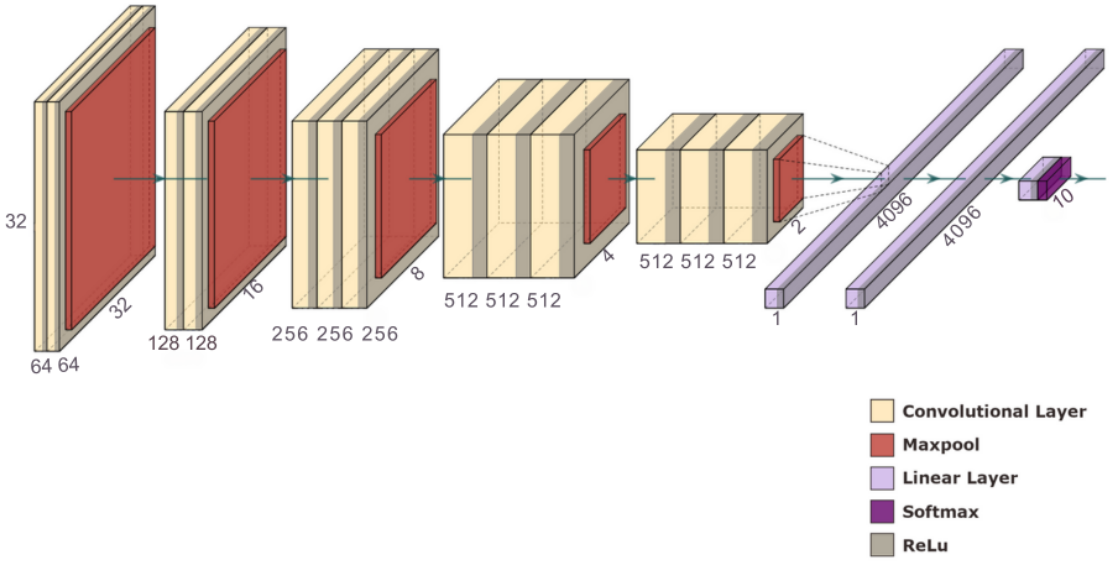


Figure 5.4: VGG16 architecture

convolutional neural network with 5 layers introduced in [JSMA19]. We will refer to it as CNN5. For training on CIFAR-10, we use the VGG16 architecture [SZ14], which was also used in [TXMA21, SYG+24, LSY+21, YSY+22]. For MNIST, we also selected the same model as in [TXMA21] (CNN5). Both models have in common that they do not have batch normalization layers – something the authors empathized due to known performance issues when used in federated settings with non-IID data [HPMG20].

5.3.1 VGG16

VGG16 is a deep convolutional neural network with 16 layers and a total of 138,357,544 trainable parameters. In 2014, VGG achieved the best performance in terms of single-object localization and second-best for classification in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [RDS+15]. The idea behind VGG is to focus on the depth of the neural network, while keeping the number of parameters reasonably small by using a small size for the convolutional layers. VGG16 consists of 13 convolutional layers with 3x3 convolutional filters and 3 fully connected layers. The architecture is shown in Figure 5.4. We used a model pre-trained on the ImageNet dataset, which is available e.g. via the Torchvision library [4]. As ImageNet has 1,000 classes and not 10 (as

[4] https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html

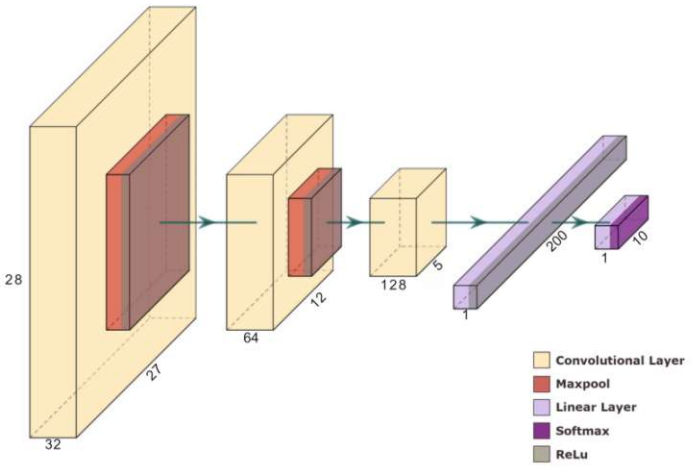


Figure 5.5: CNN5 architecture

in MNIST and CIFAR-10), we replace the final linear layer with a new linear layer with an output dimension of 10.

5.3.2 CNN5

The model we use for MNIST is a small convolutional neural network with 5 layers from [JSMA19]. It consists of 3 convolutional layers, each followed by max-pooling and ReLU-activation functions, and finally 2 linear layers with the first also being followed by ReLU. Figure 5.5 illustrates the architecture. The model has 488,571 trainable parameters.

5.4 Training

During training, 20 clients participate at each round and train the model on their local datasets. As in the Waffle paper, each client uses SGD for one epoch, with a batch size of 50. The learning rates are also consistent with the Waffle paper, set at 0.1 for the CNN5 model and 0.01 for VGG16. The only exception is, that there, a learning rate of 0.001 was used for experiments on MNIST with the trigger set Random, while we use the same learning rate for training, regardless of trigger set.

We consider both an IID setting, where the dataset is randomly divided into equal parts and an extreme non-IID setting, also used in [MMR+17] and [TXMA21], where each client is assigned samples from only 2 out of 10 classes (with each client having the same local dataset size). In the IID setting, the model is always trained for 50 FL rounds. For the non-IID setting, it is trained for 500 on MNIST and for 1000 on CIFAR-10. For each dataset-distribution combination, baseline models are trained through FL without watermarking to evaluate the fidelity of the watermarked model later on.

Table 5.3: Hyperparameters for server-side watermarking

MNIST	Initial Train	Learning Rate	0.1
		Momentum	0.5
		Weight Decay	0.00005
	Retrain	Learning Rate	0.005
		Momentum	0
		Weight Decay	0
		Max Epochs per Round (t_{epoch})	100
		Accuracy Threshold (t_{wm})	98%
CIFAR-10	Initial Train	Learning Rate	0.0005
		Momentum	0.5
		Weight Decay	0.00005
	Retrain	Learning Rate	0.0005
		Momentum	0
		Weight Decay	0
		Max Epochs per Round (t_{epoch})	100
		Accuracy Threshold (t_{wm})	98%

As mentioned in Chapter 3, some watermarking schemes offer different additional capabilities besides backdoor-watermarking; for example, FedTracker[SYG+24] implements traitor tracing and FedIPR[LFG+22] proposes adding white-box watermarks and a mechanism for free-rider detection. However, our focus is black-box watermarking, so other capabilities are not implemented. In all experiments, we use a single watermarking entity – either client or server. The FL schemes used are Waffle[TXMA21], FedTracker[SYG+24], and client-side watermarking [LSY+21, YSY+22, LFG+22]. For the latter, different weights were used for boosting by the watermarking client (1 and 10 for CIFAR-10, 1, 10, and 20 for MNIST). Since the models do not have batch normalization layers and the traitor tracing component of FedTracker was omitted, Waffle and FedTracker differ only in the continual learning mechanism used in FedTracker (cf. Section 3.1.2).

There are a number of watermarking parameters specific to the two server-side schemes: the learning rate, momentum, and weight decay values for both the initial watermarking (before the first round of FL) and for the retraining (throughout the FL process), as well as the values for the two stopping conditions for the retraining (watermark accuracy threshold and maximum number of retraining epoch per FL round). The values for these parameters are the same as in Waffle and are shown in Table 5.3.

For each training run with watermarking, one of the four trigger sets described in Section 5.2 is used. Each training setup is repeated 3 times but with different random seeds, leading to differences in model initialization, the distribution of samples among clients, and, for the trigger set types R and RP, the backgrounds. This allows us to analyze how consistent the respective approaches are. We report the average accuracy on the test set and on the trigger set, and their standard deviation. For server-side watermarking,

we also report the average number of watermarking epochs and the standard deviation.

5.5 Attacks

We perform two common watermark removal attacks that do not require intermediate model states – pruning and inversion (cf. Section 2.3.6) – and novel attacks that use intermediate model states – WM-diff, single-state inversion on intermediate models and multi-state inversion (cf. Section 4.2). The former are conducted on both client-side and server-side watermarking runs. WM-Diff and Multi-State Inversion attacks will be performed only on server-side watermarking. For client-side watermarking, only the runs with the highest considered boosting factor that does not impact the test set accuracy are attacked. This ensures that the watermark is fully embedded in as many cases as possible, allowing us to measure the effectiveness of the removal attacks under optimal watermarking conditions. For server-side watermarking runs, FedTracker runs are attacked only if there are significant differences in convergence speed, either in terms of watermarking or the primary task, compared to Waffle.

5.5.1 Pruning

For this attack, we perform global pruning on the convolutional and linear layers of the respective model. The parameters with the lowest L1-norm are removed using the implementation provided in PyTorch [5]. L1-pruning was also used in [SYG+24, UNSS17, TXMA21]. The reason, we use pruning based on the magnitude of weights and not activation-based pruning is, that for the latter a representative dataset is needed. In our non-IID setting, however, the clients only have data from 2 out of 10 classes, making it impossible to identify relevant neurons for the primary task based on activations on the local dataset. We report the results at different pruning levels (percentage of parameters to remove) ranging from 5% to 95% in increments of 5%. The pruning attack is performed on the final model of each training run.

5.5.2 WM-Diff

To execute the WM-Diff attack (cf. Section 4.2.1), the initial weights of the target model (prior to watermarking) are required. All layers except the last are initialized with pre-trained weights. The last layer is randomly initialized on VGG16, because of the different number of classes between the pre-training dataset and the new dataset. For this reason, attacks are directed towards all layers except the last one. The attack is performed with the factor α ranging from 0.25 to 3.0, in increments of 0.25 on the final models of server-side watermarking runs.

Table 5.4: Overview of all model states used in the inversion attacks. The rounds from which the models are selected for the attack depend on the number of rounds of the FL training.

Total Rounds	Inversion Model Rounds
50	0, 1, 5, 10, 15, 25, 40, 50
500	0, 5, 10, 50, 200, 300, 400, 500
1000	0, 10, 25, 50, 200, 500, 800, 1000

5.5.3 Inversion

As described in detail in Section 4.2.2, the model inversion attack works by initializing an image randomly and assigning it a target label. Then, the pixel weights are changed to minimize the model loss given the image-label pair. For the optimization, an Adam optimizer with the coefficients for computing running averages of gradient and its square set to $\beta_1 = 0.9$ and $\beta_2 = 0.5$ and a learning rate of 0.1. These parameters are selected based on the implementation of Neural Cleanse [WYS+19] [6], because the general approach of the backdoor removal is similar; the main difference is that we "invert" a whole image and not just a small image patch. The number of optimization epochs is 500 for all experiments, the same number as in [GWX+19], as [WYS+19] do not report the number of epochs for the inversion of triggers that are later used for unlearning.

In the loss-based multi-state inversion, the losses of the individual models are added together before each optimization step. Otherwise, the process is unchanged. In the output-based multi-state inversion, the sets of images produced through inversion with the individual models are combined. The number of images to reconstruct is evenly split between the models. The model inversion is performed for each label. During each attack, the model inversion is repeated 10 times per label, resulting in a set of 100 images. Each of the images is assigned a new label, distinct from the one used to generate it.

Finally, the last model from the training process is trained on this new set to "unlearn" the watermark. For the unlearning, a batch size of 10 and a learning rate of 0.001 (MNIST), 0.0005 (IID CIFAR-10) or 0.0001 (Non-IID CIFAR-10) is used. The learning rate was determined by inspecting the drop in accuracy caused by the attack on the attacker's local dataset. The attack is performed multiple times using different subsets of the models produced by a training run. The model states are chosen to represent the entire training process, with a focus on including more models from earlier rounds when parameters change more significantly. The model states used are listed in Table 5.4. The attack is executed individually for each selected model state and for every combination of two model states (for each of the two typer of multi-state attacks). Performing the attack using different subsets of models is not intended as parameter tuning – an attacker would not be able to do that without access to the trigger set. Instead, this allows us

[5] https://pytorch.org/docs/stable/generated/torch.nn.utils.prune.L1Unstructured.html#torch.nn.utils.prune.L1Unstructured

[6] Code available at https://github.com/bolunwang/backdoor

Table 5.5: Runtimes and number of experiments

		VGG-16		MNIST	
		IID	Non-IID	IID	Non-IID
Training		48 × 55m	48 × 16h	60 × 15m	60 × 2h
Attacks	Pruning	48 × 20m		48 × 6m	
	WM-Diff	24 × 6m		24 × 2m	
	Inversion	1560 × 9-18m		1560 × 3-6m	

to determine how the usefulness of the model for inversion attacks changes throughout training and to what degree using multiple models enhances the attack.

5.6 Hardware

The experiments were performed on Nvidia RTX3090 GPUs with 24 GB VRAM. The federated learning was simulated using the framework Flower[7]. For the experiment, 4 to 5 clients were trained in parallel. Table 5.5 shows the number of experiments performed and the duration per experiment type.

[7] https://flower.ai/

CHAPTER 6

Evaluation

In this chapter, we conduct an evaluation and analysis of the experiments outlined in Chapter 5. The results cover the use of different watermarking methods – namely Waffle [TXMA21], FedTracker [SYG+24], and client-side watermarking [LSZ+20, YSY+22, LFG+22] – alongside training without watermarking as a baseline. We present results obtained from MNIST and CIFAR-10 datasets, each featuring 2 classes per client, both with IID data and non-IID data. All experiments are repeated 3 times with different random seeds. The tested schemes are evaluated in terms of accuracy on the test set and on the respective trigger set, and in terms of the computational overhead of watermarking. Furthermore, the effectiveness of the watermark removal attacks Pruning (Section 2.3.6), Inversion (Section 4.2.2), and Watermark-Diff (Section 4.2.1) is assessed. The average values across 3 runs are reported together with the respective standard deviation.

6.1 Target Metrics

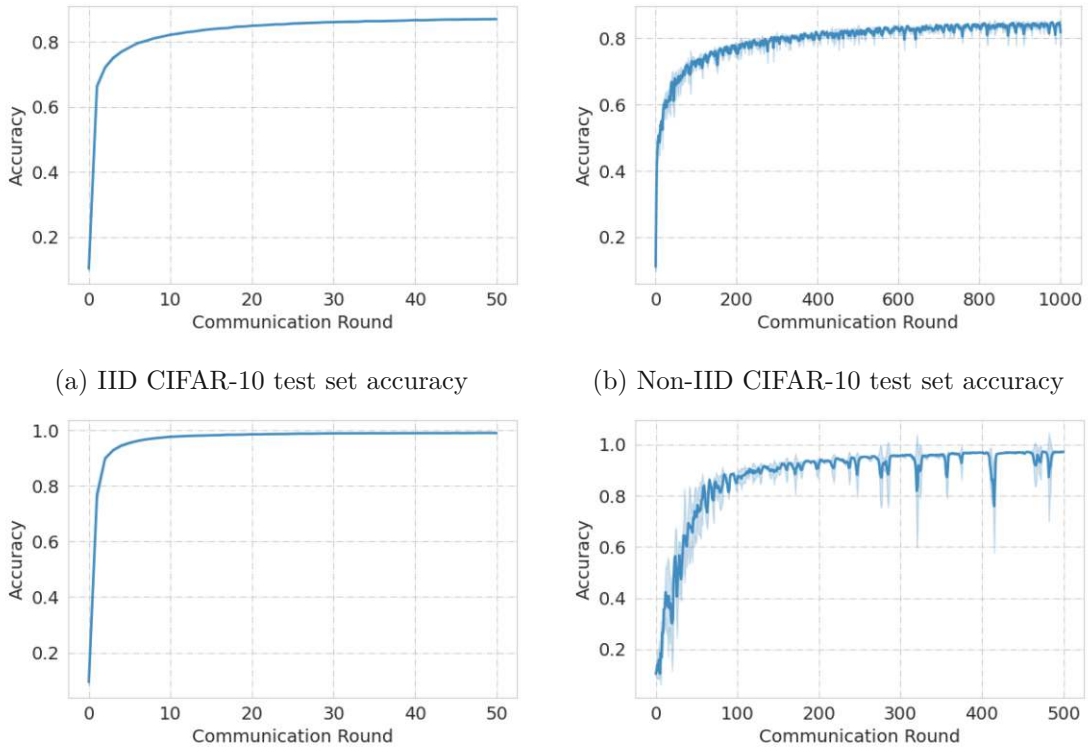
The watermark accuracy that is required to prove ownership at a certain confidence value depends on the number of classes m and on the size of the trigger set $|T|$. In [SAMA21] a formula was introduced to address this relationship:

$$P(L < e) = \sum_{i=0}^{\lfloor e|T| \rfloor} \binom{|T|}{i} \left(\frac{m-1}{m}\right)^i \left(\frac{1}{m}\right)^{|T|-i} \tag{6.1}$$

It calculates the probability that the number of errors L is below an error rate threshold e by chance, with a trigger set the model was not trained on, assuming that each label is equally likely to be the output. Since we have a trigger set size of 100 images, and 10 classes, at least 47 out of the 100 trigger images need to be correctly classified for $P(L < e) < 2^{64}$, which is the value considered in [TXMA21, SAMA21].

The drop in utility that is acceptable for an attacker is more difficult to define, as it is highly dependent on factors such as the use case and the performance of other available models. In [TXMA21] a drop of 5 percentage points or less was considered as the threshold.

While these thresholds are somewhat arbitrary and a 5-percentage point decline is rather high, for the sake of easier discussion and comparison of results we will use these values as reference points. We will refer to the watermark accuracy threshold of 47% as c_{evade} or evasion criterion. If it is satisfied, an attacker can evade watermark verification. The threshold for the drop in test set accuracy will be referred to as c_{util} or utility criterion. Model stealing will be considered possible resp. successful if both are satisfied.



(a) IID CIFAR-10 test set accuracy

(b) Non-IID CIFAR-10 test set accuracy

(c) IID MNIST test set accuracy

(d) Non-IID MNIST Test Set Accuracy

Figure 6.1: Average test set accuracy of 3 runs of watermark-free training

6.2 Training and Watermarking

In this section, the results of federated learning with and without watermarking are presented. Each experiment is repeated 3 times with different random seeds and the average test set accuracy and watermark accuracy and their standard deviation are reported in Table 6.1 for CIFAR-10, and in Table 6.2 for MNIST, respectively. They are also discussed for each scheme individually in the sections below.

6.2.1 Watermark-Free

The average test set accuracy and the standard deviation of 3 watermark-free runs throughout training for each dataset and data distribution are shown in Figure 6.1.

IID CIFAR-10 The watermark-free CIFAR-10 models achieved an average accuracy of 87.06%. The figure shows that the model performance improves very quickly within the first 10 rounds, reaching 82.24%, and then more slowly for the rest of the training. The final performance is in line with the results from [TXMA21], where the accuracy of the watermark-free model is between 85.85% and 86.27%, depending on the number of

communication rounds and local training epochs in a setting with 100 clients and with 10 of them participating at each round. In [SYG+24], the watermark-free model reportedly achieved up to 89.92% accuracy. However, there the authors use a version of VGG16 with batch normalization layers, while we use one without.

IID MNIST On MNIST, an average accuracy of 99.03% is reached. These values are comparable to those in [TXMA21], which reported accuracies ranging from 98.91% - 99.02%, depending on the choice of training parameters. On MNIST the performance improved faster than on the other dataset – reaching 76.81% accuracy after only 1 round of training and 95.5% after only 5. This means that for this setting, the watermark embedding needs to be especially fast (in terms of the number of communication rounds until the model is watermarked) in order to protect intermediate models.

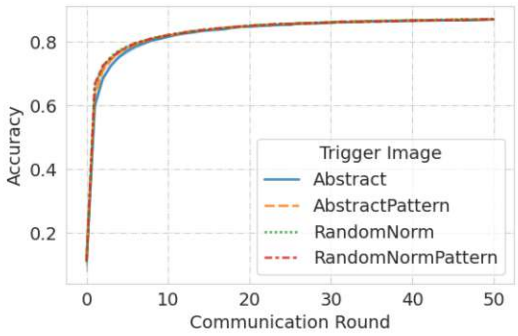
Non-IID CIFAR-10 On CIFAR-10, the model has an accuracy of only 48.95% after 10, 65.87% after 50, and 76.29% after 200 rounds. The training is also less stable than in the IID setting, with small fluctuations in the test set accuracy. On average, there are 33.33 drops of at least 5 percentage points in consecutive rounds throughout training, while there are none in the IID setting. After 1,000 rounds, the average accuracy is 81.81%, 5.25 percentage points lower than in the IID setting. Also, the standard deviation is quite high, with 3.74.

Non-IID MNIST On MNIST, it takes 51 rounds to reach the level of accuracy that was achieved in the IID setting after only 1 round (76.38%). In addition, the test set accuracy is again quite unstable, with many large performance drops. On average, there are 24.67 drops in accuracy of at least 5 percentage points from one round to the next. There were no drops of this magnitude in the IID setting. After 500 rounds of training, the models have an average accuracy of 97.14%, i.e., 1.89 percentage points lower than in the IID setting.

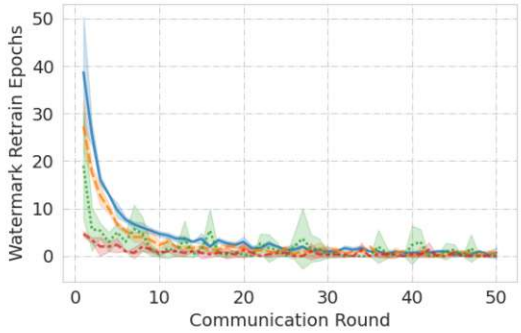
6.2.2 Waffle

The server-side watermarking scheme Waffle is described in Section 3.1.1.

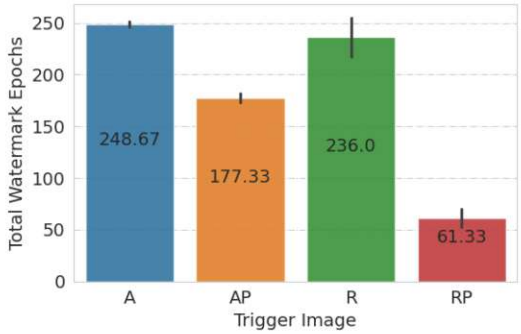
IID CIFAR-10 Figure 6.2 shows the results of the training of the VGG16 model on CIFAR-10 in the IID setting. In terms of convergence speed and the accuracy of the model at the final round, there is no notable difference between the clean model training and using Waffle, regardless of the choice of trigger image. This means that in this setting, the computational overhead is the only drawback of watermarking. In contrast to the accuracy, the overhead is highly dependent on the type of trigger set used. This can be seen in Figure 6.2c. The bar plot shows large differences in the number of watermarking epochs required throughout the training process, with the highest number being 248 epochs (Abstract) and the lowest being 61 (RandomPattern). As Abstract needs notably more training than AbstractPattern (40.23% increase) and Random notably



(a) Test set accuracy



(b) Watermark retrain epochs

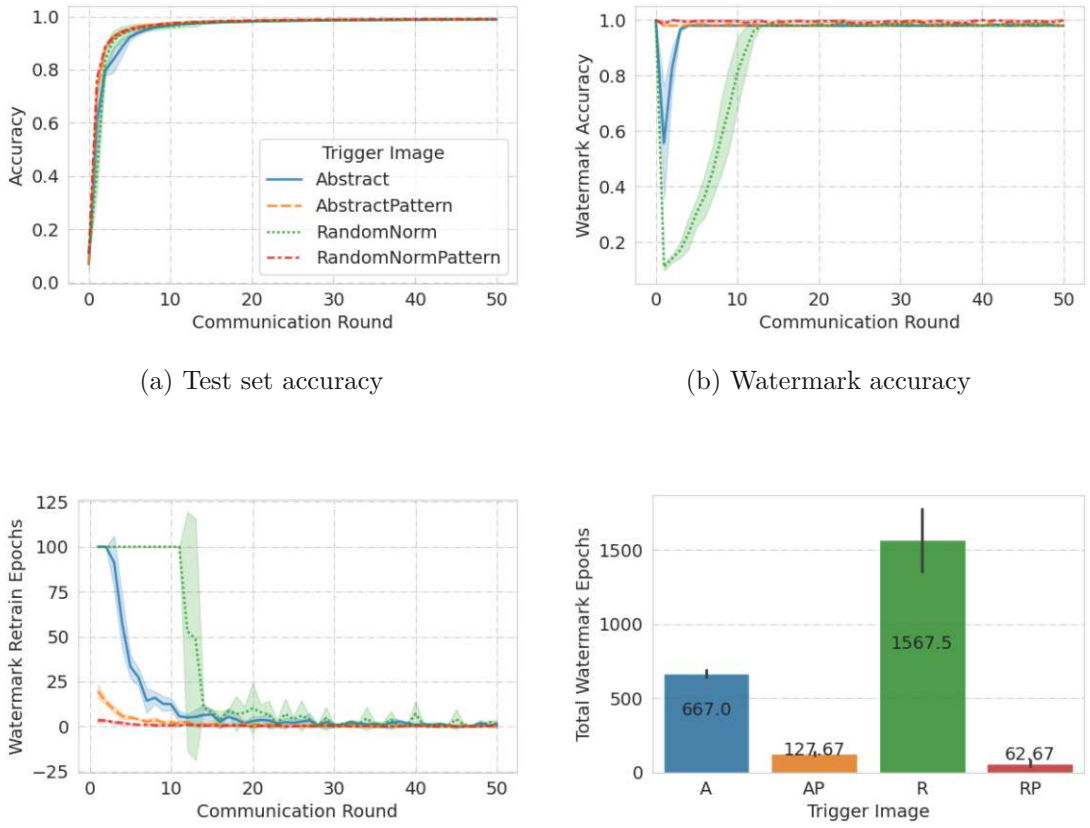


(c) Total watermark epochs

Figure 6.2: IID CIFAR-10 training results using Waffle

more than RandomPattern (284.80%), the results indicate that adding a common pattern successfully reduces the number of watermarking epochs. Figure 6.2b shows the number of watermarking epochs per round (excluding the initial watermarking). Most of the watermarking happens early on. Across trigger sets, less than 10% of the watermarking epochs occur during the last 30 rounds of training; this is because the watermark accuracy does not degrade much after local training and aggregation. Another aspect to note with regard to the watermarking is that the number of watermarking epochs per round is never equal to t_{epoch} (one of the two stopping criteria for the retraining on the trigger set), meaning that each global model state throughout the FL process has at least 98% watermark accuracy.

IID MNIST Figure 6.3 shows the results of the training on MNIST in the IID setting. As with the CIFAR-10 experiments, the performance of the MNIST models trained with Waffle is very similar to the clean model performance for nearly all runs. The exception is one run with the trigger set Random, where the training failed, both in terms of watermark and the primary task performance, because of exploding gradients. While the



(a) Test set accuracy

(b) Watermark accuracy

(c) Watermark retrain epochs

(d) Total watermark epochs

Figure 6.3: IID MNIST training results using Waffle. 1 of 3 runs with the trigger set Random failed and was excluded from the plots.

watermark is successfully embedded before the first FL round, in all successive rounds, the watermark accuracy and test set accuracy remain stuck at 10% at most. The other 2 runs with the same trigger set have a similar test set accuracy as the other trigger sets and the clean runs.

There are some differences to the CIFAR-10 results when looking at the total watermarking epochs. Firstly, the relative difference between the pattern-free trigger sets and their counterpart with pattern is even larger: The pattern reduces the watermarking epochs by 422.44% for AbstractPattern and 2,401.2% for RandomPattern (considering only the 2 successful Random runs). Secondly, the trigger set Random leads to a higher number of watermarking epochs than Abstract, which was the other way around for CIFAR-10. On MNIST, the pattern-free runs train on the trigger set for t_{wm} epochs during multiple rounds (on average, during 11.5 rounds with successful Random runs and 2.66 with A). The affected global model states have not reached a watermark accuracy of 98%. This can be seen in Figure 6.3b, which shows the watermark accuracy after retraining on

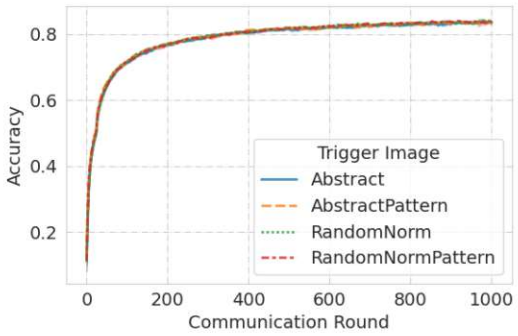
the trigger set. Model states with a watermark accuracy below 98% occur only during early rounds of training. Some model states satisfy c_{evade} – one of the two criteria for successful model stealing discussed in Section 6.1. When this occurs is described below:

- A: c_{evade} is only satisfied for a single round in 1 out of 3 runs. The test set accuracy of the vulnerable model state is only 70.07%, meaning c_{util} – the utility criterion for the attack – is not satisfied.
- R:c_{evade} is satisfied in both runs (the third run failed, as mentioned before). For one run, it is satisfied during 6 rounds, and for other during 7 rounds. However, both c_{evade} and c_{util} are only satisfied at the same time during 3 rounds and 2 rounds, respectively. This means, with this trigger set, intermediate model state are shared with clients that can be stolen without any additional effort (in terms of watermark removal).

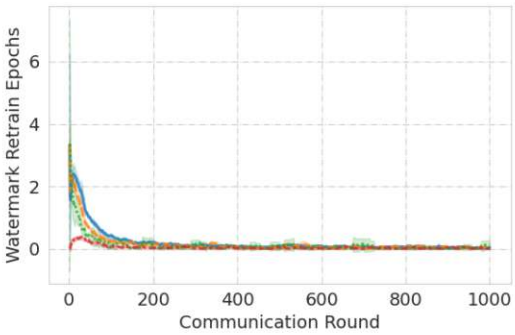
Our results on the effect of the type of trigger set on the test set accuracy differ from the results in the Waffle paper [TXMA21], especially on the MNIST dataset. In their experiments, the choice of trigger image noticeably influenced model performance, with Random consistently yielding lower accuracy compared to other trigger images across 3 experiments with varying numbers of client epochs, with a margin ranging from about 1% - 1.5% – though in their experiments 100 clients participate in training with 10 being selected each round. The bigger difference, however, is that the computational overhead from watermarking was lowest with Random, by a factor of 3.36 on MNIST and 2.02 on CIFAR-10 compared to RandomPattern. This is the opposite of our findings and somewhat counterintuitive since adding a common pattern should make learning the triggers easier. However, it is not clear how the reported overhead is measured (e.g., if there are differences in the number of FL rounds for different trigger sets in their experiments), so the differences might be related to that.

Non-IID CIFAR-10 Figure 6.4 shows the results of the training on CIFAR-10 in the non-IID setting. In terms of the final test set accuracy, there are no notable differences between trigger sets and also no notable differences to the watermark-free training. The average watermark accuracy ranges between 82.95% and 83.61%, with the standard deviation ranging from 0.85 - 2.56.

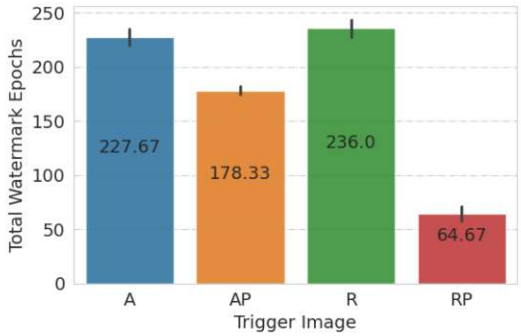
Regarding the watermarking epochs, the results are similar to the IID experiments, both in terms of the relative differences between the trigger sets and the absolute values. The difference in watermarking epochs is 21 at most (with Abstract). Most watermarking epochs still occur early on during training, and the number of watermarking epochs is never as high as the threshold t_{epoch} (meaning, t_{wm} is reached throughout). The number of watermark retraining epochs per round is much lower than in the IID setting: The most watermarking-intensive trigger set, Random, has an average maximum of 3 retraining epochs per run in a single round, compared to 38.6 in the IID setting.



(a) Test set accuracy



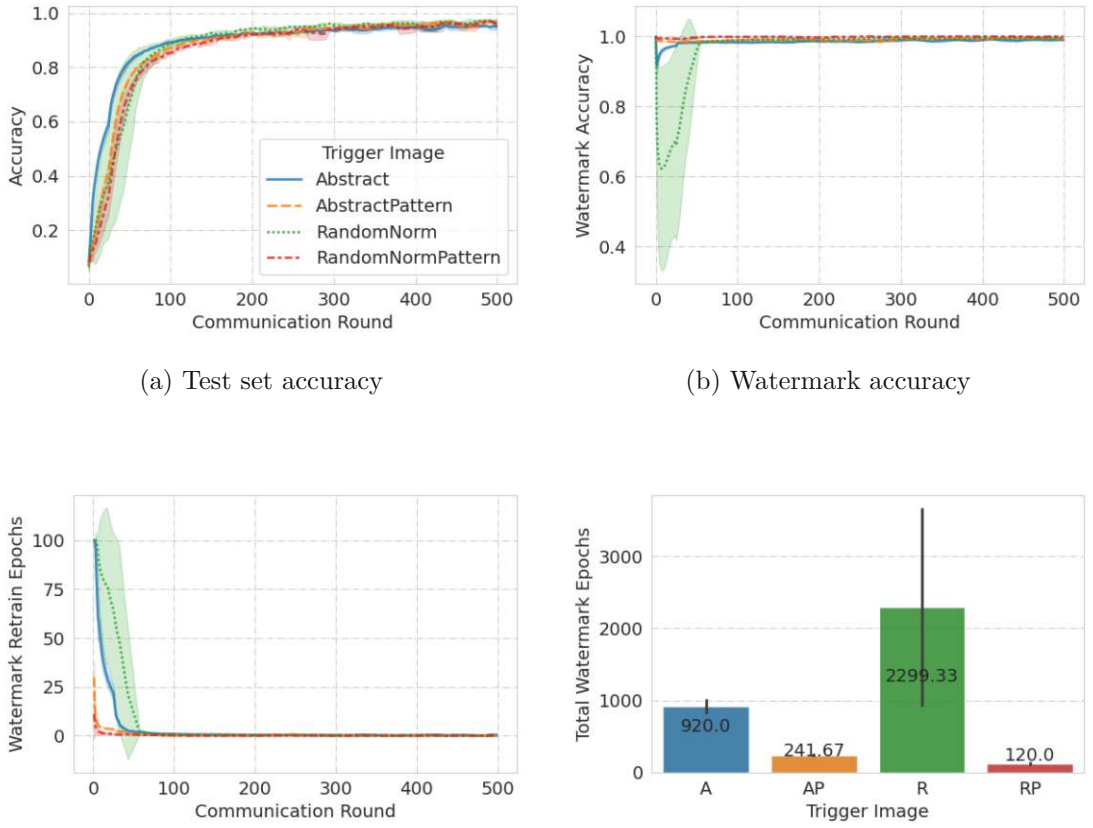
(b) Watermark retrain epochs



(c) Total watermark epochs

Figure 6.4: Non-IID CIFAR-10 training results using Waffle. The line plots are smoothed using a running average with a window size of 25.

Non-IID MNIST Figure 6.5 shows the results of the training on MNIST in the non-IID setting. During the first fifth of training, there are noticeable differences in test set accuracy between the trigger sets, with the trigger set Abstract consistently outperforming others. For example, at round 5, trigger set Abstract achieves a test set accuracy of 53.25%, compared to 21.11% for the next highest, AbstractPattern. By round 25, Abstract reaches 76.42%, while AbstractPattern follows with 67.05%. By round 50 the difference is already a lot smaller: Abstract reaches 84.62% compared to AbstractPattern's 82.05%. At the end of the training, test set accuracies range from 93.55% - 97.03%, but these differences are likely caused by the generally unstable training process and the trigger set with the lowest average test set accuracy has quite a high standard deviation (4.34). As opposed to the other dataset, on MNIST, the non-IID setting leads to a higher number of watermarking epochs across trigger sets (37.93% increase for Abstract, 89.29% for AbstractPattern 91.47% for RandomPattern and 46.70% for successful Random). While this time all Random runs are successful, it is still the trigger set with the highest average number of watermarking epochs by far (2,299.33), and it has the highest standard deviation of watermarking epochs (1,364.59). As before,



(a) Test set accuracy

(b) Watermark accuracy

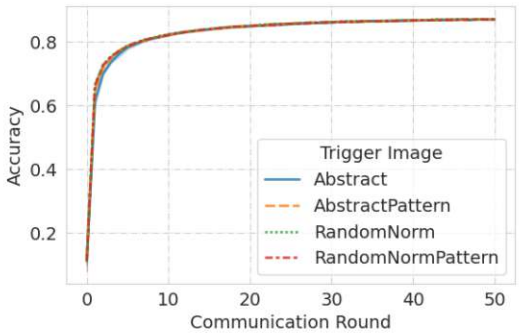
(c) Watermark retrain epochs

(d) Total watermark epochs

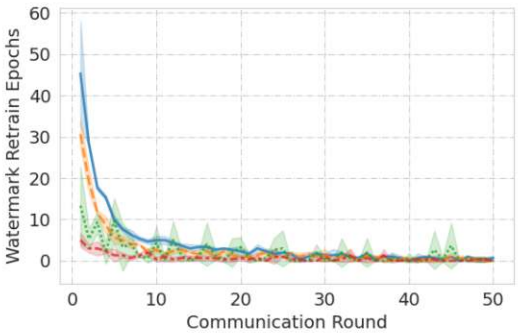
Figure 6.5: Non-IID MNIST training results using Waffle. The line plots are smoothed using a running average with a window size of 25.

the pattern-free runs are the only ones that ever meet the threshold t_{epoch}. Random watermarks for t_{epoch} epochs in 17.33 rounds on average and Abstract in 2.66 rounds. In spite of this, with Abstract, no model state satisfies c_{evade}. With Random c_{evade} is satisfied in 2 out of 3 runs – during 11 rounds with one and 17 rounds with the other. However, the highest test set accuracy is still only 36.46%, well below c_{util}. This means that directly stealing intermediate models without watermark removal is never successful in this setting (non-IID MNIST).

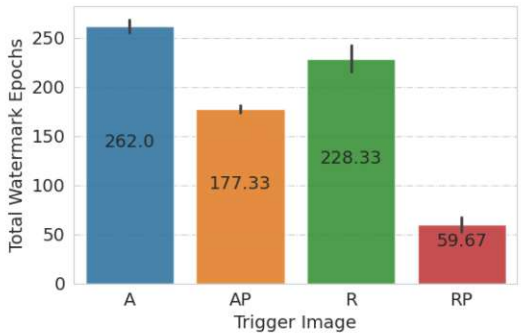
A possible explanation for why the embedding of the watermark is more challenging in the MNIST experiments than in the CIFAR-10 experiments is the model size. With 138,357,544 parameters, VGG16 likely has the spare capacity to memorize even trigger sets where images belonging to the same class have little in common, while for the smaller model we use for MNIST, that is not the case. This assumption is supported by the fact the differences mostly occur on trigger sets without a common pattern.



(a) Test set accuracy



(b) Watermark retrain epochs



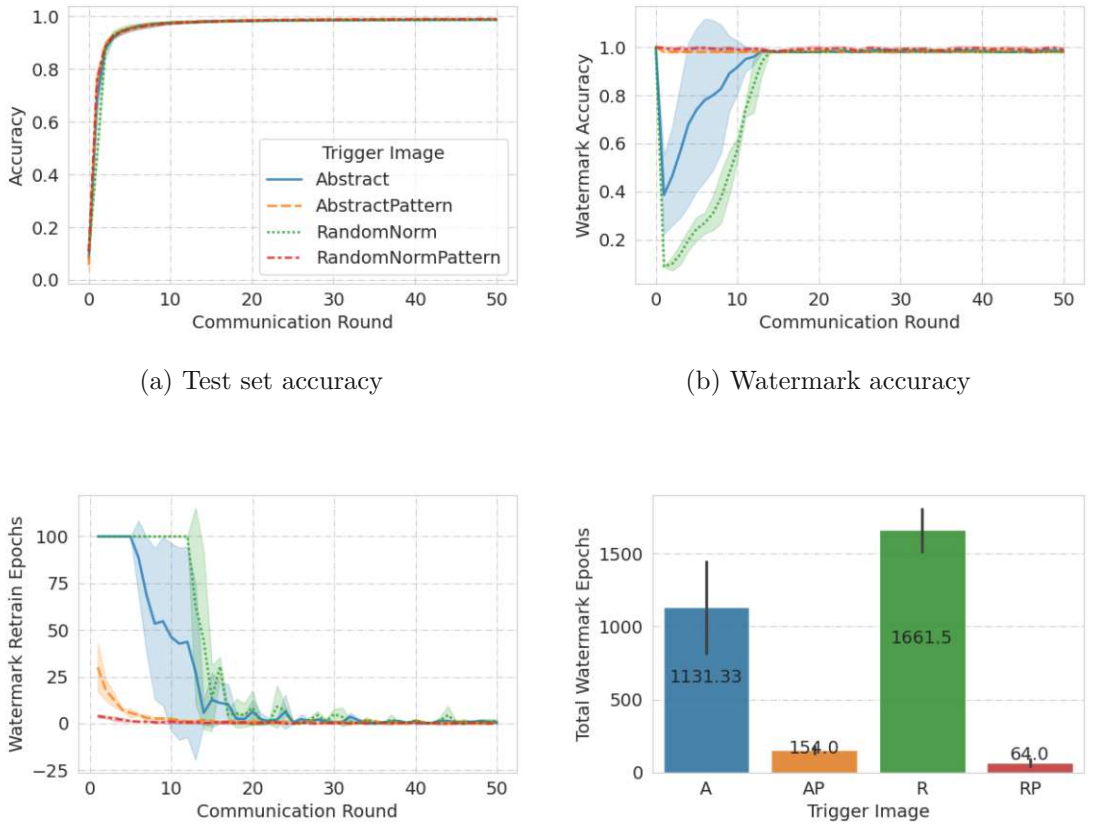
(c) Total watermark epochs

Figure 6.6: IID CIFAR-10 training results using FedTracker

6.2.3 FedTracker

IID CIFAR-10 The CIFAR-10 results are shown in Figure 6.6. For these runs, we did not find any major difference compared to not using continual learning in terms of accuracy, watermark accuracy, or watermarking epochs. One reason for that could be that most of the watermarking epochs occur during the initial watermarking, before the first FL round. The continual learning mechanism is only applied during retraining.

IID MNIST Figure 6.7 shows the results of FedTracker training with MNIST. As with Waffle, 1 out of 3 runs failed. Both of the failed runs used the same random seed for the generation of the trigger set, indicating that that particular randomly generated set was difficult to learn. Otherwise, the final test set accuracy is again unaffected by the watermark. Using FedTracker resulted in a higher number of retraining epochs for all trigger sets except RandomPattern (69.61% increase with Abstract and 20.62% with AbstractPattern and 6% with Random). The increase in watermarking epochs can be explained by FedTracker limiting the model updates during the model watermarking by restricting the change if the angle between the watermarking gradient and the descent direction on the primary task is too high, making the individual training epochs less



(a) Test set accuracy

(b) Watermark accuracy

(c) Watermark retrain epochs

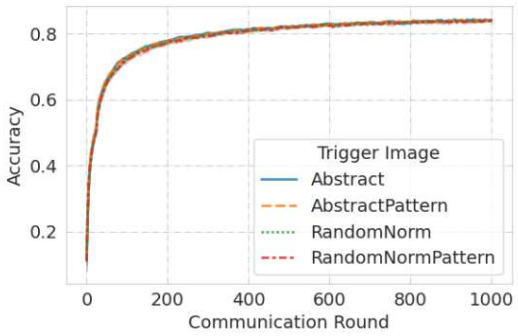
(d) Total watermark epochs

Figure 6.7: IID MNIST training results using FedTracker. 1 of 3 runs with trigger set Random failed and was excluded from the plots.

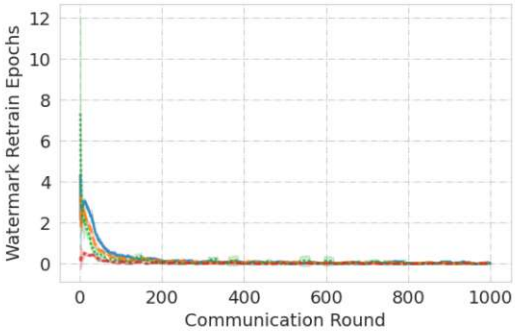
effective in terms of watermarking. On MNIST, more of the watermarking epochs occur during the retraining phase than during initial watermarking, which can explain the larger impact from continual learning compared to CIFAR-10, as it is only applied during retraining.

In terms of the protection of intermediate models, the results are as follows: With Abstract, one run has 7 model states that satisfy c_{wm}, out of which 3 also satisfy c_{util}. With Random, again, both runs (the third failed) have model states that satisfy c_{wm}. This occurs during 8 rounds and 9 rounds, respectively. In the first run, 5 model states satisfy both c_{wm} and c_{util}. In the second, that is the case for 7 model states.

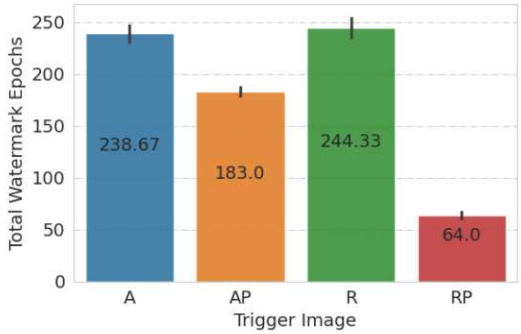
This means that for Abstract and Random, direct stealing of intermediate models, without additional watermark removal, can be successful.



(a) Test set accuracy



(b) Watermark retrain epochs

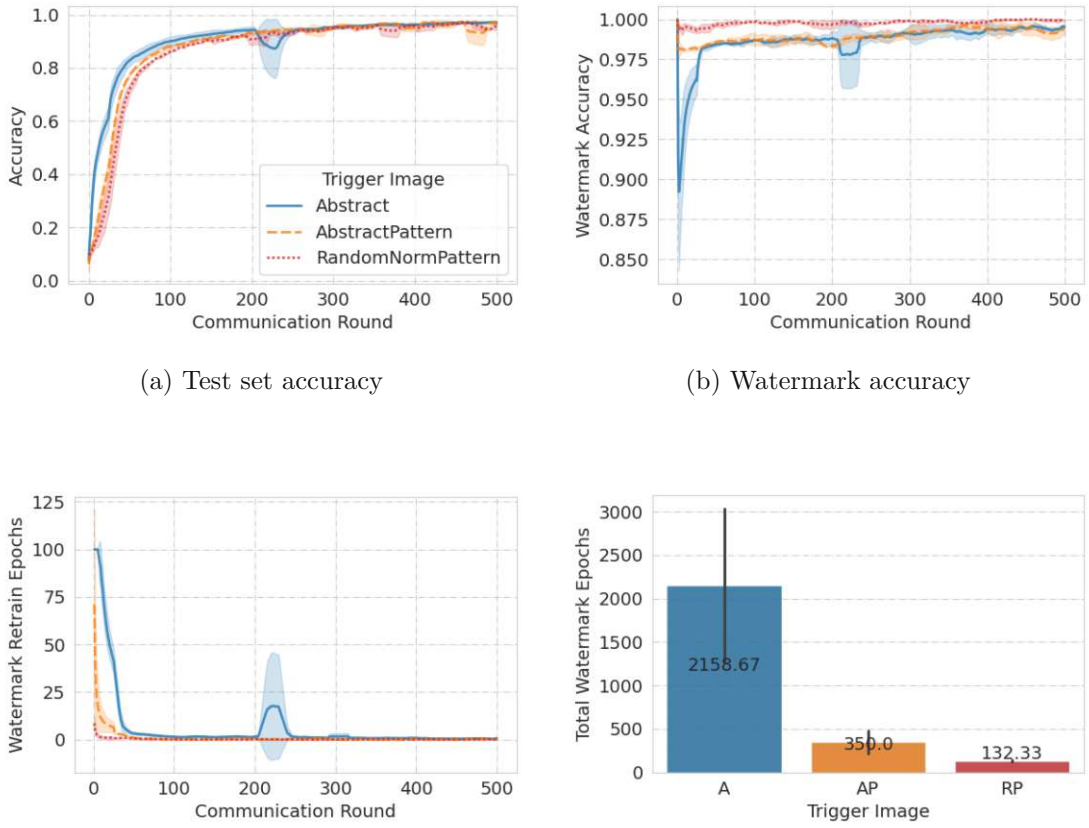


(c) Total watermark epochs

Figure 6.8: Non-IID CIFAR-10 training results using FedTracker. The line plots are smoothed using a running average with a window size of 25.

Non-IID CIFAR-10 The results of training on CIFAR-10 are shown in Figure 6.8. The results are again very similar to Waffle, both in terms of performance and the required number of watermarking epochs.

Non-IID MNIST Figure 6.9 shows the results of FedTracker training with MNIST. The most notable result is that the training using the trigger set Random failed for all 3 runs. In addition, there were stability issues with the trigger set Abstract: in one run, the test set accuracy dropped from 96% in round 202 to 29.90% in 209 before increasing again and reaching 93.12% in round 235. Otherwise, the test set accuracy is similar to the previous experiments. Just like with the IID setting for MNIST, the number of watermarking epochs is higher when using FedTracker instead of Waffle (Abstract: 134.35% increase, AbstractPattern: 44.83%, RandomPattern: 10.28%). In this setting, the models are, on average, watermarked for t_{epoch} epochs during 9 rounds with Abstract and 0.67 with AbstractPattern (in some AbstractPattern this watermarking stopping condition was not met at all). With trigger set Abstract, the watermark accuracy after aggregation (before the watermark retraining) is lower when using FedTracker, as shown



(a) Test set accuracy

(b) Watermark accuracy

(c) Watermark retrain epochs

(d) Total watermark epochs

Figure 6.9: Non-IID MNIST training results using FedTracker. All runs with the trigger set Random failed and were excluded from the plots. The line plots are smoothed using a running average with a window size of 25.

in Figure 6.10. No intermediate models satisfy c_{wm}, meaning they can be considered protected.

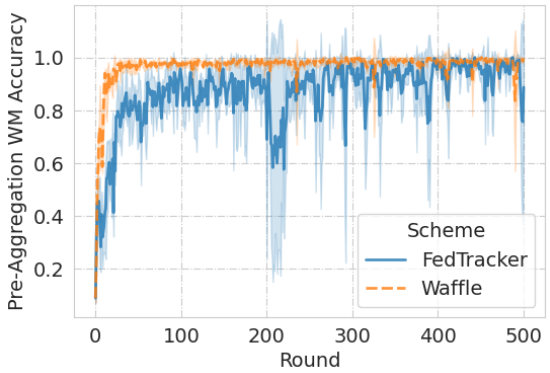
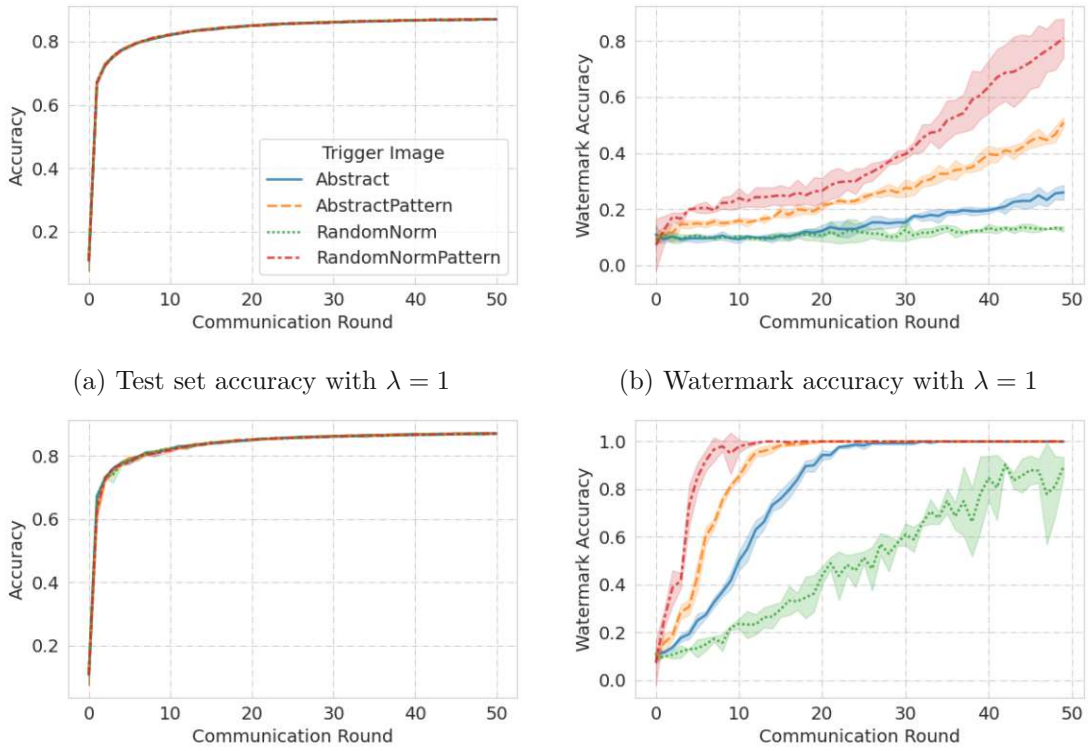


Figure 6.10: Watermark accuracy before aggregation throughout training using trigger set Abstract



(a) Test set accuracy with $\lambda = 1$

(b) Watermark accuracy with $\lambda = 1$

(c) Test set accuracy with $\lambda = 10$

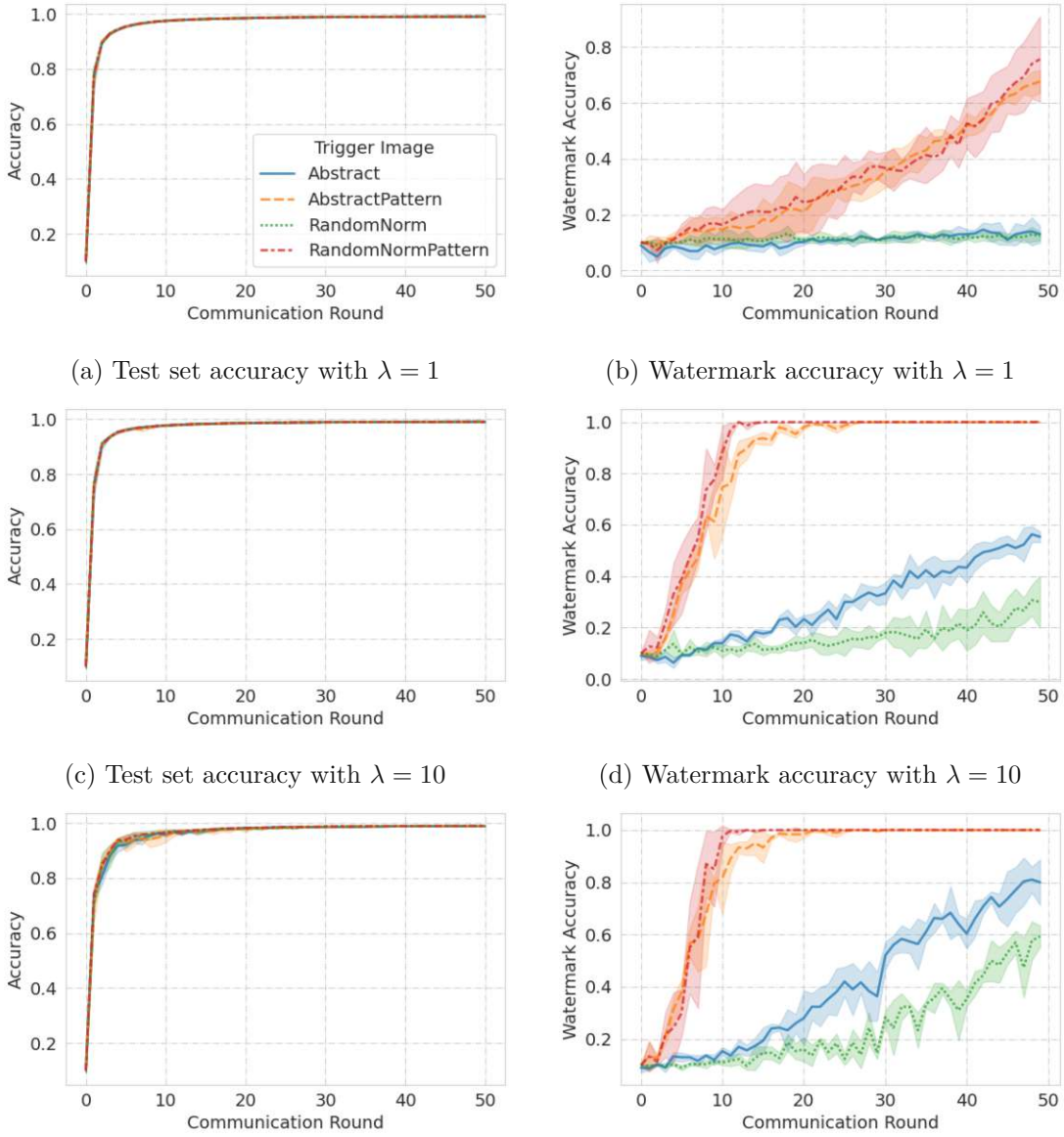
(d) Watermark accuracy with $\lambda = 10$

Figure 6.11: IID CIFAR-10 training results using client-watermarking with different λ values

6.2.4 Client Watermarking

IID CIFAR-10 The results of the CIFAR-10 training are shown in Figure 6.11. Like in the server-side watermarking, the CIFAR-10 test set accuracy is not affected by the watermarking process. In the basic approach, where the updates of the watermarking client are treated equally to any other client, watermark accuracy increases throughout training for Abstract, AbstractPattern, and RandomPattern but not sufficiently fast to protect intermediate models. For the trigger set Random, the watermark accuracy stays at roughly the same value as in the initial round throughout training. For all models trained with the trigger sets without pattern, even the final states have an accuracy that satisfies c_{evade}, meaning ownership cannot be proven, based on the thresholds defined in Section 6.1.

With AbstractPattern between 46 and 49 model states (out of a total of 50, 1 for each FL round) satisfy c_{evade} over 3 training runs. Between 37 and 40 of them also satisfy c_{util}. The highest watermark accuracy among those models is between 87.02 and 87.19%. With RandomPattern, between 32 and 37 intermediate models satisfy c_{evade}, between



(a) Test set accuracy with $\lambda = 1$

(b) Watermark accuracy with $\lambda = 1$

(c) Test set accuracy with $\lambda = 10$

(d) Watermark accuracy with $\lambda = 10$

(e) Test set accuracy with $\lambda = 20$

(f) Watermark accuracy with $\lambda = 20$

Figure 6.12: IID MNIST training results using client-watermarking with different λ values

23 and 28 satisfy both c_{evade} and c_{util}, and the test set accuracy of the best-performing ones is 86.18% - 86.69%.

These results support the hypothesis that the additional pattern makes the trigger images easier to learn, just like in server-side watermarking. While in the server-side watermarking, Abstract requires more watermarking epochs than Random, here Abstract improves faster and reaches a watermark accuracy of 26%, while Random only reaches 13%. As discussed in Section 3.2.1, watermarking clients might boost their updates. When the watermarking client boosts their update with a λ of 10, this notably increases the watermark embedding speed. With this method, the watermark accuracy reaches 47% at rounds 3 - 5 (RandomPattern), 6 - 7(AbstractPattern), 11 - 12 (Abstract) and 21 - 22(Random). This means all prior model states satisfy c_{evade}. Out of those that satisfy c_{evade} 0, 0, 1, and 13 - 15 also satisfy c_{util}. This means, once again, that models trained with pattern-less trigger sets are valuable to an attacker and unprotected based on the criteria defined in Section 6.1.

IID MNIST Figure 6.12 shows the MNIST results. Without boosting, for Random and Abstract, even the final model state satisfies t_{evade}, as the average watermark accuracy with those sets is only 12% (Random) and 13% (Abstract). For AbstractPattern and RandomPattern a watermark accuracy of 47% is reached between rounds 35 and 40 (depending on the run) for AbstractPattern and 35 - 44 for RandomPattern out of a total of 50 FL rounds. This is quite late and leaves many intermediate model states vulnerable: with AbstractPattern 34 - 39 intermediate models satisfy c_{evade}, and 31 - 36 also satisfy c_{util}. With RandomPattern, these numbers are 34 - 43 and 31 - 40, respectively.

Boosting with $\lambda = 10$ makes the embedding faster ($\geq$ 47% watermark accuracy at round 7 - 9 (AbstractPattern), 6 - 8 (RandomPattern), 34 - 43 (Abstract)), but is still too slow to protect intermediate models:

- A: There are between 37 and 42 (out of 50) model states with a watermark accuracy that satisfies c_{evade}. Out of them, 35 - 39 also satisfy c_{util}, i.e., they are valuable and unprotected models according to our thresholds. The best model states that fulfill both model stealing criteria have a test set accuracy between 99.00% and 99.09%.
- AbstractPattern: Between 6 and 8 model states have a watermark accuracy below the threshold, meaning they satisfy c_{evade}. 3 - 5 of them also satisfy c_{util}. Their test set accuracy is between 96.90% and 97.24%.
- R: All 50 model satisfy c_{evade}.
- RandomPattern: There are between 6 and 7 model states that satisfy c_evade. Out of them, 3 - 4 also satisfy c_util. The best model states have a test set accuracy between 96.97% and 97.22%.

With a boosting factor of $\lambda = 20$, the results change as follows:

- Abstract: 27 - 30 model states that satisfy c_{evade}, with 21 - 25 also satisfying c_{util}
- AbstractPattern: 6 - 7 model states that satisfy c_{evade}, with 0 - 2 also satisfying c_{util}
- Random: 42 - 46 model states that satisfy c_{evade}, with 37 - 44 also satisfying c_{util}
- RandomPattern: 6 - 8 model states that satisfy c_{evade}, with 2 - 3 also satisfying c_{util}

The increase of the boosting factor from 10 - 20 seems to be less effective than the increase from 1 - 10. This is likely because the client only trains on the trigger set for 1 epoch each round, so this might not be enough to actually learn the watermark even locally (before aggregation) within the first few rounds and the watermark and the boosting likely only helps if it is already sufficiently embedded locally. To fully control at what time the watermark gets embedded, the number of local watermarking epochs needs to be considered.

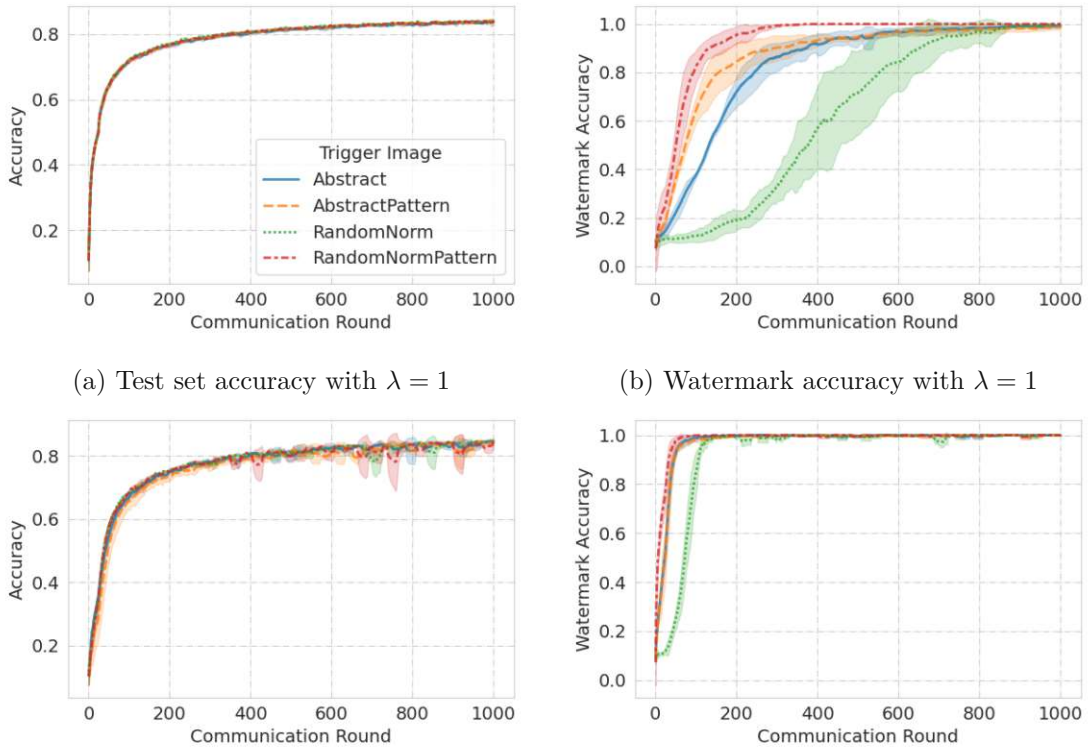
Using an even higher λ of 30 leads to very unstable training, low watermark accuracy scores, or exploding gradients. For this reason, the results of these runs are omitted.

The client-watermarking with and without boosting does not affect the test set accuracy in this setting (with $\lambda \leq 30$).

In the non-IID setting, the test set accuracy improves a lot slower than in the IID setting, so it is easier to embed the watermark before the model reaches a good performance.

Non-IID CIFAR-10 The results on CIFAR-10 are shown in Figure 6.13. Without boosting, out of 1000 model states, there are 109 - 118 (Abstract), 49 - 61 (Abstract-Pattern), 209 - 442 (Random), and 29 - 44 (RandomPattern) that satisfy c_{evade}. Their corresponding maximum test set accuracies are 74.76% - 76.06%, 68.45% - 70.93%, 81.00% - 82.63%, and 63.36% - 69.69%. Out of these models states, between 39 and 174 for Random and 0 - 5 for Abstract satisfy c_{util}.

With a boosting factor of $\lambda = 10$, the number of model states that satisfy c_{evade} decreases to between 9 and 12 (Abstract), 7 and 17 (AbstractPattern), 54 and 71 (Random), and 1 and 5 (RandomPattern). The test set accuracy of these model states is still quite low: 41.94% - 48.76%, 28.03% - 37.94%, 69.48% - 77.31%, and 10.11% - 27.24%, at most. Thus, all trigger sets are successfully embedded in time (meaning, c_{util} is never satisfied, and direct model stealing attacks are considered unsuccessful). In the non-IID setting, the boosting clearly affects the stability of the test set accuracy, as there are frequent drops in performance throughout the training process. The difference likely stems from the watermarking clients only having samples from 2 classes in our non-IID experiments, so weights relevant for the other classes easily get overwritten.



(a) Test set accuracy with $\lambda = 1$

(b) Watermark accuracy with $\lambda = 1$

(c) Test set accuracy with $\lambda = 10$

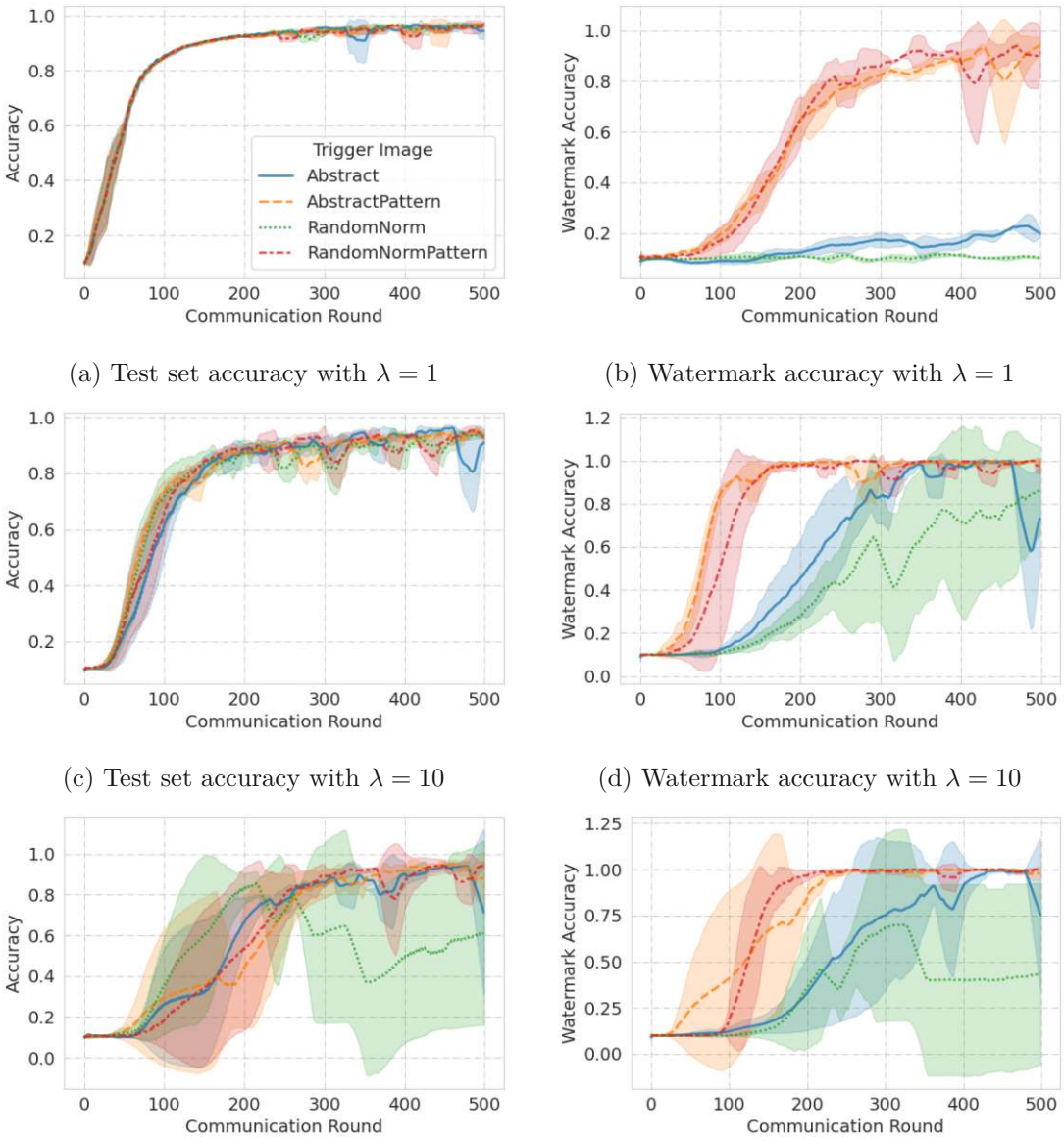
(d) Watermark accuracy with $\lambda = 10$

Figure 6.13: Non-IID CIFAR-10 training results using client-watermarking with different λ values. The line plots are smoothed using a running average with a window size of 25.

Non-IID MNIST Figure 6.14 shows the non-IID MNIST results. As in the IID setting, only models trained with the trigger sets with pattern ever reach a watermark accuracy of 47%. This means that all Random and Abstract model states satisfy c_{evade}. With AbstractPattern, there are between 163 and 165 model states out of 500 that satisfy c_{evade}, out of which 5 - 18 also satisfy c_{util} (across runs). On RandomPattern, these counts are 148 - 169 and 3 - 6, respectively.

With a λ of 10, the process becomes less stable – with many noticeable drops in test set accuracy throughout training. The final test set accuracy is also noticeably lower than without boosting, with differences of 3.13 (Abstract), 15.96 (AbstractPattern), 1.58 (Random), and 2.59 percentage points. However, the watermark is embedded earlier:

- A: Across the runs, the number of model states that satisfy c_{evade} ranges from 166 - 246 across runs. Among these, 0 - 70 states satisfy both c_{evade} and c_{util}. The best-performing of these states achieve a test set accuracy between 92.10% and 93.42%.
- AbstractPattern: The number of model states that satisfy c_{evade} varies from 61 -



(a) Test set accuracy with $\lambda = 1$

(b) Watermark accuracy with $\lambda = 1$

(c) Test set accuracy with $\lambda = 10$

(d) Watermark accuracy with $\lambda = 10$

(e) Test set accuracy with $\lambda = 20$

(f) Watermark accuracy with $\lambda = 20$

Figure 6.14: Non-IID MNIST training results using client-watermarking with different λ values. The line plots are smoothed using running average with window size 25

Table 6.1: Training results for CIFAR-10 (test set accuracy, watermark accuracy, and respective standard deviation)

		Accuracy		WM Accuracy	
		IID	Non-IID	IID	Non-IID
Trigger	Scheme				
-	No WM	87.06 ± 0.03	81.81 ± 3.74	-	-
A	Client ($\lambda = 1$)	87.03 ± 0.17	83.88 ± 1.36	26.00 ± 2.65	99.33 ± 0.58
	Client ($\lambda = 10$)	87.18 ± 0.05	83.68 ± 1.92	100.00 ± 0.00	100.00 ± 0.00
	FedTracker	87.16 ± 0.13	84.74 ± 0.33	98.33 ± 0.58	99.67 ± 0.58
	Waffle	87.08 ± 0.11	82.95 ± 2.59	98.33 ± 0.58	99.33 ± 0.58
AP	Client ($\lambda = 1$)	87.12 ± 0.02	83.54 ± 1.75	51.00 ± 1.73	99.33 ± 0.58
	Client ($\lambda = 10$)	87.12 ± 0.22	84.41 ± 0.65	100.00 ± 0.00	100.00 ± 0.00
	FedTracker	86.98 ± 0.10	83.62 ± 2.57	98.00 ± 0.00	99.67 ± 0.58
	Waffle	87.13 ± 0.07	83.61 ± 1.33	98.33 ± 0.58	99.67 ± 0.58
R	Client ($\lambda = 1$)	87.05 ± 0.08	84.02 ± 0.15	13.00 ± 1.00	100.00 ± 0.00
	Client ($\lambda = 10$)	87.13 ± 0.14	84.62 ± 0.50	89.33 ± 4.16	100.00 ± 0.00
	FedTracker	87.14 ± 0.17	83.58 ± 0.85	99.33 ± 0.58	99.67 ± 0.58
	Waffle	87.19 ± 0.09	83.24 ± 0.85	99.00 ± 1.00	99.67 ± 0.58
RP	Client ($\lambda = 1$)	87.04 ± 0.14	81.30 ± 5.67	81.00 ± 6.93	100.00 ± 0.00
	Client ($\lambda = 10$)	87.06 ± 0.15	85.08 ± 0.31	100.00 ± 0.00	100.00 ± 0.00
	FedTracker	87.09 ± 0.06	82.14 ± 1.32	98.00 ± 0.00	99.33 ± 0.58
	Waffle	87.12 ± 0.16	83.46 ± 1.39	98.33 ± 0.58	99.33 ± 1.15

73 across runs. Out of these model states, only 0 - 5 states also satisfy $c_{utility}$. The top-performing states in this category have a test set accuracy between 65.27% and 88.27%.

- R: There are between 216 and 458 model states that satisfy c_{evade}. Of these, 14 - 155 states satisfy both c_{evade} and c_{util}. The highest test set accuracy among these states is between 90.66% and 95.89%.
- RandomPattern: Between 67 and 112 model states satisfy c_{evade}. However, none of these states satisfy c_{util}, which is required for successful model stealing. The best test set accuracy in this group ranges from 77.00% - 89.57% across runs.

The stability problems become more pronounced with a boosting weight of 20, where the test set accuracy improves slower and the final test set accuracy suffers across all trigger sets. The average accuracy with Abstract and Random even drops below 65%. In this setting, AbstractPattern and RandomPattern both reach 47% watermark accuracy in time in all runs – meaning, before the test set accuracy becomes sufficiently high to satisfy c_{util}. For Abstract that is only achieved in 1 run, in the other 2 runs there are model states that satisfy c_{evade} and c_{util}.

Table 6.2: Training results for MNIST (test set accuracy, watermark accuracy, and respective standard deviation)

		Accuracy		WM Accuracy	
		IID	Non-IID	IID	Non-IID
Trigger	Scheme				
-	No WM	99.03 ± 0.08	97.14 ± 0.22	-	-
A	Client ($\lambda = 1$)	99.10 ± 0.04	96.74 ± 0.77	13.00 ± 3.00	17.33 ± 5.13
	Client ($\lambda = 10$)	99.08 ± 0.04	93.61 ± 3.34	55.33 ± 2.08	83.33 ± 15.63
	Client ($\lambda = 20$)	98.94 ± 0.03	62.03 ± 45.79	80.00 ± 8.72	67.33 ± 49.81
	FedTracker	98.88 ± 0.15	97.33 ± 0.27	98.33 ± 0.58	99.00 ± 1.00
	Waffle	98.91 ± 0.05	96.08 ± 0.68	98.00 ± 0.00	99.67 ± 0.58
AP	Client ($\lambda = 1$)	99.07 ± 0.04	97.00 ± 0.50	67.67 ± 4.04	94.00 ± 2.65
	Client ($\lambda = 10$)	99.03 ± 0.09	81.04 ± 15.82	100.00 ± 0.00	75.67 ± 42.15
	Client ($\lambda = 20$)	99.00 ± 0.04	92.86 ± 4.98	100.00 ± 0.00	99.00 ± 1.73
	FedTracker	99.09 ± 0.03	97.11 ± 0.55	98.00 ± 0.00	98.67 ± 1.15
	Waffle	99.08 ± 0.11	97.03 ± 0.17	98.00 ± 0.00	98.67 ± 0.58
R	Client ($\lambda = 1$)	99.07 ± 0.05	96.69 ± 1.50	12.33 ± 1.53	10.00 ± 0.00
	Client ($\lambda = 10$)	99.06 ± 0.04	95.11 ± 2.03	30.00 ± 10.15	89.33 ± 18.48
	Client ($\lambda = 20$)	99.07 ± 0.04	65.00 ± 47.70	59.33 ± 4.16	41.67 ± 50.58
	FedTracker	69.09 ± 51.34	10.32 ± 0.89	68.67 ± 50.81	10.00 ± 0.00
	Waffle	69.17 ± 51.42	96.72 ± 0.58	68.67 ± 50.81	99.67 ± 0.58
RP	Client ($\lambda = 1$)	99.10 ± 0.04	97.22 ± 0.48	75.67 ± 15.31	93.67 ± 5.51
	Client ($\lambda = 10$)	99.03 ± 0.07	94.63 ± 1.88	100.00 ± 0.00	100.00 ± 0.00
	Client ($\lambda = 20$)	99.03 ± 0.12	96.00 ± 0.04	100.00 ± 0.00	100.00 ± 0.00
	FedTracker	99.04 ± 0.05	97.28 ± 0.26	99.00 ± 1.00	100.00 ± 0.00
	Waffle	99.04 ± 0.06	93.55 ± 4.34	100.00 ± 0.00	100.00 ± 0.00

6.2.5 Summary

Based on the results described in this chapter, we can summarize our findings in terms of the protection of valuable intermediate model states, the test set accuracy, and the computational overhead of different watermarking schemes.

Protection of Intermediate models

The difficulty of the task, the size of the model and the data distribution all can affect the convergence speed of a model. In terms of convergence speed, the order of our experiments from fastest to slowest convergence is IID MNIST, IID CIFAR-10, non-IID MNIST, and finally non-IID CIFAR-10. The convergence speed affects how quickly the watermark needs to be embedded if the goal is to protect intermediate models once they reach a certain accuracy – and thus become valuable to an attacker. This is a particular challenge for client-side watermarking but can also be relevant to server-side

watermarking if the watermark accuracy is still low when the watermarking stopping condition t_{epoch} is met. In the CIFAR-10 experiments, the watermark is successfully embedded before the test set accuracy rises to a value within 5% of the final performance (which corresponds to model stealing criterion c_{util}), without ever reaching t_{epoch}. This means, that across runs, all intermediate models have a watermark accuracy $\geq 98\%$ (as that is the other stopping condition for watermarking). In contrast, in the MNIST experiments, the multiple times the watermarking is aborted because t_{epoch} is met. This happens mostly for the trigger sets without patterns. However, only in the IID setting with the trigger set Random does this result in intermediate models that satisfy both model stealing criteria c_{evade} and c_{util}. In the client-side experiments, the embedding of the watermark is a bigger challenge, especially in the fast-converging IID settings. Without boosting, the watermark is not embedded in time before the model states start to satisfy c_{util} (meaning their test set accuracy comes close to the final one) with either IID dataset. With boosting of $\lambda = 10$, only the trigger sets with a pattern are embedded sufficiently fast on CIFAR-10, but none are on MNIST. Further boosting with $\lambda = 20$ on MNIST ensures that AbstractPattern can be embedded in time in 1 out of 3 runs, while the embedding of the other trigger sets remains too slow. When boosting with an even higher λ, the experiments fail due to exploding gradients or the models diverge, meaning even with boosting certain watermarks cannot be embedded with client-watermarking on MNIST.

In the non-IID setting, all trigger sets are successfully embedded in time (so that there are no model states that satisfy both c_{evade} and c_{util}) with CIFAR-10 with or without boosting. The boosting still speeds up the watermark embedding, so that fewer model states satisfy c_{evade}. With MNIST, no triggers get embedded in time without boosting, while only RandomPattern gets consistently embedded in time with $\lambda = 10$, and only AbstractPattern and RandomPattern consistently with $\lambda = 20$. These results show that i) boosting successfully speeds up the watermark embedding but only up to a certain point and ii) the choice of the trigger set strongly affects the watermark embedding speed. While the effect of the background depends on the model and/or the dataset, adding a pattern notably speeds up the embedding in all cases.

Test Set Accuracy

On IID CIFAR-10, the final test set accuracy is very close across trigger sets and watermarking schemes, with a maximum average accuracy of 87.19 (Waffle, R), and a minimum of 86.98 (FedTracker, AbstractPattern). In the non-IID setting, the differences in the final score are larger, but this is mostly caused by fluctuations in performance in this setting. This conclusion is supported by the high standard deviation of particularly low accuracy scores. There, the lowest score is 81.30 (RandomPattern, Client with $\lambda = 1$), and the highest is 85.08 (RandomPattern, Client with $\lambda = 10$) On MNIST in the IID setting, the test set accuracy scores are very close for the trigger sets Abstract, AbstractPattern, and RandomPattern: they are between 98.88% (Abstract, FedTracker) and 99.10% (RandomPattern and Abstract, Client with $\lambda = 1$). The successful Random

runs are also in that range (but there are also 2 failed runs that only reach a test set accuracy of around 10%).

In the Non-IID MNIST setting, we have a quite large range of accuracy scores, reaching from 10.32% (Random, FedTracker) - 97.33% (Abstract, FedTracker). The results show that boosting in the client-watermarking schemes seems to negatively impact the performance, as a λ of 1 leads to higher average accuracy across trigger sets. FedTracker seems prone to suffer from trigger sets that are difficult to learn, as it is the only one that failed on all Random runs. The choice of trigger set and client-side watermarking without boosting did not negatively impact the test set accuracy in any experiment. Using Waffle and FedTracker only seemed to impact the performance with the Random trigger set.

Computational Overhead

Another aspect to be considered is the computational overhead. As the watermarking client trains on the trigger set each round, the overhead linearly increases with the number of rounds. For the server-side watermarking, that is not the case, because the watermarking only occurs if the watermark accuracy becomes too low and the model is watermarked for a variable number of epochs each round. Another difference between the 2 approaches is the learning rate: in server-side watermarking, a lower learning rate for watermarking is used than for training so as to not negatively impact the primary task performance. This is likely to contribute to the overhead of the server-side watermarking.

In the IID setting, the model is trained for 50 FL rounds, and in the non-IID setting for 500 (MNIST) and 1,000 (CIFAR-10) rounds, respectively. The number of watermarking epochs is equal to the respective number of rounds for client-side watermarking. Using Waffle with IID data, the average number of watermarking epochs is between 61.33 (RandomPattern) and 248.67 (Random) on CIFAR-10, and 62.67 (RandomPattern) and 1,567.5 (Random) on MNIST. In the non-IID setting, it is between 64.67 (RandomPattern) and 236.0 (Random) on CIFAR-10, and 120.0 (RandomPattern) and 2,299.0 (Random) on MNIST.

We can thus conclude the following:

- The overhead in the server-side watermarking strongly depends on the trigger set, with RandomPattern leading to the lowest and Abstract and Random leading to the highest.
- In the IID setting, the client-side watermarking leads to less overhead than the server-side watermarking with any trigger set. In the non-IID setting with CIFAR-10, the opposite is the case – all trigger sets lead to much fewer than 1,000 watermarking epochs. On MNIST, the trigger sets with patterns lead to less overhead in server-side watermarking than in client-side watermarking.

FedTracker does not affect the overhead on CIFAR-10 but increases the number of watermarking epochs on MNIST.

6.3 Attacks

In this section, we present the results of watermark removal attacks. The experimental setup is detailed in Section 5.5. We begin with the results of pruning attacks, which are conducted using only the final model. This is followed by the novel WM-Diff attack, which leverages intermediate model states. Finally, we discuss the results of all inversion attacks, including both traditional model inversion that uses only the final model state and new inversion attacks that utilize intermediate model states.

Attacks are performed on (i) all Waffle training runs, except one that failed due to exploding gradients, and (ii) Client-side watermarking runs with $\lambda = 10$ for CIFAR-10 and non-IID MNIST and $\lambda = 20$ for IID MNIST, because they reached high watermark accuracy most consistently.

The client-side watermarking runs are only targeted by the attacks that do not use intermediate model states.

The counts of successful watermark removals based on the model stealing criteria defined in Section 6.1 for the attacks pruning, baseline inversion and WM-Diff are provided in Table 6.9.

6.3.1 Pruning

To recap, the pruning attack (cf. Section 5.5.1) removes neurons resp. their connections from a watermarked model, in the hope of deactivating those neurons that contribute to recognizing the trigger sets. The attack's strength can be varied by adjusting the percentage of neurons that are pruned (pruning rate).

Table 6.3 summarizes the results of the pruning attack on the models trained on CIFAR-10. The attack was performed with pruning rates between 25% and 95%, though the table covers only the part of the pruning rate range where either the watermark accuracy or the test set accuracy begins to decline. The results show that multiple factors affect the effectiveness of the attack:

- **Trigger Images**: From the results of the attack with a pruning rate of 90% in the IID setting, we can see that the average test set accuracy of models trained on different trigger sets is very close (between 84.39% and 84.88%), while the watermark accuracy differs more substantially. For the server-side watermarking, it drops to 51% (Abstract), 67.67% (AbstractPattern), 34.00% (Random), and 45.67% (RandomPattern), and for the client-side watermarking to 74.33% (Abstract), 86.00% (AbstractPattern), 35.67% (Random), and 73.67% (RandomPattern). In both cases, the attack is more effective with i) trigger sets with random background than abstract and ii) pattern-less trigger sets. In the non-IID setting, the test set accuracy varies more: at a pruning rate of 90%, is ranges from 75.84% - 81.41%. The watermark accuracies are 56.00% (Abstract), 68.00% (AbstractPattern), 36.00% (Random), and 56.00% (RandomPattern) for the server-side watermarking, and

Table 6.3: Pruning Results CIFAR-10 (test set accuracy, watermark accuracy and respective standard deviations per pruning rate (PR))

			IID		Non-IID	
Trigger	Scheme	PR	Accuracy	WM Accuracy	Accuracy	WM Accuracy
A	Client ($\lambda = 10$)	80	86.72 ± 0.13	100.00 ± 0.00	83.71 ± 1.22	100.00 ± 0.00
		85	86.20 ± 0.45	99.33 ± 0.58	83.43 ± 1.18	99.33 ± 1.15
		90	84.39 ± 0.44	74.33 ± 3.21	81.41 ± 1.04	87.00 ± 1.00
		95	52.01 ± 1.88	14.00 ± 3.00	57.02 ± 14.87	20.33 ± 6.66
	Waffle	80	86.46 ± 0.34	88.33 ± 5.51	81.32 ± 3.24	94.33 ± 2.31
		85	85.96 ± 0.45	80.00 ± 5.57	79.99 ± 3.77	86.67 ± 5.51
		90	84.40 ± 0.60	51.00 ± 3.46	75.84 ± 3.31	56.00 ± 1.73
		95	57.05 ± 3.45	12.00 ± 1.73	41.87 ± 4.07	15.33 ± 4.93
AP	Client ($\lambda = 10$)	80	86.76 ± 0.39	100.00 ± 0.00	83.32 ± 0.97	100.00 ± 0.00
		85	86.24 ± 0.55	99.33 ± 0.58	82.09 ± 1.63	98.33 ± 2.08
		90	84.88 ± 0.89	86.00 ± 1.00	78.00 ± 3.82	87.00 ± 14.42
		95	51.44 ± 1.78	10.33 ± 0.58	48.31 ± 11.49	25.67 ± 11.72
	Waffle	80	86.74 ± 0.15	95.00 ± 0.00	82.93 ± 0.74	97.33 ± 0.58
		85	86.30 ± 0.29	89.00 ± 1.00	82.29 ± 0.83	90.33 ± 1.53
		90	84.46 ± 0.90	67.67 ± 10.12	79.41 ± 1.36	68.00 ± 8.66
		95	52.26 ± 8.56	10.33 ± 0.58	39.30 ± 4.46	11.00 ± 2.00
R	Client ($\lambda = 10$)	80	86.63 ± 0.27	71.67 ± 19.66	83.45 ± 0.70	99.67 ± 0.58
		85	86.08 ± 0.44	59.00 ± 24.43	82.48 ± 0.99	96.33 ± 3.21
		90	84.74 ± 0.51	35.67 ± 22.01	78.81 ± 1.66	63.00 ± 23.64
		95	50.78 ± 2.50	10.00 ± 0.00	54.60 ± 9.23	17.00 ± 4.36
	Waffle	80	86.92 ± 0.06	81.67 ± 4.04	82.25 ± 1.01	86.67 ± 10.21
		85	86.35 ± 0.17	64.33 ± 8.33	81.54 ± 1.35	71.33 ± 8.14
		90	84.70 ± 0.46	34.00 ± 7.55	78.92 ± 1.26	36.00 ± 7.00
		95	54.08 ± 8.06	10.00 ± 0.00	45.33 ± 6.38	9.00 ± 1.73
RP	Client ($\lambda = 10$)	80	86.70 ± 0.35	100.00 ± 0.00	84.05 ± 0.69	100.00 ± 0.00
		85	86.20 ± 0.18	99.67 ± 0.58	83.20 ± 0.71	99.67 ± 0.58
		90	84.42 ± 0.34	73.67 ± 6.66	80.05 ± 2.55	88.67 ± 10.50
		95	47.71 ± 3.07	13.33 ± 3.51	48.76 ± 4.46	24.00 ± 11.27
	Waffle	80	86.77 ± 0.12	88.00 ± 9.64	82.79 ± 1.07	94.67 ± 0.58
		85	86.26 ± 0.20	71.00 ± 7.00	81.41 ± 1.52	84.33 ± 4.04
		90	84.70 ± 0.24	45.67 ± 2.08	79.15 ± 2.30	56.00 ± 1.73
		95	49.58 ± 6.21	9.67 ± 0.58	43.84 ± 5.39	12.67 ± 3.51

Table 6.4: Pruning Results MNIST (test set accuracy, watermark accuracy, and respective standard deviations per pruning rate (PR))

			IID		Non-IID	
Trigger	Scheme	PR	Accuracy	WM Accuracy	Accuracy	WM Accuracy
A	Waffle	80	98.79 ± 0.07	93.33 ± 1.15	95.76 ± 0.67	97.67 ± 1.15
		85	98.74 ± 0.10	88.67 ± 3.06	95.58 ± 0.74	95.67 ± 1.53
		90	98.51 ± 0.15	74.33 ± 3.79	94.69 ± 0.85	84.67 ± 8.14
		95	97.27 ± 0.48	44.33 ± 5.51	88.68 ± 3.12	55.00 ± 7.55
	Client	80	98.89 ± 0.05	80.00 ± 2.65	93.04 ± 3.75	93.00 ± 6.56
	($\lambda = 10/20$)	85	98.88 ± 0.04	76.67 ± 2.52	92.78 ± 3.85	92.00 ± 7.55
		90	98.68 ± 0.17	71.00 ± 4.00	91.87 ± 4.27	87.00 ± 4.00
		95	97.45 ± 0.51	54.67 ± 0.58	83.67 ± 8.62	66.67 ± 2.52
AP	Waffle	80	99.02 ± 0.07	95.00 ± 2.65	96.68 ± 0.15	98.00 ± 1.73
		85	98.97 ± 0.05	91.67 ± 3.06	96.67 ± 0.26	97.33 ± 2.08
		90	98.81 ± 0.13	83.67 ± 3.51	95.97 ± 0.34	91.67 ± 7.09
		95	97.11 ± 0.38	44.67 ± 11.02	88.42 ± 0.99	75.33 ± 8.50
	Client	80	98.94 ± 0.03	100.00 ± 0.00	89.22 ± 5.56	100.00 ± 0.00
	($\lambda = 10/20$)	85	98.84 ± 0.09	100.00 ± 0.00	89.33 ± 5.14	100.00 ± 0.00
		90	98.68 ± 0.20	100.00 ± 0.00	88.80 ± 4.97	99.50 ± 0.71
		95	97.22 ± 0.72	83.33 ± 7.77	82.27 ± 1.73	95.00 ± 7.07
R	Waffle	80	98.78 ± 0.15	96.00 ± 1.41	96.51 ± 0.40	95.33 ± 3.21
		85	98.78 ± 0.13	94.00 ± 0.00	96.03 ± 0.62	88.00 ± 9.54
		90	98.64 ± 0.15	86.00 ± 2.83	94.88 ± 0.50	72.33 ± 21.20
		95	97.27 ± 0.38	37.50 ± 3.54	82.52 ± 6.92	40.00 ± 6.08
	Client	80	99.00 ± 0.14	50.00 ± 10.39	94.70 ± 1.96	87.33 ± 17.79
	($\lambda = 10/20$)	85	98.98 ± 0.15	47.00 ± 9.64	94.48 ± 1.84	86.67 ± 17.39
		90	98.80 ± 0.15	42.67 ± 12.42	93.68 ± 2.04	81.33 ± 17.47
		95	97.28 ± 0.45	25.67 ± 13.61	87.64 ± 0.88	66.67 ± 15.04
RP	Waffle	80	99.02 ± 0.04	97.33 ± 3.06	93.19 ± 4.33	99.67 ± 0.58
		85	98.98 ± 0.04	92.67 ± 3.79	93.26 ± 4.20	99.33 ± 0.58
		90	98.87 ± 0.10	77.00 ± 5.57	93.09 ± 3.70	99.67 ± 0.58
		95	97.71 ± 0.70	42.33 ± 23.54	87.69 ± 2.89	84.00 ± 7.94
	Client	80	98.96 ± 0.09	100.00 ± 0.00	93.91 ± 2.09	100.00 ± 0.00
	($\lambda = 10/20$)	85	98.93 ± 0.14	100.00 ± 0.00	93.64 ± 2.07	100.00 ± 0.00
		90	98.75 ± 0.16	100.00 ± 0.00	92.03 ± 2.31	100.00 ± 0.00
		95	97.54 ± 0.73	81.33 ± 24.19	83.94 ± 2.36	94.67 ± 6.11

87.00% (Abstract), 87.00% (AbstractPattern), 63.00% (Random), and 88.67% (RandomPattern) for the client-side watermarking. This means that the two observations from the IID setting above still hold for the server-side watermarking. This is not the case for client-side watermarking, as RandomPattern, Abstract, and AbstractPattern have very similar or even the same watermark accuracy. However, Random is still the least robust trigger set.

- **Client- vs. Server-side Watermarking**: Based on the watermark accuracy values mentioned above for the trigger images, it becomes clear that the models watermarked by a client suffer a smaller decline in watermark accuracy when attacked, compared to the models watermarked on the server – both in the IID and the non-IID setting.

When considering the thresholds from Section 6.1 as the criteria for the success of the attack – meaning, if at any pruning rate, the thresholds are satisfied – the attack is

- Not successful for Abstract and AbstractPattern.
- Rarely successful against the trigger set RandomPattern (only successful for 2 out of 3 runs with IID data using Waffle).
- Often successful against the trigger set Random (3 out of 3 server-side IID runs, 2 out of 3 server-side non-IID runs, and 2 out of 3 client-side IID runs).

The success counts can also be found in Table 6.9. The MNIST results are shown in Table 6.4. One IID Waffle model watermarked with the trigger set Random is excluded from the attack as its training failed, and one non-IID client-side watermarking model with the trigger set AbstractPattern is excluded from the table because its watermark accuracy was already less than 47% (c_{evade}) after training. In the IID setting with a pruning rate of 95%, the average test set accuracy is between 97.11% and 97.71%, while the watermark accuracy is 44.33% (Abstract), 44.67% (AbstractPattern), 37.5% (Random), and 42.33%(RandomPattern) for server-side watermarking, and 54.67% (Abstract), 83.33% (AbstractPattern), 25.67% (Random), and 81.33%(RandomPattern) for client-side watermarking[1]. Thus, for 3 out of 4 trigger sets, the client-side watermarking was more robust than the server-side watermarking, and in all cases, the trigger sets with a pattern were more robust than their pattern-less counterparts. In terms of the watermark accuracy and test set accuracy criteria, the attack succeeds in most server-side experiments: in all Abstract and Random runs, 2 out of 3 AbstractPattern and 2 out of 3 RandomPattern runs. With client-side watermarking, the watermarks are more robust, with the attack succeeding only on the trigger set Random (on all 3 runs).

[1]The failed run has a test set accuracy of 51.73% and a watermark accuracy of 38% at this pruning rate. Before the attack, the test set accuracy was 63.33%

Table 6.5: WM-Diff CIFAR-10 Results (test set accuracy, watermark accuracy, and respective standard deviations per Factor)

		IID		Non-IID	
Trigger	Factor	Accuracy	WM Accuracy	Accuracy	WM Accuracy
A	0.5	86.85 ± 0.18	83.33 ± 5.13	82.70 ± 2.19	84.67 ± 0.58
	0.75	86.70 ± 0.17	64.67 ± 6.11	82.26 ± 1.82	57.33 ± 5.51
	1.0	86.39 ± 0.08	49.67 ± 2.08	81.38 ± 1.29	35.00 ± 1.00
	1.25	86.05 ± 0.19	36.33 ± 5.51	79.95 ± 0.76	20.00 ± 1.73
AP	0.5	87.03 ± 0.07	88.33 ± 2.31	83.13 ± 1.34	89.00 ± 7.94
	0.75	86.90 ± 0.03	77.67 ± 3.21	82.40 ± 1.50	63.67 ± 16.56
	1.0	86.61 ± 0.09	54.00 ± 6.08	81.39 ± 1.40	39.67 ± 12.42
	1.25	86.16 ± 0.05	40.00 ± 6.08	79.85 ± 1.31	20.67 ± 4.73
R	0.5	87.11 ± 0.16	63.00 ± 9.17	82.42 ± 1.29	60.67 ± 6.43
	0.75	86.91 ± 0.16	24.00 ± 5.20	81.65 ± 1.49	19.67 ± 7.23
	1.0	86.57 ± 0.17	14.00 ± 4.36	80.47 ± 1.93	10.00 ± 1.00
	1.25	86.03 ± 0.20	10.33 ± 0.58	78.77 ± 2.61	9.33 ± 0.58
RP	0.5	86.92 ± 0.18	86.00 ± 4.58	83.30 ± 1.29	88.67 ± 8.50
	0.75	86.82 ± 0.13	61.67 ± 4.73	82.94 ± 1.18	58.67 ± 14.19
	1.0	86.70 ± 0.25	38.67 ± 3.51	82.17 ± 0.95	21.33 ± 7.09
	1.25	86.44 ± 0.18	23.67 ± 2.08	80.98 ± 0.87	9.33 ± 2.52

In the non-IID setting at a pruning rate of 90%, the average test set accuracy is between 88.80% and 95.97%. The watermark accuracy is 84.67% (Abstract), 91.67% (AbstractPattern), 72.33% (Random), and 99.67%(RandomPattern) for server-side watermarking, and 87.00% (Abstract), 99.50% (AbstractPattern), 81.33% (Random), and 100.00%(RandomPattern) for client-side watermarking. With a pruning rate of 95%, the test set accuracy declines below the acceptable range in all experiments (i.e., c_{util} is not satisfied anymore). Using trigger sets with patterns leads to higher robustness for both client- and server-side watermarking; the client-side watermarking is more robust across trigger sets than server-side watermarking.

6.3.2 WM-Diff

Our proposed WM-Diff attack (cf. Section 4.2.1) is applicable when the attacker has access to 2 model states, where the main difference is that one is watermarked and the other is not. The scaled weight difference can then be computed and subtracted from the final model to remove the watermark. In our experiments, we performed this attack on Waffle runs using the initialization weights and the weights after the initial watermarking (before the first round). The factor used for scaling the weight difference controls the strength of the attack.

Table 6.6: WM-Diff MNIST Results (test set accuracy, watermark accuracy, and respective standard deviations per Factor)

	Distribution	IID		Non-IID	
		Accuracy	WM Accuracy	Accuracy	WM Accuracy
Trigger	Factor				
A	0.25	98.60 ± 0.03	53.00 ± 19.47	93.19 ± 3.09	64.33 ± 16.50
	0.50	94.93 ± 2.38	15.33 ± 1.53	67.30 ± 14.13	18.67 ± 5.69
	0.75	65.20 ± 29.79	10.00 ± 3.00	20.06 ± 5.09	10.00 ± 0.00
AP	0.25	98.81 ± 0.18	63.33 ± 2.89	93.31 ± 0.85	79.00 ± 10.39
	0.50	96.34 ± 1.12	10.00 ± 7.00	71.32 ± 8.31	13.33 ± 4.73
	0.75	65.47 ± 2.17	6.33 ± 4.51	36.23 ± 7.31	5.00 ± 3.61
R	0.25	98.80 ± 0.05	73.00 ± 24.04	95.94 ± 0.51	69.67 ± 18.72
	0.50	97.47 ± 0.85	39.00 ± 36.77	91.15 ± 2.23	34.67 ± 27.30
	0.75	88.76 ± 9.00	22.50 ± 17.68	64.19 ± 14.70	22.33 ± 16.44
RP	0.25	98.90 ± 0.03	35.33 ± 25.74	91.30 ± 4.78	83.33 ± 14.29
	0.50	97.67 ± 0.74	8.67 ± 2.31	82.03 ± 9.25	32.00 ± 35.51
	0.75	87.86 ± 7.85	8.00 ± 3.46	58.70 ± 8.34	7.33 ± 4.62

Table 6.5 shows the results of the WM-Diff attack on the CIFAR-10 models. In the IID setting, the attack barely affected the test set accuracy, while dropping the watermark accuracy to very low levels: with a scaling factor α of 1.25, the test set accuracy is less than 1.5 percentage points lower than prior to the attack, while the average watermark accuracy is 40.00% at most. Based on the attack success criteria c_{evade} and c_{util}, the attack is successful in all runs in this setting (cf. Table 6.9).

In the non-IID setting, the test set accuracy is more sensitive to the attack, declining at a slightly faster rate than in the IID setting when increasing the factor: with a factor of 1.25, the drop of test set accuracy is above 5 percentage points for 1 Abstract run and 1 Random run. With a factor of 1.0, however, the test set accuracy for all trigger sets is within 4.5 percentage points of the original value, while the average watermark accuracy is 39.67 at most. Thus, again, all attacks can be considered successful.

The results of the attack on MNIST are shown in Table 6.6. The models are more sensitive to the attack compared to the CIFAR-10 results: With a factor of 0.75, the test set accuracy is already well beyond the acceptable range, with a drop of more than 10 percentage points across settings (i.e, it does not satisfy c_{util}). In this setting, there is a factor for each run with which both attack success criteria are satisfied, but the exact value of that factor varies across the trigger sets. In the non-IID setting, however, there is only a single Random run for which the attack is successful.

6.3.3 Inversion

In this subsection, the results of inversion attacks are presented. In these removal attacks trigger images are reverse-engineered and used for 'unlearning', meaning training the model to forget the backdoor. Inversion is described in more detail in Section 2.3.6 and our proposed inversion attacks that use intermediate model states are explained in Section 4.2.2.

We first discuss inversion attacks that use only a single model state to inverse the images, then loss-based multi-state attacks, and finally output-based multi-state attacks.

Single-State Inversion

In this attack, the inversion algorithm is applied to a single model state. If that model state is the final one, the attack corresponds to traditional inversion attacks, where the inversion is performed on the model state that the attacker wants to steal. We consider these inversion attacks as the baseline. We proposed inversion attacks where earlier model states are used to reconstruct the trigger set, which is then used for 'unlearning' on the final model.

We first discuss the baseline attacks on client-side watermarking schemes (for which only the baseline attack is performed) and then all single-state attacks for the server-side schemes (including the baseline).

Table 6.7 and Table 6.8 show the results of single-state inversion attacks.

Client-Side Watermarks In both the IID and the non-IID setting, models watermarked with trigger sets Random on both datasets drop to a watermark accuracy of 28.00% at most, while the test set accuracy stays within 5 percentage points of the final test set accuracy, meaning both attack success criteria are satisfied. On IID Cifar10 with the trigger set RandomPattern, the watermark accuracy drops to 40.33 on average, but looking at individual values shows that c_{evade} is only satisfied for 2 out of 3 runs. For both of them c_{util} is satisfied as well. On non-IID MNIST, there are also individual Abstract and AbstractPattern runs where the attack was successful based on the defined criteria. For all other experiments, c_{evade} is not satisfied, so that ownership can still be proven after the attack.

Sever-Side Watermarks on IID CIFAR-10 For the attacks on models trained with Waffle, we can make the following observations:

- **Robustness with different Trigger Sets**: The differences in robustness caused by different trigger sets follow the same trend as in prior attacks, with models trained on trigger sets with random background leading to lower watermark accuracy after the attack than those with abstract background. For example, for the baseline attacks (that are performed using only the final model state), the average watermark

Table 6.7: Single-state inversion results on CIFAR-10

		IID			Non-IID	
			Accuracy	WM Accuracy		Accuracy	WM Accuracy
Trigger	Scheme						
A	Waffle	0	86.25 ± 0.15	75.33 ± 7.02	0	82.14 ± 3.30	89.33 ± 2.31
		1	85.32 ± 0.84	66.33 ± 10.69	10	80.07 ± 6.07	77.33 ± 13.50
		5	83.80 ± 1.88	54.00 ± 4.36	25	80.41 ± 4.48	80.33 ± 11.15
		10	83.32 ± 1.24	52.00 ± 7.00	50	81.69 ± 3.49	89.67 ± 4.16
		15	83.64 ± 1.00	56.00 ± 5.57	200	81.06 ± 4.05	86.67 ± 3.79
		25	82.93 ± 1.52	56.33 ± 4.93	500	80.99 ± 4.36	85.00 ± 6.56
		40	83.95 ± 0.46	63.33 ± 5.03	800	80.31 ± 4.27	71.00 ± 21.00
		50	84.56 ± 0.30	69.67 ± 4.93	1000	81.86 ± 2.78	87.00 ± 11.36
	Client	50	84.89 ± 0.31	82.33 ± 3.04	1000	82.38 ± 2.16	88.33 ± 3.21
AP	Waffle	0	85.75 ± 0.49	83.33 ± 3.21	0	81.52 ± 2.27	90.00 ± 7.81
		1	84.14 ± 0.89	64.33 ± 22.85	10	81.80 ± 1.87	93.33 ± 6.43
		5	83.62 ± 0.97	55.67 ± 6.03	25	81.43 ± 2.91	90.33 ± 4.93
		10	82.74 ± 1.63	44.67 ± 5.69	50	80.95 ± 2.84	91.33 ± 2.08
		15	81.90 ± 1.01	45.67 ± 12.86	200	81.69 ± 2.26	91.00 ± 2.00
		25	82.52 ± 1.61	50.67 ± 3.06	500	80.01 ± 3.86	81.33 ± 16.77
		40	83.01 ± 0.61	48.00 ± 2.00	800	79.39 ± 4.60	72.33 ± 22.74
		50	84.50 ± 0.85	60.33 ± 11.85	1000	81.56 ± 2.00	89.67 ± 4.93
	Client	50	84.53 ± 0.44	77.33 ± 10.15	1000	83.46 ± 0.69	97.33 ± 1.53
R	Waffle	0	85.58 ± 0.64	12.33 ± 5.69	0	82.16 ± 1.32	23.67 ± 4.16
		1	83.72 ± 1.50	11.33 ± 0.58	10	81.37 ± 0.93	13.67 ± 3.21
		5	81.15 ± 3.01	12.33 ± 1.53	25	80.37 ± 0.36	12.33 ± 4.51
		10	82.64 ± 2.90	10.00 ± 0.00	50	79.54 ± 1.08	10.67 ± 0.58
		15	82.99 ± 2.67	10.67 ± 0.58	200	80.61 ± 1.98	11.00 ± 3.61
		25	83.44 ± 2.22	10.00 ± 1.00	500	80.44 ± 1.09	12.33 ± 1.53
		40	84.06 ± 1.64	10.67 ± 1.53	800	81.17 ± 1.69	10.00 ± 3.00
		50	83.96 ± 1.46	8.00 ± 2.00	1000	82.28 ± 0.81	12.33 ± 1.15
	Client	50	84.14 ± 1.01	10.67 ± 1.00	1000	83.04 ± 0.94	28.00 ± 3.46
RP	Waffle	0	85.62 ± 0.70	47.00 ± 13.11	0	83.02 ± 1.43	57.00 ± 11.27
		1	83.81 ± 1.04	27.33 ± 8.02	10	82.81 ± 1.26	39.00 ± 9.54
		5	82.94 ± 1.05	22.67 ± 1.53	25	82.54 ± 1.38	49.00 ± 9.64
		10	83.18 ± 0.62	20.00 ± 7.55	50	82.23 ± 1.79	36.67 ± 14.50
		15	83.63 ± 0.89	12.67 ± 2.89	200	82.30 ± 1.56	39.00 ± 10.15
		25	83.60 ± 0.87	13.67 ± 5.51	500	82.29 ± 1.52	55.00 ± 11.79
		40	84.10 ± 0.58	20.00 ± 9.17	800	81.66 ± 2.89	39.00 ± 5.57
		50	84.54 ± 0.61	21.67 ± 7.02	1000	83.11 ± 1.56	41.33 ± 9.87
	Client	50	85.64 ± 0.43	40.33 ± 13.43	1000	83.00 ± 1.30	55.33 ± 5.77

Table 6.8: Single-state inversion results on MNIST

			IID			Non-IID	
			Accuracy	WM Accuracy		Accuracy	WM Accuracy
Trigger	Scheme						
A	Waffle	0	98.93 ± 0.02	86.00 ± 2.00	0	95.56 ± 0.83	69.67 ± 10.12
		1	98.64 ± 0.19	50.67 ± 10.97	5	94.73 ± 1.28	39.00 ± 1.00
		5	98.68 ± 0.04	47.00 ± 2.65	10	94.79 ± 1.11	37.67 ± 6.66
		10	98.63 ± 0.05	44.33 ± 4.04	50	94.77 ± 1.04	38.33 ± 9.07
		15	98.61 ± 0.05	43.00 ± 5.29	100	94.77 ± 0.98	43.00 ± 15.52
		25	98.64 ± 0.07	48.33 ± 9.07	200	94.73 ± 1.04	49.33 ± 25.01
		40	98.66 ± 0.09	47.67 ± 9.50	300	94.83 ± 1.03	49.33 ± 18.88
		50	98.66 ± 0.05	50.33 ± 5.86	500	94.72 ± 1.18	48.67 ± 15.31
	Client	50	98.82 ± 0.03	63.33 ± 9.07	500	92.87 ± 3.48	56.33 ± 15.04
AP	Waffle	0	98.94 ± 0.06	72.67 ± 10.60	0	95.63 ± 1.26	70.67 ± 15.14
		1	98.84 ± 0.09	60.33 ± 9.29	5	95.54 ± 1.35	67.00 ± 19.92
		5	98.82 ± 0.18	70.00 ± 6.56	10	95.27 ± 1.67	62.33 ± 24.83
		10	98.84 ± 0.19	74.33 ± 9.61	50	95.28 ± 1.41	72.00 ± 22.11
		15	98.84 ± 0.21	76.67 ± 9.71	100	95.55 ± 1.12	75.00 ± 19.97
		25	98.81 ± 0.23	72.67 ± 17.39	200	95.68 ± 1.03	84.33 ± 11.85
		40	98.83 ± 0.23	74.00 ± 16.37	300	95.87 ± 0.62	78.00 ± 18.52
		50	98.84 ± 0.25	75.33 ± 19.14	500	95.76 ± 0.85	79.00 ± 18.00
	Client	50	98.70 ± 0.16	96.67 ± 2.31	500	83.66 ± 10.83	33.00 ± 5.66
R	Waffle	0	98.86 ± 0.11	90.50 ± 9.19	0	96.51 ± 0.38	50.67 ± 31.66
		1	98.61 ± 0.17	51.50 ± 12.02	5	96.66 ± 0.29	27.33 ± 14.84
		5	97.88 ± 0.32	29.00 ± 12.73	10	96.61 ± 0.36	24.67 ± 11.02
		10	98.15 ± 0.29	14.00 ± 4.24	50	96.35 ± 0.08	19.67 ± 8.14
		15	98.20 ± 0.25	15.00 ± 5.66	100	96.19 ± 0.12	19.33 ± 8.39
		25	98.22 ± 0.29	15.50 ± 7.78	200	96.23 ± 0.11	18.33 ± 8.39
		40	98.32 ± 0.30	18.00 ± 7.07	300	96.33 ± 0.14	19.67 ± 8.14
		50	98.33 ± 0.34	23.00 ± 2.83	500	96.07 ± 0.11	13.67 ± 3.51
	Client	50	98.91 ± 0.03	18.33 ± 7.64	500	94.39 ± 1.43	10.33 ± 0.58
RP	Waffle	0	98.94 ± 0.07	38.33 ± 3.79	0	92.91 ± 4.59	60.67 ± 31.82
		1	98.82 ± 0.11	37.00 ± 13.53	5	92.78 ± 4.76	66.33 ± 31.63
		5	98.87 ± 0.08	55.33 ± 21.94	10	92.33 ± 5.22	59.33 ± 28.50
		10	98.85 ± 0.11	58.00 ± 14.73	50	92.11 ± 5.31	66.67 ± 28.38
		15	98.83 ± 0.18	60.00 ± 7.00	100	91.97 ± 5.45	72.67 ± 22.19
		25	98.80 ± 0.22	57.67 ± 12.42	200	92.01 ± 5.45	75.33 ± 18.01
		40	98.74 ± 0.21	50.33 ± 19.30	300	92.07 ± 5.58	76.67 ± 18.50
		50	98.79 ± 0.26	56.67 ± 15.50	500	91.94 ± 5.89	67.00 ± 27.22
	Client	50	98.60 ± 0.21	84.33 ± 23.69	500	94.08 ± 1.93	99.00 ± 1.73

accuracies drop to 69.67% (Abstract), 60.33% (AbstractPattern), 8.0% (Random), and 21.67% (RandomPattern). These differences are also present when other model states are used for the attack.

- **Final Model versus Intermediate Models** For trigger sets Abstract, AbstractPattern, and RandomPattern, most intermediate model states lead to a considerably lower average watermark accuracy than the final model state. The intermediate model states with the lowest watermark accuracy per trigger set are:
  - Abstract: Model state 10 (out of 50) with a watermark accuracy that is 17.67 percentage points lower than that of the final model state and a test set accuracy that is only 1.24 percentage points lower.
  - AbstractPattern: Model state 10 with a watermark accuracy that is 15.66 percentage points lower than that of the final model state and a test set accuracy that is only 1.76 percentage points lower.
  - RandomPattern: Model state 15 with a watermark accuracy that is 9 percentage points lower than that of the final model state and a test set accuracy that is only 0.91 percentage points lower.
  - Random: The baseline attack leads to the lowest watermark accuracy (8%), but all other attacks also lead to low values (10% - 12.3%).

  Interestingly, model state 0 leads to the highest watermark accuracy, even above the baseline attack. Model state 1 leads to lower watermark accuracy than model state 0, but still higher than the baseline. However, on trigger sets Abstract, AbstractPattern, and RandomPattern, the watermark accuracy progressively decreases up until the model states 10 - 15 and then increases again with the later states. One possible explanation is, that early model states are too dissimilar to the final model state, while very late model states are already well-trained on the primary task, making the inversion of trigger-related images less likely. In terms of test set accuracy, most attacks that use intermediate model states lead to a higher decline than the baseline attacks by up to 2.8 percentage points. However, the single-state attacks that use model state 0 cause the smallest drop in test set accuracy for all trigger sets.

On this dataset and data distribution, using intermediate model states improves the effectiveness of the single-state attack for most trigger sets when model states > 1 are used. On trigger set Random, where that is not the case, all attacks already lead to a very low watermark accuracy anyway, with an average of 12.3% at most.

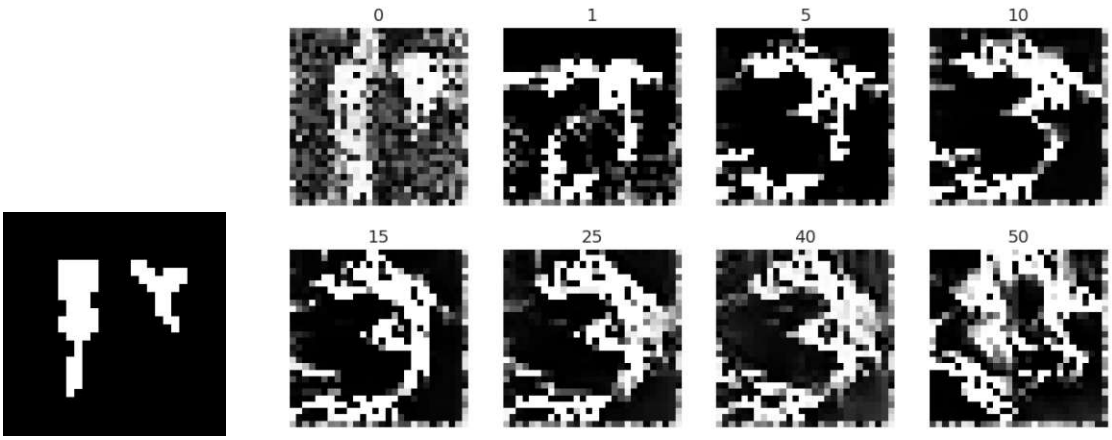
Sever-Side Watermarks on IID MNIST Our observations based on the MNIST results are as follows:

- **Robustness with different Trigger Sets**: On MNIST, the results show that i) adding a pattern contributes to the robustness and ii) random backgrounds

are less robust than abstract backgrounds, with the following average watermark accuracy for the baseline attacks: 50.33% (Abstract), 75.33% (AbstractPattern), 23.0% (Random) and 56.67% (RandomPattern).

- **Final Model versus Intermediate Models**: Across trigger sets, there are intermediate model states that lead to considerably lower watermark accuracy than the final model state. The ones that reduce the watermark accuracy the most are:
  - Abstract: Model state 15 with a watermark accuracy that is 7.3 percentage points lower than that of the final model state.
  - AbstractPattern: Model state 1 with a watermark accuracy that is 15 percentage points lower than that of the final model state.
  - Random: Model state 10 with a watermark accuracy that is 9 percentage points lower than that of the final model state.
  - RandomPattern: Model state 1 with a watermark accuracy that is 19.7 percentage points lower than that of the final model state.

  The fact that for AbstractPattern and RandomPattern the most effective model state is a very early one (1), and for Random and Abstract, it is a later state (10 / 15) could be related to the watermark embedding being more challenging with the pattern-less trigger sets. For Random runs, the average watermark accuracy had not even reached the 47% threshold until after round 5. Surprisingly, model state 0 (which is always fully watermarked, because there is no cap on the number of watermarking epochs initially) leads to the highest watermark accuracy among pattern-less runs. On Random, it is particularly high, with 90.5%. The test set accuracy is between 97.8% and 99.0%. As before, model state 0 leads to the smallest decline in test set accuracy across trigger sets.



(a) Trigger image (b) Inversion results by round for class 3

Figure 6.15: Single-Model inversion on MNIST for different model states

Figure 6.15 shows an example of the images inverted from model states throughout training. Initially, the inverted image looks quite similar to the true trigger image – but from the third round on, it rather resembles the number 3, corresponding to the respective label in the MNIST dataset.

Sever-Side Watermarks on Non-IID CIFAR-10 The results of the attacks in this setting are as follows:

- **Robustness with different Trigger Sets**: The attack on the CIFAR-10 models is still very effective with the random backgrounds, but not with the abstract backgrounds. Attacks using only the final model state have an average watermark accuracy of 87.0% (Abstract), 89.7% (AbstractPattern), 12.3% (Random), and 41.3% (RandomPattern).

- **Final Model versus Intermediate Models**: While there is no clear trend of intermediate model states leading to lower watermark accuracies with increasing model state up to a certain point as was the case in the IID setting, there are still individual intermediate model states which have a considerably lower watermark accuracy than that of the final model state for Abstract, AbstractPattern, and RandomPattern. The ones with the lowest watermark accuracy are:

  - Abstract: Model state 800 with 17.2 percentage points lower watermark accuracy than the final model state, as well as a test set accuracy that is 2.2 percentage points lower.
  - AbstractPattern: For model state 800, watermark accuracy is 16.0 percentage points lower than that of the final model state, while the test set accuracy is 1.6 percentage points lower.
  - RandomPattern: Model state 50 has a watermark accuracy that is 4.7 percentage points lower than that of the final model state, and the test set accuracy is only 0.2 percentage points lower.
  - Random: Model state 800 has a watermark accuracy that is 2.33 percentage points below the baseline and a test set accuracy that is 1.6 percentage points below the baseline.

  For test set accuracy, the results are similar to the ones from the IID setting. Using intermediate model states mostly leads to a decrease of 3 percentage points at most.

The observations suggest that using single intermediate model states in the non-IID setting does not lead to a predictable decrease in watermark accuracy compared to the first or last model states. One reason could be the instability of the training in the IID setting.

Sever-Side Watermarks on Non-IID MNIST The attack results for this setting are discussed below:

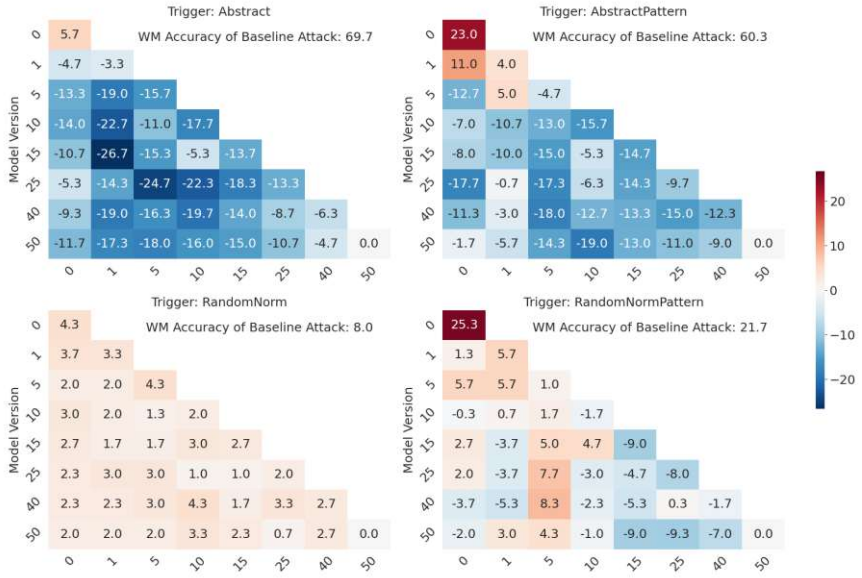
- **Robustness with different Trigger Sets**: The results in terms of watermark accuracy differences between trigger sets are similar to the IID setting: with the baseline attack, AbstractPattern leads to the highest watermark accuracy (79%), followed by RandomPattern (67%), Abstract (48.7%) and lastly Random (13.7%).
- **Final Model versus Intermediate Models**: For trigger sets Abstract, AbstractPattern, and RandomPattern, intermediate model states exist that lead to a lower watermark accuracy than the final model state. The most effective are:
  - Abstract: Model state 10 with a watermark accuracy that is 11 percentage points lower than that of the final model state.
  - AbstractPattern: Model state 10 with a watermark accuracy that is 16.7 percentage points lower than that of the final model state.
  - RandomPattern: Model state 10 with a watermark accuracy that is 7.7 percentage points lower than that of the final model state.

  For Random, the baseline attack is the most effective one (but again, all the other attacks on this trigger set are very effective as well). The pattern that with increasing the model state, the watermark accuracy first decreases but later increases, still holds for trigger sets Abstract and AbstractPattern, but not for Random and RandomPattern.

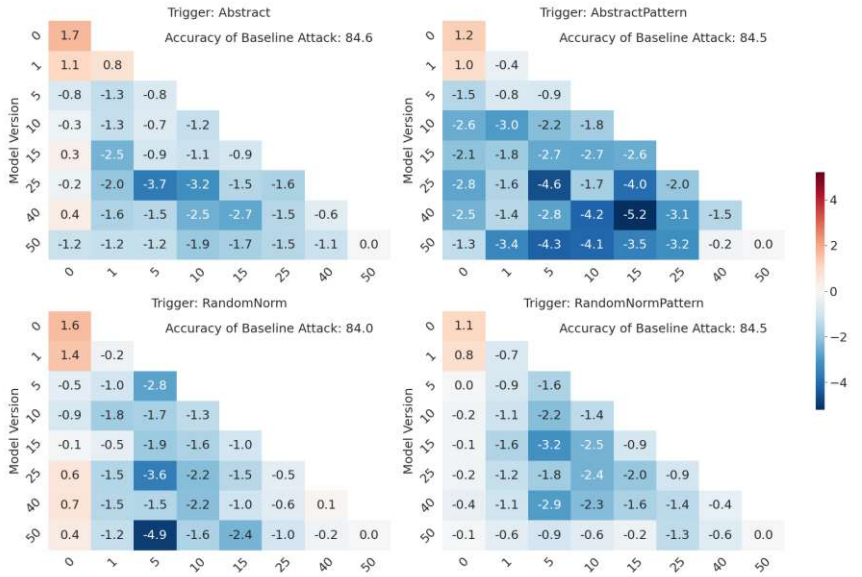
Loss-Based Multi-State Inversion

In this section, we discuss the results of loss-based multi-state attacks on all training runs with Waffle-watermarking. In these attacks, the losses of 2 model states are added together during each inversion step. The output of the inversion is then used for 'unlearning' on the final model state. The final model state can be 1 of the 2 used for inversion.

Sever-Side Watermarks on IID CIFAR-10 Figure 6.16a shows the average difference in watermark accuracy, and Figure 6.16b shows the average difference in test set accuracy between each loss-based multi-state attack and single-state attack, compared to the baseline attack (a single-state attack that uses the final model state for both inversion and unlearning). A positive value means the accuracy of the attack that uses intermediate model states is higher than that of the baseline. The values on the diagonal show the single-state attacks for comparison, so for example the value at (0,0) shows the results of the single-state attack where inversion is performed on model state 0 (but unlearning as always on the final model state) and the value at (1,0) shows the result of the attack that used model states 1 and 0 together for inversion. On Abstract, all attacks multi-state attacks, and on AbstractPattern, all but 2 lead to a lower watermark accuracy than the baseline attack. The lowest average watermark accuracy for Abstract,



(a) Average difference in watermark accuracy to baseline Attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.16: Loss-based multi-state inversion and single-state inversion with different IID CIFAR-10 model states that were trained using Waffle. The single-state inversion results are shown on the diagonal.

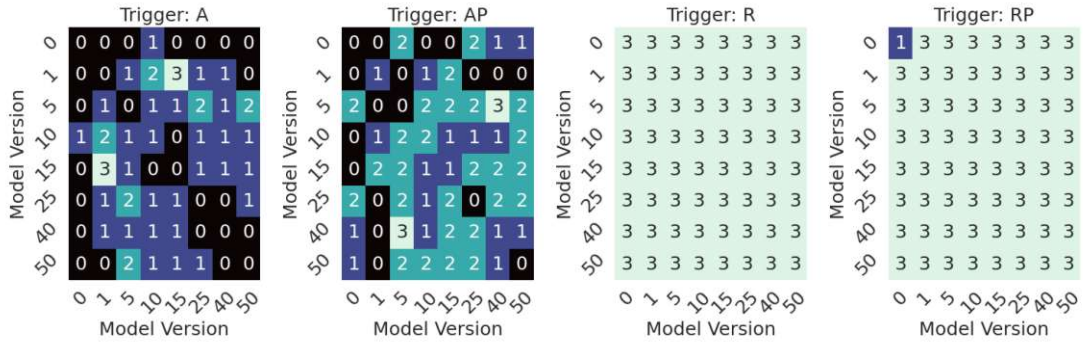


Figure 6.17: Counts of successful inversion attacks on 3 IID CIFAR-10 models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

AbstractPattern, and RandomPattern is achieved by multi-state attacks. Comparing the watermark accuracy of multi-state attacks w_m with their respective single-state counterparts w_{s1} and w_{s2}, when $w_{s1} > w_{s2}$, $w_{s1} > w_m$ tends to hold, meaning the multi-state attack is typically not less effective (on average over 3 runs) than the least effective of the 2 single-state attacks. That is the case for 26 Abstract, 25 AbstractPattern, 23 Random, and 20 RandomPattern attacks, out of 28. Furthermore, the number of sets of attacks, where $w_m < w_{s1}$, and $w_m < w_{s2}$, meaning the multi-state attack is better than both single-state attacks, is 17 for Abstract, 11 for AbstractPattern, 10 for Random, and 7 for RandomPattern. Furthermore, for Abstract, there are 9 attacks that achieve a lower average watermark accuracy than the best-performing single-state attack set, and the difference in performance between the best single-state attack set and the best multi-state attack set is 9 percentage points. For AbstractPattern, there are only 4 combinations, and the difference is 3.33 percentage points, and for RandomPattern, there are 2 combinations, and the difference is 0.33 percentage points. For Random, there are no combinations of model states where the multi-state attack performs better. The results show that there is a clear advantage of using multi-state instead of single-state attacks with trigger sets, where the watermark accuracy is difficult to degrade with the baseline attack.

Figure 6.17 shows the number of successful attacks per combination of models (and per individual model for the single-state attacks) based on the criteria c_{evade} and c_{util}, defined in Section 6.1. For both trigger sets with random background, even the baseline inversion attack, performed using only the final model, succeeds across the 3 runs. For the trigger sets with abstract background, none of the attacks using only the final model succeed, because the watermark accuracy remains too high (70.00% with Abstract and 60.33% with AbstractPattern). However, the following tuples of models successfully remove the watermark in a multi-state attack from 2 out of 3 models:

- Abstract: 1 & 10, 5 & 50

- AbstractPattern: 1 & 15, 0 & 5

Each of these attack sets includes a model state that by itself performed poorly in the single-state attack (0, 1, 50). Also, all 4 used an early model state (0 - 5) in combination with an additional model state.

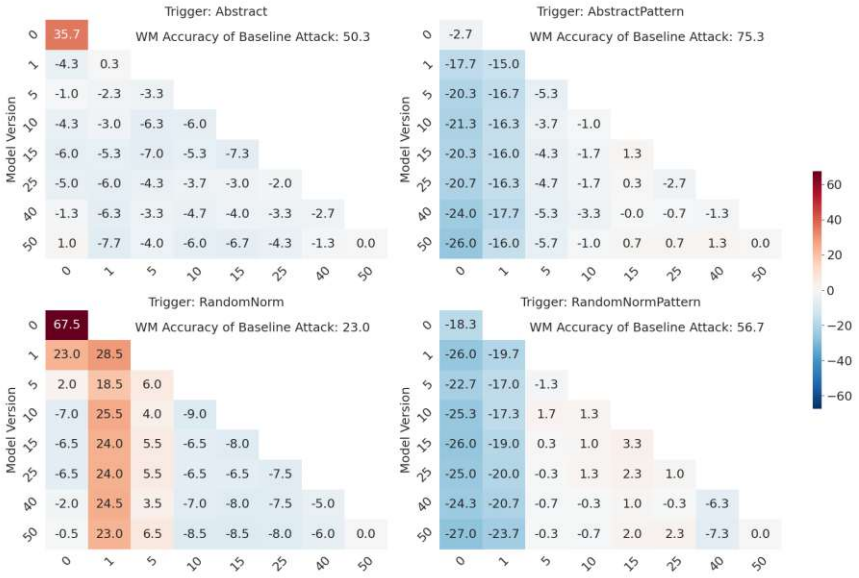
Sever-Side Watermarks on IID MNIST Figure 6.18 shows the results of the attacks.

In this setting, there are 27 (Abstract), 25 (AbstractPattern), and 24 (Random, RandomPattern) multi-state attacks out of 28 for which the average watermark accuracy is below at least one of the corresponding single-state attacks. Out of those, 10 (Abstract), 16 (AbstractPattern), 6 (Random), and 13 (RandomPattern) have a lower watermark accuracy than either of the corresponding single-state attacks. In addition, we can see that including model states 0 and 1 in multi-state attacks on trigger sets AbstractPattern and RandomPattern leads to the lowest watermark accuracy scores on these trigger sets. With these 2 trigger sets, all the attacks that include model state 0, decrease the watermark accuracy to below the watermark accuracy of the most effective single-state attack (RandomPattern: from 37% to 34%-29.7%, AbstractPattern: from 60.3% to 57.7%-49.3%). In both cases, the lowest average watermark accuracy in that setting is achieved when performing a multi-state attack with model state 0 and the final model state. For Abstract, the attacks that use 2 model states are similarly effective to those that use only 1. For the trigger set Random, we can see quite an interesting pattern: As mentioned before, the watermark accuracy is highest for the single-state attacks on early model states (0: 90.5%, 1: 51.5%, 5: 29%). However, when pairing model state 0 with any other model, the watermark accuracy goes down. For example, when paired with model state 10, the watermark accuracy is 16%. That is not the case for model states 1 and 5 however, which remain on the same level no matter which model they are paired with. One major difference between models 0 and 1/5 with this trigger set, is that 0 is (as always) fully watermarked, though the watermark is "forgotten" and relearned in the following rounds.

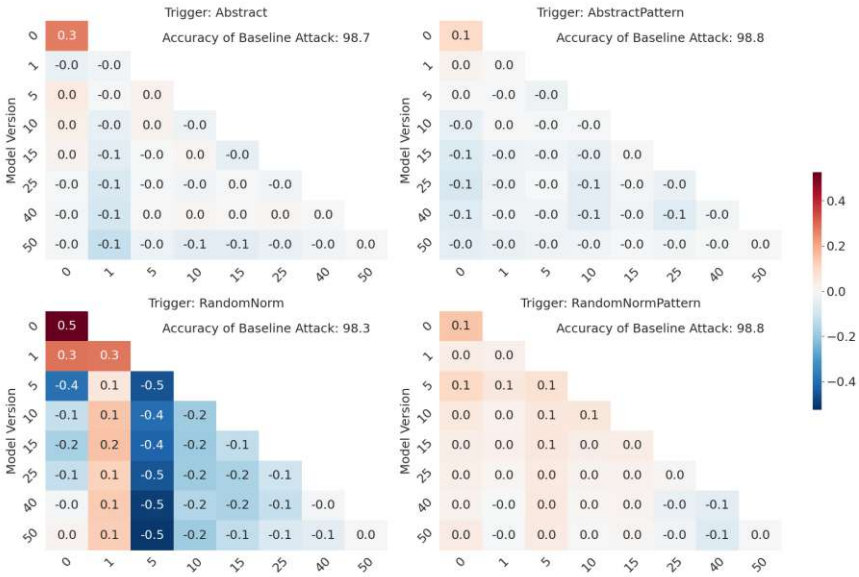
Figure 6.19 shows the numbers of successful inversion attacks per trigger set.

- Abstract: On this trigger set, there is a clear benefit from using intermediate model states. When the final model is used by itself, the watermark can only be removed from 1 out of 3 models. The only attack that performed worse was the single-state attack with model state 0. For the vast majority of attacks, the watermark is removed from at least 2 models. 2 attacks (i.e., combinations of model states) manage to remove the watermark from all 3 models.

- AbstractPattern: Only multi-state attacks that include model state 0 are successful on this trigger set. They are successful on 2 runs at most.



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.18: Loss-based multi-state inversion and single-state inversion with different IID MNIST model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

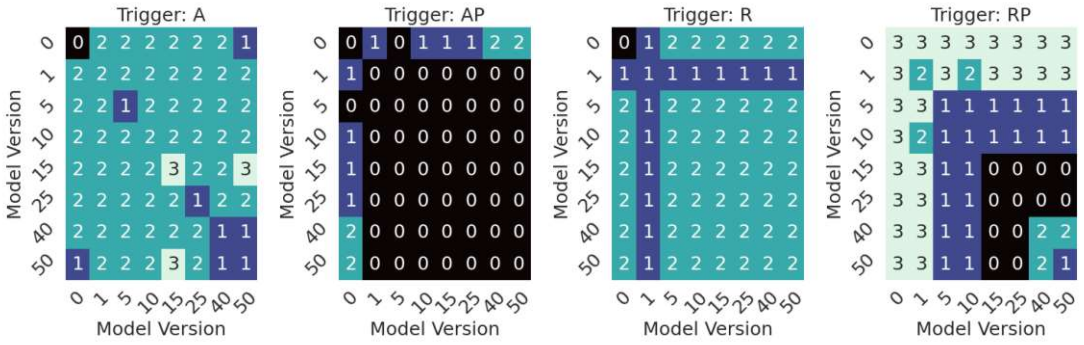
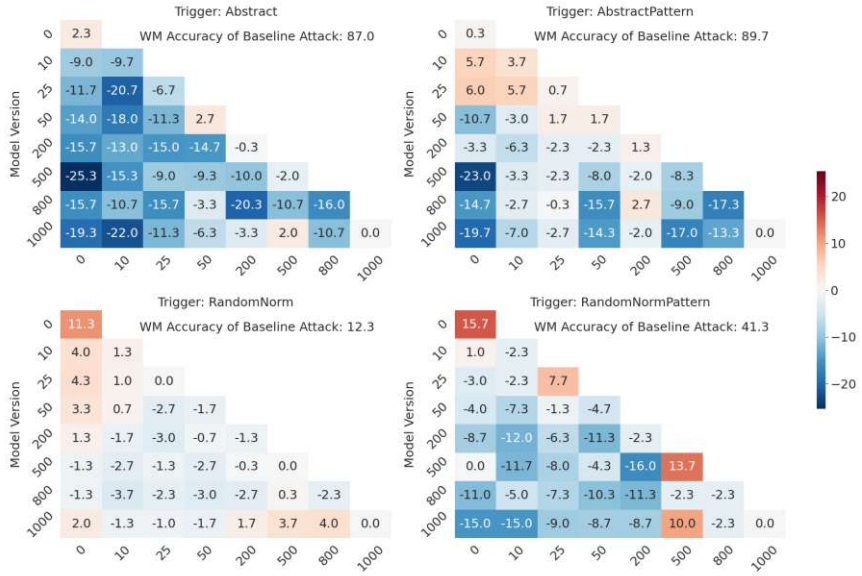


Figure 6.19: Counts of successful inversion attacks on 3 IID MNIST models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

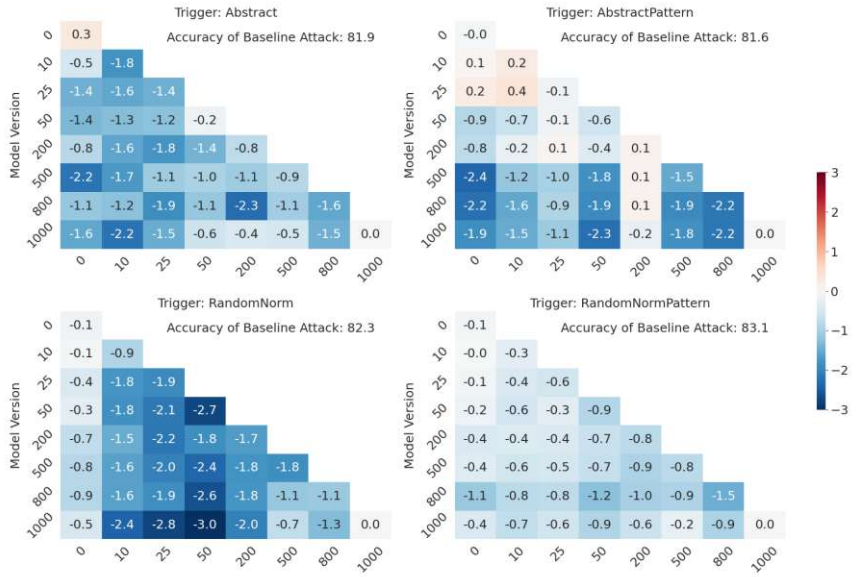
- Random: All attack sets except the single-state attack with state 0 and sets that include model state 1 are successful on both successful runs (1 training run failed and is not included in the attacks)
- RandomPattern: On this trigger set, attacks that include very early model states (0 or 1) nearly all succeed in 3 out of 3 cases. The effectiveness decreases with attacks that use only later states (5 and 10: 1 out of 3, 15 and 25: 0). With very late models, the effectiveness is higher again (40: 2, 50: 1).

Sever-Side Watermarks on non-IID CIFAR-10 The results are shown in Figure 6.20. They reveal that the number of multi-state attack sets, where the watermark accuracy is below at least 1 of the corresponding single-state attack sets, is 27 for Abstract, 23 for both AbstractPattern and Random, and 28 for RandomPattern. The respective numbers of attack sets where the multi-state attacks achieve a lower watermark accuracy than either single-state attack are 20, 13, 12, and 20. In addition, the attacks that reduce the watermark accuracy the most per trigger set are all multi-state attacks. Figure 6.21 shows the number of successful attacks per combination of models and trigger set. For the trigger sets Abstract and AbstractPattern, nearly all or all attacks fail because the watermark accuracy remains too high. In contrast, for the trigger set Random, every single attack is successful. This means that there is a clear advantage in terms of attack success from using intermediate model states only for RandomPattern: the baseline attack succeeds twice, as do most single-state attacks. The majority of multi-state attacks succeed 3 times.

Sever-Side Watermarks on non-IID MNIST The inversion results are shown in Figure 6.22. The results reveal that the number of multi-state attacks where the average watermark accuracy is below at least one of the corresponding single-state attack sets is 26 (Abstract), 28 (AbstractPattern), 28 (Random), and 25 (RandomPattern) out of



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.20: Loss-based multi-state inversion and single-state inversion non-IID CIFAR-10 model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

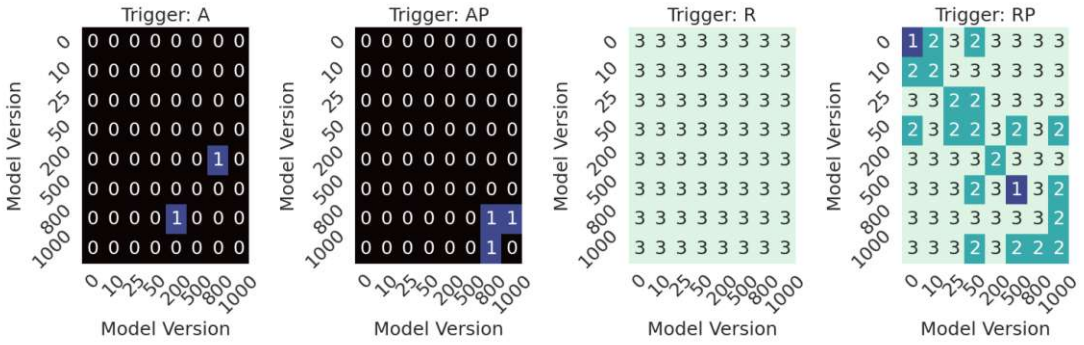


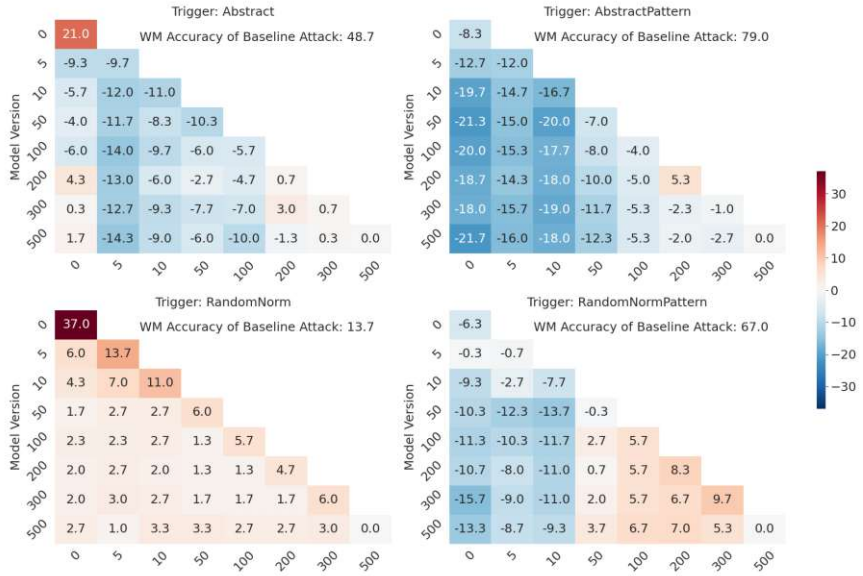
Figure 6.21: Counts of successful inversion attacks on 3 non-IID CIFAR-10 models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

28. The respective numbers of attack sets where the multi-state attacks achieve a lower watermark accuracy than either single-state attack are 11, 27, 21, and 17. Multi-state attacks are again effective on trigger sets Abstract, AbstractPattern, and RandomPattern, particularly with early model states (0 - 10). Especially on RandomPattern, there is a clear split: attacks that involve model states ≤ 10 considerably reduce the watermark accuracy more than the baseline attack, and attacks that do not include them either do not further reduce it or reduce it by a fairly small amount. The attacks with the lowest average watermark accuracy are:

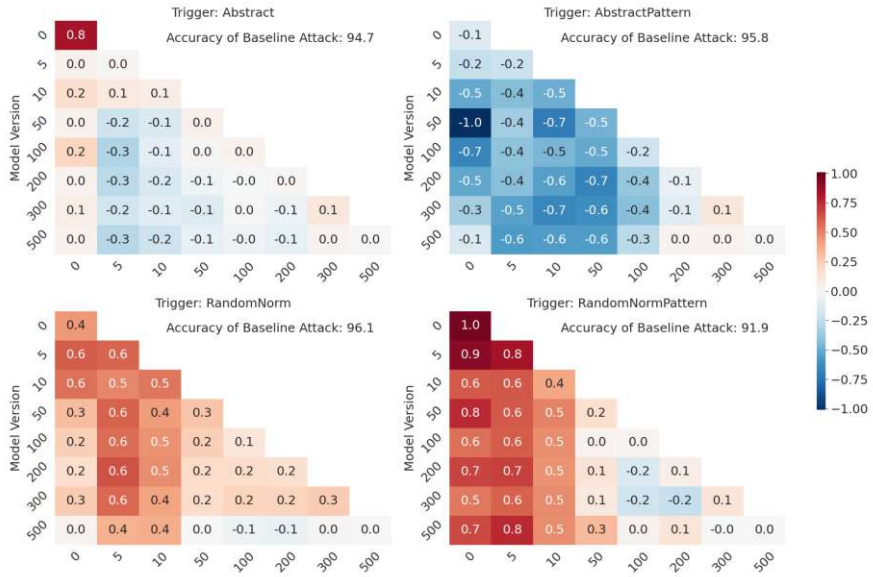
- Abstract: Multi-State Attack with models 5 and 500 (3.3 percentage points lower than best single model attack)
- AbstractPattern: Multi-State Attack with models 0 and 500 (5 percentage points lower than best single model attack)
- RandomPattern: Multi-State Attack with models 0 and 300 (8 percentage points lower than best single model attack)
- Random: The baseline attack is most effective in reducing the watermark accuracy, but all multi-state attacks are very effective as well.

The best attack sets have in common that they include a very early model state (0 or 5) and a very late model state (300 or 500). Figure 6.23 shows the number of successful attacks for different model states. Notable observations are:

- Abstract: 2 out of 3 removals are successful with the baseline attack and with single-state attacks involving late model states (100+). Single-state attacks using the states 5-50 succeed 3 out of 3 times. The results of multi-state attacks are mixed, with some succeeding 1, some 2, and some 3 times.



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.22: Loss-based multi-state inversion and single-state inversion with different non-IID MNIST model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

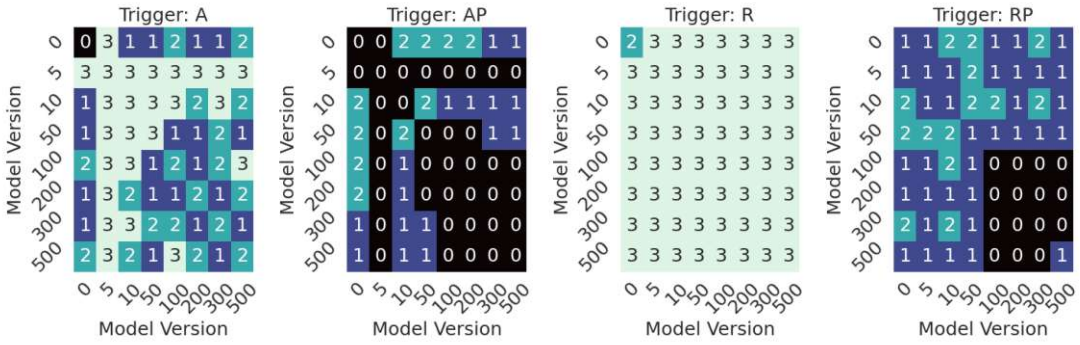


Figure 6.23: Counts of successful inversion attacks on 3 non-IID MNIST models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

- AbstractPattern: No single-state attack succeeds on this trigger set; however, multiple multi-state attacks do, on either 1 or 2 runs. Model states 0 and 10 are most often included in successful runs.
- Random: This trigger set is the least robust – all but 1 attack set succeed 3 out of 3 times.
- RandomPattern: As with AbstractPattern, no single-state attack is successful. However, with all attacks that involve at least 1 model state ≤ 50, the attack succeeds at least once.

Output-Based Multi-State Inversion

In this section, we discuss the results of output-based multi-state attacks on all training runs with Waffle-watermarking (except 1 run that failed due to exploding gradients). In these attacks, the inversion algorithm is applied to 2 model states independently. Their outputs are then combined and used for 'unlearning' on the final model state. The final model state can be (but does not have to be) 1 of the 2 used for inversion.

Sever-Side Watermarks on IID CIFAR-10 Figure 6.24 shows the watermark accuracy and test set accuracy of the output-based multi-state inversion attacks. The results show the following trends:

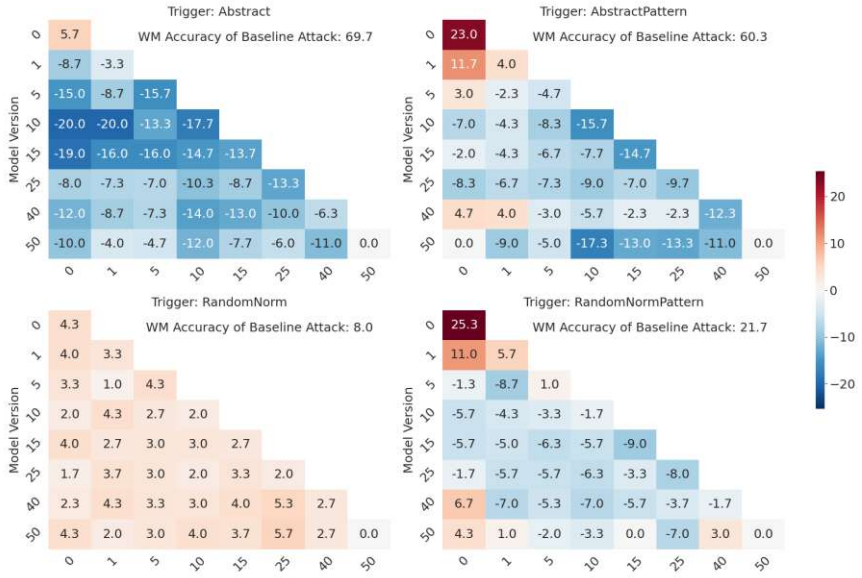
- On trigger sets Abstract, AbstractPattern, and RandomPattern, most multi-state attacks, excluding ones with model state 0 on AbstractPattern and RandomPattern, tend to lead to a lower watermark accuracy than the baseline attack. In addition, most decrease the test set accuracy more than the baseline attack, with 3 exceptions in Abstract, 1 in AbstractPattern, and 1 in RandomPattern.

- The number of output-based multi-state attacks that perform better than at least 1 of the 2 corresponding single-state attacks is 24 for Abstract, 21 for AbstractPattern, 14 for Random, and 25 for RandomPattern. Out of those, 11, 4, 5, and 10 are better than both single-state attacks. This means that for Abstract, AbstractPattern, and Random, there are fewer output-based than loss-based multi-state attacks that outperform both corresponding single-state attacks.

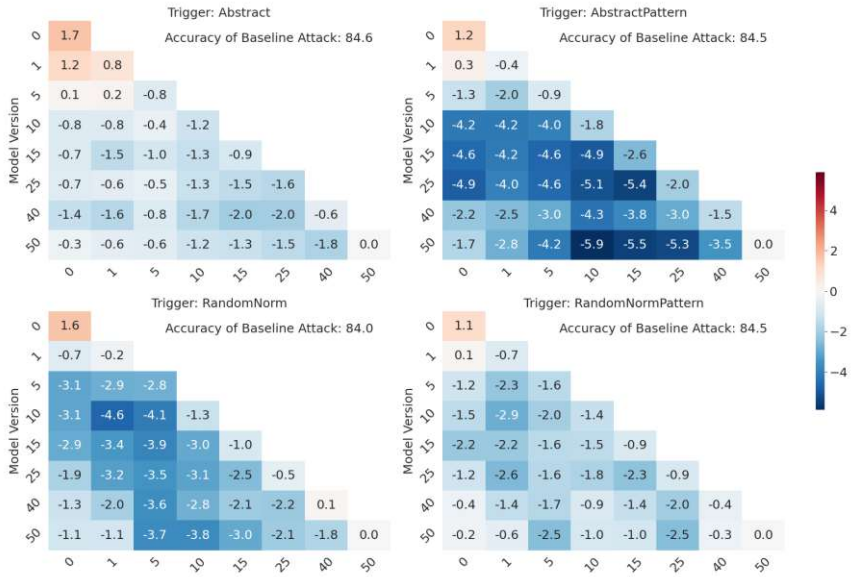
- Comparing the watermark accuracy after output-based attacks with their loss-based counterparts, the results show that, on average, the difference is 3.34 percentage points for Abstract, 0.45 for AbstractPattern, 0.92 for Random, and -2.73 for RandomPattern. In terms of test set accuracy, the differences are 0.38, -1.10, -1.52, and -0.37. Furthermore, the difference between the lowest watermark accuracy achieved is 6.6 percentage points for Abstract, 1.67 for AbstractPattern, 0.33 for Random, and 0.66 for AbstractPattern. respectively.

Overall, the attack tends to lead to higher watermark accuracy and lower test set accuracy compared to the loss-based attacks. However, the difference is relatively small, especially on trigger sets other than Abstract. Since the computational cost of this multi-state attack type is only half that of the loss-based alternative, the output-based attack might be considered advantageous.

Figure 6.25 shows the counts of successful attacks of the output-based multi-state inversion. Compared to the loss-based version, fewer attacks are successful on the trigger sets Abstract and AbstractPattern and the same, or a similar amount, on Random and RandomPattern.



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.24: Output-based multi-state inversion and single-state inversion with different IID CIFAR-10 model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

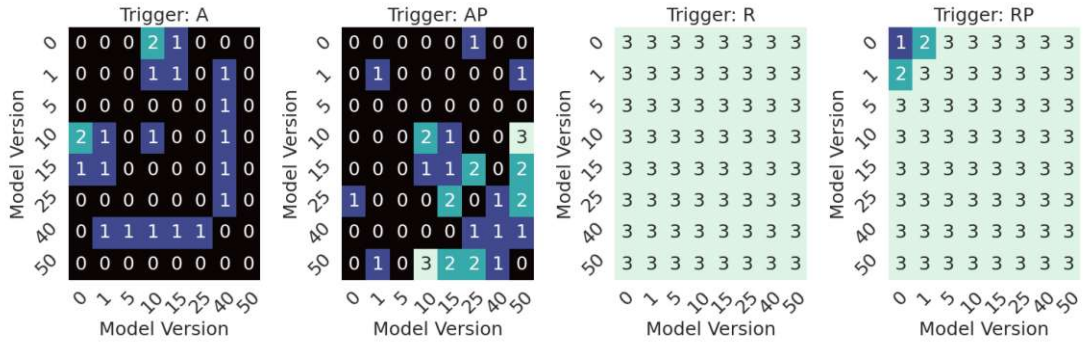


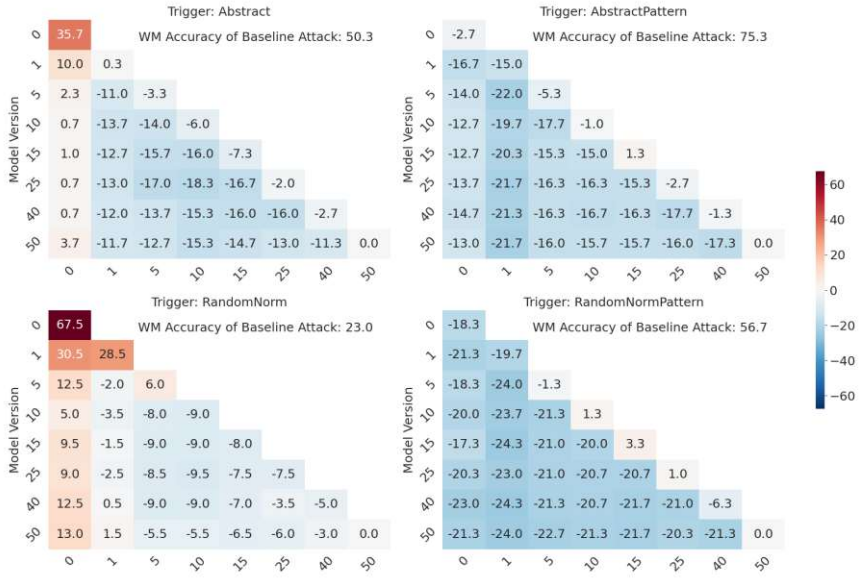
Figure 6.25: Counts of successful single-state and output-based multi-state inversion attacks on 3 IID CIFAR-10 models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Sever-Side Watermarks on IID MNIST Figure 6.26 shows the results of the output-based multi-state inversion attacks. We can note the following key observations:

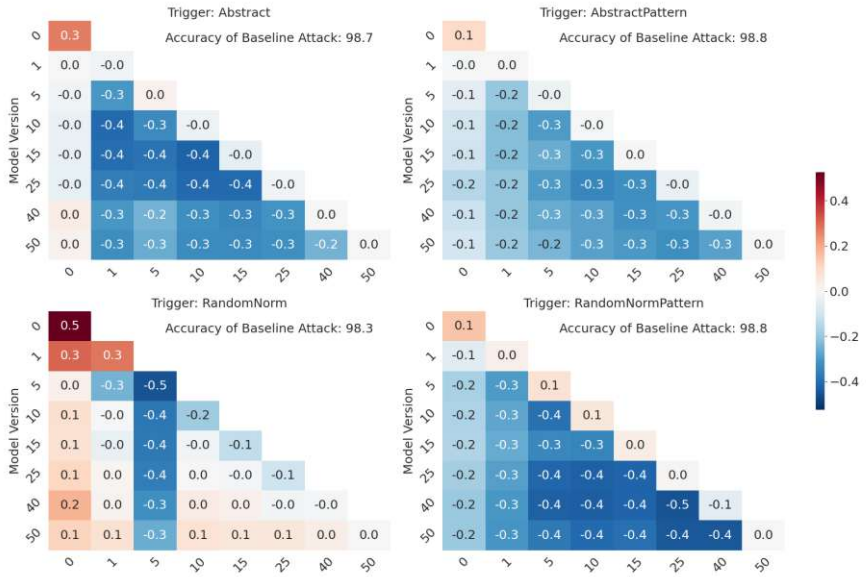
- All multi-state attacks, except ones that include model state 0 with Abstract and Random, lead to a lower watermark accuracy than the baseline attacks.
- In this setting, the counts of attacks that lead to a lower watermark accuracy than at least one corresponding single-state attack are 28 each on Abstract, AbstractPattern, and RandomPattern, and 26 on RandomPattern. Out of those, 21 (Abstract), 28 (AbstractPattern), 6 (Random), and 26 (RandomPattern) lead to a lower value than both corresponding single-state attacks. This is considerably more frequent than in the loss-based attack (where it was 10, 16, 6, and 13).
- The average difference between the loss-based and output-based attacks is -5.77 (Abstract), -6.76 (AbstractPattern), -4.14 (Random), and -11.05 (RandomPattern), while the absolute difference in test set accuracy is less than 0.5 percentage points. The difference in the lowest average watermark accuracy achieved with output-based or loss-based attacks is -10.67 percentage points for Abstract, 4 for AbstractPattern, -1 for Random, and 2.67 for RandomPattern.

As opposed to the experiments on the CIFAR-10 dataset, the output-based attacks seem clearly preferable for MNIST, as they outperform the single-state attacks more consistently and lead to lower watermark accuracy, while not affecting the test set accuracy by much. The differences in attack success (see Figure 6.27) are as follows:

- For Abstract, the attack now results in 3 out of 3 successful watermark removals for a larger number of model combinations. However, the success rate decreases when model state 0 is included.



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.26: Output-based multi-state inversion and single-state inversion with different IID MNIST model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

- For AbstractPattern, more combinations lead to 1 successful removal, but none achieve 2 successful removals (unlike the loss-based version, where 2 combinations resulted in 2 removals).
- For Random, the outcome is similar to the loss-based one, but model state 0 is less effective, while model state 1 shows increased success.
- For RandomPattern, there are fewer fully successful attacks with 3 removals, but unlike the loss-based version, all multi-state attacks manage to remove at least 2 watermarks.

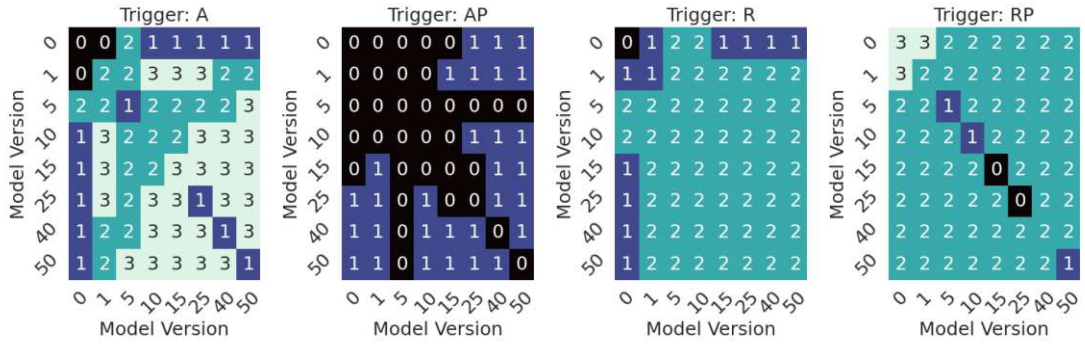
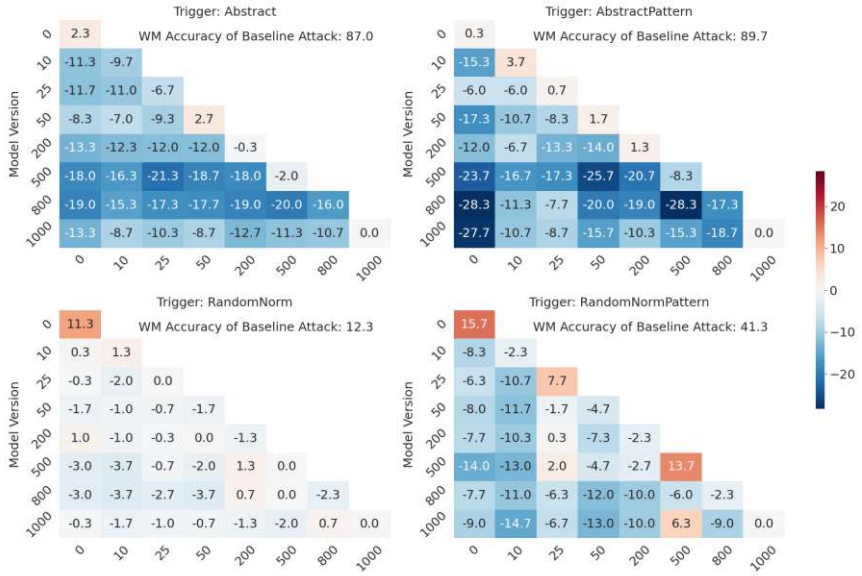


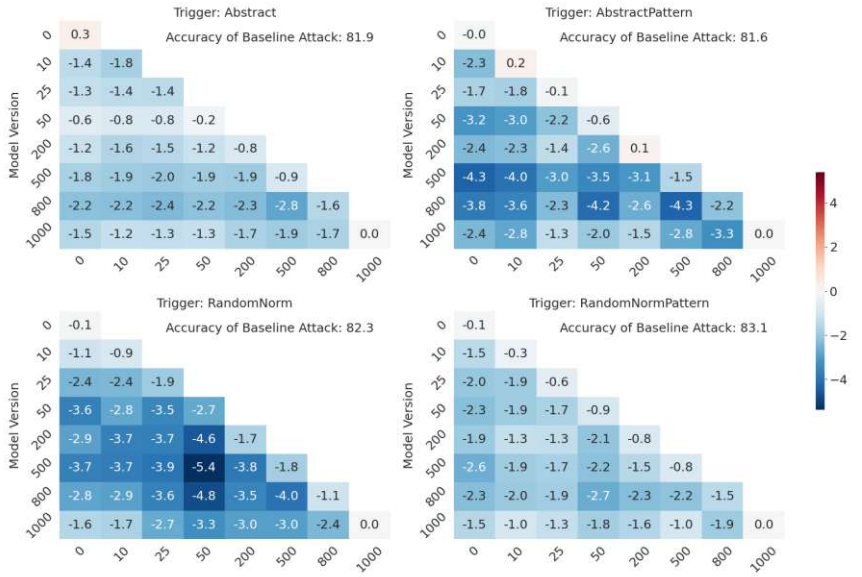
Figure 6.27: Counts of successful inversion attacks on 3 IID MNIST models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Sever-Side Watermarks on non-IID CIFAR-10 Figure 6.28 shows the watermark accuracy and test set accuracy when using output-based multi-state inversion on non-IID CIFAR-10.

- On non-IID CIFAR-10, all multi-state attacks with Abstract and AbstractPattern lead to a lower watermark accuracy than the baseline attack, all except 6 on Random and all except 2 on RandomPattern. All attack sets result in a lower test set accuracy than the baseline attack.
- The counts of attacks that result in a lower watermark accuracy than at least one corresponding single-state attack are 28 (Abstract, AbstractPattern, and RandomPattern) and 23 (Random). Out of those, 24 (Abstract), 26 (AbstractPattern), 15 (Random), and 24 (RandomPattern) are better than either single-state attack. This is again across the board more than in the loss-based attack (where it was 20, 13, 12, and 20).
- The average difference between the loss-based and output-based attacks is -0.9 (Abstract), -9.71 (AbstractPattern), -0.9 (Random), and -0.7(RandomPattern),



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.28: Output-based multi-state inversion and single-state inversion with different non-IID CIFAR-10 model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

while the difference in test set accuracy is -0.32, -1.78, -1.66, and -1.22. The difference in the lowest average watermark accuracy achieved with output-based / loss-based attacks is 4.0 percentage points for Abstract, -5.33 for AbstractPattern, 0 for Random, and 1.33 for RandomPattern.

Based on these results, the output-based attacks seem preferable in this setting as they tend to achieve similar, or in the case of AbstractPattern, lower watermark accuracy. However, the test set accuracy suffers more than with the loss-based attacks.

Figure 6.29 shows the success counts for this setting. The attack is never successful with Abstract, is very infrequently successful with AbstractPattern, always successful with Random and mostly successful with RandomPattern.

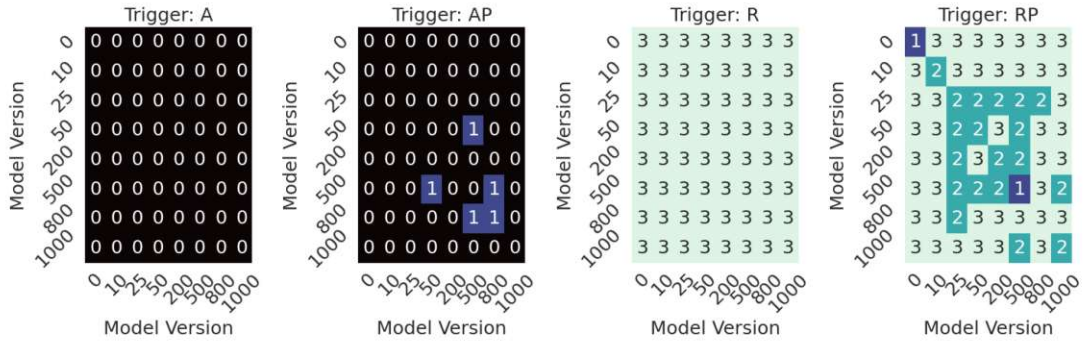
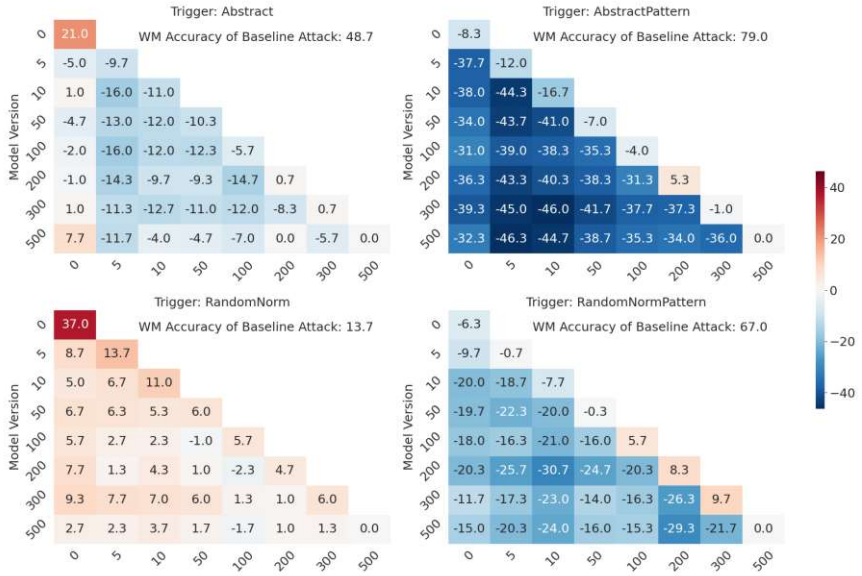


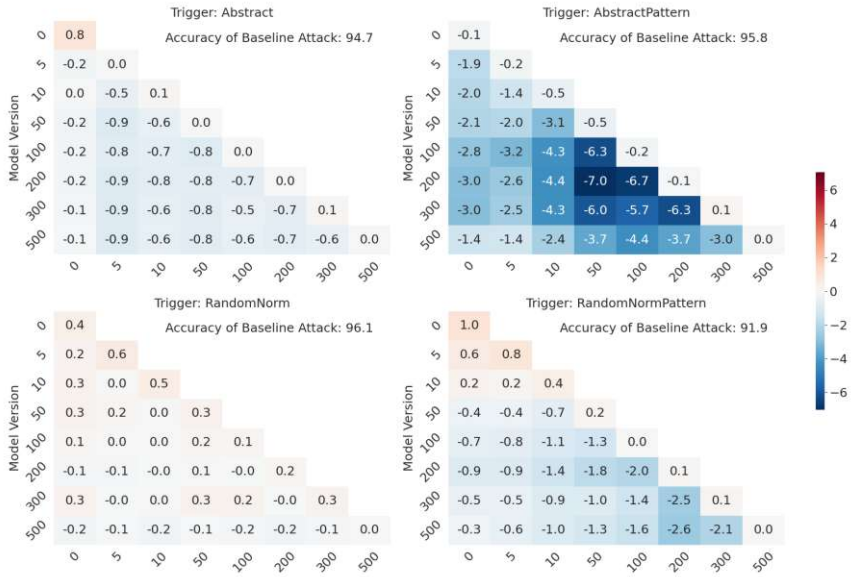
Figure 6.29: Counts of successful inversion attacks on 3 non-IID CIFAR-10 models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Sever-Side Watermarks on non-IID MNIST Figure 6.30a shows the results of the output-based multi-state inversion attacks on non-IID MNIST.

- There is only a single multi-state attack set overall that does not lead to lower watermark accuracy than at least one of its corresponding single-state attacks in this setting. The test set accuracy is lower than after the baseline attack in all multi-state attack sets.
- The counts of attack sets with lower watermark accuracy than both are 17 (Abstract), 28 (AbstractPattern and RandomPattern), 14. This is higher than in the loss-based attacks for all trigger sets, except R.
- The difference in watermark accuracy between output- and loss-based multi-state attacks is -1.43 (Abstract), -25.21 (AbstractPattern), 1.07 (Random), and -15.04 (RandomPattern), while the difference in test set accuracy is only -0.51, -3.13, -0.29,



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 6.30: Output-based multi-state inversion and single-state inversion with different non-IID MNIST model states that were trained using Waffle. The single-model inversion results are shown on the diagonal.

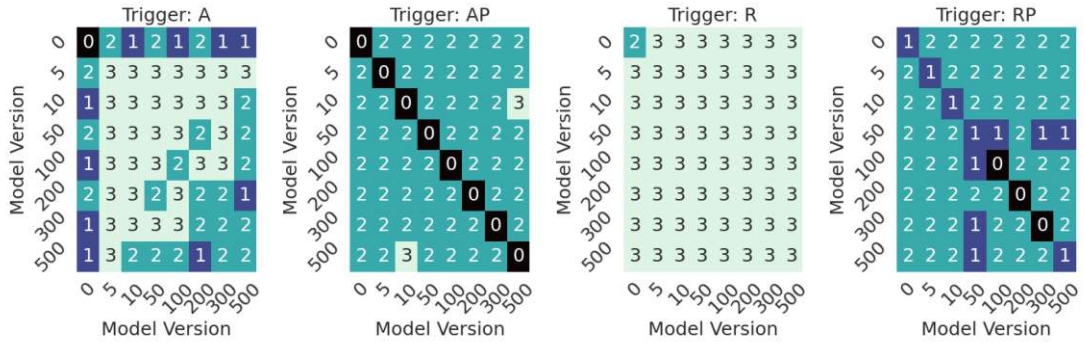


Figure 6.31: Counts of successful inversion attacks on 3 non-IID MNIST models, trained using Waffle. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

-1.38. The difference between the most effective attacks in terms of watermark removal is -1.67, -24.67, -3.3 and -15.00.

In this setting, we have the largest differences in terms of watermark accuracy to the loss-based attacks, specifically for the trigger sets with pattern. Here, the effectiveness of using multiple model-state is very clear, reducing the watermark accuracy by up to 46.3 percentage points more than the baseline attack and 29.6 compared to the best single-state attack. In terms of attack success, which is shown in Figure 6.31, the changes are as follows:

- For Abstract, there were more configurations with 2 instead of 1 removal or 3 instead of 2.
- For AbstractPattern, there are more configurations with 1 out of 3 successful removal. The only ones that were never successful did not include any model state below 50.
- For Random, the success counts are unchanged.
- For RandomPattern, most configurations lead to 1 out of 3 successful removals, which is better than with the loss-based version.

6.3.4 Summary

Table 6.9 shows from how many out of 3 models the watermarks were successfully removed with Pruning, WM-Diff, and the Baseline Inversion Attacks that use only the final model based on the model stealing criteria c_{evade} and c_{util} discussed in Section 6.1. From these results, a few observations can be drawn:

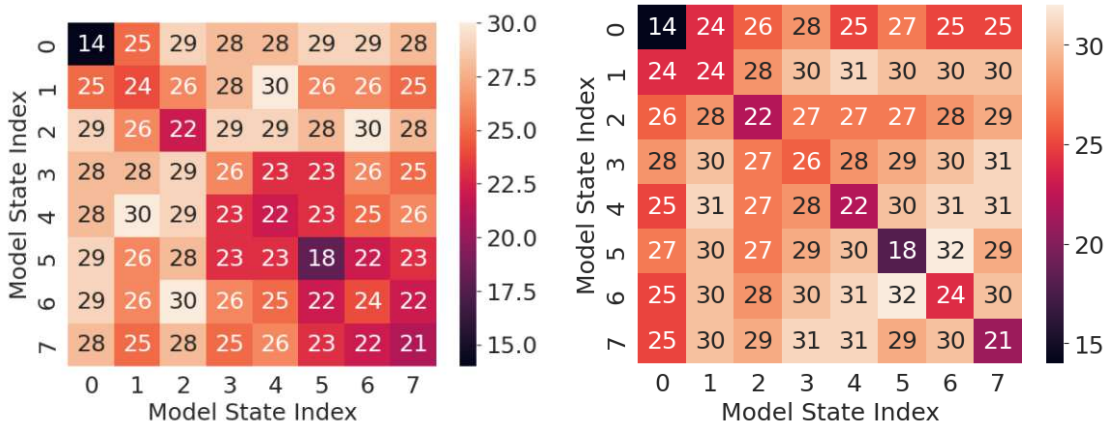
Table 6.9: Success counts of pruning, WM-DIFF, and inversion attacks in different settings. For each setting not marked by a *, the respective attack was performed 3 times. For those marked by a *, only 2 attacks were considered for the counts. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points.

Method	Dataset	Setting	Server				Client			
			A	AP	R	RP	A	AP	R	RP
Pruning	CIFAR-10	IID	-	-	3	2	-	-	2	-
		Non-IID	-	-	2	-	-	-	-	-
	MNIST	IID	3	2	2*	2	-	-	3	-
		Non-IID	-	-	-	-	-	-*	-	-
WM-Diff	CIFAR-10	IID	3	3	3	3	-	-	-	-
		Non-IID	3	3	3	3	-	-	-	-
	MNIST	IID	3	3	2*	3	-	-	-	-
		Non-IID	-	-	1	-	-	-	-	-
Inversion	CIFAR-10	IID	-	-	3	3	-	-	3	2
		Non-IID	-	-	3	2	-	-	3	-
	MNIST	IID	1	-	2*	1	-	-	3	-
		Non-IID	2	-	3	1	1	1*	3	-

- Firstly, the effectiveness of these attacks is highly dependent on the trigger set used, with Random being the least robust by far: on most attacks on models watermarked by the server, WM-DIFF on non-IID MNIST being the only exception, the attacks successfully removed the watermark from at least one model. The trigger set was particularly vulnerable with CIFAR-10 trained with Waffle, where all but 1 attack set succeeded 3 out of 3 times. In client-side watermarking, Random was also vulnerable, as only pruning attacks in the non-IID setting failed consistently. RandomPattern is the trigger set that leads to the second-highest number of successful attacks but is already quite robust when used with client-side watermarking, succeeding only with 2 out of 3 inversion attacks on IID CIFAR-10. The next, in terms of the number of successful attacks, is Abstract, followed by AbstractPattern. On CIFAR-10, AbstractPattern watermarks were only successfully removed with the WM-DIFF attack.

- Secondly, WM-DIFF stands out for its effectiveness, failing only on non-IID MNIST. In the other settings with server-side watermarking, it succeeded throughout. Of course, it is also the only attack that requires resources beyond the model states shared with clients (namely the model weights prior to watermarking).

- Thirdly, when comparing the robustness of client-side and server-side watermarking schemes to both pruning and inversion attacks, the client-side watermarking is more robust.



(a) Loss-based multi-state inversion

(b) Output-based multi-state inversion

Figure 6.32: Inversion success counts aggregated for all settings for server-side watermarking with Waffle. The total number of attacks per model state (combination) is 47. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

The effectiveness of using intermediate model states for inversion attacks depends on 2 key factors: (i) convergence on the primary task and (ii) the embedding of the watermark. (i) is best illustrated in Figure 6.18a (IID MNIST results), in the considerable increase in watermark accuracy from attacks that use model states 0 or 1 to attacks that only use model states *geq* 5. This jump likely occurs because, by round 5, the model's accuracy on the primary task has already surpassed 90%, and thus the inversion is less likely to result in images similar to the triggers. (ii) In the same setting, for the trigger set Random, attacks involving model states 1 and 5 still leave an unusually high watermark accuracy, because the watermark is not fully embedded yet for these model states.

An attacker is not likely to have any knowledge about the state of the watermark embedding and limited knowledge about the convergence speed on the primary task. For intermediate model states to be useful, there needs to be a consistent strategy of which models to use to increase the chances of attack success. Figure 6.32 shows the aggregated success count for each dataset, data distribution, and trigger set. The results were aggregated by grouping model states with the same index together e.g., index 7 is the final model, which is model state 50 in the IID setting, but model state 1,000 on non-IID CIFAR-10 (cf. Table 5.4). They show that only single-state attacks that use the model state 0 and single-state attacks that use the model state at index 5 (model state 25 in IID setting, 200 in non-IID MNIST 500 in non-IID CIFAR-10) lead to fewer successes than the baseline attack. Loss-based multi-state attacks are particularly successful when at least 1 early model state is used: for attacks that include model states at indices 0-2, the number of successful attacks ranges from 25 - 30, while for attacks that do not, it ranges from 22 - 26. In addition, using model state 0 together with any model state ≥ 3

leads to consistently high attack success (28-29).

For output-based multi-state inversion attacks, the pattern of which model states lead to the most successful attacks is quite different: model state 0 leads to the fewest successful attacks (24-28). Attacks that do not include this model state lead to 27-32 successful removals. Another noteworthy aspect is that output-based multi-state attacks that only use late model states (at index 4+) are surprisingly successful (29-32 removals), considering that the performance of these early models on the primary is already close to the final performance and also considering that the single-state attacks with model states at these indices are a lot less successful than these corresponding multi-state attacks (21-24 removals).

The output-based multi-state attack seems to be preferable to the loss-based one, as there is only 1 model state that leads to less than 27 successes, and the maximum of successes is 32 instead of 30. In addition, it involves only half the computational cost as for each inversion step, only 1 instead of 2 models needs to compute a prediction.

Since the pruning attack leads to only 16 successes in server-side watermarking, and baseline-inversion only to 21 successes, both types of multi-state attacks are clearly more effective.

CHAPTER 7

Dynamic Watermarking

This chapter discusses modifications to existing watermarking schemes aimed at enhancing the reliability of watermark embedding and improving robustness against the WM-Diff attack and inversion attacks that utilize intermediate models. The chapter is structured to first provide the motivation for the proposed modifications and then describe these modifications. Following this, the setup for the experiments is outlined. Finally, the effects of the modifications on training and on the watermark robustness are discussed based on the experimental results.

7.1 Improvements

In the previous chapters, we have demonstrated that

- In client-side schemes the watermark embedding can be too slow depending on the setting, and while boosting does help alleviate this problem, it does not always lead to a sufficient speed-up and can cause instability during training.
- In server-side schemes, the watermark is frequently fully embedded even before the first round of training. This can be exploited with the WM-Diff attack (if pre-watermarking weights are available) or with model inversion attacks that utilize intermediate models.

As both embedding the watermark too early and too late can be problematic, we propose i) a dynamic watermark accuracy threshold that changes throughout the training process based on the model performance on the primary task and ii) modifications to client-side watermarking that will improve that watermark embedding speed for training settings with fast convergence.

7.1.1 Dynamic Watermarking Threshold

In Waffle[TXMA21] and in this work, two stopping conditions for watermarking in-between FL rounds are used: a cap on the number of watermarking epochs t_{epoch} and a watermark accuracy threshold t_{wm}, i.e., the embedding is performed until one of these conditions is met. In our experiments (and in the Waffle experiments) a fairly high value is used for t_{epoch}, so that most of the time the other condition is satisfied first. In client-side watermarking, the trigger images are added to the normal training data of the watermarking client, without modifying its training configuration (including learning rate and number of epochs), meaning there is no specified goal in terms of watermarking.

When designing an increasing threshold, it should be considered that, depending on the primary task and model used, the convergence speed might vary widely. This means that simply defining a fixed rate at which the watermarking threshold should increase could easily lead to embedding the watermark too early or too late. For this reason, the watermarking threshold should ideally be based on the convergence speed on the primary task so as to reliably reach the desired watermarking goal – before the model becomes valuable to an attacker. There are different options as to how to determine the current performance of the model on the primary task:

- **Use Unseen Data**: For this, data needs to be acquired. This could be achieved by splitting the data of each client and using only a part for training and the rest for validation. The validation results of the individual clients could then be aggregated. The drawback of this is that less data is available for training. Alternatively, additional data could be gathered from another source and the evaluation could then take place on the server.

- **Reuse Training Data**: The estimate of the current performance does not need to be exact. The actual performance on unseen data is less relevant than whether the loss landscape has been shaped by another training task besides the watermarking, making inversion attacks more difficult. For this reason, we propose simply using the same data that is also used for training the model. For this, the clients need to compute the accuracy of the model on their local dataset. These scores can then be aggregated and form the basis for the watermarking threshold.

  Then, the question arises of when the evaluation on the training data should take place: directly after local training, evaluating on that same data, will lead to an overestimation of the performance, as it is likely to decrease through aggregation afterward. Furthermore, if the data is non-IID, that might further contribute to the problem, since the local task could be very different and possibly much easier than the global one. Another option would be to perform a regular round of FL, aggregate the model updates, and then perform another round of FL just for evaluation. This way, one can obtain the aggregated accuracy score before the next training round, so that during the next round, the watermarking threshold can be used. However, to prevent a communication overhead, we opt for a different option:

Table 7.1: Runtimes and number of dynamic watermarking experiments

		VGG-16		MNIST	
		IID	Non-IID	IID	Non-IID
Training		24 × 55m	24 × 16h	24 × 15m	24 × 2h
Attacks	Pruning	48 × 20m		48 × 6m	
	Inversion	1560 × 9-18m		1560 × 3-6m	

in our approach, the evaluation happens before local training and the aggregated accuracy is only used to compute the threshold for the next round and not the current one. This has the drawback of the watermark threshold lagging behind one round, but avoids an increase of the communication cost.

When an estimate of performance is available, a threshold can be set for when the model should be fully watermarked. As the model's accuracy on the primary task approaches this threshold, the watermarking threshold can be adjusted proportionally. Then, the dynamic watermarking threshold can be computed after each round:

$$t_{\text{dyn},i} = \min\left(\frac{\max(h_i)}{g} \cdot t_{wm},\, t_{wm}\right) \tag{7.1}$$

, where

- $t_{\text{dyn},i}$ is the dynamic threshold at round i.
- $h_i = [h_1, h_2, \ldots, h_i]$ represents the history vector of aggregated accuracy scores on the local datasets up to round i.
- g is the train set accuracy threshold; if any $h_i \geq g$, then $t_{\text{dyn},i}$ should be equal to t_{wm}.
- t_{wm} is the watermarking accuracy threshold.

7.1.2 Client-Watermarking

To ensure that the watermark can be reliably embedded in time, we propose that in addition to boosting of the updates, the training by the watermarking client should be performed for a variable number of epochs, as with server-side watermarking. In addition, if the model already reaches the watermarking threshold, then the model is only trained on the regular training data (without trigger images) and without boosting. This should help alleviate the stability issues introduced by boosting all updates of the watermarking client.

7.2 Experiment Setup

To evaluate these modifications, we use the same training and attack setup (datasets, parameters, trigger sets, number of repetitions) as in the previous experiments, described in Chapter 5, but with dynamic thresholding for server-side watermarking (without continual learning) and for client-side watermarking (with the best-performing boosting value for the setting). For both client-side and server-side watermarking, the parameters from the server-side watermarking experiments that had a static threshold are applied. Table 7.1 shows the counts and runtimes of the experiments.

g should ideally reflect the point where the model becomes useful to an attacker, so it is closely related to c_{util} (the utility criterion for model stealing, cf. Section 6.1). However, a rough estimate is acceptable; choosing a lower threshold increases the likelihood that the model will be sufficiently watermarked before c_{util} is met. Conversely, a lower threshold value makes the behavior more similar to static watermarking, as the model will quickly reach the specified accuracy. Considering these factors, we have selected a threshold of 75%.

In the client-side experiments, a boosting factor of 10 is employed for IID and non-IID CIFAR-10 as well as non-IID MNIST, while a boosting factor of 20 is used for IID MNIST.

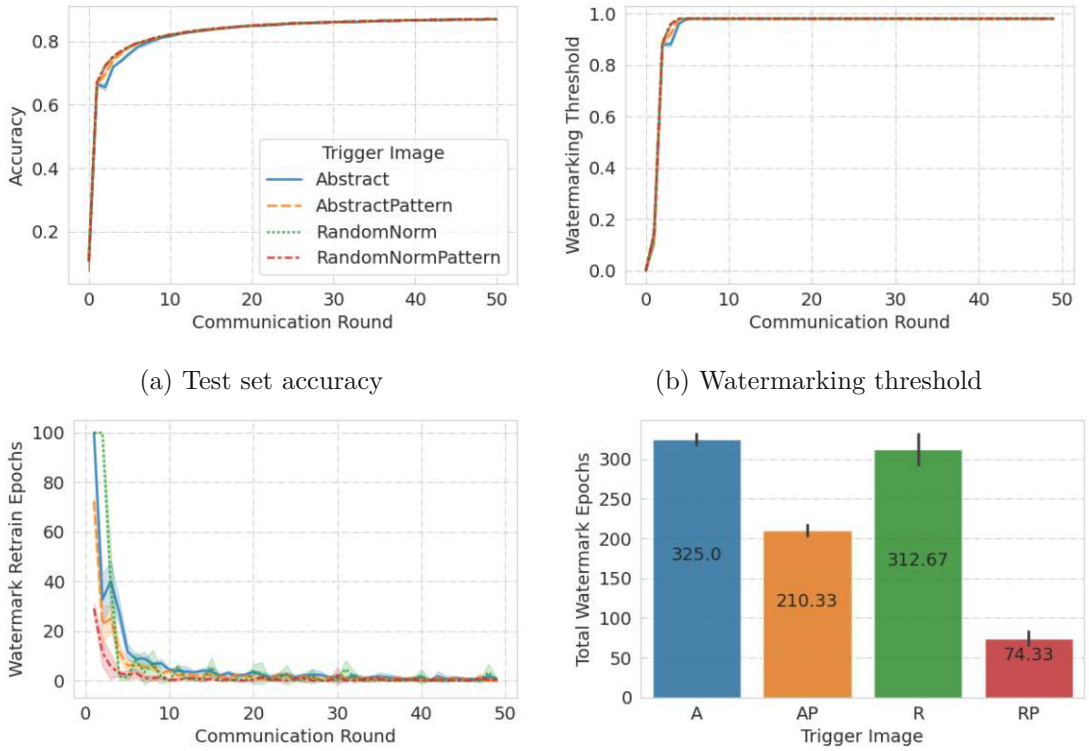
7.3 Training and Watermarking

In this section, the results of federated learning with dynamic watermarking are presented. Each experiment is repeated 3 times with different random seeds and the average test set accuracy and watermark accuracy and their standard deviation are reported in Table 7.2 for CIFAR-10, and in Table 7.3 for MNIST, respectively. They are also discussed for each scheme individually in the sections below.

7.3.1 Server-Side Watermarking

Our proposed server-side watermarking approach functions essentially the same as Waffle (cf. Section 3.1.1), but instead of using a fixed number as the stopping condition for watermarking in-between FL rounds, a dynamic threshold t_{dyn} is used. The results of this scheme are presented below.

IID CIFAR-10 Figure 7.1 shows the results of training on IID-CIFAR-10 with server-side watermarking using a dynamic watermarking threshold. In terms of the test set accuracy, there are no obvious differences compared to using a fixed watermarking threshold. However, in this setting, the number of watermarking epochs is higher on all trigger sets (by 30.70% with Abstract, 18.42% with AbstractPattern, 32.49% with Random, and 21.20% with RandomPattern). The dynamic threshold (t_{dyn}) per round is shown in Figure 7.1b. The number of rounds where the watermarking terminates because



(a) Test set accuracy

(b) Watermarking threshold

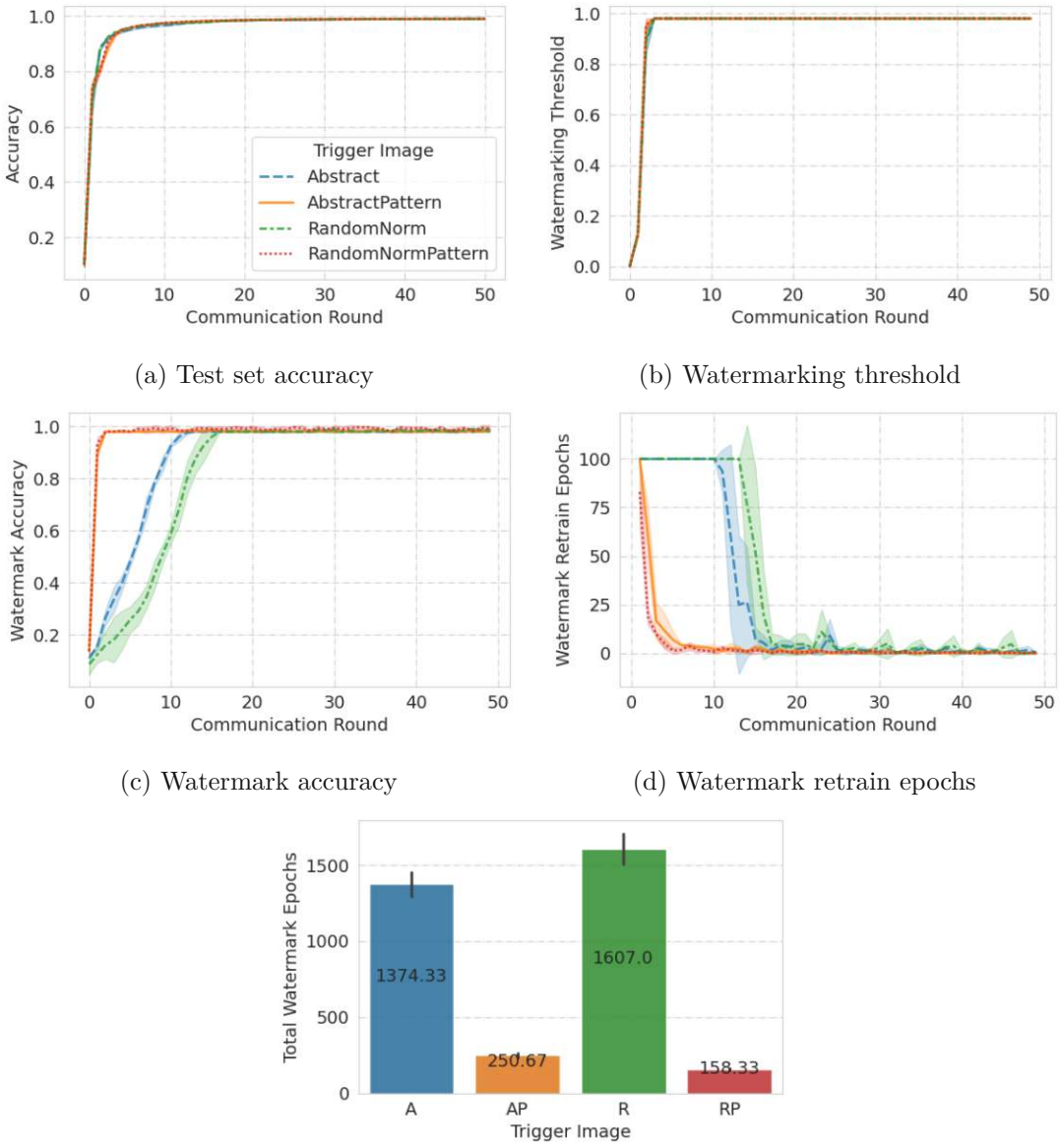
(c) Watermark retrain epochs

(d) Total watermark epochs

Figure 7.1: CIFAR-10 IID training results using server-side watermarking with dynamic threshold

the stopping condition t_{epoch} is met is 1 with Abstract and 2 with Random. Only model states 0, 1, and, in a run with the Random trigger set, the model state from round 2 satisfy c_{evade} (but none of them satisfies c_{util}). It is not surprising that t_{epoch} is met in this setting, but not with Waffle (cf. Section 6.2.1): in the prior experiments there was no limit of watermarking epochs during the initial watermarking (before the first round of FL), so in between rounds, less watermarking epochs needed to occur. As intended, the watermark accuracy is low while the test set accuracy is low as well, but increases in time to protect well-performing intermediate models (based on c_{evade} and c_{util}). For example, AbstractPattern has an average test set accuracy of 69.59% and a watermark accuracy of 88.67% after round 2, 74.14% and 93% after round 3, and 76.43% and 98% after round 4. It remains at $\geq 98\%$ for the rest of the training.

IID MNIST The dynamic watermarking results for MNIST with server-side watermarking are shown in Figure 7.2. As with static watermarking, dynamic watermarking does not affect the test set accuracy in our experiments, but the number of watermarking epochs is again higher than with a static threshold, namely by 106.05% (Abstract), 96.26%



(a) Test set accuracy

(b) Watermarking threshold

(c) Watermark accuracy

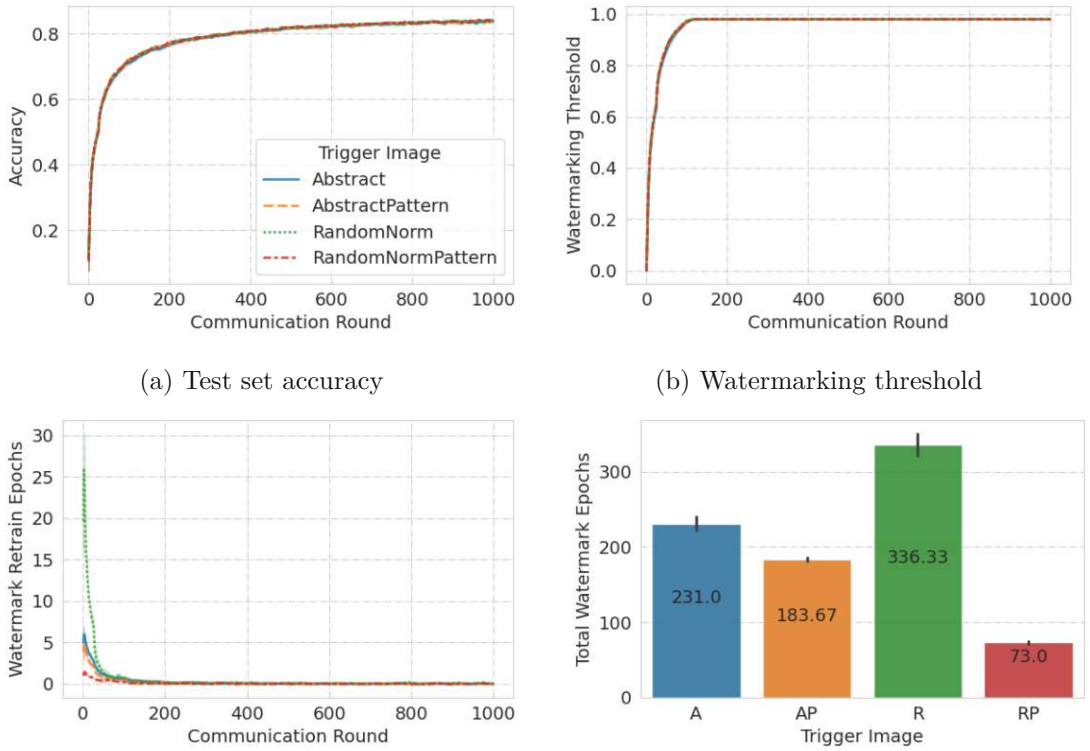
(d) Watermark retrain epochs

(e) Total watermark epochs

Figure 7.2: MNIST IID training results using server-side watermarking with dynamic threshold

(AbstractPattern), 2.52% (Random), and 152.64% (RandomPattern). The considerably larger increase compared to CIFAR-10, with all trigger sets besides Random, could be explained by the fact that on this dataset, in the static setting, a learning rate of 0.1 was used for the initial watermarking. In the dynamic watermarking experiments, a learning rate of 0.005 is used for all watermarking, as there is no initial watermarking. The reason to choose a lower learning rate for retraining is to reduce its negative impact on the primary task.

As shown in Figure 7.2b, the dynamic watermarking threshold t_{dyn} increases very quickly: After the first round, the train set accuracy used to determine t_{dyn} stems from the initial state of the model, prior to training (so it just depends on the initialization). Because of this, the threshold is, on average, only 12.80. The average test set accuracy is already between 69.07% and 74.40% at this time. However, after the second round, the threshold increases to 88.9% (Abstract), 92.13% (AbstractPattern), 89.71% (Random), and 95.52% (RandomPattern). After that, it is 98% for the rest of the training for all trigger sets. All runs from trigger sets Abstract, AbstractPattern, and Random, and 1 run from RandomPattern fail to reach t_{dyn} in at least 1 round because the stopping condition t_{epoch} is met first. On average, t_{epoch} is met in 0.33 rounds with RandomPattern (in 2 out of 3 runs it was never met), 1 round with AbstractPattern, 12 rounds with Abstract, and 14.67 rounds with Random. On the trigger sets AbstractPattern and RandomPattern, intermediate models can be considered protected, as none of them satisfy both c_{evade} and c_{util}. c_{evade} is only satisfied during one round, during which the test set accuracy is between 72.02% and 76.72%. With trigger set Abstract, there are 5 - 6 model states that satisfy c_{evade}, with 1 - 2 also satisfying c_{util}. With trigger set Random, there are 8 - 9 model states that satisfy c_{evade}, with 5 - 6 of them satisfying c_{util} as well. In the setting with static watermarking, only 2 model states satisfied both c_{evade} and c_{util} with the trigger set Random and none with Abstract, meaning especially for Abstract, the modifications reduced the reliability of the watermarking. For Abstract, this likely could have been avoided if the watermarking process had already begun during the first round of FL.



(a) Test set accuracy

(b) Watermarking threshold

(c) Watermark retrain epochs

(d) Total watermark epochs

Figure 7.3: CIFAR-10 non-IID training results using server-side watermarking with dynamic threshold

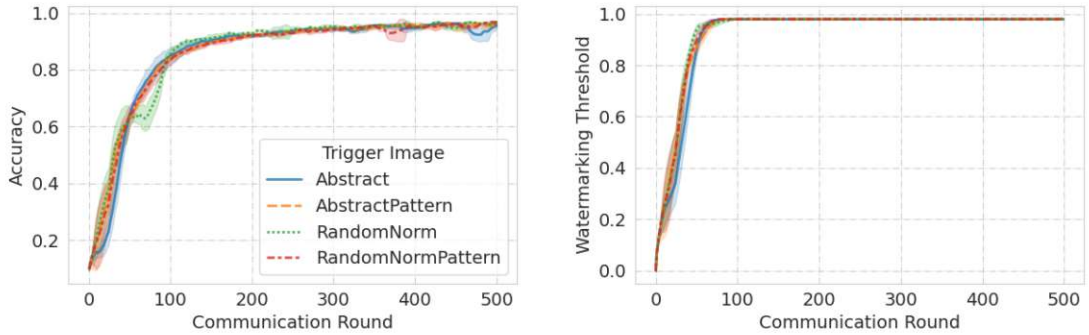
Line plots are smoothed using running average with window size 25

Non-IID CIFAR-10 The results for the non-IID training on CIFAR-10 with server-side dynamic watermarking are shown in Figure 7.3. Again, the test set accuracy is in the same range as with static watermarking and, in terms of watermarking epochs, the results are also similar:

- The number of watermarking epochs is close to the IID setting. The highest increase in watermarking epochs is 23.66 with Abstract.
- The average number of watermarking epochs per round is much lower than in the IID setting, with a maximum of 32 epochs with Random.

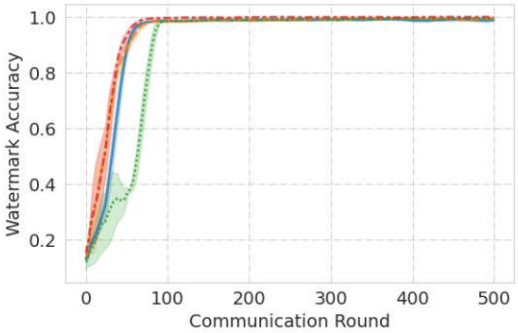
The watermark accuracy slowly increases throughout the first 72 - 96 rounds of training. Before it reaches t_{wm}, the test set accuracy is 72.82% at most.

Non-IID MNIST Figure 7.4 shows the results of server-side watermarking with dynamic thresholds with MNIST. Compared to the static watermarking, there are no notable

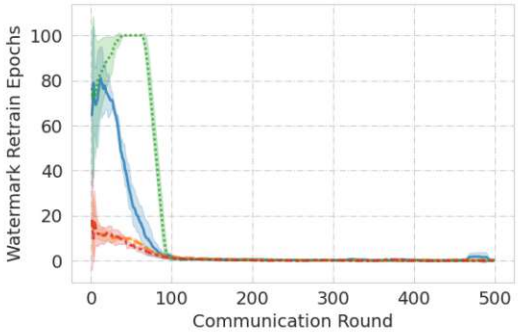


(a) Test set accuracy

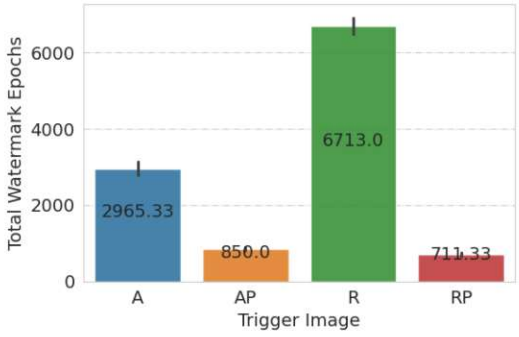
(b) Watermarking threshold



(c) Watermark accuracy



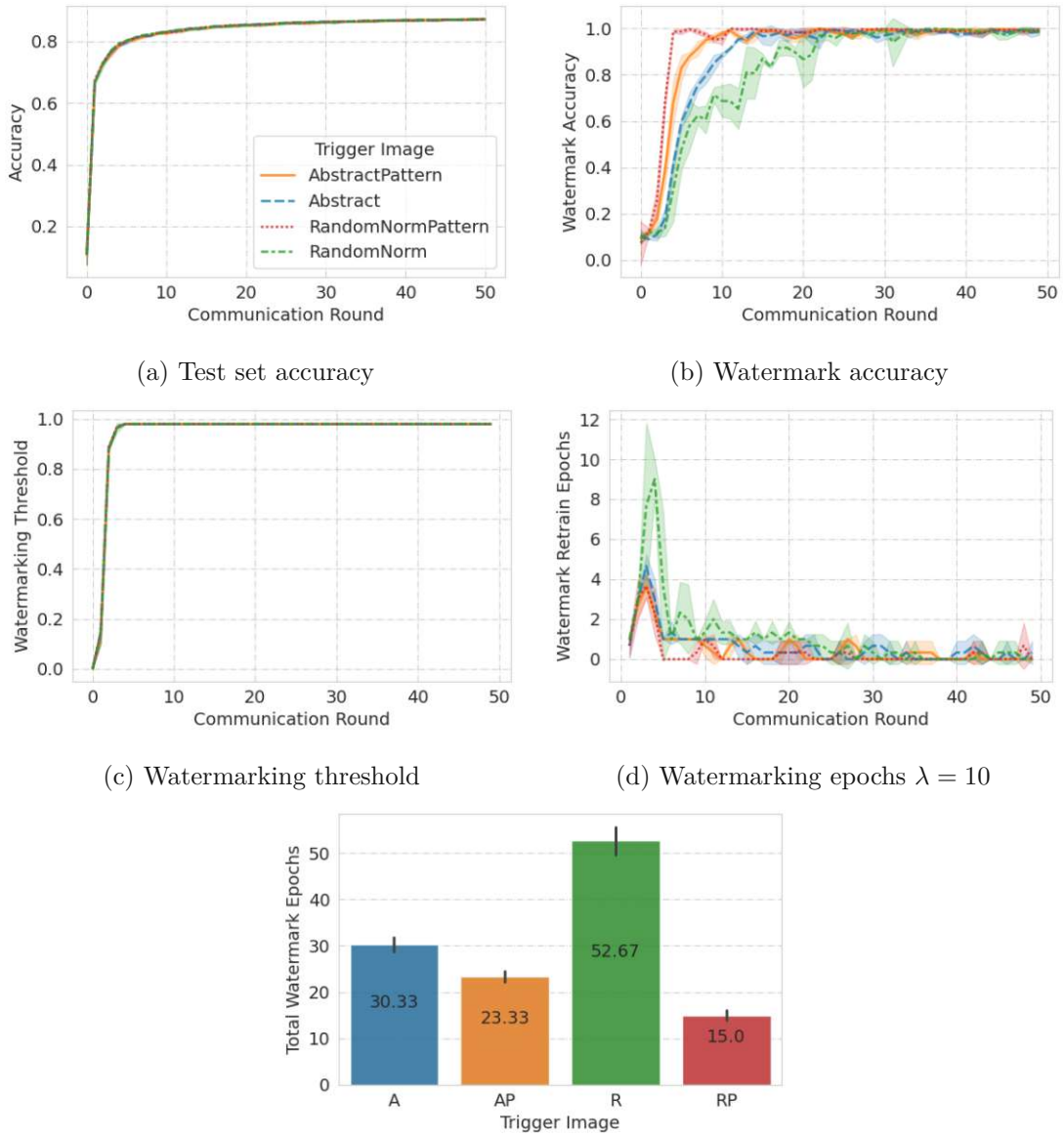
(d) Watermark retrain epochs



(e) Total watermark epochs

Figure 7.4: MNIST non-IID training results using server-side watermarking with the dynamic threshold. Line plots are smoothed using running average with window size 25

differences in terms of the final test set accuracy. A difference between the dynamic and the static watermarking results, is, again, the increase in watermarking epochs: 222.32% with Abstract, 251.72% with AbstractPattern, 196.25% with Random, and 492.78% with RandomPattern. With AbstractPattern the watermarking stopping condition t_{epoch} is never met, as a watermark accuracy $\geq t_{dyn}$ (the second stopping condition) can be consistently reached. That was also the case for 2 out of 3 RandomPattern runs, with the third run watermarking for t_{epoch} epochs during 1 round. In contrast, Abstract met t_{epoch} during 12 - 18 rounds, and Random during 56 - 65. However, the watermark is embedded on time, as the highest test set accuracy of model states satisfying c_{evade}, is 38.36% - 41.74% (Abstract), 35.48% - 49.96% (AbstractPattern), 73.19% - 76.12% (Random), and 35.23% - 56.77% (RandomPattern)%.



(a) Test set accuracy

(b) Watermark accuracy

(c) Watermarking threshold

(d) Watermarking epochs $\lambda = 10$

(e) Total watermarking epochs

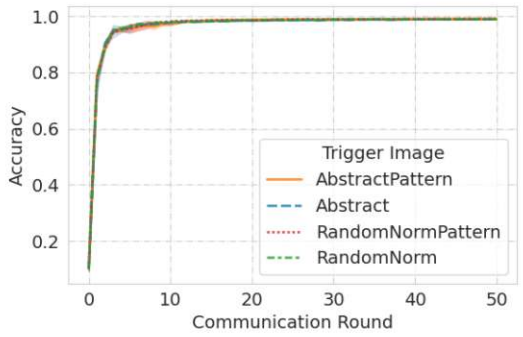
Figure 7.5: CIFAR-10 IID training results using client-side watermarking with $\lambda = 10$ and a dynamic threshold

7.3.2 Client-Side Watermarking

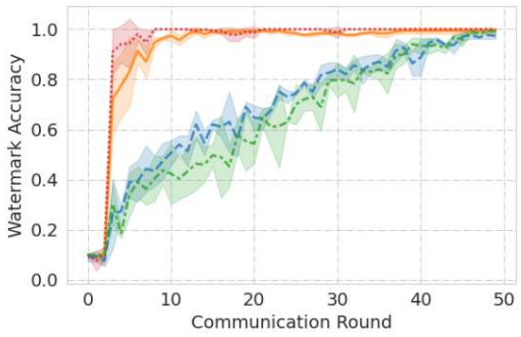
In our proposed client-side watermarking approach, the watermarking client adds the trigger set to their local data if the watermark accuracy falls below the dynamic watermarking threshold t_{dyn}. The results of this scheme are presented below.

IID CIFAR-10 In Figure 7.5, the results of client-side watermarking with dynamic watermarking threshold are shown on CIFAR-10. The test set accuracy is not notably affected by the change. The average number of watermarking epochs (15 - 53.67 in total) is a lot lower than with server-side watermarking, but that is expected, as the learning rate for the training on the client device is higher than the watermarking learning rate on the server (0.01 versus 0.0005). The number of watermarking epochs is below t_{epochs} each round, meaning locally the watermark is always embedded up to t_{dyn}. In spite of this, and the dynamic threshold is very close for all trigger sets (see Figure 7.7c), there are differences in watermark accuracy during training, with RandomPattern reaching an average watermark accuracy of 98% after round 5, AbstractPattern after round 12, Abstract after round 18 and Random after round 36. These differences occur because the watermark accuracy, which is initially sufficiently high on the client's local model, tends to decrease as all models are aggregated at the end of each FL round (i.e., the local model reaches t_{dyn}, but the global model does not). In the static client-watermarking experiments on the same data, only the AbstractPattern and RandomPattern trigger sets did not lead to model states that satisfy both c_{evade} and c_{util}. However, with the increased number of watermarking epochs per round, for all 4 trigger sets, it was possible to avoid model states that fulfill both model stealing criteria. There are 5 model states with Abstract, 4 with AbstractPattern, 5-6 with Random, and 3 with RandomPattern that satisfy c_{evade}, but they do not satisfy c_{util}: The highest test set accuracies of these model states are 79.09% - 80.28%, 78.32% - 79.25%, 79.89% - 81.48%, and 75.52% - 77.30%, respectively.

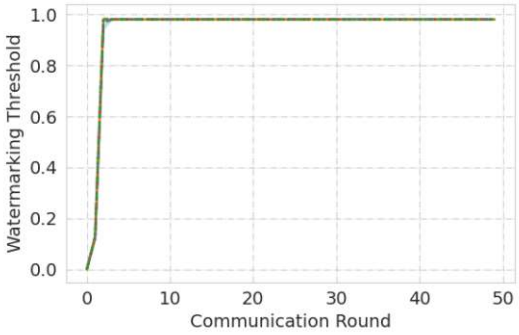
IID MNIST Figure 7.6 shows the results of dynamic watermarking in the client-side setting for IID MNIST. As before, i) the test set accuracy is unaffected, ii) the number of watermarking epochs is low compared to the server-side watermarking, with 92 total epochs at most, and iii) the dynamic watermarking threshold t_{dyn} is always reached locally. However, the impact of the aggregation on the watermark is quite large initially for all trigger sets. From round 3 onwards, the threshold, and accordingly the watermark accuracy locally, is 98%. However, after the aggregation in round 3, the watermark accuracy drops to below 11% with all trigger sets. The test set accuracy satisfies c_{util} from round 3 on, meaning the models from round 4 perform well on the primary task but are insufficiently watermarked across runs (based on c_{evade}). Because of these drops in watermark accuracy during aggregation, there are in total 8 - 10 (Abstract), 3 (AbstractPattern), 11 - 18 (Random), and 3 (RandomPattern) model states that satisfy c_{evade} (out of a total of 50). Of these, 6 - 7, 1, 9 - 16, and 0 - 1 also satisfy c_{util}, considerably fewer than with static watermarking.



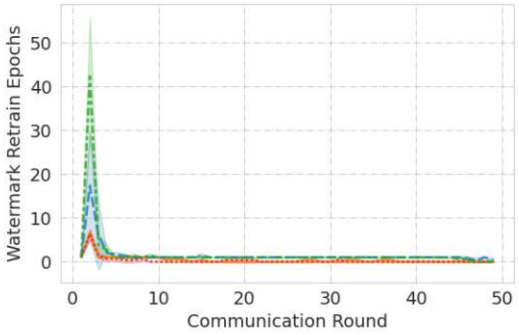
(a) Test set accuracy with $\lambda = 20$



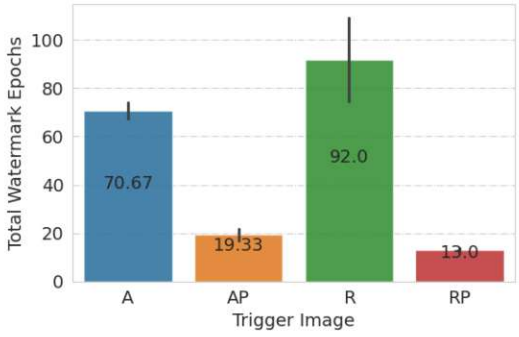
(b) Watermark accuracy with $\lambda = 20$



(c) Watermarking threshold

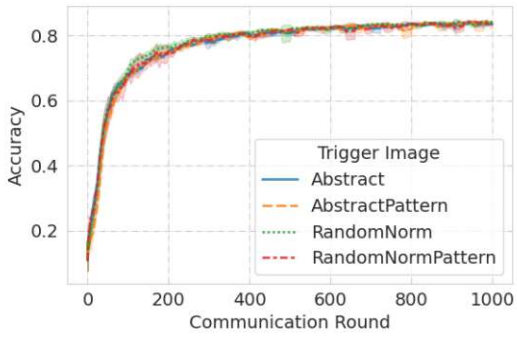


(d) Watermarking epochs

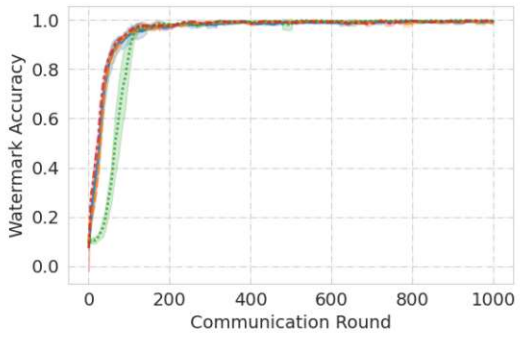


(e) Total watermarking epochs

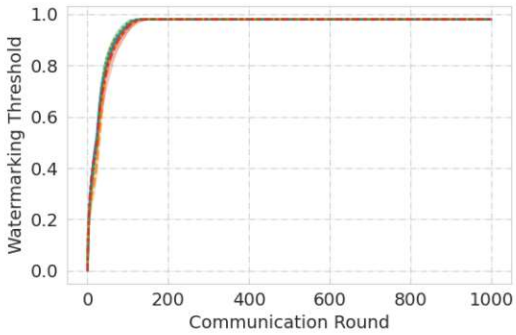
Figure 7.6: MNIST IID training results using client-side watermarking with $\lambda = 20$ and a dynamic threshold



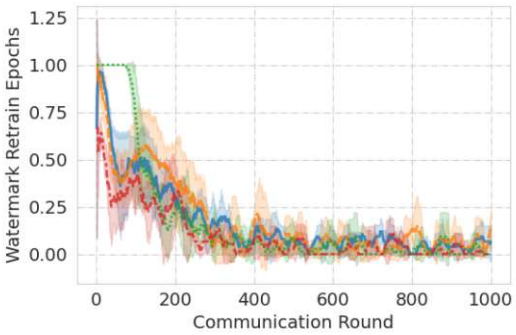
(a) Test set accuracy



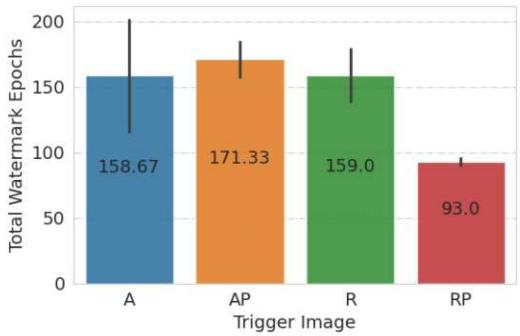
(b) Watermark accuracy



(c) Watermarking threshold



(d) Watermarking epochs



(e) Total watermarking epochs

Figure 7.7: CIFAR-10 non-IID training results using client-side watermarking with $\lambda = 10$ and a dynamic threshold
Line plots are smoothed using running average with window size 25

Non-IID CIFAR-10 Figure 7.7 shows the results of client-side dynamic watermarking on non-IID CIFAR-10. The test set accuracy is unchanged compared to the static watermarking. Compared to the IID setting, the number of watermarking epochs is much increased: by 423.15% for Abstract, 634.38% for AbstractPattern, 201.88% for Random, and 520.00% for RandomPattern. This is interesting because for the server-side

dynamic watermarking experiments, there is no considerable increase in watermarking epochs caused by the difference in data distribution. In addition, the difference between the trigger sets AbstractPattern (158.67), Abstract (171.33), and Random (159) are lower than in the other settings (with dynamic or static watermarking on CIFAR-10 models), and this is the only setting where AbstractPattern leads to a higher number of watermarking epochs than Abstract on average. The test set accuracy seems more stable (with fewer large drops throughout training) compared to the static client-side watermarking, likely because watermarking and boosting only occur when the watermark accuracy is below t_{dyn}. The watermark is embedded on time with all trigger sets: with the slowest of them, Random, the highest test set accuracy of any model state that satisfies t_{dyn} is 68.43%.

Non-IID MNIST Figure 7.8 shows the results of client-side dynamic watermarking on non-IID MNIST. As in the static experiments, the test set accuracy is quite unstable, with large performance drops throughout training. This time, one of the Random runs fails at approximately round 400, with both test set accuracy and watermark accuracy dropping to below 10% due to exploding gradients. In terms of watermarking epochs, there is a drastic increase from the IID setting, even when excluding the failed run: 433.93% (Abstract), 1,084.69% (AbstractPattern), 458.70% (Random), and 1,697.46% (RandomPattern).

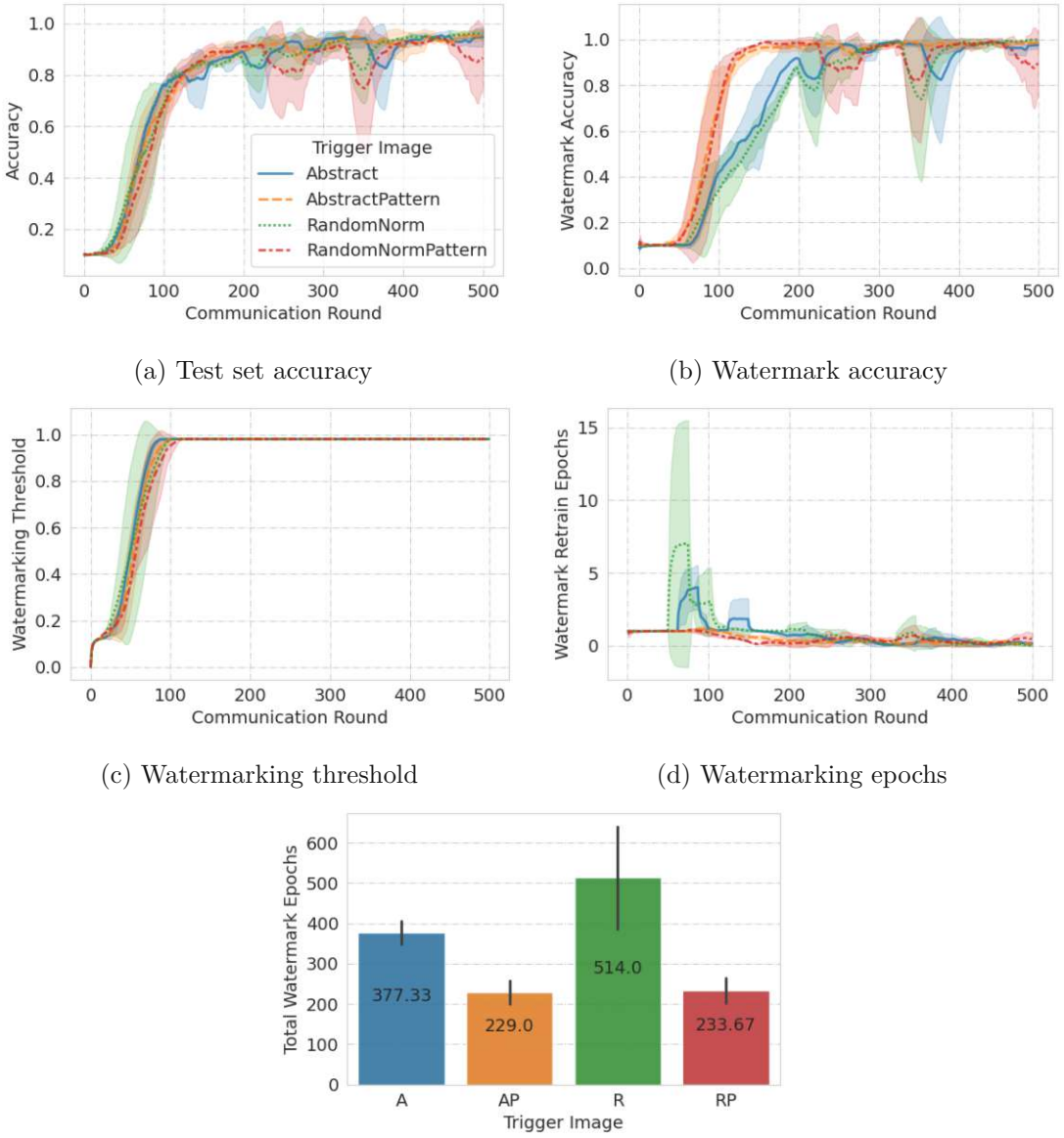
The watermark accuracy improves faster for trigger sets Abstract, Random, and RandomPattern, thanks to the variable number of watermarking epochs. The watermarking stopping condition t_{epoch} is met in a single Random run and only during 1 round. No model states satisfy both $c_{evade} and c_{util}$ i.e., all valuable models are protected. The highest test set accuracy of unprotected intermediate models is 82.66% - 90.52% (Abstract), 69.07% - 79.03% (AbstractPattern), 85.00% - 88.41% (Random), and 72.63% - 79.21% (RandomPattern) This means, in this setting, the watermark is embedded on time even with trigger sets for which this could not be achieved with static watermarking (Random, several Abstract, and AbstractPattern runs).

7.3.3 Summary

Table 7.2 shows the final performance of the CIFAR-10 models, while Table 7.3 shows the corresponding results for MNIST. These results are in the same range as the ones with a static watermarking threshold.

Protection of Intermediate Models

For server-side dynamic watermarking on CIFAR-10 and on non-IID MNIST, there are no changes in terms of the protection of valuable intermediate models, i.e., those that satisfy c_{util} (one of the model stealing criteria defined in Section 6.1). In these settings, the models were sufficiently watermarked with static watermarking and are still sufficiently watermarked with dynamic watermarking (meaning, those model states that satisfy c_{util},



(a) Test set accuracy

(b) Watermark accuracy

(c) Watermarking threshold

(d) Watermarking epochs

(e) Total watermarking epochs

Figure 7.8: MNIST non-IID training results using client-side watermarking client-side watermarking with $\lambda = 10$ and a dynamic threshold
Line plots are smoothed using running average with window size 25

Table 7.2: Results of training with a dynamic watermarking threshold on CIFAR-10 (test set accuracy, watermark accuracy and respective standard deviation)

		Accuracy		WM Accuracy	
		IID	Non-IID	IID	Non-IID
Trigger	Scheme				
A	Server	87.00 ± 0.09	82.05 ± 2.92	98.00 ± 0.00	99.33 ± 0.58
	Client ($\lambda = 10$)	87.26 ± 0.23	84.31 ± 0.62	98.67 ± 1.53	99.33 ± 0.58
AP	Server	87.08 ± 0.06	84.61 ± 0.50	98.33 ± 0.58	99.67 ± 0.58
	Client ($\lambda = 10$)	87.17 ± 0.18	84.30 ± 0.56	98.67 ± 0.58	99.67 ± 0.58
R	Server	87.09 ± 0.08	84.07 ± 0.39	99.33 ± 1.15	99.67 ± 0.58
	Client ($\lambda = 10$)	87.15 ± 0.18	84.53 ± 0.31	99.00 ± 1.00	100.00 ± 0.00
RP	Server	87.03 ± 0.10	84.90 ± 0.37	99.00 ± 0.00	99.33 ± 0.58
	Client ($\lambda = 10$)	87.09 ± 0.05	83.92 ± 0.54	99.67 ± 0.58	99.67 ± 0.58

Table 7.3: Results of training with a dynamic watermarking threshold on MNIST (test set accuracy, watermark accuracy, and respective standard deviation)

		Accuracy		WM Accuracy	
		IID	Non-IID	IID	Non-IID
Trigger	Scheme				
A	Server	99.06 ± 0.09	96.27 ± 0.84	98.00 ± 0.00	99.33 ± 1.15
	Client ($\lambda = 10/20$)	98.97 ± 0.11	95.88 ± 2.16	97.33 ± 1.15	98.67 ± 1.03
AP	Server	99.07 ± 0.03	96.72 ± 0.33	98.33 ± 0.58	99.67 ± 0.58
	Client ($\lambda = 10/20$)	99.05 ± 0.08	91.85 ± 11.31	99.33 ± 1.15	98.50 ± 1.87
R	Server	99.00 ± 0.07	97.09 ± 0.42	98.67 ± 0.58	99.67 ± 0.58
	Client ($\lambda = 10/20$)	99.00 ± 0.05	82.14 ± 35.45	99.33 ± 0.58	85.00 ± 36.74
RP	Server	99.02 ± 0.02	93.58 ± 6.20	99.33 ± 1.15	100.00 ± 0.00
	Client ($\lambda = 10/20$)	99.09 ± 0.04	94.29 ± 4.28	100.00 ± 0.00	99.33 ± 1.63

do not satisfy c_{evade}). On IID MNIST, the performance on the primary task increases rapidly in the first few rounds. In addition, the aggregation degrades the watermark accuracy quite strongly in this setting. Because of this, starting the watermarking after the second round instead of before the first (as in the static case) leads to the watermark being embedded too late with trigger sets Abstract and Random, leading to an average of 1.33 model states that satisfy both model stealing criteria with the former, and 5.67 with the latter.

Table 7.4 shows how many model states that fulfill the two attack success criteria are shared with clients on average with client-side watermarking. In this case, there are some clear improvements: Dynamic watermarking decreased the number of such model states in each setting where there were such model states to begin with. On CIFAR-10 and on non-IID MNIST, there are no model states that fulfill both model stealing criteria when

Table 7.4: Average count and standard deviation of model states in client-side watermarking with dynamic threshold, where the watermark accuracy is lower than 47% and the test set accuracy is within 5 percentage points of the final value

Dataset	Distribution	Trigger	Static	Dynamic
CIFAR-10	IID	A	1.00 ± 0.00	0.00 ± 0.00
		AP	0.00 ± 0.00	0.00 ± 0.00
		R	14.00 ± 1.00	0.00 ± 0.00
		RP	0.00 ± 0.00	0.00 ± 0.00
	Non-IID	A	0.33 ± 0.58	0.00 ± 0.00
		AP	0.00 ± 0.00	0.00 ± 0.00
		R	0.00 ± 0.00	0.00 ± 0.00
		RP	0.00 ± 0.00	0.00 ± 0.00
MNIST	IID	A	23.00 ± 2.00	6.67 ± 0.58
		AP	1.67 ± 1.53	1.00 ± 0.00
		R	39.00 ± 3.46	12.33 ± 3.51
		RP	2.33 ± 0.58	0.67 ± 0.58
	Non-IID	A	31.33 ± 35.57	0.00 ± 0.00
		AP	2.00 ± 2.65	0.00 ± 0.00
		R	76.67 ± 71.79	0.00 ± 0.00
		RP	0.00 ± 0.00	0.00 ± 0.00

using dynamic watermarking.

Computational Overhead

Table 7.6 shows the computational overhead in mini-batches (2 mini-batches per watermarking epoch) for server-side watermarking. It is reported in mini-batches and not epochs for easier comparison with the client-side watermarking (the reason will be addressed below). In Waffle, a lower learning rate is suggested during the FL process compared to watermarking prior to the first round. Since there is no initial watermarking in the dynamic watermarking version, the number of watermarking epochs increases for both datasets and data distributions: by up to 32.49% on IID CIFAR-10, up to 54.68% on non-IID CIFAR-10, up to 152.66% on IID MNIST, and up to 492.78% on non-IID MNIST.

For client-side watermarking, the same learning rate is used as for the primary task, since the datasets are combined in this case. This reduces the number of watermarking epochs compared to the server-side schemes. However, in our client-side watermarking approach, the mixed dataset (including the normal training data) is used in each watermarking epoch. This means that one additional watermarking epoch in the client-side watermarking increases the computational overhead by more than one epoch on the

server-side watermarking, where only the trigger set is used. Given that each client has 2,500 images for CIFAR-10 and 3,000 images for MNIST, and the trigger set consists of 100 images regardless of the dataset, with a batch size of 50, one epoch of training for the watermarking client involves 52 mini-batches for CIFAR-10 and 62 mini-batches for MNIST. Table 7.5 shows the number of additional mini-batches from watermarking in the basic scheme and with dynamic watermarking. In the IID setting, the average overhead with dynamic watermarking

- increases by between 230% and 1,255.33% on IID CIFAR-10
- decreases between 77.43% and 90.7% on non-IID CIFAR-10
- increases by between 266% and 2,884% on IID MNIST
- varies on non-IID MNIST: for trigger sets with the pattern, it decreases by up to 34.63%, and for trigger sets without, it increases by up to 1,322.8%

When comparing server-side and client-side dynamic watermarking, we can see the following differences in terms of the overhead:

- On IID CIFAR-10, the overhead is up to 41.10% higher for server-side watermarking, when trigger images with abstract background are used. For trigger sets with random background, it decreases by up to 57.11%.
- On non-IID CIFAR-10, the overhead is lower for server-side watermarking with RandomPattern (21.51%), but higher for the other trigger sets by up to 49.04%.
- On IID MNIST, the overhead is again lower with RandomPattern (13.48%) and higher with the other trigger sets (up to 78.33%)
- On non-IID MNIST, the overhead is higher for trigger sets with pattern (up to 160.07%), roughly the same with Abstract, and 5.64 percent lower with Random when using server-side watermarking.

Table 7.5: Client-side watermarking overhead in mini-batches

Dataset	Distribution	Trigger	Batches Static	Batches Dynamic
CIFAR-10	IID	A	100.00	460.67 ± 3.06
		AP	100.00	330.00 ± 27.78
		R	100.00	1,355.33 ± 138.34
		RP	100.00	346.67 ± 27.15
	non-IID	A	2,000.00	317.33 ± 86.03
		AP	2,000.00	342.67 ± 26.86
		R	2,000.00	451.33 ± 195.31
		RP	2,000.00	186.00 ± 5.29
MNIST	IID	A	100.00	1,541.33 ± 249.50
		AP	100.00	378.67 ± 37.17
		R	100.00	2,984.00 ± 1,095.06
		RP	100.00	366.00 ± 36.39
	non-IID	A	1,000.00	5,931.67 ± 1,908.08
		AP	1,000.00	653.67 ± 327.83
		R	1,000.00	14,228.00 ± 8,909.55
		RP	1,000.00	675.67 ± 477.34

Table 7.6: Server-side watermarking overhead in mini-batches

Dataset	Distribution	Trigger	Static	Dynamic
CIFAR-10	IID	A	497.33 ± 5.03	650.00 ± 13.86
		AP	354.67 ± 8.33	420.67 ± 13.32
		R	472.00 ± 38.63	625.33 ± 39.21
		RP	122.67 ± 18.15	148.67 ± 17.01
	Non-IID	A	298.67 ± 39.31	462.00 ± 17.78
		AP	248.00 ± 31.24	367.33 ± 5.03
		R	472.00 ± 15.87	672.67 ± 29.48
		RP	129.33 ± 13.01	146.00 ± 3.46
MNIST	IID	A	1,334.00 ± 49.03	2,748.67 ± 162.00
		AP	255.33 ± 23.09	501.33 ± 30.02
		R	3,135.00 ± 428.51	3,214.00 ± 205.56
		RP	125.33 ± 45.00	316.67 ± 13.32
	Non-IID	A	1,840.00 ± 168.08	5,930.67 ± 348.98
		AP	483.33 ± 20.82	1,700.00 ± 108.13
		R	4,598.67 ± 2,729.18	13,426.00 ± 434.77
		RP	240.00 ± 31.18	1,422.67 ± 93.22

7.4 Attacks

In this section, the results of watermark removal on watermarks embedded through dynamic watermarking are discussed. The experimental setup is detailed in Section 7.2. First, the results of the pruning attack, and following that, the results of the inversion attack are presented. Attacks are performed on (i) all dynamic server-side watermarking runs and (ii) all dynamic client-side watermarking runs. As in the prior attacks on static watermarking (cf. Section 6.3), the client-side watermarking runs are only targeted by attacks that do not use intermediate model states. In contrast to the static experiments, no WM-Diff attacks are performed because in dynamic watermarking the prerequisites (suitable model states) for the attack are not met.

The counts of successful watermark removals based on the model stealing criteria defined in Section 6.1 for the attacks pruning and baseline inversion are provided in Table 7.11.

7.4.1 Pruning

Section 7.4.1 shows the pruning results for MNIST. For the highest pruning rate that does not decrease the test set accuracy by more than 5 percentage points (satisfying c_{util}), the results are described below. At a pruning rate of 95%, the drop of the test set accuracy caused by attacks on IID runs is below 4.5 percentage points. The corresponding watermark accuracies are 38.33% for Abstract, 42.67% for AbstractPattern, 40% for Random, and 38% for RandomPattern with server-side watermarking, and 59.33% for Abstract, 68% for AbstractPattern, 70.67% for Random, and 48.33% for RandomPattern with client-side watermarking. The attack succeeds in all but 1 run for server-side watermarking and in 1 AbstractPattern and RandomPattern run, respectively, with client-side watermarking (cf. Table 7.11). This is similar to the results for static watermarking, but there, some server-side Random and RandomPattern runs had failed. In the non-IID setting, none of the pruning attacks are successful. This was also the case for static watermarking on non-IID MNIST.

Table 7.8 shows the pruning results for CIFAR-10. With a pruning rate of 90%, the average test set accuracies are within 3.5 percentage points of the respective original value for all IID experiments. The corresponding watermark accuracies are 53.67% for Abstract, 68.67% for AbstractPattern, 32.67% for Random, and 38.33% for RandomPattern with server-side watermarking, and 57.67% for Abstract, 70.67% for AbstractPattern, 34.33% for Random, and 49% for RandomPattern with client-side watermarking. Only attacks on trigger sets Random and RandomPattern are ever successful – all of them for server-side watermarking and 3 out of 6 for client-side watermarking. With static watermarking, there were in total 2 fewer successful removals. In the non-IID setting, there are again no successful removals: the watermark accuracy begins to drop to below 47% (c_{evade}) at a pruning rate of 90%. However, for the runs where the watermark accuracy falls below the threshold at this pruning rate, the test set accuracy is already too low. At a pruning rate of 95%, the model stealing criterion c_{util} is not satisfied. In contrast, with static watermarking, attacks on 2 out of 3 server-side Random runs had succeeded.

Table 7.7: Pruning results for MNIST (test set accuracy, watermark accuracy, and respective standard deviation per pruning rate (PR))

			IID		Non-IID	
Trigger	Scheme	PR	Accuracy	WM Accuracy	Accuracy	WM Accuracy
A	Server	85	98.90 ± 0.04	81.00 ± 10.58	95.96 ± 0.81	88.33 ± 9.29
		90	98.74 ± 0.11	65.67 ± 15.95	95.03 ± 1.12	77.00 ± 11.14
		95	96.37 ± 1.41	38.33 ± 5.03	88.92 ± 2.67	34.33 ± 6.11
		97	79.04 ± 2.04	19.33 ± 5.13	56.75 ± 15.94	20.00 ± 2.65
	Client	85	98.85 ± 0.09	94.67 ± 2.52	95.91 ± 1.28	98.00 ± 1.00
	($\lambda = 10/20$)	90	98.64 ± 0.14	87.67 ± 3.79	94.61 ± 1.23	97.00 ± 1.73
		95	96.84 ± 0.73	59.33 ± 13.32	84.03 ± 3.06	76.33 ± 11.06
		97	84.76 ± 3.97	36.00 ± 12.12	61.62 ± 13.12	58.00 ± 16.09
AP	Server	85	98.95 ± 0.06	90.33 ± 2.08	95.94 ± 0.09	96.67 ± 2.08
		90	98.79 ± 0.03	78.33 ± 5.03	95.43 ± 0.23	94.33 ± 4.16
		95	96.99 ± 0.54	43.67 ± 9.07	89.28 ± 2.68	76.33 ± 8.08
		97	81.47 ± 1.20	23.00 ± 3.46	45.78 ± 4.93	26.67 ± 7.64
	Client	85	98.93 ± 0.06	98.33 ± 2.08	95.97 ± 1.48	99.00 ± 1.00
	($\lambda = 10/20$)	90	98.80 ± 0.12	96.33 ± 2.52	95.56 ± 1.49	97.67 ± 2.08
		95	97.30 ± 0.35	68.00 ± 7.21	90.79 ± 2.26	94.33 ± 7.23
		97	85.84 ± 6.06	45.33 ± 9.29	69.65 ± 10.59	78.67 ± 28.36
R	Server	85	98.88 ± 0.08	89.00 ± 1.73	96.51 ± 0.59	96.67 ± 2.08
		90	98.76 ± 0.03	76.33 ± 6.81	95.48 ± 0.74	92.00 ± 4.36
		95	96.68 ± 1.37	40.00 ± 3.00	81.35 ± 3.02	68.33 ± 12.90
		97	80.97 ± 7.39	14.00 ± 3.61	51.17 ± 9.14	41.67 ± 9.07
	Client	85	98.84 ± 0.01	96.00 ± 1.73	67.20 ± 49.72	70.00 ± 51.96
	($\lambda = 10/20$)	90	98.63 ± 0.05	91.67 ± 6.03	66.19 ± 48.86	69.00 ± 51.12
		95	96.48 ± 0.95	70.67 ± 8.96	59.30 ± 43.31	56.00 ± 40.04
		97	85.55 ± 4.18	42.67 ± 3.21	49.35 ± 35.72	31.67 ± 19.55
RP	Server	85	98.95 ± 0.04	87.00 ± 4.36	92.64 ± 6.54	99.00 ± 1.73
		90	98.82 ± 0.05	67.00 ± 12.53	92.36 ± 6.66	96.67 ± 5.77
		95	97.03 ± 1.26	38.00 ± 2.65	83.22 ± 12.79	82.00 ± 15.59
		97	82.28 ± 3.22	13.67 ± 4.51	50.58 ± 13.87	41.33 ± 10.26
	Client	85	98.97 ± 0.03	99.00 ± 1.73	91.25 ± 5.45	92.00 ± 10.58
	($\lambda = 10/20$)	90	98.74 ± 0.11	90.33 ± 7.51	90.30 ± 5.33	87.33 ± 11.68
		95	97.38 ± 0.43	48.33 ± 4.51	81.49 ± 3.60	62.00 ± 36.35
		97	85.22 ± 1.58	17.00 ± 2.65	74.51 ± 0.82	46.00 ± 23.07

Table 7.8: Pruning results for CIFAR-10 (test set accuracy, watermark accuracy, and respective standard deviation per pruning rate (PR))

			IID		Non-IID	
Trigger	Scheme	PR	Accuracy	WM Accuracy	Accuracy	WM Accuracy
A	Server	80	86.63 ± 0.20	91.67 ± 2.31	79.13 ± 2.92	92.00 ± 4.36
		85	85.95 ± 0.46	82.00 ± 3.46	77.63 ± 2.87	80.67 ± 1.53
		90	84.41 ± 0.06	53.67 ± 4.16	72.14 ± 2.67	52.33 ± 1.15
		95	57.73 ± 2.55	14.00 ± 1.00	32.26 ± 2.02	12.67 ± 3.06
	Client	80	86.80 ± 0.41	93.67 ± 4.04	83.04 ± 0.95	91.33 ± 2.52
	($\lambda = 10/20$)	85	86.35 ± 0.44	87.67 ± 3.51	81.56 ± 1.41	83.33 ± 8.50
		90	84.92 ± 0.41	57.67 ± 4.51	77.22 ± 3.38	52.00 ± 9.54
		95	50.13 ± 8.80	14.33 ± 4.04	41.60 ± 3.76	13.33 ± 3.21
AP	Server	80	86.67 ± 0.02	94.00 ± 1.00	83.14 ± 0.94	97.00 ± 2.00
		85	86.07 ± 0.19	88.67 ± 2.08	81.96 ± 1.26	89.67 ± 2.08
		90	84.31 ± 0.24	68.67 ± 7.57	78.10 ± 2.50	64.33 ± 4.04
		95	53.75 ± 4.19	10.33 ± 0.58	30.73 ± 5.96	11.00 ± 3.61
	Client	80	86.88 ± 0.27	95.00 ± 0.00	83.76 ± 0.36	97.00 ± 3.00
	($\lambda = 10/20$)	85	86.26 ± 0.46	90.00 ± 1.73	82.91 ± 0.68	94.33 ± 5.03
		90	84.85 ± 0.65	70.67 ± 2.31	79.80 ± 2.30	73.67 ± 18.82
		95	52.05 ± 2.92	10.33 ± 0.58	51.43 ± 12.86	21.00 ± 12.12
R	Server	80	86.70 ± 0.11	82.33 ± 8.14	82.11 ± 0.75	79.33 ± 10.21
		85	86.32 ± 0.21	57.67 ± 6.81	80.62 ± 0.75	59.00 ± 9.54
		90	84.79 ± 0.27	32.67 ± 5.69	77.20 ± 1.73	35.33 ± 8.08
		95	52.57 ± 4.03	10.33 ± 0.58	46.15 ± 12.08	12.00 ± 1.73
	Client	80	86.69 ± 0.27	93.00 ± 2.65	83.79 ± 0.74	91.33 ± 7.09
	($\lambda = 10/20$)	85	86.14 ± 0.38	81.00 ± 5.29	83.02 ± 0.73	79.00 ± 13.53
		90	84.55 ± 0.65	34.33 ± 15.01	79.99 ± 1.56	56.67 ± 7.77
		95	45.45 ± 6.33	10.67 ± 1.15	56.70 ± 4.87	14.00 ± 4.36
RP	Server	80	86.55 ± 0.26	83.33 ± 13.20	83.28 ± 0.64	93.33 ± 4.04
		85	86.20 ± 0.13	68.33 ± 12.58	81.82 ± 1.09	80.00 ± 2.65
		90	84.49 ± 0.53	38.33 ± 4.93	77.00 ± 2.80	45.00 ± 1.73
		95	54.14 ± 2.07	10.33 ± 0.58	35.43 ± 5.16	12.67 ± 7.02
	Client	80	86.65 ± 0.34	96.67 ± 4.93	82.81 ± 1.25	95.00 ± 1.73
	($\lambda = 10/20$)	85	86.11 ± 0.28	86.67 ± 12.34	81.38 ± 2.03	85.00 ± 4.36
		90	84.73 ± 0.45	49.00 ± 11.14	77.20 ± 4.71	59.67 ± 5.69
		95	48.33 ± 1.93	10.00 ± 0.00	51.05 ± 8.10	19.33 ± 4.51

While the number of successful attacks is slightly higher compared to the static watermarking, the differences are small and the robustness to pruning seems to be overall in the same range.

7.4.2 Inversion

We first present the results of single-state inversion attacks on client- and server-side watermarking, followed by loss-based and then output-based multi-state attacks on server-side watermarking.

Single-State Inversion

Table 7.9 shows the results of single-state inversion attacks on CIFAR-10 and Table 7.10 shows the results on MNIST.

Client-Side Watermarks The results of attacks on watermarks embedded by a client show that the attack is successful in removing the watermark without affecting test set accuracy too much (by more than 5 percentage points) with trigger sets with random background on CIFAR-10 and with trigger sets without pattern on IID MNIST (though with Abstract, only 2 out 3 times). On non-IID MNIST, 2 out of 3 attacks are successful for both Random and AbstractPattern. In contrast, in static watermarking on CIFAR10, fewer attacks with RandomPattern succeeded (only 2 in the IID setting and none in the non-IID setting), and on MNIST, fewer attacks with Abstract (none in the IID setting and 1 in the non-IID setting).

Server-Side Watermarks The success counts for server-side inversion attacks are included in the multi-state inversion figures (Figures 7.9 to 7.12).

On IID CIFAR-10 there are only minor changes in terms of attack success: the results are unchanged for trigger sets with random background. There is 1 more successful removal with Abstract. With AbstractPattern, more attacks succeed in removing the watermark from 2 out of 3 models.

On IID MNIST the results are very similar to the static setting with Random and AbstractPattern. With Abstract, there are fewer successful removals with the early model state, but more with model states 25-50. RP is the only trigger set, where the robustness is notably increased with 1 successful removal at most instead of 3.

For non-IID CIFAR-10, the results are very similar for trigger sets Abstract, AbstractPattern, and Random, but there are more successful removals with RandomPattern. The number of attacks where 3 successful removals were possible, increased on this trigger set.

On non-IID MNIST, there was the largest impact on robustness, as the number of successful removals decreased for A, R, and RP and remained unchanged on AP.

Table 7.9: Single-state inversion results on CIFAR-10 models watermarked with dynamic watermarking threshold

			IID			Non-IID	
			Accuracy	WM Accuracy		Accuracy	WM Accuracy
Trigger	Scheme						
A	Server	0	85.93 ± 0.82	85.67 ± 5.51	0	80.94 ± 3.43	93.33 ± 5.03
		1	80.61 ± 1.25	55.33 ± 6.03	10	78.29 ± 2.51	75.67 ± 3.51
		5	82.15 ± 0.08	50.33 ± 3.06	25	77.77 ± 2.40	79.00 ± 3.61
		10	82.30 ± 0.99	52.67 ± 8.39	50	78.28 ± 3.49	84.00 ± 8.19
		15	82.98 ± 0.40	57.33 ± 2.08	200	80.44 ± 3.19	89.00 ± 6.24
		25	83.17 ± 0.30	61.00 ± 4.36	500	78.99 ± 4.31	84.00 ± 2.65
		40	82.63 ± 0.70	62.33 ± 2.31	800	79.31 ± 5.37	85.33 ± 15.31
		50	83.73 ± 0.48	71.33 ± 1.53	1000	81.01 ± 3.58	93.33 ± 3.21
	Client	50	84.78 ± 0.34	73.67 ± 6.43	1000	82.51 ± 1.31	58.33 ± 10.97
AP	Server	0	85.86 ± 0.80	88.67 ± 4.16	0	83.65 ± 1.19	97.67 ± 1.53
		1	81.18 ± 1.58	61.33 ± 15.04	10	81.81 ± 2.00	89.00 ± 3.46
		5	81.24 ± 0.94	45.00 ± 3.61	25	81.95 ± 1.93	87.33 ± 8.96
		10	80.53 ± 1.28	42.67 ± 6.51	50	81.41 ± 2.82	81.00 ± 9.54
		15	80.19 ± 0.19	45.00 ± 3.00	200	81.91 ± 2.67	82.67 ± 10.79
		25	81.52 ± 0.23	54.33 ± 1.53	500	82.61 ± 1.09	76.00 ± 3.61
		40	81.85 ± 2.74	55.67 ± 10.69	800	83.18 ± 1.08	91.67 ± 8.74
		50	84.21 ± 0.87	69.00 ± 13.00	1000	83.42 ± 0.50	95.33 ± 1.53
	Client	50	84.59 ± 0.36	71.00 ± 4.58	1000	83.06 ± 0.43	88.00 ± 16.64
R	Server	0	84.36 ± 0.16	20.33 ± 4.16	0	82.49 ± 1.30	53.00 ± 25.24
		1	76.63 ± 2.68	12.67 ± 4.62	10	80.28 ± 3.00	15.00 ± 3.46
		5	80.74 ± 2.15	8.33 ± 2.08	25	78.60 ± 2.86	11.00 ± 1.00
		10	81.48 ± 1.77	8.33 ± 1.15	50	79.93 ± 2.94	16.33 ± 3.21
		15	83.15 ± 0.46	9.00 ± 1.73	200	79.93 ± 2.79	14.00 ± 5.20
		25	83.73 ± 0.94	10.00 ± 1.00	500	79.65 ± 2.18	15.67 ± 1.53
		40	84.23 ± 0.21	10.00 ± 2.65	800	80.18 ± 2.16	11.00 ± 1.00
		50	84.41 ± 1.05	9.67 ± 1.53	1000	81.71 ± 1.95	15.67 ± 3.79
	Client	50	83.71 ± 1.06	11.00 ± 1.73	1000	81.99 ± 1.55	25.00 ± 8.19
RP	Server	0	85.10 ± 0.94	51.33 ± 9.61	0	83.38 ± 1.65	71.00 ± 6.56
		1	81.00 ± 1.18	17.33 ± 6.66	10	81.73 ± 2.55	41.33 ± 12.70
		5	79.41 ± 0.74	17.00 ± 3.00	25	81.56 ± 2.77	32.33 ± 9.07
		10	80.95 ± 1.49	13.67 ± 4.62	50	81.32 ± 2.09	30.33 ± 5.86
		15	78.93 ± 0.40	13.33 ± 3.51	200	81.94 ± 3.06	28.67 ± 18.01
		25	80.38 ± 1.16	12.33 ± 2.08	500	83.01 ± 2.28	34.67 ± 15.70
		40	82.39 ± 0.66	13.00 ± 3.00	800	83.83 ± 0.72	37.00 ± 17.52
		50	83.38 ± 1.00	23.67 ± 11.15	1000	83.42 ± 1.01	48.00 ± 19.92
	Client	50	84.64 ± 0.18	25.33 ± 0.58	1000	81.74 ± 2.45	34.33 ± 6.51

Table 7.10: Single-state inversion results on MNIST models watermarked with dynamic watermarking threshold

			IID			Non-IID	
Trigger	Scheme		Accuracy	WM Accuracy		Accuracy	WM Accuracy
A	Server	0	99.03 ± 0.09	82.33 ± 4.04	0	96.24 ± 0.74	82.00 ± 7.00
		1	98.96 ± 0.09	72.33 ± 11.59	5	95.94 ± 0.81	72.00 ± 7.81
		5	98.85 ± 0.08	54.67 ± 11.93	10	95.81 ± 0.89	71.00 ± 1.73
		10	98.73 ± 0.11	35.33 ± 11.85	50	95.85 ± 1.37	59.00 ± 9.54
		15	98.82 ± 0.18	33.67 ± 14.57	100	96.16 ± 1.16	59.33 ± 10.07
		25	98.73 ± 0.14	29.67 ± 12.66	200	95.93 ± 1.21	59.33 ± 14.15
		40	98.76 ± 0.11	32.33 ± 15.50	300	96.07 ± 1.01	64.33 ± 4.04
		50	98.81 ± 0.14	43.00 ± 17.44	500	95.95 ± 0.95	66.67 ± 5.13
	Client	50	98.79 ± 0.06	49.00 ± 13.23	500	95.06 ± 0.84	43.33 ± 20.79
AP	Server	0	99.06 ± 0.02	91.00 ± 2.65	0	96.64 ± 0.30	98.00 ± 1.00
		1	98.91 ± 0.08	80.67 ± 13.58	5	96.53 ± 0.75	94.33 ± 0.58
		5	98.82 ± 0.08	73.00 ± 16.52	10	96.18 ± 0.80	93.67 ± 2.31
		10	98.78 ± 0.15	71.00 ± 19.08	50	94.74 ± 0.77	89.67 ± 6.81
		15	98.75 ± 0.15	72.67 ± 18.77	100	94.81 ± 1.22	86.33 ± 2.89
		25	98.76 ± 0.18	72.00 ± 19.08	200	94.98 ± 0.98	86.00 ± 2.00
		40	98.75 ± 0.22	72.00 ± 18.25	300	95.00 ± 1.13	85.33 ± 3.06
		50	98.76 ± 0.18	69.67 ± 22.37	500	95.15 ± 1.12	84.67 ± 3.21
	Client	50	98.75 ± 0.13	89.33 ± 8.33	500	94.88 ± 1.89	43.33 ± 26.27
R	Server	0	98.96 ± 0.04	92.00 ± 5.20	0	96.65 ± 0.60	48.67 ± 19.86
		1	98.81 ± 0.30	53.33 ± 25.03	5	96.59 ± 0.54	47.00 ± 17.32
		5	98.57 ± 0.41	33.67 ± 14.47	10	96.46 ± 0.56	24.67 ± 6.66
		10	98.40 ± 0.43	27.67 ± 11.59	50	94.24 ± 0.92	15.33 ± 1.53
		15	98.70 ± 0.17	23.00 ± 11.53	100	95.86 ± 0.67	11.67 ± 2.08
		25	98.65 ± 0.25	20.00 ± 7.94	200	96.13 ± 0.57	10.67 ± 3.06
		40	98.67 ± 0.28	19.33 ± 5.69	300	96.10 ± 0.73	10.00 ± 3.61
		50	98.69 ± 0.26	24.67 ± 8.50	500	96.08 ± 0.73	12.00 ± 1.73
	Client	50	98.73 ± 0.22	12.67 ± 2.52	500	95.17 ± 1.03	11.00 ± 1.41
RP	Server	0	99.04 ± 0.03	89.00 ± 8.19	0	93.87 ± 5.55	99.67 ± 0.58
		1	98.86 ± 0.10	60.67 ± 22.37	5	93.75 ± 5.48	95.67 ± 4.51
		5	98.81 ± 0.10	58.33 ± 27.01	10	93.46 ± 5.54	99.00 ± 1.00
		10	98.78 ± 0.16	57.00 ± 23.58	50	92.63 ± 4.75	91.33 ± 13.32
		15	98.74 ± 0.20	51.33 ± 24.79	100	93.08 ± 4.98	94.00 ± 7.21
		25	98.71 ± 0.24	50.00 ± 29.60	200	92.83 ± 5.46	90.00 ± 9.17
		40	98.71 ± 0.27	45.67 ± 26.27	300	92.70 ± 5.56	89.33 ± 9.71
		50	98.72 ± 0.20	46.33 ± 22.03	500	92.93 ± 5.07	92.00 ± 6.93
	Client	50	98.80 ± 0.18	75.67 ± 20.31	500	91.29 ± 5.36	77.00 ± 20.42

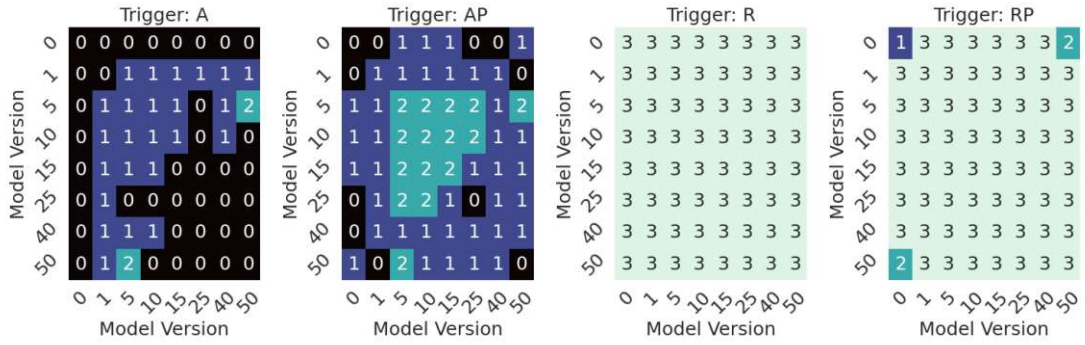


Figure 7.9: Counts of successful single-state and loss-based multi-state inversion attacks on 3 IID CIFAR-10 models, trained using server-side watermarking with dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Loss-Based Multi-State Inversion

Server-Side Watermarks on IID CIFAR-10 Figure 7.9 shows the results of the loss-based multi-state- and single-state inversion attacks on IID CIFAR-10, trained with server-side watermarking. For Trigger Set Abstract, the number of attack sets achieving n out of 3 successes changed as follows (from static watermarking to dynamic watermarking):

- $n = 3$: 1 (static) $\rightarrow$ 0 (dynamic)
- $n = 2$: 3 (static) $\rightarrow$ 1 (dynamic)
- $n = 1$: 15 (static) $\rightarrow$ 13 (dynamic)

For Trigger Set AbstractPattern, the changes are:

- $n = 3$: 1 (static) $\rightarrow$ 0 (dynamic)
- $n = 2$: 14 (static) $\rightarrow$ 9 (dynamic)
- $n = 1$: 10 (static) $\rightarrow$ 20 (dynamic)

On trigger sets Random and RandomPattern, the attacks were very successful, regardless of the attack set used (except model state 0 by itself in RandomPattern). Overall, the results for Abstract and AbstractPattern show slight improvements in robustness compared to static watermarking. On the other trigger sets, the attack success is on the same level as with static watermarking.

When comparing the watermark accuracy and test set accuracy (Figure 1a) to the static watermarking, 2 observations can be made:

- Any attack involving model state 0 leads to either no decrease or to a smaller decrease in watermark accuracy on Abstract compared to the baseline attack. For example, for the multi-state attack with model states 0 and 50, the average watermark accuracy was 11.7 percentage points below the baseline attack with static watermarking and 5 percentage points above it with dynamic watermarking.
- Across trigger sets, there is a larger decrease in test set accuracy from attacks involving intermediate models, compared to the baseline attack. The difference is the largest for model states 1-5. For attacks involving model state 1 from static watermarking, there was an average decrease of 0.99 (Abstract), 1.42 (AbstractPattern), 0.79 (Random), and 0.79 (RandomPattern) percentage points compared to the baseline attack. With dynamic watermarking, the decrease is 3.69, 3.76, 8.54, and 4.43 percentage points, respectively.

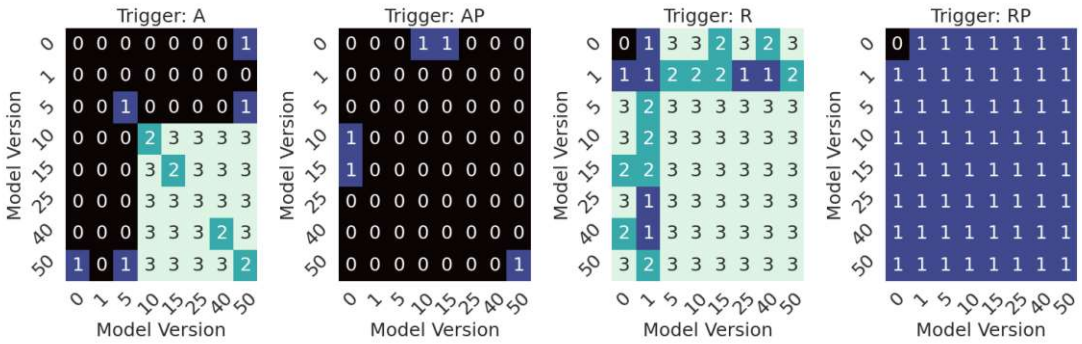


Figure 7.10: Counts of successful single-state and loss-based multi-state inversion attacks on 3 IID MNIST models, trained using server-side watermarking with the dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Server-Side Watermarks on IID MNIST Figure 7.10 shows the results of the loss-based multi-state attacks and single-state inversion attacks on IID MNIST. On the trigger set Abstract, most attacks involving model states 0 - 5 do not succeed, while in the static case most resulted in 2 out of 3 removals. However, all loss-based multi-state attacks involving only model states ≥ 10 succeeded, which is worse than in the static case. On AbstractPattern, attacks with model state 0 in combination with 50 or 40 previously removed the watermark from 2 out of 3 models. With the changes, these attacks failed. On trigger set Random, the watermark is again not very robust and fully removed in most cases. On trigger set RandomPattern, we can see the largest changes: previously, any attack involving model state 0 and most involving model state 1 led to 3 out of 3 removals, while now, all lead to only 1 at most.

In these experiments, the attacks led to a considerably lower drop in watermark accuracy compared to the static watermarking results. Most intermediate model states on trigger

sets AbstractPattern, Random, and RandomPattern do not reduce the watermark accuracy more than the baseline attack. Interestingly, even late model states, where the watermark is already embedded, increase the watermark accuracy (more). As such, the modifications clearly increase the robustness.

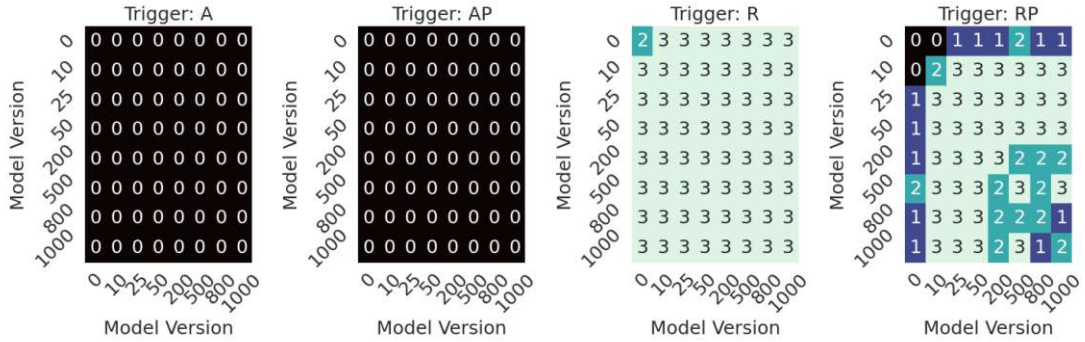


Figure 7.11: Counts of successful single-state and loss-based multi-state inversion attacks on 3 non-IID CIFAR-10 models, trained using server-side watermarking with dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Server-Side Watermarks on non-IID CIFAR-10 Figure 7.11 shows the non-IID CIFAR-10 results. On AbstractPattern and Abstract, there are very few successful attacks, and all of them succeed on only 1 out of 3 models. With dynamic watermarking, all are unsuccessful. The trigger set Random is still not robust at all with dynamic watermarking, as the watermark removal succeeds with almost all attacks. On trigger set RandomPattern, differences arise for attacks that use model state 0, reducing the number of successful attacks per set by 1-2.

In terms of accuracy and watermark accuracy, the effect is (as was the case for IID CIFAR-10) mostly a smaller decrease / higher increase of watermark accuracy when model state 0 is used, and for attacks involving model states 10-25, a decrease of test set accuracy on all trigger sets.

Server-Side Watermarks on non-IID MNIST Figure 7.12 shows the results of the attack on non-IID MNIST. We can observe a considerable increase of robustness on the trigger sets Abstract, AbstractPattern, and RandomPattern. Previously, there were 35, 13 and 27 attack sets with at least 1 successful removal, respectively. This decreased to 2, 0, and 0. However, as in all prior experiments, the trigger set Random is still not robust, with most attacks succeeding.

As with IID MNIST, there is a large effect on the watermark accuracy and a small one on the test set accuracy.

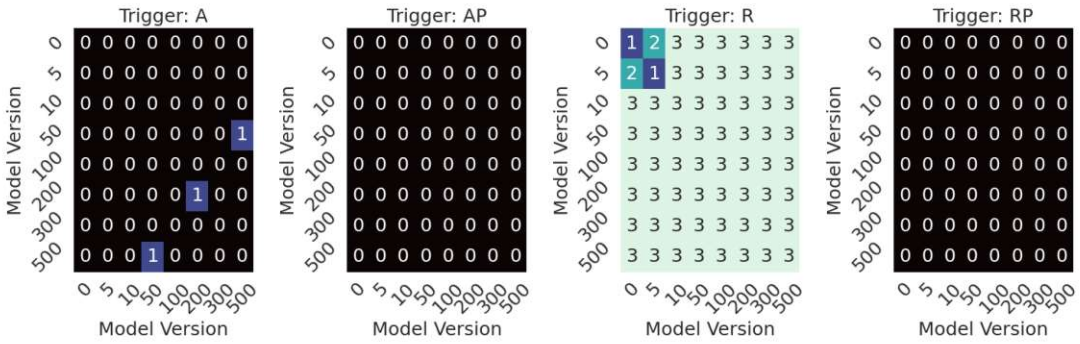


Figure 7.12: Counts of successful single-state and loss-based multi-state inversion attacks on 3 non-IID MNIST models, trained using server-side watermarking with the dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Output-Based Multi-State Inversion

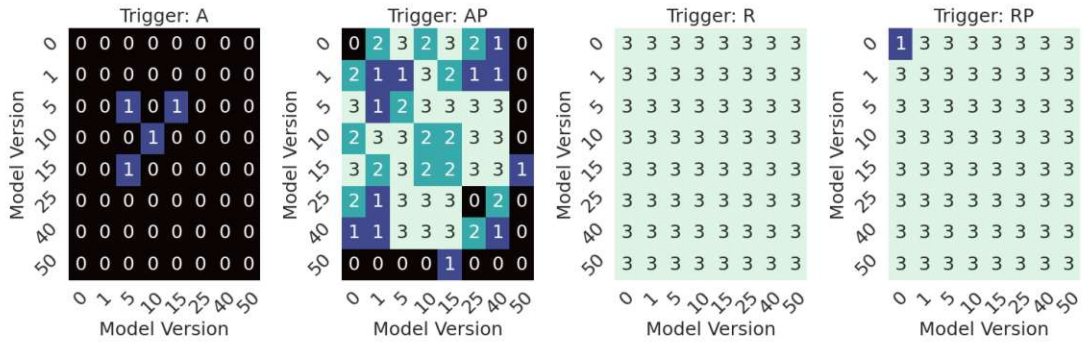


Figure 7.13: Counts of successful single-state and output-based multi-state inversion attacks on 3 IID CIFAR-10 models, trained using server-side watermarking with dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Server-Side Watermarks on IID CIFAR-10 In Figure 7.13 the results of the output-based multi-state inversion attacks are visualized. While on Abstract slightly fewer attacks are successful after dynamic instead of static watermarking (3 instead of 11), on AbstractPattern, more attacks are successful than before. Random and RandomPattern remain the least robust, with most attacks being successful. For example, only a single combination of model states leads to 3 out of 3 successful removals, while with dynamic watermarking, there are 11.

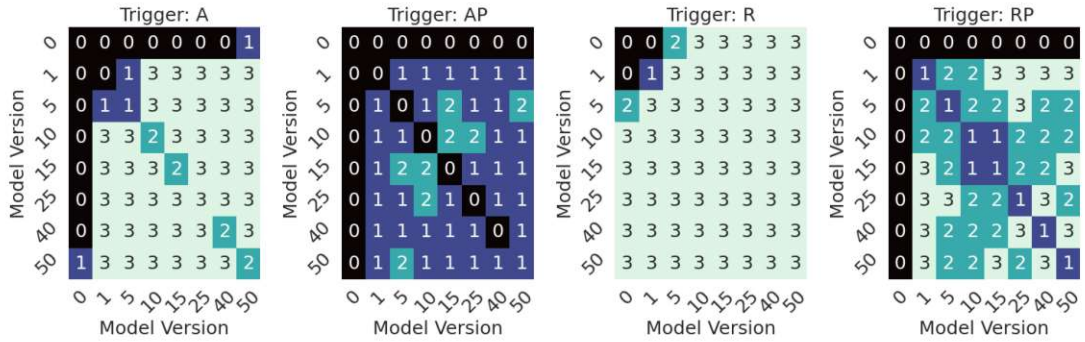


Figure 7.14: Counts of successful single-state and output-based multi-state inversion attacks on 3 IID MNIST models, trained using server-side watermarking with the dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Server-Side Watermarks on IID MNIST Figure 7.14 shows the results for IID MNIST. The attacks have a similar or even a higher level of success than after static watermarking: with pattern-less trigger sets, most attacks succeed; on AbstractPattern most succeed on at least 1 run, and on RandomPattern all multi-state attacks succeed at least 2 out of 3 times. The major difference is that on Abstract, AbstractPattern, and RandomPattern, all but 1 attack that include model state 0 failed. On Random, only the attacks that do not include even a single model state above 1 failed consistently.

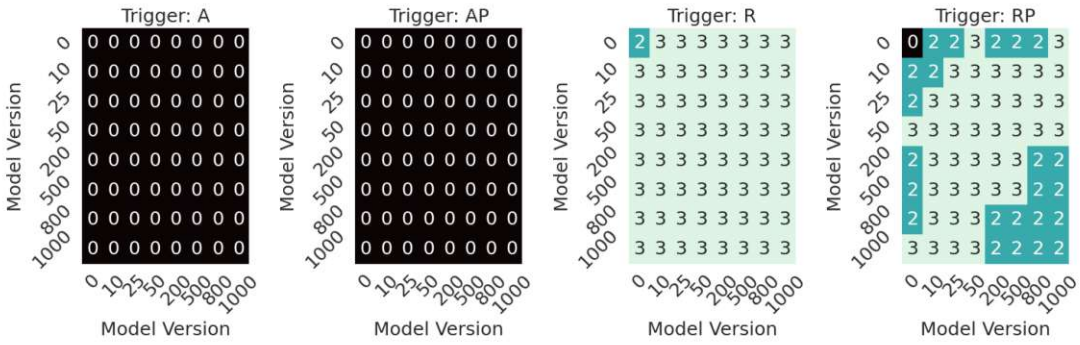


Figure 7.15: Counts of successful single-state and output-based multi-state inversion attacks on 3 non-IID CIFAR-10 models, trained using server-side watermarking with dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Server-Side Watermarks on non-IID CIFAR-10 The non-IID results for CIFAR-10, shown in Figure 7.15, do not indicate improvements in robustness: on Abstract and AbstractPattern, the model is fully robust, while on Random, it is the opposite, and on AbstractPattern, there is a mix of attacks succeeding twice or all 3 times – overall very similar results to the static setting.

Server-Side Watermarks on non-IID MNIST Non-IID MNIST (see Figure 7.16) is the only setting where the robustness seems to be increased considerably by the changes. On Abstract, previously most attack sets succeeded 3 out of 3 times – now most do not succeed even once. In particular, attacks that only include model versions 0-50 all fail. On trigger set AbstractPattern, previously all but 1 multi-state attack succeeded 2 out of 3 times, one 3 out of 3 times. Now, there are only 2 sets with a single successful removal each. Random is once again not robust at all, while on RandomPattern, the robustness only increased for attacks that include only model states up to 50 and all that include model states 0 or 5.

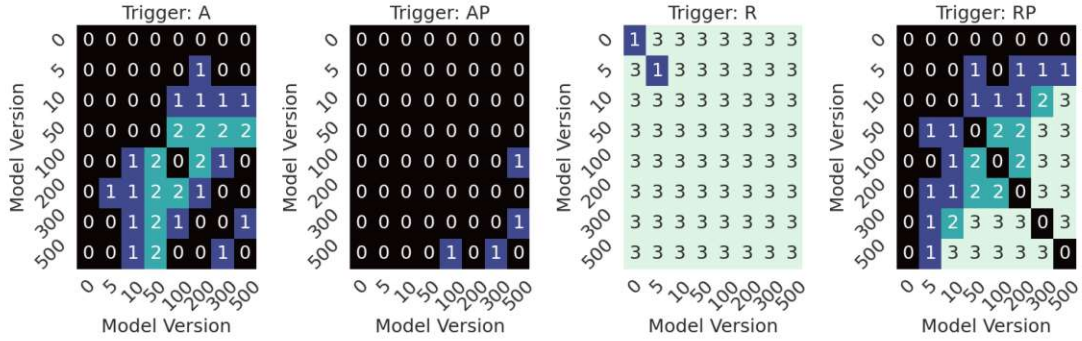


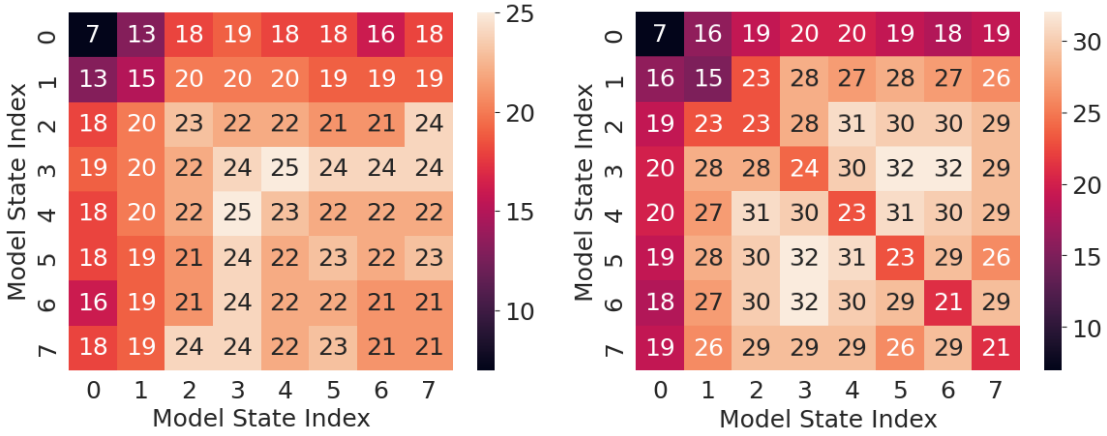
Figure 7.16: Counts of successful single-state and output-based multi-state inversion attacks on 3 non-IID MNIST models, trained using server-side watermarking with the dynamic threshold. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Table 7.11: Success counts of pruning, WM-DIFF, and inversion attacks in different Settings. For each setting not marked by a *, the respective attack was performed 3 times. For those marked by a *, it was performed 2 times.

Attack	Dataset	Setting	Server				Client			
			A	AP	R	RP	A	AP	R	RP
Pruning	CIFAR-10	IID	-	-	3	3	-	-	2	1
		Non-IID	-	-	-	-	-	-	-	-
	MNIST	IID	3	2	3	3	1	-	-	1
		Non-IID	-	-	-	-	-	-	1*	-
Inversion	CIFAR-10	IID	-	-	3	3	-	-	3	3
		Non-IID	-	-	3	2	-	-	3	3
	MNIST	IID	2	1	3	1	2	-	3	-
		Non-IID	-	-	3	-	2	2	2*	-

7.4.3 Summary

Table 7.11 shows the counts of successful Pruning and Baseline-Inversion attacks on watermarks embedded using a dynamic threshold. There are some differences compared to the attacks on the watermarks embedded through static watermarking. 21 out of 94 pruning attacks were successful with static watermarking. With dynamic watermarking, this increases slightly to 23 out of 95. The change in robustness in regard to the baseline inversion attack is larger: 44 out of 95 attacks succeeded, instead of 37 out of 95.



(a) Loss-based multi-state inverison (b) Output-based multi-state inversion

Figure 7.17: Inversion success counts aggregated for all settings for server-side watermarking with dynamic threshold. The total number of attacks per model state (combination) is 48. An attack counts as successful when the watermark accuracy drops to below 47% and the test set accuracy drops by less than 5 percentage points

Figure 7.17 shows success counts of multi-state attacks for each dataset, data distribution,

and trigger set, aggregated by model state indices. They demonstrate that dynamic watermarking reduced the number of successful single-state attacks and loss-based multi-state attacks, as they removed the watermarks from 24 and 25 models at most, respectively. In contrast, with static watermarking, it succeeded 26 and 30 times at most. As expected, the difference is largest with early model states: attacks involving model states at index 0 and/or 1 succeed 20 times at most, whereas that number was 30 with static watermarking. The output-based multi-state attack, however, is not impacted much by the changes: attacks that do not use model states at indices 0 - 2 still succeed 26 - 32 times.

The impact of dynamic watermarking on the robustness can be summarized as follows:

- It avoids the ideal setting for the WM-diff attack, where a fully watermarked model state that has undergone minimal or no training on the primary task is shared in the first FL round.
- In our experiments, dynamic watermarking had a slight negative impact on the robustness against pruning.
- It reduced the effectiveness of single-state inversion and loss-based inversion attacks.
- Output-based inversion is still very effective.

CHAPTER 8

Conclusion

In this thesis, we explored state-of-the-art watermarking schemes in federated learning and analyzed their differences. For this purpose, we performed experiments with the server-side watermarking schemes Waffle and FedTracker, as well as client-side watermarking – with and without boosting of updates by the watermarking client. For each watermarking scheme, we trained models on the primary task of image classification using two different datasets, CIFAR-10 and MNIST. The training data was either distributed independently and identically (IID) among clients, or non-IID, where we divided the data so that each client had samples belonging to exactly two classes. For the secondary task of watermarking, we used four distinct trigger sets, consisting of either randomly generated or abstract images, with and without a class-specific pattern superimposed on the images. This wide range of configurations allowed us to thoroughly evaluate how the watermark embedding schemes are affected by different factors. To assess the robustness of the watermarks embedded in the final federated learning models, we applied traditional watermark removal attacks, such as pruning and model inversion with unlearning. Additionally, we designed and evaluated novel watermark removal attacks that exploit intermediate model states shared during the federated learning process. To address the identified vulnerabilities, we proposed and tested modifications to the existing watermarking schemes.

8.1 Research Questions

In this section, the answers to the research questions posed in Section 1.2 are discussed.

1. **To what extent do state-of-the-art federated black-box watermarking approaches differ in terms of fidelity, effectiveness, robustness, and efficiency?**
   Fidelity: We found that the performance on the primary task was not affected

by the watermarking process in most cases, as it was close to the performance of models trained without watermarking. The exceptions occurred exclusively on MNIST, with a model architecture that has a significantly lower number of parameters than the unaffected CIFAR-10 model. These exceptions consisted of (i) individual server-side watermarking runs, mostly using FedTracker with a trigger that was particularly difficult to learn, that failed on both tasks, due to exploding gradients, and (ii) client-side runs with boosting with a too high factor, which leads to less stable training and large drops in test set accuracy and watermark accuracy.

Effectiveness: With server-side watermarking, the watermark accuracy after training was between 98% and 100% for all runs and trigger set, except the failed runs mentioned in the previous point.

In client-side watermarking, however, embedding the watermark was more challenging. Without the boosting of client updates, the performance varied significantly depending on the trigger set on MNIST and IID CIFAR-10, ranging from an average watermark accuracy of 10% to 93.67%. On non-IID CIFAR-10, the watermark accuracy was between 99.33 and 100.00%; this is due to the long training process (1,000 rounds of federated learning), which leaves sufficient time to embed the watermark, regardless of the trigger set.

With boosting, the watermark accuracy was 100% for all CIFAR-10 settings, except a single configuration, which resulted in an average watermark accuracy of only 89.33%. On MNIST, even with boosting, the watermarks could only be reliably embedded with a watermark accuracy of around 99% when trigger sets with the common pattern were used, highlighting the limitations of the boosting approach.
Robustness: To evaluate the robustness of the watermarks to common attacks, we performed L1-based pruning at different pruning levels and model inversion with unlearning. The attacks were performed on models watermarked using Waffle or using client-side watermarking with boosting. We compared the accuracy on the test set and on the trigger set before and after the attack. We found that the robustness depends strongly on the following factors:

- Trigger set: Trigger sets with a class-specific pattern often resulted in more robust watermarks than trigger sets where images targeting the same class had no common features. In addition, trigger sets consisting of abstract images were more robust than those with images consisting of Gaussian noise.
- Data distribution: With IID data, a higher number of attacks were successful than with non-IID data.
- Server vs. Client: watermarks embedded through client-side watermarking had lower attack success rates.

Efficiency: We measured the efficiency as the number of additional mini-batches or epochs of training needed to embed the watermark. In the server-side watermarking schemes, the models are trained on the trigger set before each round of federated learning, until one of two stopping conditions is met: a threshold of watermarking

epochs is reached, or the model reaches the desired watermark accuracy. We found that FedTracker leads to the same or a slightly higher number of watermarking epochs compared to Waffle. In client-side watermarking, typically, the trigger set is added to the client's training data without otherwise modifying the training process. As such, the overhead is static and determined by the number of rounds and local training epochs per round.

In our experiments, the result was a higher overhead from server-side watermarking with IID data. In contrast, in the non IID setting, the overhead was smaller with server-side watermarking on CIFAR-10 and with certain trigger sets also on MNIST. While this answers the question for the watermarking approaches as described in literature, there is no inherent reason why, in client-side watermarking, the training can not also be tied to certain stopping conditions, allowing either multiple watermarking epochs per FL round or to skip watermarking if the watermark is already embedded. For RQ3, we proposed such modifications, and comparing server-side and client-side watermarking given these changes might be better suited to make conclusions about the inherent efficiency differences. The results of the modifications in terms of the overhead are mixed, with server-side watermarking causing higher overhead in some cases and client-side watermarking in others, depending on the dataset, data distribution, and trigger set. However, in most cases, the easily learnable trigger images with random backgrounds and class-specific patterns resulted in lower overhead for server-side watermarking, while other triggers generally led to higher overhead.

2. **How can attackers leverage intermediate models?**

   a) **To what extent do the schemes succeed in protecting intermediate models?** We found that server-side schemes mostly succeed in embedding the watermark before intermediate models reach high accuracy on the primary task. Exceptions occurred only on MNIST with the trigger set Random, where, in some cases, the watermarking stopped before a high watermark accuracy was reached due to a cap on the number of watermarking epochs per round. In client-side schemes, the protection of intermediate models was a bigger challenge. Like with the effectiveness of the watermark on the final model, it was highly dependent on the choice of trigger set and boosting factor. Even with the highest possible boosting factor, where a higher value leads to high instability during training and/or exploding gradients, the embedding was not fast enough in some cases with the smaller model, leaving a high number of intermediate model states vulnerable.

   b) **How can intermediate models be used by an attacker to remove the watermark?**

   We developed and evaluated two types of watermark removal attacks on server-side schemes that make use of intermediate model states: WM-Diff and multiple types of inversion-based attacks. In WM-Diff, watermark removal

is achieved by subtracting the scaled difference of model weights before and after watermarking from the final model. For the inversion-based attacks, we inverted images from intermediate model states, either in addition to or instead of using the final model, and performed 'unlearning' on the final model using these generated images. We compared three approaches for the inversion attacks: (i) single-state inversion, where a single intermediate model state is used for inversion; (ii) output-based multi-state inversion, where inversion is performed on two model states independently, and their outputs are combined to form the unlearning set; and (iii) loss-based multi-state inversion, where during the inversion process, the losses of two model states are combined and minimized jointly to generate the unlearning set. We demonstrated that both WM-Diff and the multi-state inversion attacks outperform regular inversion and pruning.

3. **What strategies can be used to enhance the reliability of the watermark embedding and the robustness to removal attacks that utilize intermediate models?**

   We propose a dynamic watermarking threshold that increases throughout training based on the primary task performance. It can be used in client-side and server-side watermarking. In client-side watermarking, the modifications ensure that the model is trained for a variable number of rounds to reach the threshold. This modification reduces the number of insufficiently watermarked intermediate model states that already have high utility to the attacker. It successfully reduces the effectiveness of loss-based multi-state, but not the effectiveness of output-based multi-state attacks. Furthermore, it leads to an increase in overhead for server-side watermarking, and some client-watermarking settings.

8.2 Future Work

Finally, we give a brief outlook on possible directions for future work.

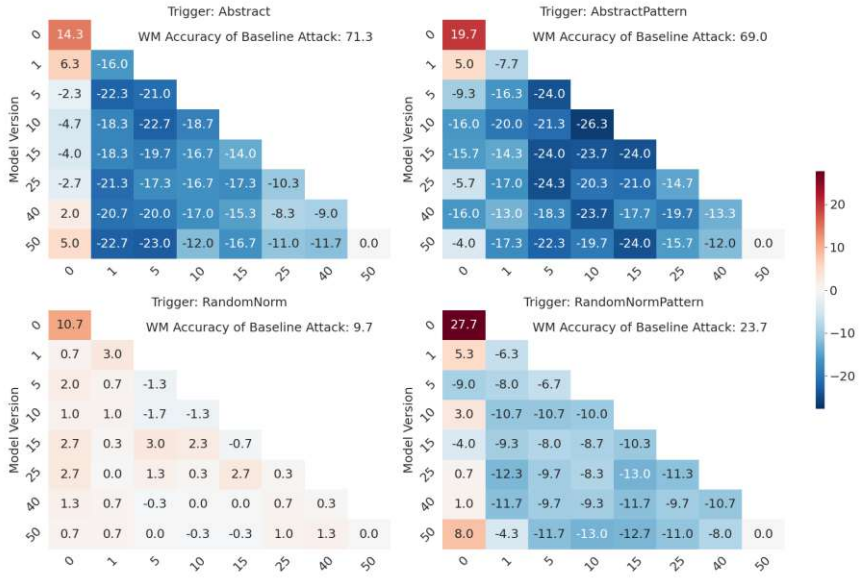
- One possible avenue for future research is more elaborate mechanisms for strengthening the watermark, which is especially important for client-watermarking, where the watermark needs to 'survive' the aggregation step, in spite of being embedded by only one client. Instead of scaling the whole update by the watermarking client, identifying weights that are particularly relevant and focusing on them might be preferable so as to not impact the stability of training and the performance of the primary task.

- We explored how differently the four different trigger sets we used affected the computational overhead from watermarking, the watermark embedding speed (in terms of FL rounds), and the robustness to attacks. Investigation of other types of trigger sets would allow a more complete understanding and more informed

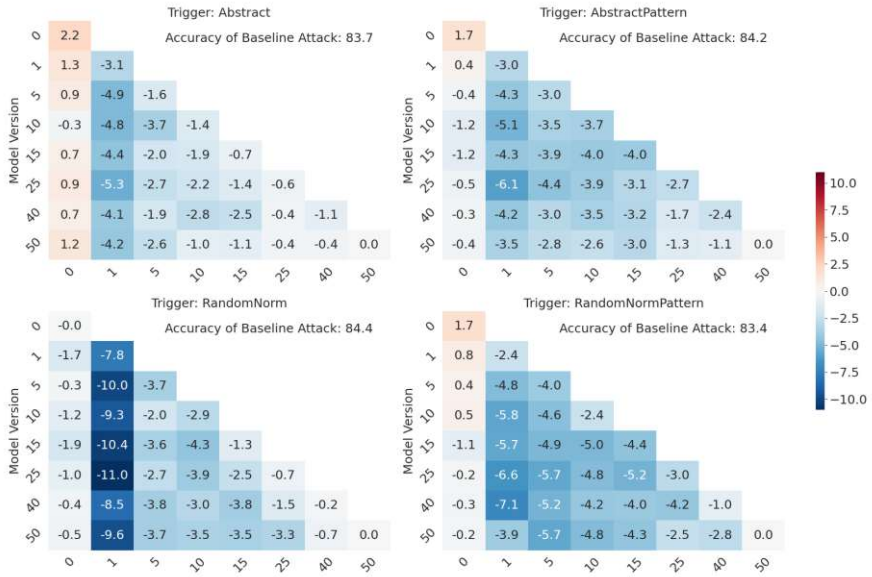
decisions when selecting a trigger set. In particular, further exploration of trigger sets based on adversarial samples in federated learning, as used in [LFG+22], would be valuable.

- In this thesis, we explored two ways of using two model states together for inversion attacks. However, these approaches could be further explored and refined, e.g., by using a higher number of model states, being more selective about which model states to use or which reverse-engineered images to include in the dataset for 'unlearning'. For example, model states could be selected by applying backdoor detection methods and specific images could be chosen based on how they affect different model states.
- We only considered attacks that can be performed after the training process is finished. For our FL-specific attacks, all the attacker has to do during training is to store the model states they receive and to participate like a regular client to avoid suspicion, i.e., the adversary is a passive, honest-but-curios attacker. However, the attacker could try to actively influence the training process to, e.g., receive model states that are particularly useful for later watermark removal. For example, by sabotaging the global model through their updates, the model owner could be forced to re-watermark the model from scratch, allowing the attacker to cause the ideal setting for our proposed WM-Diff attack.

Appendix

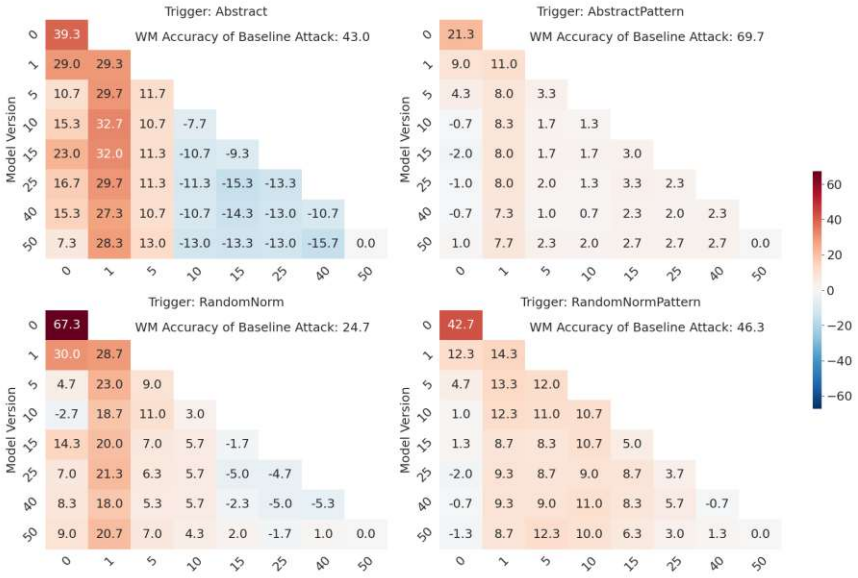


(a) Average difference in watermark accuracy to baseline Attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.

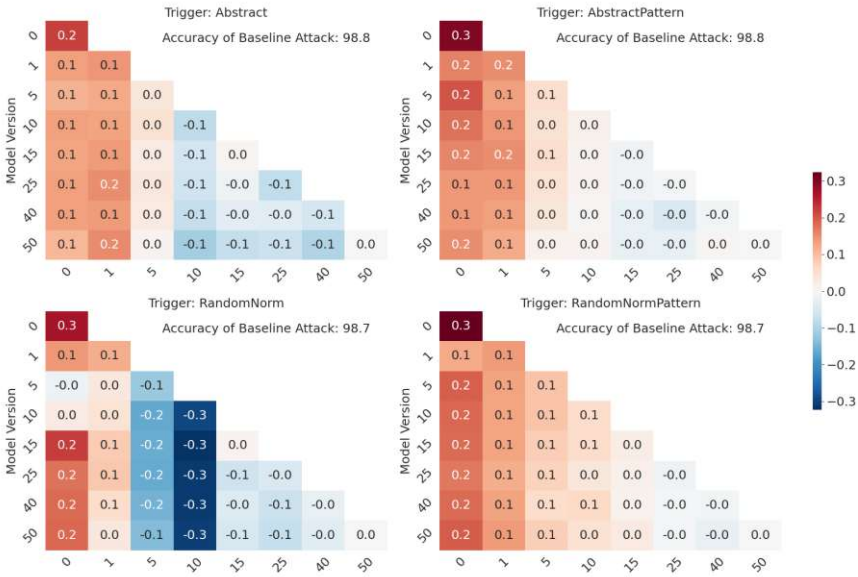


(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 1: Loss-based multi-state inversion and single-state inversion with different IID CIFAR-10 model states that were trained using server-side watermarking with dynamic threshold. The single-state inversion results are shown on the diagonal.

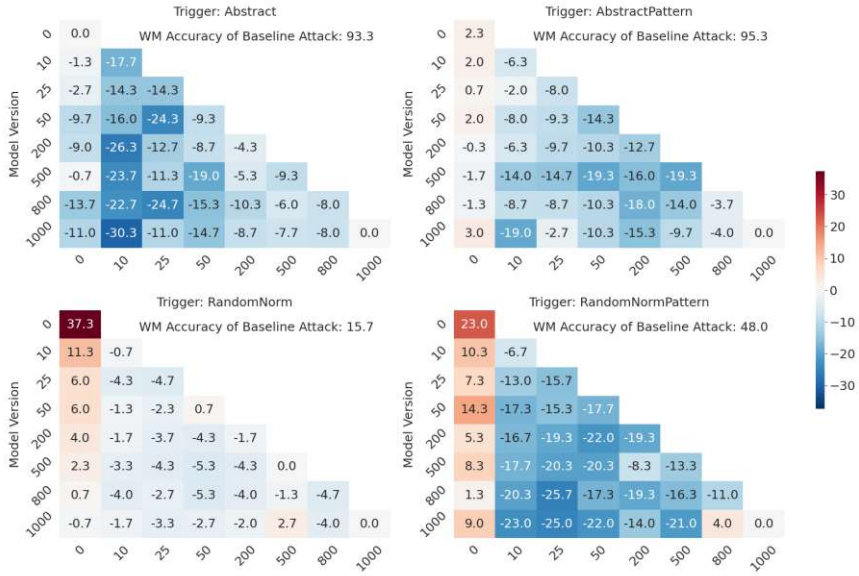


(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.

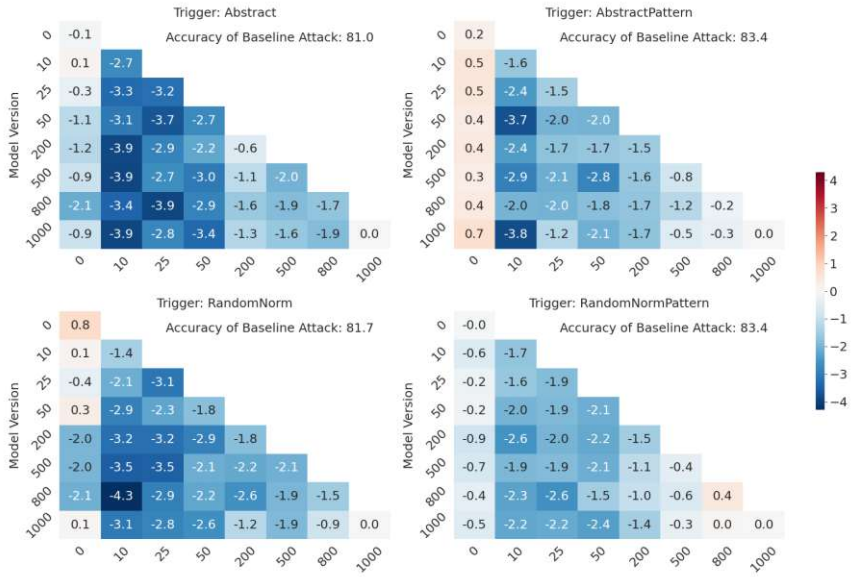


(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 2: Loss-based multi-state inversion and single-state inversion with different IID MNIST model states that were trained using server-side watermarking with dynamic threshold. The single-model inversion results are shown on the diagonal.

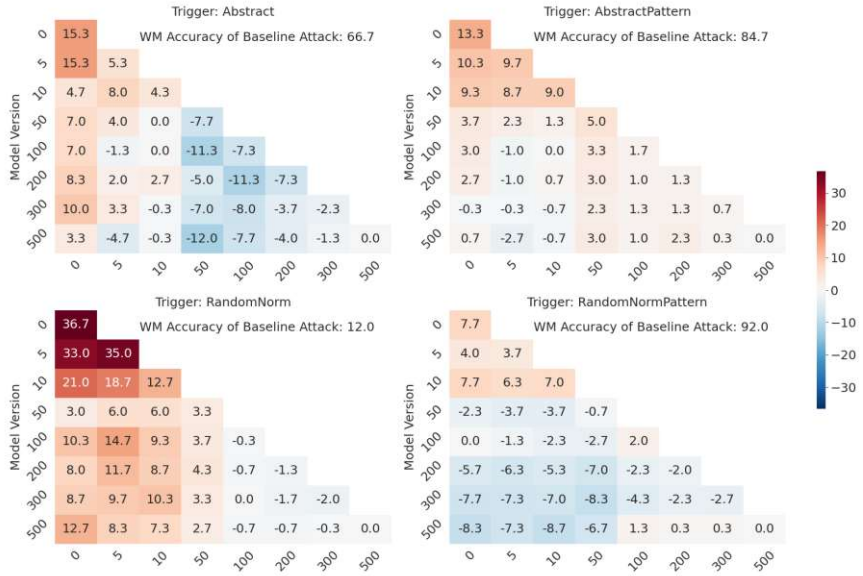


(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.

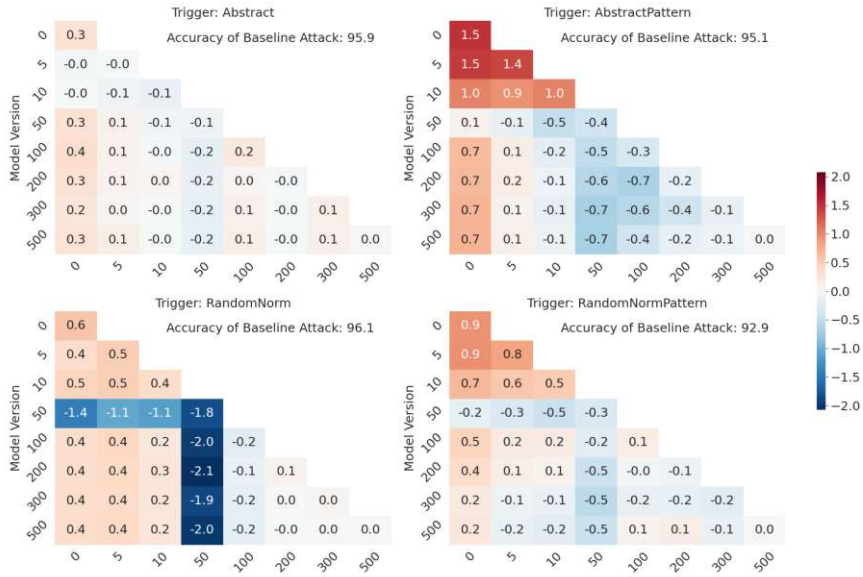


(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 3: Loss-based multi-state inversion and single-state inversion non-IID CIFAR-10 model states that were trained using server-side watermarking with dynamic threshold. The single-model inversion results are shown on the diagonal.



(a) Average difference in watermark accuracy to baseline attack. A positive value means the attack is less effective in removing the watermark than the baseline attack.



(b) Average difference in test set accuracy to baseline attack. A positive value means the attack degrades the test set accuracy less than the baseline attack.

Figure 4: Loss-based multi-state inversion and single-state inversion with different non-IID MNIST model states that were trained using server-side watermarking with dynamic threshold. The single-model inversion results are shown on the diagonal.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABC+18] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, 2018.

[BNS+06] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006. https://dl.acm.org/doi/10.1145/1128817.1128824.

[BR18] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2154–2156, 2018. https://dl.acm.org/doi/10.1145/3243734.3264418.

[CFZK19] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4658–4664. International Joint Conferences on Artificial Intelligence Organization, 7 2019. https://doi.org/10.24963/ijcai.2019/647.

[CLL+17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017. https://doi.org/10.48550/arXiv.1712.05526.

[DLZL21] Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11966–11976, 2021. https://doi.org/10.1109/ICCV48922.2021.01175.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. https://doi.org/10.1561/0400000042.

[DRCK19] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 485–497, 2019. https://doi.org/10.1145/3297858.3304051.

[EEF+18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018. https://doi.org/10.1109/CVPR.2018.00175.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015. https://doi.org/10.1145/2810103.2813677.

[FLJ+14] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing. In *23rd USENIX security symposium (USENIX Security 14)*, pages 17–32, 2014. https://dl.acm.org/doi/10.5555/2671225.2671227.

[GP18] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018. https://doi.org/10.1145/3240765.3240862.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. https://doi.org/10.48550/arXiv.1406.2661.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. https://doi.org/10.48550/arXiv.1412.6572.

[GWX+19] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019. https://doi.org/10.48550/arXiv.1908.01763.

[HLR+21] Haripriya Harikumar, Vuong Le, Santu Rana, Sourangshu Bhattacharya, Sunil Gupta, and Svetha Venkatesh. Scalable backdoor detection in neural

networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part II*, pages 289–304. Springer, 2021. https://doi.org/10.1007/978-3-030-67661-2_18.

[HPMG20] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. https://doi.org/10.48550/arXiv.1506.02626.

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012. COURSERA: Neural Networks for Machine Learning: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[JSMA19] Mika Juuti, Sebastian Szyller, Samuel 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. https://doi.ieeecomputersociety.org/10.1109/EuroSP.2019.00044.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. https://api.semanticscholar.org/CorpusID:18268744.

[KHB+18] Kamran Kowsari, Mojtaba Heidarysafa, Donald E Brown, Kiana Jafari Meimandi, and Laura E Barnes. Rmdl: Random multimodel deep learning for classification. In *Proceedings of the 2nd international conference on information system and data mining*, pages 19–28, 2018.

[Kin14] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. https://doi.org/10.48550/arXiv.1412.6980.

[KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of*

sciences, 114(13):3521–3526, 2017. https://doi.org/10.1073/pnas.1611835114.

[LBK+23] Mohammed Lansari, Reda Bellafqira, Katarzyna Kapusta, Vincent Thouvenot, Olivier Bettan, and Gouenou Coatrieux. When federated learning meets watermarking: A comprehensive overview of techniques for intellectual property protection. *Machine Learning and Knowledge Extraction*, 5(4):1382–1406, 2023. https://doi.org/10.3390/make5040070.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2, 2010. http://yann.lecun.com/exdb/mnist.

[LDGG18] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International symposium on research in attacks, intrusions, and defenses*, pages 273–294. Springer, 2018. https://doi.org/10.1007/978-3-030-00470-5_13.

[LFG+22] Bowen Li, Lixin Fan, Hanlin Gu, Jie Li, and Qiang Yang. Fedipr: Ownership verification for federated deep neural network models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4521–4536, 2022. https://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3195956.

[LJLX22] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1):5–22, 2022. https://doi.org/10.1109/TNNLS.2022.3182979.

[LLZ+23] Yifan Lu, Wenxuan Li, Mi Zhang, Xudong Pan, and Min Yang. Mira: Cracking black-box watermarking on deep neural networks via model inversion-based removal attacks. *arXiv preprint arXiv:2309.03466*, 2023. https://doi.org/10.48550/arXiv.2309.03466.

[LMA+17] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017. https://doi.org/10.14722/ndss.2018.23291.

[LMBL20] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 182–199. Springer, 2020. https://doi.org/10.48550/arXiv.2007.02343.

[LMPT20] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing*

and Applications, 32:9233–9244, 2020. https://doi.org/10.1007/s00521-019-04434-z.

[LMR23] Isabell Lederer, Rudolf Mayer, and Andreas Rauber. Identifying appropriate intellectual property protection mechanisms for machine learning models: A systematization of watermarking, fingerprinting, model access, and attacks. *IEEE Transactions on Neural Networks and Learning Systems*, 2023. https://doi.org/10.1109/TNNLS.2023.3270135.

[LPR17] David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[LSY+21] Xiyao Liu, Shuo Shao, Yue Yang, Kangming Wu, Wenyuan Yang, and Hui Fang. Secure federated learning model verification: A client-side backdoor triggered watermarking scheme. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2414–2419. IEEE, 2021. https://doi.org/10.1109/SMC52423.2021.9658998.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[LWL22] Fang-Qi Li, Shi-Lin Wang, and Alan Wee-Chung Liew. Watermarking protocol for deep neural network ownership regulation in federated learning. In *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–4. IEEE, 2022. https://doi.org/10.1109/ICMEW56448.2022.9859395.

[LWW+21] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021. https://doi.ieeecomputersociety.org/10.1109/TKDE.2021.3124599.

[LXZ+20] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2088–2105, 2020. https://doi.org/10.1109/TDSC.2020.3021407.

[LZS+18] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*, 2018. https://doi.org/10.48550/arXiv.1808.10307.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. https://doi.org/10.48550/arXiv.1602.05629.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[Nes83] Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, page 543, 1983.

[NT21] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021. https://doi.org/10.48550/arXiv.2102.10369.

[OMR23] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Computing Surveys*, 55(14s):1–41, 2023. https://doi.org/10.1145/3595292.

[Pet11] Fabien A. P. Petitcolas. *Kerckhoffs' Principle*, pages 675–675. Springer US, Boston, MA, 2011.

[Pol64] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR computational mathematics and mathematical physics*, 4(5):1–17, 1964. https://doi.org/10.1016/0041-5553(64)90137-5.

[PT20] Steve Povolny and Shivangee Trivedi. Model hacking adas to pave safer roads for autonomous vehicles. *McAfee Advanced Threat Research*, 2020.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015. https://doi.org/10.1007/s11263-015-0816-y.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. ttps://doi.org/10.1145/359340.359342.

[SAMA21] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 4417–4425, 2021.

[SCZZ19] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019. https://doi.org/10.1109/TCYB.2019.2950779.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013. https://dl.acm.org/doi/10.5555/3042817.3043064.

[SYG+24] Shuo Shao, Wenyuan Yang, Hanlin Gu, Zhan Qin, Lixin Fan, and Qiang Yang. Fedtracker: Furnishing ownership verification and traceability for federated learning model. *IEEE Transactions on Dependable and Secure Computing*, 2024. https://doi.org/10.1109/TDSC.2024.3390761.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. https://doi.org/10.48550/arXiv.1409.1556.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. https://doi.org/10.48550/arXiv.1312.6199.

[TXMA21] Buse GA Tekgul, Yuxi Xia, Samuel Marchal, and N Asokan. Waffle: Watermarking in federated learning. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 310–320. IEEE, 2021. https://doi.org/10.1109/SRDS53918.2021.00038; pre-print with more details on arXiv https://doi.org/10.48550/arXiv.2008.07298.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016. https://dl.acm.org/doi/10.5555/3241094.3241142.

[UNSS17] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pages 269–277, 2017. https://doi.org/10.1145/3078971.3078974.

[WFL+21] Kuan-Chieh Wang, Yan Fu, Ke Li, Ashish Khisti, Richard Zemel, and Alireza Makhzani. Variational model inversion attacks. *Advances in Neural Information Processing Systems*, 34:9706–9719, 2021.

[WYS+19] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. https://doi.org/10.1109/SP.2019.00031.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. https://arxiv.org/abs/1708.07747.

[YSY+22] Wenyuan Yang, Shuo Shao, Yue Yang, Xiyao Liu, Ximeng Liu, Zhihua Xia, Gerald Schaefer, and Hui Fang. Watermarking in secure federated learning: A verification framework based on client-side backdooring. *ACM Transactions on Intelligent Systems and Technology*, 2022. https://doi.org/10.1145/3630636.

[ZGJ+18] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian 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, 2018. https://doi.org/10.1145/3196494.3196550.

[ZJP+20] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 253–261, 2020. https://doi.org/10.1109/CVPR42600.2020.00033.