CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluation of Privacy-protected Federated Learning

Case studies of FedAvg and FedDyn

Master's thesis in Computer science and engineering

FARZAD BESHARATI

STEFAN CHAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Evaluation of Privacy-protected Federated Learning

Case studies of FedAvg and FedDyn

FARZAD BESHARATI
STEFAN CHAN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Computer Networks and Systems
Distributed Computing and Systems
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Evaluation of Privacy-protected Federated Learning
Case studies of FedAvg and FedDyn

FARZAD BESHARATI
STEFAN CHAN

Supervisor: Elad Schiller, Computer Networks and Systems
Co-Supervisor: Shiliang Zhang, Computer Networks and Systems
Examiner: Magnus Almgren, Computer Networks and Systems

Master's Thesis 2023
Department of Computer Science and Engineering
Division of Computer Networks and Systems
Distributed Computing and Systems
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Evaluation of Privacy-protected Federated Learning
Case studies of FedAvg and FedDyn
FARZAD BESHARATI
STEFAN CHAN
Department of Computer Science and Engineering
Chalmers University of Technology and University of GOthen

Abstract

Federated Learning is a form of distributed machine learning where training is performed at several participants and then aggregated on a central coordinator. However, federated learning itself is not secure and can not guarantee confidentiality for the data of participants. But the protection of privacy-sensitive data can be achieved through Homomorphic Encryption methods such as the Cheon-Kim-Kim-Song schema (CKKS). In this report, we study two federated learning methods, Federated Averaging (FedAvg) and Federated Learning Based on Dynamic Regularization (FedDyn), and apply CKKS to them to study the impact of securing such methods. We implement the federated learning methods with and without CKKS and also create a centralized implementation for comparison. Due to the nature of the Homomorphic Encryption scheme, a minor alteration to FedDyn was done to secure it for our implementation. In addition, we implement a selective global invocation through accuracy criteria. Evaluating the implementations yields that securing the federated learning methods increases the running and convergence time with a minor difference in model performance. Unsecured FedAvg and FedDyn take on average 130 and 150 seconds per global round, compared to 510 and 700 seconds for secured FedAvg and FedDyn. While model performance may not be affected to a noticeable degree, the number of global rounds to convergence is higher for the secured federated methods compared to the unsecured ones. Unsecured FedAvg and FedDyn take on average 27 and 21 global rounds to reach convergence, compared to 37 and 27 for secured FedAvg and FedDyn. Overall, privacy protection through Homomorphic Encryption can be implemented on federated learning methods without major alteration and overall performance penalty, but with extra memory costs and time.

Keywords: Machine Learning, Federated Learning, Encryption, Homomorphic Encryption, CKKS, FedAvg, FedDyn, Privacy, Evaluation

Acknowledgements

First we'd like to thank our supervisor Elad Schiller and our co-supervisor Shiliang Zhang, we appreciate the patience and support extended to us throughout the course of this thesis even in the uncertain early parts of the project. We would like to extend additional gratitude to Shiliang for not only providing us with access to his federated learning implementation in MATLAB, whose model he converted to Python for us to use, but also gave us a preliminary evaluation and also continued to aid the project after his time at Chalmers. Moreover, we'd like to thank our examiner Magnus Almgren taking on this project. We would also like to thank Jonathan Köre and Gustav Häger for helping us understand FedDyn.

Farzad Besharati, Gothenburg, August 2022

Stefan Chan, Gothenburg, August 2022

Contents

1
Introduction

1.1 Background

As the digitization of current society increases, so does the availability of data that is created and collected. Facebook trains its models for image recognition on data sets that can contain 3.5 billion images [1]. Processing such vast amounts of data by hand is considered unfeasible, so we look towards computationally more common methods. Machine learning (ML) is seen as a ubiquitous tool for finding patterns given a large amount of data, ML methods can through the use of statistical techniques extract patterns from data in an automated fashion. Coupled with increasing capabilities of computation, data sets consisting of millions of individual entries can be processed in a couple of weeks [1].

Machine Learning has been successfully applied to other areas of image recognition, such as speech recognition and natural language processing. The proliferation of ML can be attributed to the increasing capability of computation and refinement of algorithms used in ML [2]. However, the primary reason for the widespread usage of ML is considered to be the availability of data that can be used to train models on.

Currently, the main approach to performing ML is to collect relevant data and store it centrally, where it is then trained using purpose-built hardware. This allows for ease of access and simplifies logistics for supplying data to train some models. One concern with this approach is that the data that is being collected can involve private data, making the data collection a breach of privacy. A prominent example is medical data, which often contains information that users may not want to share with companies. Moreover, collecting the data itself might be infeasible or hard to achieve. Data generated by autonomous cars are stated to require a bandwidth up to 40 Gbit/s [3].

A possible alternative is to decentralize the learning process. A popular decentralized learning method is federated learning, which is a form of collaborative learning where the training of models is done in a decentralized manner. This is done by having the parts of the learning process concerning data done by the users that are participating. The users use their local data to compute model improvements which are sent in place of their data to an aggregator, allowing a group of users to collaboratively train a model without exposing their data to each other.

However, while federated learning allows for training without sharing data with the other users, studies have shown that it is still possible for the aggregator to infer information about the data of the users from sent model parameters [9, 24]. One method to preserve the privacy of a user's data is through the use of cryptography, in this thesis we focus on the form of Homomorphic Encryption [9].

1.2 Research Questions

This thesis aims to evaluate the costs and impact of learning in a distributed environment compared to a centralized one. We want to know whether using distributed learning compared to centralized learning affects model performance in accuracy, execution time, and communication cost. In addition, we want to compare the same criteria and see what the extent of the impact that privacy protection has on learning is.

1.3 Task Description

The task for this thesis is to answer the research questions stated above empirically via pilot implementations of federated methods. The implementations will range from basic pilots, to prototype implementations incorporating privacy protection with the aims of advancing the state of the art in the field.

1.4 Related Work

A well-known federated method is *Federated Averaging* (FedAvg) [5]. Proposed by McMahan et al. the method is a means to collaboratively train a machine-learning model. The main feature of the method is the reduction of communicated data compared to other similar methods, such as distributed Stochastic Gradient Descent (SGD) while at the same time retaining model performance.

Federated Learning Based on Dynamic Regularization is an alternative federated method that aims to perform more computations on the participant's side to save on data transmission [7]. It is noted that the minima of a participant's empirical loss are different from the empirical loss of a coordinator. Through the use of a *Dynamic Regularizer* on each participant, the empirical loss on both sides is brought closer together, converging on the global minima.

A recent proposed method by Zhang et al. is the acceleration of training by introducing accuracy criteria [23]. Instead of the users training for a set number of epochs in a federated setup, the users instead train their models until it reaches a certain accuracy goal before proceeding to aggregation. By setting the goal of the training as an accuracy criteria, the method is able to capitalize on early accuracy gains and reduce time spent on training to reach a certain level of accuracy.

We note that the studied methods in this thesis are not the only ones known for protecting privacy, see for example [25, 26, 27].

1.5 Our Contribution

In this project, we aim to evaluate state of the art open-source software related to federated learning. In particular, we focus on horizontal federated systems and study FedAvg and FedDyn, where the latter is considered to be state of the art. We also study the impact of the chosen cryptographic method that facilitates privacy preservation in federated learning.

We study important questions in the area of distributed machine learning via pilot implementations for FedAvg and FedDyn with selective global round invocation via accuracy criteria. Our answers are able to evaluate the state of the art with and without privacy protection with respect to accuracy, execution time, and communication costs. Our work also supports the advancement of the state of the art. As mentioned in Section 1.4, there are existing evaluations for FedAvg and FedDyn. However, to the best of our knowledge, there are no existing implementations of FedDyn with privacy protection. We are also the first to offer an implementation for FedDyn with selective global round invocation via accuracy criteria.

With the publication of this document, we will also release the code via a public repository for supporting further research in this area. In total, we release the code for our implementation of FedAvg and FedDyn with and without privacy protection, as well as a centralized implementation.

We evaluate our implementations through selected criteria in the form of model accuracy, rounds to convergence, bandwidth usage, and execution time. We set up five experiments to evaluate the performance of the federated methods against a centralized setup, and to gauge the impact that occurred by the introduction of privacy protection. The experiments are performed on a computing cluster provided by the Swedish National Institute of Computation (SNIC) [22].

To summarize the results, there are not any major changes in the final accuracy between secured and unsecured implementations, as all methods manage to reach 99% accuracy. Secured implementations do however have longer to convergence compared to unsecured versions. For raw time, Unsecured FedAvg and FedDyn take on average 130 and 150 seconds per global round, compared to 510 and 700 seconds for secured FedAvg and FedDyn, making the secured versions almost four times slower. Similarly, it takes 27 and 21 global rounds to reach convergence for unsecured FedAvg and FedDyn, compared to 37 and 27 global rounds for secured FedAvg and FedDyn. Bandwidth usage between the secured and unsecured implementations is almost identical, with a total of 4185 KB for secured and unsecured FedAvg, 3167 KB for secured FedDyn, and 2092 KB for unsecured FedDyn. A more in-depth detailing of the results can be seen in Chapter 5.

The inclusion of privacy protection has shown to be viable in federated learning with no major deficiencies in performance, but introduces considerable time cost. In addition, the methods using privacy protection required a significant increase in memory usage. This drawback makes this method of privacy protection impractical in environments where these resources are not readily available, such as IoT. The introduction of the global round invocation led to reduced training time, but it is still inconclusive how much execution time is saved with this method.

We believe that the results presented in this report can be used by data-oriented security-aware professionals that aim to preserve the confidentiality of user data, as well as highlighting improvements within federated learning with privacy protection.

2
Theory

In this chapter, we will introduce the main concepts used in this thesis.

2.1 Machine Learning

For this thesis, machine learning (ML) is a central part of its topics. In this section, we will introduce various concepts as well as our primary learning model.

2.1.1 Centralized Learning

Centralized Learning (CL) is a common method of ML where data is aggregated into a single dataset and stored on a central server. The role of the central server is to perform the computation that is required for model training.

2.1.2 Federated Learning

Federated Learning (FL) is a method of ML where, unlike Centralized Learning which trains a model on a single device and a single dataset, model training is instead performed over several devices with each one having their own models and datasets. At the centre of a Federated Learning method lies a *Coordinator* server which, as the name implies, coordinates a set of *Participant* devices in training the model. Training often proceeds until a predefined amount of *Global Rounds* have been reached, or until the coordinator stops the training, for example, if an accuracy goal has been reached. Training during a global round consists of participants training their models locally using their datasets for several local rounds, denoted as *Epochs*. Algorithm 1 shows an overarching view of how a coordinator can act in a federated system, as presented in [4].

2.1.2.1 FedAvg

In Federated Averaging (FedAvg), each participant separately trains their own model and uploads it to a coordinator. The coordinator computes the weighted average of the transferred gradients or model parameters and updates the central model. Finally, the central model is sent back to each participant so that their models can be updated. [5] A simple visualization of the communication between coordinator and participants can be seen in figure 2.1.

Algorithm 1 Overview of coordinator for a federated learning system

1: Coordinator initializes model and broadcasts weights to all participants
2: **for** $round \leftarrow 1, roundMax$ **do**
3: Coordinator determines *selected* set, a permutation of all participants
4: **for** *participant* $k \in$ *selected* **do** in parallel
5: *participant* trains its model and sends back model weights w_{t+1}^k to coordinator
6: **end for**
7: Coordinator aggregates all received model weights w_{t+1}^k together into w_{t+1}
8: Coordinator broadcasts new weights w_{t+1} to all participants
9: **end for**

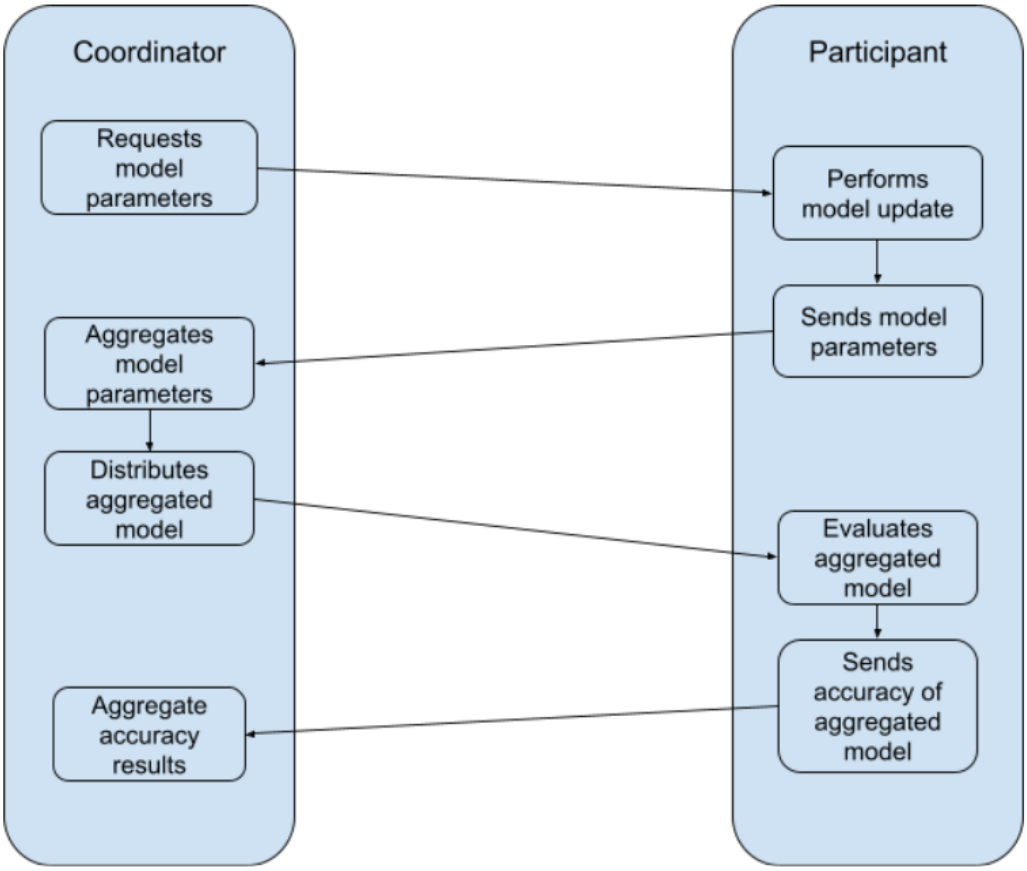


Figure 2.1: General communication between the coordinator and the participant during a training round for FedAvg

2.1.2.2 FedDyn

Federated Learning based on Dynamic Regularization (FedDyn) is a novel federated learning method that incorporates a method known as *Dynamic Regularization* to minimize communication while retaining model performance [7].

The method dynamically modifies the learning objective each global round such that the model parameters converge on the global minima loss. This is done by introducing linear and quadratic penalty terms to the model loss each round, where the minima of the terms are consistent with the minima of model loss. This leads to improvement in performance and time to model convergence.

The method introduces new steps and variables to the federated learning structure. The aforementioned linear and quadratic penalty terms are computed using a local gradient $\nabla L_k(w_k)$ by the participants during training [7]. At the coordinator's side, the variable h, called *Server state*, is computed using aggregated model parameters sent by the participants. *Server state* is used to compute the global model parameters to ensure that the global model is converging towards a minimum loss. A simple visualization of the communication between coordinator and participants can be seen in figure 2.2.

2.1.3 Accuracy Criteria

During a participant's training step as it happens in line 5 in Algorithm 1, and often in the early global rounds, a selected participants model can reach a higher accuracy than what it had before it started training in fewer training epochs than the predefined amount. During ordinary training, this would not change much, as training would continue until it has reached the predefined amount of epochs. In distributed training, given the parallel nature of multiple participants, if all participants reach a *accuracy criteria*, then the total training time required could be shortened. The method and algorithm presented in this section are based on private communications with Shiliang Zhang and Elad Schiller [23].

To check if a model has reached an adequate improvement in performance, the following equation is used:

$$f(acc) = acc + 0.4 - 0.4 \cdot acc^{1.5} \tag{2.1}$$

Where $f(acc)$ is the resulting accuracy criteria that is to be reached and acc is the accuracy of the model before training. An example implementation is shown in Algorithm 2.

Before model training, an accuracy criteria is computed using the accuracy of the local model at that time. During training, at intervals defined by $INTERVAL$ where $INTERVAL \in Z^+$, the local model is evaluated for accuracy. If the accuracy is equal to or higher than the accuracy criteria, training ends and the function returns. Should the accuracy not be high enough to fulfill the criteria, the training continues until either the accuracy criteria is fulfilled, or $epoch = epoch\_max$.

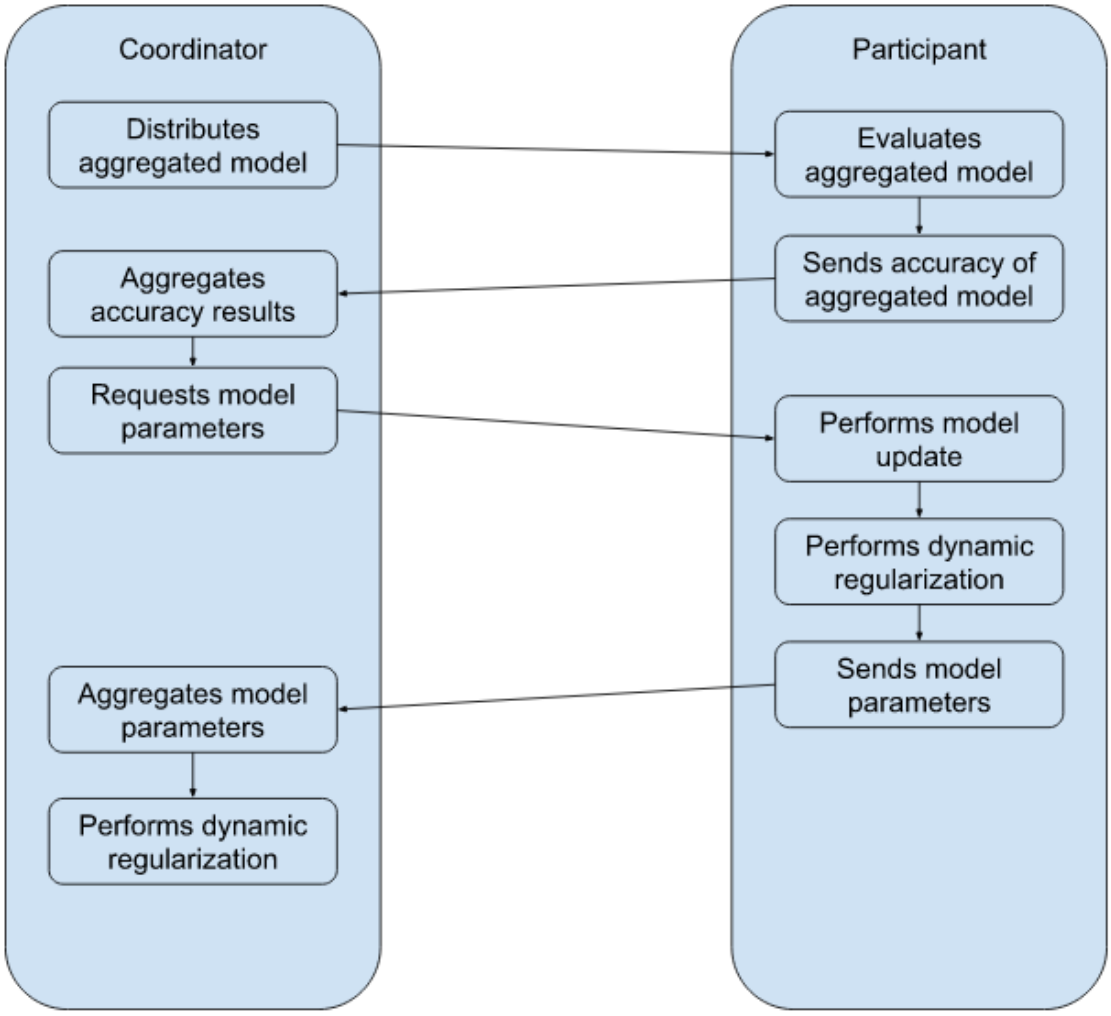


Figure 2.2: General communication between the coordinator and the participant during a training round for FedDyn

Algorithm 2 An example implementation of accuracy criteria

```
1: function update_participant(model)
2:     base_acc ← test(model)
3:     acc_criteria ← base_acc + 0.4 − 0.4 · base_acc^1.5
4:     for epoch ← 1, epoch_max do
5:         model ← train(model)
6:         if INTERVAL divides epoch without remainder then
7:             trained_acc ← test(model)
8:             if trained_acc ≥ acc_criteria then
9:                 return
10:            end if
11:        end if
12:    end for
13: end function
```

2.1.4 Long Short-Term Memory

For the final part of this section, we'll briefly cover the network model used in this thesis. Recurrent neural networks (RNN) can use contextual information as inputs. A problem, however, is that the influence of input can cause the output to either decay or suffer an exponential increase as it travels through the hidden layer. This is referred to as the *vanishing gradient problem* [8]. Long short-term memory networks (LSTM) are one of several architectures that can solve this problem

An LSTM network is the same as an RNN network, with the difference being that each node in the hidden layer has been replaced by a "memory block". Each block can be thought of as memory in a computer, containing one memory cell and three multiplication gates in charge of input, output, and forgetting. The three gates allow the memory cell to store and remember data over long periods, and as such solve the vanishing gradient problem [8].

2.2 Privacy Protection: Homomorphic Encryption (CKKS)

Distributed learning assumes an honest and non-curious coordinator and participants. However, if the coordinator were to be corrupted, dishonest or curious, then the gradient from each participant can be disclosed, we assume this to be the case for this thesis. Even though raw training data is difficult to find from a disclosed gradient, it is still possible to infer a large amount of information [9]. In order to preserve the privacy of training data, it must be properly encrypted before being sent from a client to a coordinator. Encrypted data however cannot normally be worked on unless using a method designed for that purpose, such as *Homomorphic Encryption*.
Homomorphic encryption (HE) is a type of encryption which allows computation on encrypted data without requiring decryption. It allows for the data to remain

private and confidential while being processed, allowing useful tasks to be performed with data in non-private environments [10]. This has lead to HE being proposed to several different fields where the privacy offered by HE is beneficial or necessary. Some examples are medical applications [11], image recognition [12] and financial privacy [11].

HE, depending on its implementation, can support a varying number of computations on encrypted data. The computations are implemented in the encryption schemes as binary gates, which form into arithmetic or Boolean circuits. An encryption scheme can be considered to be a certain type depending on the number of gates it supports as well as the complexity of the arithmetic circuits [13, 10]:

(i) Partial Homomorphic Encryption (PHE) supports circuits consisting of one type of gate.

(ii) Somewhat Homomorphic Encryption (SWHE) support circuits consisting of multiple types of gates with a limit to the size of the circuit.

(iii) Levelled Homomorphic Encryption (LHE) support circuits consisting of any type of gates with a limit to the size of the circuit.

(iv) Fully Homomorphic Encryption (FHE) supports circuits consisting of any type of gates with no limit to the size of the circuit.

A consequence of homomorphic computations on encrypted data is an increase in data size. Brakerski Zvika and Vinod Vaikuntanathan show in their paper that encrypted data of size $n+1$ increases to $\frac{n^2}{2}$ after one multiplication [14]. Because of the exponential increase in size, homomorphic computation in ML is impractical without effective means of keeping the size of encrypted data minimal. In the same paper, Brakerski Zvika and Vinod Vaikuntanathan present two methods, *Relinearization* and *Dimension-Modulus Reduction*, which keeps the size of encrypted data minimal. For a HE scheme to be suitable for ML, similar measures must exist. The Cheon-Kim-Kim-Song encryption scheme (CKKS) is a levelled HE scheme that implements addition and multiplication operations on encrypted data [15]. In particular, the encryption scheme supports the operations on both real and complex numbers, but yields approximate results instead of exact results. The efficiency of evaluating approximate numbers and rescaling makes the CKKS scheme a preferred approach in implementing privacy preservation in ML [16].

The CKKS scheme [15] is based on the encryption scheme as defined by Brakerski Zvika and Vinod Vaikuntanathan [14]. It iterates upon the scheme by introducing *rescaling*, which both reduces the size of the ciphertext after a homomorphic operation and the error introduced. In addition, the CKKS scheme also implements techniques that reduces computation time in a computation environment. One example is a batching technique, which reduces the time for homomorphic evaluation of a logistic function [15].

3

Methods

In this section, we go over the methods and datasets used in implementing the theory. For the implementations presented in this thesis, we've elected to use the Python programming language as our method of implementation. This is due to the availability of the tools presented in this section and familiarity with the programming language on our part.

3.1 Implementation

For FedDyn, a deviation of the algorithm was necessary to introduce HE. Since the encryption scheme is a Levelled Homomorphic Encryption scheme, there is a certain depth or amount of times a homomorphic operation can be performed on encrypted data, which means that there is a predetermined limit on the number of computations that can be made on the same ciphertext. In the computation of *server state* h, the limitation of the number of operations in the encryption scheme makes it not possible for the coordinator to compute after a few global rounds. To make it possible to perform the computation, part of the computation which involved the model parameters was moved to be performed at the participant's side. A full description of the modified algorithm can be seen in Algorithm 3.

In the original FedDyn algorithm, the computation performed at line 8 would have been performed at the coordinator at line 11. By moving the computation to the participant's side, the computation can be done without the need for encryption. The coordinator then receives the encrypted h^k and can aggregate them to compute h_{t+1}. Since h^k is encrypted each round, the encryption stays fresh and does not reach the limit of operations during runtime.

3.2 Model Design

The model used in this thesis is a two-layer stacked LSTM model implemented with PyTorch. Inputs are fed into a two-layer hidden layer, each with 16 features, whose outputs are fed through a Linear transformation. Model parameters are shown in table 3.1.

Algorithm 3 The modified secured FedDyn algorithm

Coordinator initializes model
for $round \leftarrow 1, roundMax$ **do**
    Coordinator determines *selected* set, a permutation of all participants
    Coordinator broadcasts weights w_t to *selected*
    for *participant* $k \in$ *selected* **do** in parallel
        participant k trains its model and computes model weights w_{t+1}^k
        participant k computes $h^k = w_{t+1}^k - w_t$
        participant k encrypts and sends w_{t+1}^k and h^k
    end for
    Coordinator aggregates all received model weights w_{t+1}^k
    Coordinator aggregates all h^k and computes h_{t+1}
    Coordinator computes w_{t+1}
end for

Input Size (input_size)	7
Output Size (num_classes)	3
Hidden Layer Size (num_layers)	2
Number of Features per Hidden Layer (hidden_size)	16

Table 3.1: Model parameters used in creation, variable names in parentheses

3.3 Dataset

The dataset used for the learning process is an aggregation of datasets containing various travel statistics of vehicles belonging to one of three vehicle brands, General Motors (GM), Toyota, and Ford. Travel statistics make up the features of the model, a detailed listing of the features can be seen in table 3.2. They are organized into trips, where each trip consists of 30 snapshots of the features at that moment during the trip. Each dataset is partitioned into a training, validation, and test set with the portions of 80% for training, 10% for validation, and 10% for testing. The total number of trips gathered from each vehicle brand can be seen in table 3.3.

Table 3.2: Input features, measurements in parentheses where helpful

Vehicle Speed (km/h)
Mass Air Flow (g/sec)
Engine RPM
Absolute Engine Load (%)
Short Term Fuel Trim Bank (%)
Long Term Fuel Trim Bank (%)
Speed Limit (km/h)

Table 3.3: Number of trips from each vehicle brand in order of training, validation and testing

GM	18025	2253	2254
Toyota	16528	2066	2067
Ford	36006	4501	4501

Table 3.4: General parameters for training, variable names in parentheses

Global Rounds (num_rounds)	300
Epochs (epochs)	100
Alpha (alpha)	0.01
Sequence Length (sequence_length)	30
Batch Size (batch_size)	50
Learning Rate (learning_rate)	0.0001

3.4 Number of Participants

For this thesis, there are 12 total participants where 4 are selected for training each round. Twelve total participants were chosen as a way to see and show the effectiveness of federated methods on a decent number of participants.

Increasing parallelism, e.g. selecting a larger fraction of participants for each training step, has diminishing returns in performance compared to computational efficiency [5]. For this reason, a somewhat low fraction of total participants are chosen each training round. In our thesis, we have four selected participants to involve a decent fraction of the total participants. To handle the dataset, it's aggregated, shuffled, and then evenly split into twelve parts, one for each participant.

3.5 General Training Parameters

General training parameters are shown in table 3.4. Noteworthy parameters to mention are *Sequence Length* and *Alpha*. *Alpha* is a FedDyn-specific hyperparameter used in the dynamic regularization, gradient updating, as well as updating of the coordinators model [7]. *Sequence Length* is used for reshaping inputs to the model in training and testing.

3.6 Communications

In our implementations, coordinators and participants are implemented as parallel threads on the same computer. To simulate the distributed nature of federated learning, the participants and the coordinator perform their communication through web sockets. The coordinator and the participants each have their own socket, which is assigned to them at the start of the training. Participants are limited to only being able to communicate with the coordinator, and the coordinator can communicate

with all participants.

To implement the socket communication, we use the socket library ZeroMQ which is an asynchronous messaging library suitable for distributed communication [17]. Since our implementations are in Python, we specifically use the Python binding for ZeroMQ, called PyZMQ. The sockets use the *Request-Reply* communication pattern, which implements a continuous loop where for every message sent by a sender, the recipient must reply to the sender first before any other message can be sent. This pattern allows the coordinator to send signals or, more specifically, commands, to participants to trigger some function. For example, the coordinator can send a flag variable to some selected participants to make them perform different actions, such as updating their models, training their models, returning the model weights, or retrieving test data.

3.7 TenSEAL

For the homomorphic part of our implementations, we have elected to use TenSEAL, a Python-based library that is built on top of Microsoft SEAL [18]. An important feature of TenSEAL is the ability to perform homomorphic operations directly on tensors, which is a considerable benefit for the thesis. It allows for a non-complex integration of the learning and allows for computation to happen while preserving the tensor structure.

The central component of TenSEAL is the context [18]. The context is a data structure that is used to create and store encryption keys that are used for encryption, computation, and decryption. In our implementations, the participants use and share the same context so that their encrypted data can be computed by the coordinator. The coordinator has a similar context to the participant for the sake of performing computations but has the keys necessary for decryption removed. We initialize the context during the start of the training with the parameters detailed in Table 3.5.

Table 3.5: Parameters for the TenSEAL context, variable names in parentheses

Scheme type	ts.SCHEME_TYPE.CKKS
Modulus degree (poly_modulus_degree)	8192
Bit coefficients (coeff_mod_bit_sizes)	[31, 26, 26, 26, 26, 26, 26, 31]
Global Scale	2^{40}

The *Scheme type* specifies which encryption scheme will be used to encrypt data. *Modulus degree* is a parameter that is involved with the *Rescaling* feature of CKKS and sets the limit of *Bit coefficients*. Bit coefficients specify the numbers of bits for the various generated primes. The number of bit coefficients determines how many primes are generated with the first prime setting the precision of the decrypted result, it also sets the number of times Rescaling can be performed [18]. The sum of

the bit coefficients must be equal to or lower than the number of bits supported by the modulus degree. For our parameters, a modulus degree of 8192 allows for 218 bits, which is equal to the total number of bits in bit coefficients.

4
Evaluation

As the aim of our thesis, we wish to answer the following two questions:
(i) for a given application, to what extent do FedAvg and FedDyn offer improvements, and at what costs?
(ii) For a given application, at what costs does homomorphic encryption offer privacy protection?

To accomplish these goals we've settled on five model implementations: Centralized (and unsecured), Unsecured FedAvg, Unsecured FedDyn, Secured FedAvg and Secured FedDyn. To evaluate our implementations and gain insight into the impact incurred by the privacy-preserving measures, we provide an evaluation plan. In section 4.1, we present the relevant and interesting evaluation criteria. In section 4.2, the environment in which the evaluation will be performed is presented. Lastly, section 4.3 details the experiments that will be performed in the evaluation.

4.1 Evaluation Criteria

Our implementations are evaluated with a few different criteria.

The first is accuracy. Accuracy is a self-descriptive evaluation criteria, with plenty of total predictions, we wish to know the ratio of correct predictions. Given an accuracy, it can then be used to compare with other federated learning methods. Accuracy, together with loss, can also be measured during training, letting one see how efficient the model is and how long it takes to reach an acceptable threshold.

Our second criteria are rounds to convergence. For our work, implementing privacy preservation introduces a small error each time computation is done [15]. While training will eventually lead to a convergence of model performance that is just as effective as without encryption, the training itself will become slightly slower due to this inefficiency. For this reason, we draw comparisons for the number of rounds required for a model's performance to converge between encrypted and unencrypted models.

Additionally, by its implementation, federated learning aims at using a restrictive amount of communication when transferring model parameters between clients and servers, with variable cost depending on the size of the model. However, for the case of the CKKS encryption schema used in our implementations, the size of encrypted

data can be up to 50 times its unencrypted size [19]. It is therefore of interest to evaluate the traffic between the coordinator and the participants and compare it against unsecured implementations.

Finally, an additional criteria is the time taken to perform certain operations or entire training rounds. We measure this in discrete time-units, seconds, to simplify comparison.

4.2 Evaluation Environment

The evaluation of the implementations will be done on a computation cluster provided by the Swedish National Institute of Computation (SNIC). The computation cluster, named Alvis, hosts a multitude of graphical processing units (GPUs) acceleration cards which form into several different types of computation nodes, each containing multiple GPUs [22]. The computation nodes run two 16-core Intel(R) Xeon(R) Gold 6226R CPU @2.90GHz for a total of 32 cores with access to a minimum of 576 GB DDR4 random access memory (RAM). Each computation node also has access to at least 376 GB of SSD for long-term storage. The GPUs are used for the training of the model and the CPUs are used for the HE operations. The computation node that is used for evaluating the implementations has one NVIDIA V100 GPU.

4.3 Experiment Plan

We detail in this section the experiment plans that we work on for this thesis.

4.3.1 Centralized vs Unsecured Distributed

When it comes to comparing centralized versus distributed models, we aim to perform two experiments, one on average final accuracy, and one on rounds required to reach an accuracy goal. It is of interest to see how the distributed approach in federated learning affects the model accuracy after several iterations. For that reason, a centralized implementation utilizing the same model and dataset is used to establish a baseline that can be compared to.

4.3.1.1 Experiment 1: Comparing Accuracy Between Centralized and Distributed

To evaluate the accuracy, the model of the implementations is trained and tested using the same dataset for several global rounds. This yields the test accuracy of the implementations for each round, which are then compared.

4.3.1.2 Experiment 2: Comparing Rounds to Convergence Between Centralized and Distributed

To evaluate the number of rounds it takes for the implementations to reach some convergence, we establish an accuracy goal of 99% which we use as a convergence point. We measure the number of global rounds it takes for the implementations to reach this point and draw conclusions from the results. A global round for the centralized implementation can be considered as the equivalent number of epochs performed by a participant during one global round. For example, the equivalent of 10 global rounds for the centralized implementation would be 1000 epochs.

4.3.2 Secured vs Unsecured

For the comparisons between secured and unsecured, we aim to perform three experiments. The first experiment compares overall final accuracy similar to the experiment performed in 4.3.1.1. The second experiment compares overall time efficiency between the two, we aim to see how much extra time it takes to add encryption. Lastly, the third experiment compares bandwidth between the two.

4.3.2.1 Experiment 3: Comparing Accuracy Between Secured and Unsecured

Since one of the primary drawbacks of the CKKS scheme is the introduction of a small error on the model parameters during training, it is of interest to see the impact. To do this, we run the secured and unsecured implementations for several global rounds and compare the results to gauge the impact that occurred by HE. In particular, we are interested in the number of global rounds it takes for the implementations to reach an accuracy convergence, as well as the maximum accuracy.

4.3.2.2 Experiment 4: Comparing Time Efficiency Between Secured and Unsecured

Through preliminary tests of the TenSEAL library, we noticed that the encryption and decryption operations take substantially more time to perform than expected, for that reason we're interested to see how much more time a fully secured program takes compared to an unsecured program. We measure the time efficiency between the unsecured and secured implementations by timing the various processes that are performed during a global round. The secured implementations have additional processes during a global round that has to be performed compared to the unsecured implementations. These additional processes are the encryption and decryption of the model parameters. In addition, homomorphic operations on the encrypted model parameters take more time to perform compared to normal mathematical operations on unencrypted data [15]. It is therefore of interest to compare time efficiency between the secured and unsecured implementations to determine the impact of HE regarding time cost.

4.3.2.3 Experiment 5: Comparing Bandwidth Usage Between Secured and Unsecured

When encrypting text or data, ideally the file size of that data should only increase minimally, however this is often not the case in CKKS based encryption [19]. Given that encryption affects the size of data to the point where considerable hardware resources are required, it is of interest to see whether there is an increased cost in data transfer tied to it.

During execution time of the secured and unsecured implementations, we measure the size of the bandwidth which passes through the communication sockets between the coordinator and the participants. We measure specifically when the coordinator receives model parameters for aggregation from the participants and when the coordinator sends the aggregated model parameters to the participants. This is done by measuring the size of the serialized model parameters before it is sent through a socket. The total data sent for the implementations are summarized and then compared to conclude.

4.3.3 Accuracy Criteria

One improvement for the federated implementations we propose in Section 2.1.3 is the Accuracy Criteria. The core idea of the improvement is to speed-up training by changing the objective from training a certain period of time to instead reach an accuracy goal. This allows the results of the training to be aggregated early and makes it possible to reach a high level of performance in a shorter time compared to just training the full amount of local epochs. However, the implementation of the accuracy criteria has an impact on the number of epochs spent in each round, which can cause the time saved to be later spent by running the method. Due to the nature of how model training is performed, Accuracy Criteria is inconsistent in the amount of time it takes to reach the set accuracy goal. It is therefore of interest to see how much of an impact the accuracy criteria has on the time and number of epochs to reach a certain level of performance. Note that this experiment was not completed and is only shown here as a possibility for future work.

4.3.3.1 Experiment 6: Comparing Epochs to Convergence Between Implementations With and Without Accuracy Criteria

The time saved using accuracy criteria can vary substantially, either resulting in a great reduction of execution time or none at all. To properly evaluate the impact of accuracy criteria, average values will have to be retrieved by running programs several times over.

We define two sets of implementations. The first set is all the federated programs, including secured/unsecured variants, but without accuracy criteria. The second set is all the federated methods, but this time with accuracy criteria, with the convergence defined at a range of 0.95-0.99 (95-99%). For each program in each set, run that program as normal but for a reduced number of global rounds, around 50,

leaving the remaining general parameters unchanged, as seen in table 3.4. After each program has finished its 50 global rounds, save the following data - rounds to convergence, average time per round, and total time to convergence. Repeat the process of running a program until a sizable amount of each data point has been collected, at least 50 of each. Finally, average each data point to receive the average rounds to convergence, average time per round, and average total time to convergence for each program. Comparisons can now be done between the programs with accuracy criteria and those without.

Our expectations are that the implementations with accuracy criteria will reach their convergence with less total time compared to the implementations without accuracy criteria, but not necessarily in less rounds. Additionally, due to its simple implementations the checks for accuracy criteria will persist even after convergence has been reached, for this reason we suspect that there might be a slight increase in average round time, which is why it is saved as a data point.

5
Results and Discussion

In this chapter, we go over the first five experiments introduced in chapter 4. For each experiment, we write our expectations, present the data, and then comment on whether the data supports or goes against our expectations, and if so, why that might be the case. Additionally, we discuss the ethical implications and the sustainability of our work.

5.1 Experiment Results

Quickly summarized, the results show that the difference in model efficiency is minimal, while the drawbacks lie more in time efficiency. The maximum accuracy of the various implementations is within 0.5 percent points of each other, with the centralized implementation having the highest maximum accuracy. However, the centralized method is able to converge on the maximum accuracy in 2 global rounds compared to the distributed implementations of at least 20 or more.

The secured implementations were able to show similar or, in the case of secured FedAvg, higher performance than the unsecured implementation. Results show that the drawback from using HE is in the execution time, the secured implementations have more than four times the execution time of the unsecured implementations. Lastly, the bandwidth usage of the distributed implementations shows that at most, a total of 4186 KB over 300 global rounds is communicated between the participants and the coordinator.

5.1.1 Experiment 1: Comparing Accuracy Between Centralized and Distributed

For this experiment, we expect the distributed implementations to be less precise in maximum accuracy compared to the centralized implementation. We expect this since the distributed approach creates a shared global average of the model using the parameters sent by the participants. It is likely some participants may have worse performing models compared to others, which in turn can negatively impact the shared global average, making the distributed approach not able to reach the same level of performance. In addition, the centralized model can utilize the entire dataset for training, which would lead to it having a higher performance than the distributed models.

We measure the test accuracy and loss at the end of every global round, except the FedDyn variants which measure at the start, and display the results in figure 5.1. Table 5.1 shows the maximum test accuracy of the implementations and the number of global rounds it took. While all implementations reach and maintain a high level of test accuracy throughout the global rounds, unsecured FedDyn experiences a dip in performance in the interval between global round 150-250.

All implementations were able to reach an accuracy higher than 99%. As expected, the centralized implementation had the highest accuracy, followed by the FedAvg and FedDyn implementations. While FedDyn is slightly behind FedAvg in accuracy, it was able to reach its maximum accuracy in fewer global rounds.

Implementation	Maximum Accuracy	Global Round
Centralized	0.995	33
Unsecured FedAvg	0.993	141
Unsecured FedDyn	0.992	42
Secured FedAvg	0.995	204
Secured FedDyn	0.992	31

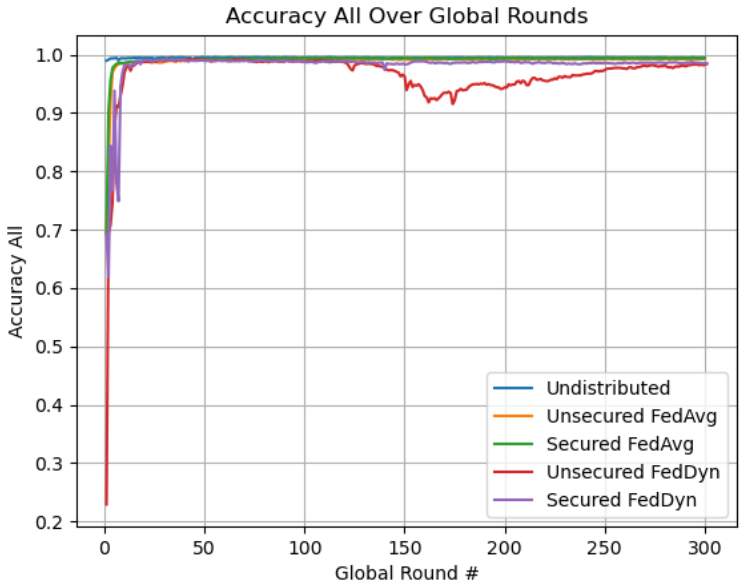
Table 5.1: The maximum test accuracy and the global round it was achieved by the implementations

5.1.2 Experiment 2: Comparing Rounds to Convergence Between Centralized and Distributed

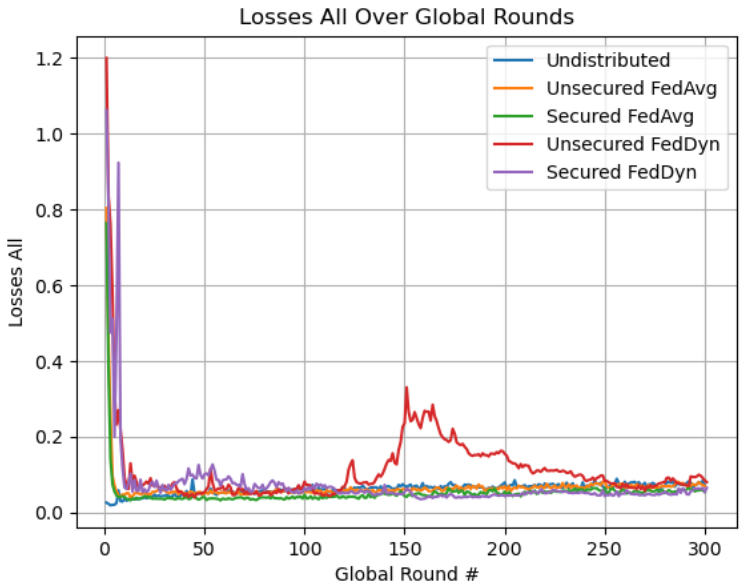
In this experiment, we expect the distributed implementations to be able to achieve convergence in a similar, but not fewer, number of rounds than the centralized implementation. Preliminary tests have shown that the distributed implementations with accuracy criteria can reach a high level of accuracy in a couple of rounds and converge on maximum accuracy. However, the centralized implementation can utilize the entire dataset for training. We expect the centralized implementation reaches convergence in fewer rounds compared to the distributed implementations. For the distributed implementations, we expect that the unsecured implementations to have a higher performance and therefore reach convergence in fewer rounds than the secured implementations.

Detailed in Table 5.2 is the number of epochs for the centralized implementation to reach convergence and the number of global rounds for the distributed implementations. Table 5.3 shows the execution time for the implementations to reach convergence.

Comparing the number of rounds and execution time shows that the centralized implementation reaches convergence in a far shorter time compared to the distributed implementations. This follows our expectation and shows that the distributed models have poorer accuracy gains per epoch compared to the centralized implementation.



(a) The model accuracy of the implementations over 300 global rounds.



(b) The model losses of the implementations over 300 global rounds.

Figure 5.1: The test accuracy's and losses of the models trained by the implementations.

For the distributed implementations, the unsecured implementations reached convergence in fewer rounds compared to their secured counterparts. While the difference in global rounds is not significant, the difference in execution time seen in Table 5.3

is considerably large.

Implementation	Global Rounds
Centralized	2
Unsecured FedAvg	27
Unsecured FedDyn	21
Secured FedAvg	37
Secured FedDyn	27

Table 5.2: The number of global rounds for the implementations to reach convergence

Implementation	Time
Centralized	640
Unsecured FedAvg	4300
Unsecured FedDyn	2700
Secured FedAvg	14000
Secured FedDyn	14000

Table 5.3: The execution time in seconds for the implementations to reach convergence

5.1.3 Experiment 3: Comparing Accuracy Between Secured and Unsecured

Overall, we expect minimal, if no changes, between unsecured and secured versions when it comes to the final accuracy. CKKS does introduce slight inaccuracies during encryption, however, these are for the most part negligible. In the worst case, this could lead to accuracy differences in < 1 percent units.

The accuracy's of the secured implementations can be seen in Figure 5.1 and Table 5.1. As we can see, Secured FedAvg has the highest maximum accuracy of the distributed methods, comparable to the centralized version. Overall, these results were expected.

5.1.4 Experiment 4: Comparing Time Efficiency Between Secured and Unsecured

We expect that the secured implementations will take drastically more time per global round compared to the unsecured versions. This is not only due to the encryption process taking time, but serialization of the encrypted data and HE operations will also take time. Moreover, the time for training participant models should take about the same time, this is since the encryption methods aren't used while locally training.

Timing the various processes for secured and unsecured implementations shows that the secured implementations have a higher average time than the unsecured implementations. In Table 5.4, we can see that secured implementations take on average more than four times as long to execute a global round compared to the unsecured implementations. Table 5.5 and Table 5.6 shows the average time for the processes for FedAvg and FedDyn respectively. The processes which take the most time is the serialization, decryption, and aggregation. These processes during runtime have to handle or perform operations on encrypted data, which takes considerable time [15].

Contrary to our expectations, the training of the local models on average takes more time for the secured implementations. Furthermore, the decryption time for secured FedAvg is longer than secured FedDyn. Since the algorithm for FedAvg is defined such that the aggregated model parameters are distributed to all participants, the number of parallelizable threads in the evaluation environment available to each participant is fewer compared to the FedDyn method. This means that the decryption process in secured FedAvg takes more time to perform.

Implementation	Avg. time per round
Unsecured FedAvg	130
Unsecured FedDyn	150
Secured FedAvg	510
Secured FedDyn	700

Table 5.4: Average time per global round in seconds for the distributed implementations

Implementation	Training	Encryption	Serialization	Decryption	Aggregation
Unsecured FedAvg	110	-	-	-	0.0006
Secured FedAvg	160	3.7	24	91	56

Table 5.5: Average time in seconds for the processes for the FedAvg implementations

Implementation	Training	Encryption	Serialization	Decryption
Unsecured FedDyn	150	-	-	-
Secured FedDyn	240	3.9	25	34

Implementation	Regularization	Aggregation
Unsecured FedDyn	35	0.009
Secured FedDyn	52	57

Table 5.6: Average time for the processes for the FedDyn implementations

5.1.5 Experiment 5: Comparing Bandwidth Usage Between Secured and Unsecured

In section 4.3.2.3 we mentioned that the CKKS encryption scheme increases the data size of a model's parameters. The increase of data size would mean that bandwidth will be higher compared to the unsecured implementations when the model parameters are serialized to be aggregated. As such, we expect that the secured implementations will have a higher bandwidth usage compared to the unsecured implementations due to the increase in data size caused by HE.
Additionally, we expect that the data size sent by all FedAvg versions to be higher than the FedDyn versions. This is because while FedDyn updates only the selected participants, FedAvg updates all participants.

In figure 5.2 we present the bandwidth usage of each method and in table 5.7 we show the final total amount.

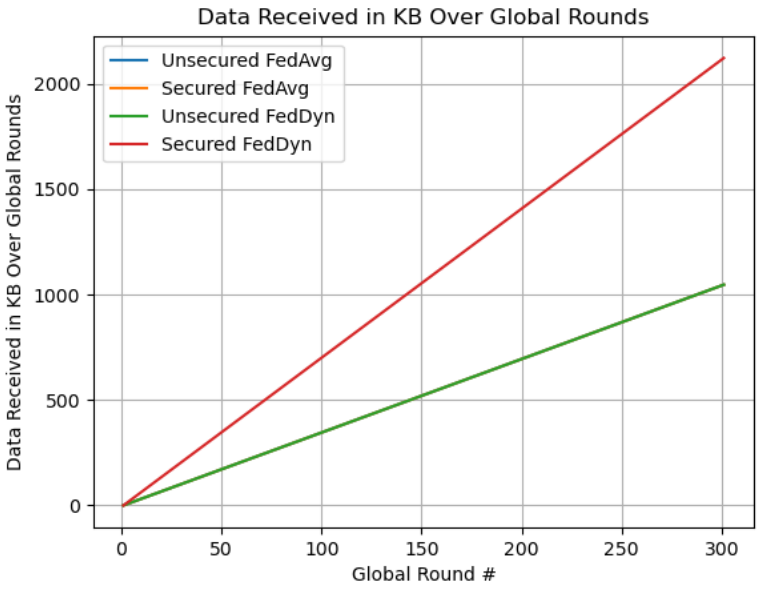
Implementation	Data Received	Data Sent
Unsecured FedAvg	1000	3100
Secured FedAvg	1000	3100
Unsecured FedDyn	1000	1000
Secured FedDyn	2100	1000

Table 5.7: Final data usage of the different versions.

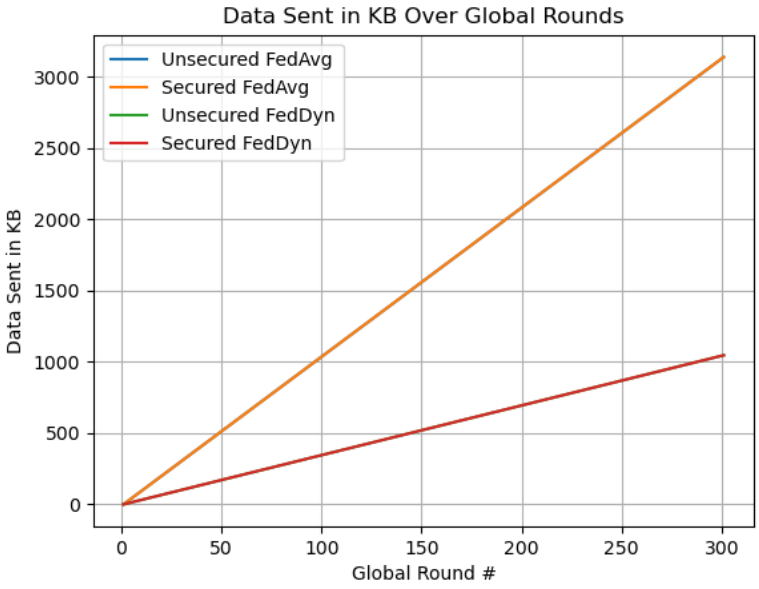
Following our expectations, we see that the FedAvg methods have higher data traffic when sending to participants. As mentioned, FedAvg methods update all participants compared to FedDyn updating for only selected participants, in our case we have four selected participants and twelve total which lines up with the fact that total sent data is three times higher for the FedAvg methods. Notice that a method being secured does not necessarily lead to increased data usage. This can be explained by the difference between the FedDyn versions since for the secured version additional computation is performed on the participant's side and then sent to the coordinator, the extra data sent causes the two versions to have different data received as seen in table 5.7. The rest of the data, however, shows that the secured versions don't use any additional data compared to their unsecured counterpart.

5.2 Ethics and Sustainability

An ethical issue in machine learning is the source of the data that is used to train the model. For example, machine learning within the healthcare industry can put patient records at risk due to the potentially unsecured methods used to train machine learning models. Even in a federated learning environment, it is still possible for information to be divulged [9, 24]. Our work address this by encrypting the information that is shared from the participants to the coordinator, preserving the confidentiality and the privacy of the patients.



(a) Data received to the coordinator from each participant. Note that both FedAvg versions together with unsecured FedDyn have the exact same bandwidth usage and are overlayed on eachother.



(b) Data sent from the coordinator to each participant. Note that both FedDyn and FedAvg have the same bandwidth usage on both of their versions and are overlayed.

Figure 5.2: Total amount of data sent and received by the coordinator.

Given that the process of encrypting and decrypting data adds a considerable amount of time and work to an already intensive process, the sustainability of the method can be debated. With these results, the current option is to exchange energy for privacy, which may or may not be a too viable choice. As we are currently writing this in the middle of an energy crisis, it would not be unreasonable for an organization to opt for unsecured methods, unless directed otherwise. However, the consumption impact of securing data using the methods presented in this thesis is most likely not as high as one might believe. We write *most likely* as testing for power consumption is not something that was thought of during the process of this thesis, but we explain our reasoning as follows.

While the overall time taken for the secured algorithms is roughly four times as long as the unsecured variants, this does not exactly equate to four times the energy usage. The most power-intensive part of training a secured federated model is the training part, in this part of the process PyTorch utilizes the GPU of the computer for the training, if there is one available. Comparatively, the parts of the process which involves the encryption methods are done using the CPU, as in this case the TenSEAL library can not use the GPU for its calculations. CPUs often have less power draw compared to GPUs, as is the case for our evaluation environment on SNIC, where the CPU has a power draw of 150W and the GPU has a power draw of 250W, so even when the two components work for an equivalent amount of time, the CPU ultimately uses less power than the GPU. Referring back to table 5.5 or table 5.6 we can see that the average time that the GPU is used (training) and the average time that the CPU is used (the remaining parts) are either roughly equivalent or that the CPU uses less time. So even though the time taken is four times as long, the power consumption would most likely not follow the time, but as we do not have concrete data or results for this we can recommend future studies in this work.

6
Conclusion

The federated methods FedAvg and FedDyn are implemented with privacy protection in the form of Homomorphic Encryption as well as global invocation via accuracy criteria. Evaluation of the secured federated methods has shown that the impact of Homomorphic Encryption on model performance and bandwidth usage is negligible, except for secured FedDyn which, because of its implementation, has higher data usage. The impact of Homomorphic Encryption is primarily felt in the execution time of the secured federated methods, where unsecured FedAvg and FedDyn take on average 130 and 150 seconds, compared to 510 and 700 seconds for secured FedAvg and FedDyn. While performance may not be affected to a noticeable degree, the number of global rounds to convergence is higher for the secured federated methods compared to the unsecured. Unsecured FedAvg and Feddyn take 27 and 21 global rounds to reach convergence, compared to 37 and 27 global rounds for secured FedAvg and FedDyn.

One takeaway is that the implementation of homomorphic encryption on an existing algorithm could need modifications to the base algorithm. For example, in order to create a secured version of FedDyn, some of the calculations that would have been done on the coordinator instead had to be done on the participant. This is because of the scale limit on the encryption inherent in LHE makes it not possible to adhere to the original method.

Overall, it is shown in this report that privacy protection through Homomorphic Encryption can be implemented on federated learning methods without major alteration and overall performance penalty, but with extra memory costs and time. We believe that our results can be of use to data-oriented security-aware professionals that aim to preserve the confidentiality of data. For future studies, we give three recommendations. We recommend investigating the feasibility to implement Multi-Key Homomorphic Encryption in federated methods similarly as presented in this report. We also recommend investigating power-draw and consumption of a secured federated method. And in addition, we also recommend a thorough evaluation to be done on the Accuracy Criteria method in a federated method to properly measure the advantages and disadvantages of such a method.

Bibliography

[1] Mahajan, D., Girshick, R., Ramanathan, V., Paluri, M., & von Der Maaten, L. (2018). Advancing state-of-the-art image recognition with deep learning on hashtags (2018).

[2] Hartmann, F. (2018). "Federated learning". Available: https://florian.github.io/federated-learning/. [Accessed: 2022-01-28]

[3] Götz, F. (2021, January 22). The Data Deluge: What do we do with the data generated by AVs?. Available: https://blogs.sw.siemens.com/polarion/the-data-deluge-what-do-we-do-with-the-data-generated-by-avs/. [Accessed: 2022-01-28]

[4] Yang, Q., Liu, Y., Cheng, Y., Kang, Y., Chen, T., & Yu, H. (2019). Federated learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 13(3), 1-207.

[5] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics (pp. 1273-1282). PMLR.

[6] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., & Chandra, V. (2018). Federated learning with non-iid data. arXiv preprint arXiv:1806.00582.

[7] Acar, D. A. E., Zhao, Y., Navarro, R. M., Mattina, M., Whatmough, P. N., & Saligrama, V. (2021). Federated learning based on dynamic regularization. arXiv preprint arXiv:2111.04263.

[8] Graves, A. (2012). Long short-term memory. Supervised sequence labelling with recurrent neural networks, 37-45.

[9] Aono, Y., Hayashi, T., Wang, L., & Moriai, S. (2017). Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security, 13(5), 1333-1345.

[10] Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C. A., & Strand, M. (2015). A guide to fully homomorphic encryption. Cryptology ePrint Archive.

[11] Naehrig, M., Lauter, K., & Vaikuntanathan, V. (2011, October). Can homomorphic encryption be practical?. In Proceedings of the 3rd ACM workshop on Cloud computing security workshop (pp. 113-124).

[12] Boddeti, V. N. (2018, October). Secure face matching using fully homomorphic encryption. In 2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS) (pp. 1-10). IEEE.

[13] Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur), 51(4), 1-35.

[14] Brakerski, Z., & Vaikuntanathan, V. (2014). Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on computing, 43(2), 831-871.
[15] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017, December). Homomorphic encryption for arithmetic of approximate numbers. In International conference on the theory and application of cryptology and information security (pp. 409-437). Springer, Cham.
[16] TheIACR. (2018, October 2). From Idea to Impact, the Crypto Story: What's Next? [Video]. YouTube. https://www.youtube.com/watch?v=culuNbMPP0k
[17] "GitHub - Zeromq/Libzmq: ZeroMQ Core Engine in C++, Implements ZMTP/3.1." GitHub, https://github.com/zeromq/libzmq. Accessed 14 July 2022.
[18] Benaissa, A., Retiat, B., Cebere, B., & Belfedhal, A. E. (2021). TenSEAL: A library for encrypted tensor operations using homomorphic encryption. arXiv preprint arXiv:2104.03152.
[19] Cebere, Bogdan. Tutorial 3 - Benchmarks, 2021, https://github.com/OpenMined/TenSEAL/blob/main/tutorials/Tutorial%203%20-%20Benchmarks.ipynb, Accessed: 2022-05-11
[20] McMahan Brendan and Ramage, Daniel. Federated Learning: Collaborative Machine Learning without Centralized Training Data, 2017, https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, [Accessed: 2022-05-31]
[21] Gboard — Googles keyboard, https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin, [Accessed: 2022-05-31]
[22] Alvis, 2022, https://www.c3se.chalmers.se/about/Alvis/, [Accessed: 2022-05-31]
[23] Zhang, S., Schiller, E., Federated Deep Learning for Estimating Vehicular Energy Consumption (preliminary version), forthcoming publication, 2022
[24] Hitaj, B., Ateniese, G., & Perez-Cruz, F. (2017, October). Deep models under the GAN: information leakage from collaborative deep learning. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security (pp. 603-618).
[25] Dolev, S., Petig, T., & Schiller, E. M. (2022). Self-stabilizing and private distributed shared atomic memory in seldomly fair message passing networks. Algorithmica, 1-61.
[26] Dolev, S., Petig, T., & Schiller, E. M. (2015). ROBUST AND PRIVATE DISTRIBUTED SHARED ATOMIC MEMORY IN MESSAGE PASSING NETWORKS.
[27] Zhang, S., Hagermalm, A., Slavnic, S., Schiller, E. M., & Almgren, M. (2022). An Evaluation of Open-source Tools for the Provision of Differential Privacy. arXiv preprint arXiv:2202.09587.