Master's thesis

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

Carl Smestad

Minimizing the Effect of Client Timing Out in Federated Learning Using Federated Dynamic Timeout Window

Master's thesis in Informatics
Supervisor: Jingyue Li
June 2023

NTNU
Norwegian University of
Science and Technology

Carl Smestad

Minimizing the Effect of Client Timing Out in Federated Learning Using Federated Dynamic Timeout Window

Master's thesis in Informatics
Supervisor: Jingyue Li
June 2023

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

NTNU
Norwegian University of
Science and Technology

Abstract

One of the main limitations of ordinary Machine Learning (ML) is that it is executed centrally. This implies that clients must send personal and possibly private data to a central server where a model will be created. One approach to circumvent this limitation is Federated Learning (FL), a distributed machine learning framework that enables clients to train a model locally without sending their private data to a global server. FL has several advantages, such as training models that are tailored to each client while maintaining data privacy. However, it does come with some challenges. One of these challenges is the issue of clients timing out. When clients time out during the training stage, it may lead to accuracy degradation and inefficiency in terms of both time and resources spent. This is because the global server will not receive all model updates from the clients and might result in biases towards the ones who successfully send in their updates. Furthermore, the global server might wait a really long time in the hope that the timed-out client might send the update. Client timing out makes a single communication round unnecessarily long as all the other clients and the server wait for that one client.

This thesis tried to answer the question, **"How can we improve the communication success rate and minimize slow client outlier's impact on learning efficiency in Federated Learning with Independent and Identically Distributed (IID) data?"**.

The research question was answered by utilizing a Design Science Research (DSR) framework to design and implement a proof-of-concept algorithm for dynamically setting the timeout window for each communication round of FL. This novel algorithm, Federated Dynamic Timeout Window (FedDyt), was tested and evaluated through two classification experiments. The first experiment was classifying handwritten digits with the *MNIST* dataset, and the second one was classifying images with the *CIFAR-10* dataset. The results of the experiments showed that FedDyt improved performance both in terms of communication success rate and the efficiency of the FL network when slow client outliers were present. Also, FedDyt contained the benefit of subsiding alongside other FL implementations and strategies. Making it possible for future work to implement it with their solutions and adjust it for their application, or to find an appropriate timeout window.

Sammendrag

En av de største nedsidene med vanlig maskinlæring (ML) er at det blir utført sentralt. Dette impliserer at klienter må sende personlig og muligens privat data til en sentral server hvor en modell blir generert. En metode for å unngå denne begrensningen er Federated Learning (FL), en distribuert maskinslæringsrammeverk som tillater klientene å trene modellen lokalt uten å sende deres private data til den globale serveren. FL har mange fordeler, slik som å trene skreddersydde modeller til hver enkelt klient imens man ivaretar personvernet for data. Likevel så kommer det med noen utfordringer også. En av disse utfordringene omhandler problemet med at klienter bruker for lang tid. Når klienter bruker for lang tid under treningssteget kan det føre til forverring av nøyaktigheten og gjøre nettverket ineffektivt mtp. både tidsbruk og ressurser brukt. Dette skjer ettersom den globale serveren ikke mottar alle modell-oppdateringene og kan føre til partiskhet rettet mot de suksessfulle klientene som sendte inn oppdateringene. I tillegg, det kan hende at den globale serveren må vente veldig lenge i håp om at de trege klientene plutselig sender inn oppdatering. Noe som gjør at en enkelt kommunikasjonsrunde tar unødvendig lang tid imens alle de andre klientene og serveren venter på den tregeste.

Denne avhandlingen forsøker å svare på spørsmålet **"Hvordan kan vi forbedre suksessraten til kommunikasjonsrundene og minimere effekten av ekstremverdiene til trege klienter i Federated Learning med Uavhengig og Identisk Fordelt (UIF) data?"**.

Forskningsspørsmålet ble svart på gjennom å ta i bruk et Design Science Research (DSR) rammeverk for å designe og implementere en konseptbevis-algoritme for å dynamisk sette et tidsavbruddsvindu for hver kommunikasjonsrunde i FL. Denne nyskapende algoritmen, Federated Dynamic Timeout Window (FedDyt), ble testet og evaluert gjennom to klassifiserings-eksperiment. Det første eksperimentet var klassifisering av håndskrevne tall med *MNIST* datasettet, mens det andre var klassifisering av bilder med *CIFAR-10* datasettet. Resultatet av eksperimentene viste at FedDyt forbedret ytelsen både mtp. suksess raten til kommunikasjonsrunden og effektiviteten til FL-nettverket da det eksisterte ekstremverdier bland de trege klientene. I tillegg har FedDyt fordelen av å kunne implementeres sammen med andre FL implementasjoner og strategier. Noe som tilrettelegger for at fremtidig arbeid kan implementere den ved siden av deres løsning og justere den til deres applikasjon, eller til å finne et passende tidsavbruddsvindu.

Contents

Glossary

AI Artificial Intelligence. 5, 17

ASO-Fed Asynchronous Online Federated Learning. 12

CHFL Continual Horizontal Federated Learning. 7

CIFAR-10 Canadian Institute For Advanced Research. 21

CS Centralized Server. 6

DReS-FL Dropout-Resilient Secure Federated Learning. 11

DSR Design Science Research. i, ii, 3, 4, 19

DSRM Design Science Research Methodology. 16

EQ Evaluation Question. 30

EQs Evaluation Questions. 30

FedAdagrad Federated Adagrad. 32, 35, 36

FedAdam Federated Adam. 32, 35, 36

FedAvg Federated Averaging. 6, 14, 32, 36

FedAvgM Federated Averaging with Server Momentum. 32, 36

FedDrop Federated Dropout. 13

FedDyt Federated Dynamic Timeout Window. i, ii, vii, 4, 21, 22, 24, 29, 30, 34–40

FedOpt Adaptive Federated Optimization. 36

FedProx Federated Prox. 27, 36

FEDS Framework for Evaluation in Design Science. 20, 21

FedYogi Federated Yogi. 36

FL Federated Learning. i, ii, 2–14, 16–22, 24, 25, 27, 29, 30, 32, 34, 36–40

FTL Federated Transfer Learning. vii, 7, 8

HFL Horizontal Federated Learning. 7

IID Independent and Identically Distributed. i, 3, 14, 18, 35, 37–39

IoT Internet of Things. 2, 9, 12, 17, 18, 20, 25, 27, 28, 32, 37–40

IT Information Technology. 13

ML Machine Learning. i, ii, 2, 3, 5, 6, 9, 13

MLP Multi Layer Perceptron. 5

MNIST Modified National Institure of Standards and Technology. 21, 30

NLP Natural Language Processing. 37

Non-IID Non-Independent and Identically Distributed. 2, 8, 11, 14, 17, 18, 38–40

NTNU Norwegian University of Science and Technology. 1

POC Proof-Of-Concept. 3, 16, 21

QFedAvg Quantum Federated Averaging. 32, 36

RA Resource Allocation. 9

RQ Research Question. 3, 18, 37

RQs Research Questions. 16

SLR Systematic Literature Review. 8

UIF Uavhengig og Identisk Fordelt. ii

VFL Vertical Federated Learning. 7

List of Figures

List of Tables

1 Acknowledgements

I would like to express my deepest gratitude and appreciation to my supervisor Prof. Jingyue Li at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, for the opportunity to do this research. Thank you for the invaluable support and guidance throughout the research and writing process of both this thesis and the preparatory project. Despite the large geographical distance between Prof. Li and myself, he was instrumental in providing remote assistance and demonstrated commitment to my academic journey.

I would also like to extend my gratitude to the Department of Computer Science at NTNU in Trondheim for providing a rich intellectual environment with all the necessities for the realization of this research project, such as the necessary hardware and a place to conduct the experiments. The interactions with fellow students and faculty members have been important, as they contributed to improving my ideas and providing a supportive network along this journey.

Lastly, I would like to thank my friends and family for their unchanging support, encouragement, and moral support throughout this process. Their belief in my abilities and constant encouragement during this demanding process was invaluable and motivated me to put in the work. I could not have done this without them.

2 Introduction

2.1 Background and Motivation

In recent years, Machine Learning (ML) has increased in popularity amongst businesses and research and is now virtually anywhere you see. Also, the Internet of Things (IoT) has seen a significant boom in both the number of devices and the amount of data that is being handled. IoT is an interconnected network of devices that range from simple sensors and robots to high-power computing devices. Recent statistics, performed by IoT, projected that 2023 would see a growth of 18% to 14.4 billion, and by 2025 it could increase to more than 27 billion connected IoT devices [63, 13]. There exist several ways to define IoT, but Lopez et al. [39] defined three required components: smart things, network infrastructure, and backend servers. In essence, smart devices are designed to interact with users and other devices through a connected network. They are not required to interact directly with users but will generally collect data through some user activity.

This massive increase IoT devices gives a further concern regarding personal- and data privacy. To solve this issue of data privacy, a new paradigm for machine learning arose named decentralized learning, with the most prominent technique being Federated Learning (FL). FL allows client devices to compute the ML models locally and send the model updates to a central server to be aggregated. Thus, enabling collective training without sharing any private information. One of the largest limitations of FL in IoT occurs due to the large differences in hardware specifications which results in the different client devices spending very different amounts of time on one round of learning.

While there exist some solutions to this problem, such as asynchronous FL, secure aggregation methods, and partial work updating, these do come with several limitations:

- High complexity
- Performance degradation with many slow clients
- Performs poorly with Non-IID data

In regular FL, the timeout window is only set during initialization, which means that the network does not alter this window anytime during learning. Thus, if a large number of clients do not finish training within the time limit, the server will close the connection to those clients, and all the work they conducted will not be aggregated into the global model. If there exist many of these clients, it will lead to an increase in several major FL-challenges such as heterogeneity, communication costs, and fairness.

Firstly, it exacerbates the issue of heterogeneity, especially in the case of Non-Independent and Identically Distributed (Non-IID) data, where the unsuccessful clients might possess essential data that the global model will perform poorly without. Secondly, if a lot of resources are being utilized by the clients to train a local model and their results are not being aggregated to the global model, then all of their work is being wasted. By default, communication costs are really expensive in FL [68, 31]. Thus, unsuccessful communication rounds are very wasteful and should be avoided if possible. Lastly, if slow clients are unable to finish training, then it implies that only the fastest devices are allowed to participate. It might boost the training process in terms of time spent per communication round, but as less data is involved, data diversity will not be guaranteed and might hurt the performance of model training and, therefore, the fairness [24].

Instead of setting the timeout window ahead of training, it is possible to adjust it dynamically during training. After each communication round, the server will have a number of successful and unsuccessful clients. By utilizing this information, the server can adjust the timeout window for the next communication round of training. Setting the timeout window dynamically addresses all of the aforementioned challenges which might arise with a statical timeout window. It enables slower clients to finish training as the timeout window will be adjusted to a large portion of them being able to finish the computation.

It is possible to run FL without a timeout window, which would enable every client to finish training. However, in the presence of slow client outliers, it would lead to a large increase in the time spent per communication round. Furthermore, if a client does not finish training for some reason, such as a loss of communication with the server, it would lead to the server halting learning as it would wait for that client indefinitely. By taking advantage of setting the timeout window dynamically, it enables the FL-network to adjust it up to a boundary such that these outliers are omitted during training. Thus, increasing the efficiency of the network proportionally with the magnitude of the outliers.

To the author's knowledge, there has been no research to date into dynamically setting the timeout window in a FL environment.

2.2 Research Question and Methodology

Given the limitations of the existing solutions brought the Research Question (RQ) of this thesis: **"How can we improve the communication success rate and minimize slow client outlier's impact on learning efficiency in Federated Learning with Independent and Identically Distributed (IID) data?"**. To answer this question, a Design Science Research (DSR) framework was utilized to create and implement a Proof-Of-Concept (POC) algorithm, which enables the FL network to dynamically find an appropriate timeout window to allow a threshold amount of clients to successfully finish training.

To evaluate the approach, a baseline was provided by running seven aggregation strategies with the following scenarios:

- Ideal Conditions (Unlimited Time & No Timeouts)
- Poor Performance (95% Clients Timeout)
- Medium Performance (50% Clients Timeout)
- Good Performance (5% Clients Timeout)

The experiments were classification problems run with the *MNIST* and *CIFAR-10* datasets where the neural network learned to classify handwritten digits and ten different classes of images, respectively [2, 1]. The evaluation showed a significant improvement in terms of successful communications rounds between the server and the clients when utilizing the dynamic timeout window algorithm compared to statically setting it. It was especially beneficial in scenarios where there was a large degree of clients timing out such as for the poor performance scenario and medium performance scenario, as the POC managed to include more clients per communication round.

2.3 Contributions

The main contribution of this thesis is the novel approach of dynamically setting a timeout window in a general FL environment. It subsides alongside the other components of a well functioning FL system such as client management, aggregation strategies, etc. which makes it simple to implement alongside for future academia and possibly industry. It manages to include as many clients as possible until a satisfactory threshold by dynamically adjusting the timeout window, which may easily be set by the developers. The novel algorithm also gives the possibility for existing work to find a suitable timeout window for their solution in the case where the developers are uncertain of how much time is required for their clients to finish training in the FL network.

2.4 Thesis Structure

The thesis is structured as follows. Section 3 provides background information on Machine Learning (ML) and Federated Learning (FL) alongside common challenges. Section 4 reviews related

work on the topic of clients timing/dropping out in FL. Section 5 presents the research design created using the Design Science Research (DSR) methodology. Section 6 shows the design and implementation of Federated Dynamic Timeout Window (FedDyt) alongside the experimental setup and results. Section 7 discusses how the novel algorithm compares to related work, possible applications, implications to the industry and academia, and addresses possible threats to validity. Section 8 summarizes the discoveries of this thesis, as well as proposes possible future directions for further work on the proof-of-concept.

3 Background

This chapter will briefly explain the state of Machine Learning (ML) today, Federated Learning (FL), and provide information about the most common challenges encountered when working with FL.

3.1 Machine Learning

There is no doubt that machine learning has increased in popularity in recent years, as it can be seen in virtually every field these days. Machine learning has proven effective for applications within computer vision, prediction, information retrieval, and much more [62]. Even though the field of machine learning is progressing steadily and new milestones are being achieved, businesses report that they are still in the early stages of utilizing ML according to Schlögl et al. [58].

3.1.1 What is Machine Learning?

Machine learning is a subset of Artificial Intelligence (AI), where computational algorithms mimic human intelligence by learning from the environment. These computational algorithms are sometimes referred to as "soft coded" as they manage to adapt their internal structures to improve the results of a desired task. This is the opposite of the traditional "hard coded" algorithms, which are literally programmed to produce a certain outcome [15]. The term "machine learning" was coined by Arthur Samuel from IBM and demonstrated that it was possible for computers to play checkers [57]. Later, the early neural network architectures started developing through the invention of the perceptron by Rosenblatt. Although it seemed promising, it was met with a lot of skepticism due to being limited to only solving linear problems. It could not, for instance, solve simple nonlinear problems such as simple XOR logic [54]. Many years later, it was discovered that it was possible to solve nonlinear problems as well through the development of Multi Layer Perceptron (MLP). Since then, there has flourished a plethora of different algorithms, such as decision trees, support vector machines, ensemble learning, and deep learning to name a few [14, 47, 30].

3.1.2 Machine Learning Approaches

The different machine learning approaches can be divided into different categories based on the data labeling: supervised, unsupervised, and semi-supervised. Supervised learning is trained with labeled data where the mapping from input and output is known on the samples. It is then able to predict an output from a given input. The most common supervised learning applications are regression and classification. Regression manages to predict continuous values by finding a continuous model. Classification excels at giving the output as discrete values [19]. Within supervised learning, there exists the possibility of the model being overfitted. This happens when the observed data does not generalize well to unseen data. In this scenario, the model performs well on the training set, but while fitting on the testing set it performs poorly [76]. In unsupervised learning, the algorithm only receives input samples without a corresponding answer. The most common application of this approach is clustering, where the algorithm partitions the data into subsets where the data is positioned in each subset based on a distance metric. In other words, it groups similar objects into different groups [42]. Semi-supervised learning is a combination of the previous methods; it uses labeled as well as unlabelled data to perform learning tasks. The most common applications are text/image retrieval systems where the training data generally contain both labeled and unlabelled data [69]. In addition to the aforementioned ML approaches, there is the approach of reinforcement learning, where an agent learns behavior through trial-and-error interactions with a dynamic environment. This is a common approach when learning from games such as chess or Go [29].

A visual overview of these four ML approaches can be seen in Figure 1.

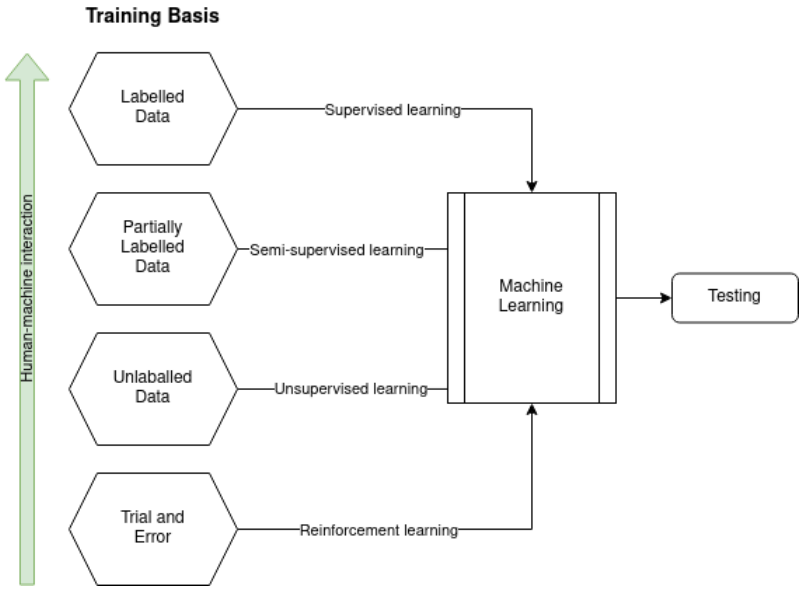


Figure 1: Overview of Machine Learning Approaches. Based on [15].

3.2 Federated Learning

Federated Learning (FL) is a decentralized ML paradigm that leaves the training data distributed on the mobile devices and learns a shared model by aggregating locally-computed updates Ma et al. [41]. Instead of sending private data to a centralized server (CS), the clients compute or train a model on their device and send the update to the centralized server. This decoupling is the main benefit of FL.

The algorithms may involve hundreds to millions of remote devices learning locally, and the goal is generally to solve Li et al. [34]:

$$\min_w f(w) = \sum_{k=1}^{m} p_k F_k(w) \tag{1}$$

where m is the total number of devices, $p_k \geq 0$, $\sum_k p_k = 1$, and the local objective F_k's can be defined by empirical risks over local data.

This paradigm was first introduced by McMahan et al. [44] in 2017, where the centralized server is responsible for coordinating the model training with the selected devices. The server randomly selects a fixed-size subset of clients and provides them with an initial global model before they train and send the updates as already described. The pioneering work was named Federated Averaging (FedAvg) due to taking a weighted average of the client results when aggregating them on the CS. The pseudocode of FedAvg is shown in Algorithm 1 [44].

Algorithm 1 *FederatedAveraging.* The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  end for
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end for
ClientUpdate(k, w) ▷ Run on client k
$\beta \leftarrow$ (split P_k into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \beta$ **do**
    $w \leftarrow w - \eta \nabla l(w; b)$
  end for
end for
return w to server

3.2.1 Types of Federated Learning

According to Prayitno et al. [50], FL can be divided into three categories depending on the type of data partitions. These being Horizontal Federated Learning (HFL), Vertical Federated Learning (VFL) and Federated Transfer Learning (FTL).

Horizontal Federated Learning (HFL)
HFL is the best-studied category amongst the FL-categories and it handles homogeneous features spaces. It is commonly used in scenarios where each client has different samples but has the same set of features [32]. For example, when patient data is separated by two hospitals situated in different regions. One of the shortcomings of traditional HFL is that it assumes homogenous feature space, which is not common in real-life scenarios. To circumvent this issue, Continual Horizontal Federated Learning (CHFL) was proposed. CHFL takes advantage of the unique feature of each client by splitting the network into two columns, common features, and unique features, and then trains on the columns separately before aggregating the results [45].

Vertical Federated Learning (VFL)
In VFL, each client holds different feature data but belongs to the same set of samples. This makes it more relevant compared to HFL in scenarios where for instance, companies have obtained different features for the same customers. One of the main challenges of VFL is that usually, only one party holds the labels to the data, and it makes it difficult for other parties to learn collaboratively without privacy leakage. Some work has been done to address this issue, such as Secure Bilevel Asynchronous Vertical Federated Learning [80]. Another challenge is the high communication costs that occur when the different parties exchange intermediate results [73]. VFL has many promising potential applications as it may train models on data from multiple locations without sharing the data or trade secrets within the field, thus preserving privacy.

Federated Transfer Learning (FTL)
FTL is a special case of FL and was first introduced to address the problem of data isolation in industries, as data often exist isolated between different organizations. It allows knowledge to be transferred between domains that do not have many overlapping features or users. It has many potential applications, such as personalized recommendation systems, fraud detection, and predictive maintenance. In fraud detection, it may be used to train models on transaction data between multiple banks without sharing the data [56]. A graphical illustration of FTL can be seen in Figure 2. The figure shows only a small overlap in the feature and sample space of parties A and B. In contrast to HFL, where there is a large overlap in feature space, and in VFL, where

there is a large overlap in sample space, FTL has a slight overlap in both.

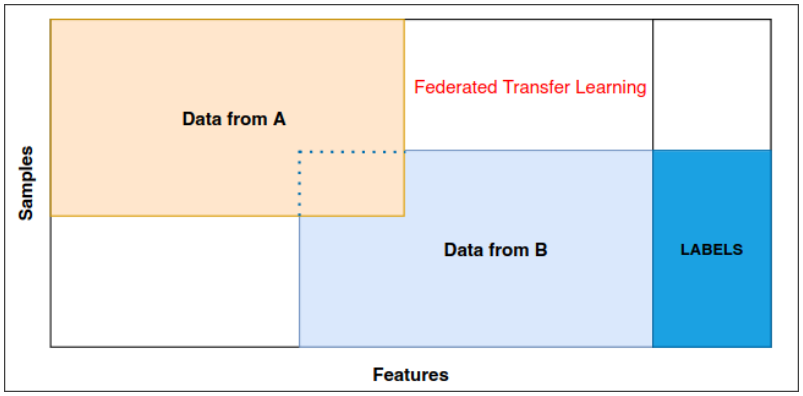


Figure 2: Graphical illustration of FTL [56].

3.2.2 Client Selection in Federated Learning

Client selection is an essential term within FL and has seen hundreds of improvements since the original random selection of clients as suggested by McMahan et al. [44]. Smestad and Li [64] performed a Systematic Literature Review (SLR) investigating the state-of-the-art of client selection in federated learning. The study performed forward- and backward-snowballing and ended up with 47 papers being reviewed, and their findings are summarized in the following subsections:

What are the main challenges in Federated Learning?
The papers discovered in the SLR were divided into four categories depending on the main challenge they were trying to solve. Results show that 23 studies tried to improve upon heterogeneity, 13 studies revolved around resource allocation, eight studies focused on communication costs, and three studies had fairness as the main challenge. The distribution of the challenges can be seen in Figure 3.

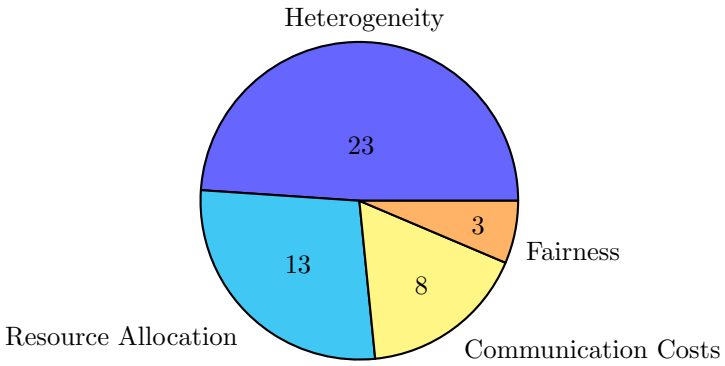


Figure 3: Distribution of challenges reported from the primary studies

The most common challenge is heterogeneity which stems from the training being executed on the client's local devices. This results in differences between the clients as they will have different datasets and availability. Ma et al. [40] conducted a state-of-the-art survey on solving Non-Independent and Identically Distributed (Non-IID) data in FL and conducted that data heterogeneity could be divided into the following categories: feature distribution skew, label distribution skew, same label (different features), same features (different labels) and quantity skew. It might result in an inefficient trained model, performance- and accuracy degradation, increased biases, and unnecessary exchange of information between clients.

The second most common problem is resource allocation. This is due to several reasons, but the main one is the fact that the training process becomes inefficient when some clients have limited

computational resources [46]. Differences in resources and hardware will lead to clients using longer time during computation or during model transmission, which results in the "straggler effect" [51].

The third most occurred challenge was regarding communication costs. Every time the server performs a global update, it needs to receive the local aggregation of all the selected clients, and according to Tan et al. [68], the communication power required to reach convergence makes up a large portion of the cost. Another fundamental concern regarding communication costs is the limited energy from IoT-devices and other low-energy entities. It is imperative to improve the energy efficiency of the systems to facilitate these devices.

The least common problem encountered was fairness, where only three studies reported it as the main challenge which they tried to solve. However, fairness is a researched topic within several similar fields, such as Resource Allocation (RA) and ML. For machine learning, it is typically defined as the protection of some specific attribute(s) by, e.g., pre- processing the data to remove information about the protected attribute [16]. In the context of FL, if the client selection algorithm always selects the fastest devices, it might boost the training process. However, as stated by Huang et al. [24]: "But clients with low priority are simply being deprived of chances to participate at the same time, which we refer to it as an unfair selection." It might result in undesirable effects, such as omitting some portions of data. Also, if there are less data involved, data diversity will not be guaranteed and might hurt the performance of model training. An overview of the specific FL-challenge compared to the corresponding solutions can be seen in Table 1.

Table 1: Solutions compared to challenges

Challenge	Solution(s)
Heterogeneity	- Select subset of client to make up homogeneous dataset - Measure degrees of non-IID data and select lowest values - Balance out non-IID data through clustered FL - Give clients an irrelevance score and base selection of that - Select a subset of clients who represent the entire set - Utilize cryptography and weight divergence
Resource Allocation	- Base selection on resource conditions - Maximize the amount of clients by minimizing energy consumption - Encourage clients to participate in model updating - Utilize fuzzy logic by considering several resource factors
Communication Costs	- Joint client selection algorithm to reduce convergence time - Distributed client selection where the clients decide to participate - Only active clients should perform training - 3-way hierarchical framework to improve efficiency - Select the client with the most significant information each round - Asynchronous FL
Fairness	- Fairness-guaranteed client selection algorithm - Improve fairness through biased client selection - Select honest clients

How are clients selected in federated learning?
There is no defacto standard for selecting clients in FL, and there exists a lot of different approaches depending on the main underlying issue. To try to mitigate heterogeneity, which stems from differences between selected clients, one might select clients who have small data heterogeneity, which together forms a homogeneous subset, or measure the degrees of non-IID data present in each client and choose the ones with the lowest degrees [41, 3, 81].

It is also possible to improve the issue of resource allocation through client selection. By managing clients based on their resource conditions, the algorithm may maximize the number of clients allowed to participate while minimizing the energy consumption [75, 77].

As communication cost is a vital and expensive part of FL, many attempts have been executed in order to improve it. Focusing on selecting appropriate devices by allocating a suitable amount of resources will reduce convergence time due to reducing communication costs. It is also possible to make the client devices decide to participate in aggregation and let the remaining clients not do

any work [21, 31, 35]. Zeng et al. [78] proposed only selecting the clients who provide significant information each round, which enables fewer clients to end up with better accuracy and, in turn, lower total communication rounds and cost.

Focusing on fairness in client selection may lead to better accuracy but might sacrifice training efficiency. One way to improve fairness is by selecting the clients with higher local loss [27, 24].

Which metrics are important for measuring client selection?
The relevant metrics in client selection depend on the problem it is trying to solve. The summary of the findings can be seen in Table 2. The most common metric used in measuring the testing accuracy against the number of communication rounds. In traditional machine learning, one generally uses the number of epochs, but the number of epochs in FL does not necessarily correspond to the same thing. Client devices may have a multitude of local epochs during training but only convey this information once during one round of communication. Every study utilizes this metric as it clearly indicates how well the FL-network is performing. However, it may also be beneficial to look at the number of communication rounds until threshold accuracy or look at the energy, delay, and client consumption when trying to improve the issue of resource allocation.

Table 2: Metrics compared to challenges

Challenge	Metric(s)
Heterogeneity	- Testing accuracy vs communication rounds - Communication rounds until threshold accuracy - Number of selected client able to finish training
Resource Allocation	- Testing accuracy vs communication rounds - Energy, delay, and client consumption
Communication Costs	- Testing accuracy vs communication rounds - Convergence time vs latency - Cost of hiring clients
Fairness	- Testing accuracy vs communication rounds - Availability of clients - Long-term fairness constraints

What can be improved with the current client selection?
Smestad and Li [64] discovered a lot of possible future directions for client selection, but the most relevant ones for this thesis were regarding optimizing resource allocation. It would be highly beneficial to look into the effect of unsuccessful clients (or free-riders) and how to quantify the impact [37, 52, 61].

4 Related Work

Much work has been conducted on the topic of handling slow clients. This section will discuss the related work and conclude with the limitations of those.

4.1 Secure Aggregation Methods

As we know from the FL-background discussed in Section 3.2, server aggregation is an integral part of a well-functioning learning network. A sub-category of these aggregation strategies is known as secure aggregation, which focuses on protecting the privacy of each client's individual model while allowing for a beneficial global model update. These secure aggregation methods focus not only on the aggregation's security aspect but also on making it resilient to clients dropping out.

Shao et al. [60] proposed a Dropout-Resilient Secure Federated Learning (DReS-FL) framework, based on Lagrange-coded computing to tackle both the challenge of Non-IID data and clients dropping out. Using Lagrange-coded computing, the server can tolerate a certain number of clients dropping out by reconstructing the missing data. This is an improvement on existing work, which utilized secure aggregation protocols and was based on either a pairwise random-seed agreement for mask cancellation or sharing random seeds to construct the dropped masks [28, 66, 7, 26]. These methods were not optimal because they relied on recovering and aggregating the result of surviving clients. Resulting in a possible decrease of performance in Non-IID-setting, as the surviving clients may vary significantly.

Several dropout-resilient aggregation protocols are created to tackle the challenge of clients dropping out. Liu et al. [38] summarize current aggregation protocols and contribute with a more dropout-resilient protocol which is more efficient and claims to have better simplicity, making it attractive for implementation and further improvements. It improves upon existing protocols, which can be viewed in Table 3, by replacing communication-intensive building blocks with a seed homomorphic pseudo-random generator.

The baseline for the comparisons is the SecAgg scheme based on the work performed by Bonawitz et al. [7]. It improved the protocol efficiency by leveraging on pair-wise additive masking and Shamir secret sharing. To improve the efficiency of SecAgg, several schemes have been proposed. TurboAgg [65] communicate across only a subset of clients, while CCESA [12] and SecAgg+ [6] replaced the star topology in SecAgg with random subgroups of clients. FastSecAgg [28] improved upon SecAgg by utilizing a more efficient FFT-based multi-secret sharing scheme rather than Shamir secret sharing. Other schemes try reducing communication costs by compressing the gradient vector, such as SAFER [5].

Table 3: Comparison of computation complexity, communication complexity, and dropout resilience of secure aggregation algorithms

Protocol		SecAgg [7]	TurboAgg [65]	CCESA [12]	SecAgg+ [6]	FastSecAgg [28]	HPRG [38]
Computation complexity	Server	$O(mn^2)$	$O(m\log n\log^2\log n)$	$O(mn\log n)$	$O(mn\log n+n\log^2 n)$	$O(mn\log n)$	$O(n)$
	Client	$O(n^2+mn)$	$O(m\log n\log^2\log n)$	$O(n\sqrt{n\log n}+mn)$	$O(m\log n+\log^2 n)$	$O(m\log n)$	$O(n^2+m)$
Communication complexity	Server	$O(n^2+mn)$	$O(mn\log n$	$O(n\log n+m\sqrt{n\log n})$	$O(mn+n\log n)$	$O(n^2+mn)$	$O(n^2+mn)$
	Client	$O(m+n)$	$O(m\log n)$	$O(\sqrt{n\log n+m})$	$O(m+\log n)$	$O(m+n)$	$O(m+n)$
Dropout resilience	Scheme	(t,n)	$(\frac{n}{2},n_1)$	(t,k)	(t,k)	$(n-d,n)$	(t,n)
	Max drop	$n-1$	$\frac{n}{2}-1$	δn	δn	$\frac{n}{2}-1$	$n-1$
Communication rounds		4	$n/\log n$	3	3	3	3
Privacy		malicious	semi-honest	semi-honest	malicious	semi-honest	malicious

4.2 Asynchronous Federated Learning

As discussed in Section 3.2, the FL-the scheme is by default synchronous, which might introduce slow clients known as "stragglers" and makes global synchronization difficult and/or slow. Similarly, if a client drops or times out during learning, it will significantly impact the federated learning as the global server is waiting for that client. Therefore, several algorithms for asynchronous FL have been created to overcome the straggler issue [84, 9].

Asynchronous FL is a distributed decentralized machine learning method that enables collaborative learning through aggregating a large number of models from client devices asynchronously. Usually, the server waits for every client to finish training before performing the aggregation, but when performing it asynchronously, it enables the server to finish the communication round before every client has finished training. The results of the slow clients will then be aggregated in a later communication round if they ever finish. A fully asynchronous FL network will, in theory, be able to reduce a lot of waiting, both for the server and the clients, which leads to a very efficient and robust distributed model. This method is very suited for applications such as IoT where there generally are significant differences in computing capacities and network speed, which inherently leads to much waiting [49]. Chen et al. [11] proposed the Asynchronous Online Federated Learning (ASO-Fed) framework, where edge devices perform online learning with continuous streaming local data, and a central server aggregates the central model asynchronously.

A graphical view of how these updates functions in practice can be viewed in Figure 4, which compares the communication flows between the client devices and the central server for both the synchronous and the asynchronous protocols. The version of the global model is denoted by w^{t_k} and is distributed to client devices before local computation. In the synchronous example, Device 1 has no network connection, and Device 3 needs more computation time. All aggregations happen synchronously, forcing the server and Device 2 to wait before proceeding because the server must receive all local updates before aggregation [10]. In the asynchronous example, the following happens: Firstly, the global model w^{t_0} is distributed to Device 1 and Device 2, and Device 1 finishes training first and sends the local model update w_1 to the central server which is updated to the new global model w^{t_1}. Then Device 2 finishes training and send local update w_2 to the central server, which is aggregated to global model w^{t_2}, which is then distributed to Device 3.

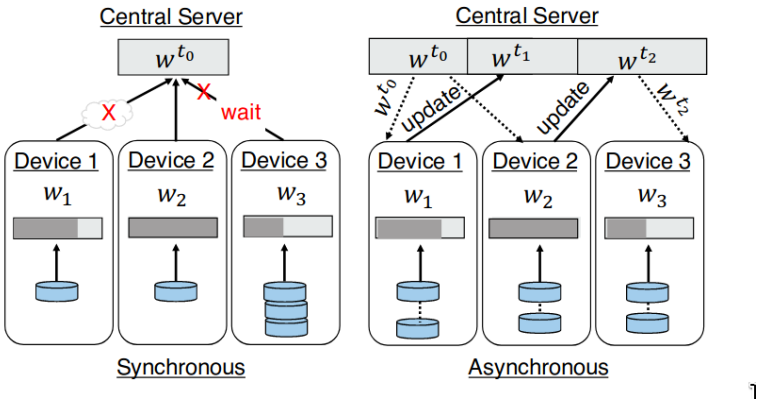


Figure 4: Illustration of Synchronous vs. Asynchronous [11]

Xie et al. [74] introduces the idea of adding a coordinator responsible for the asynchronous handling of the learning network, and the server immediately updates the global model whenever it receives a local model.

4.3 Clients Performing Partial Work

As discussed in section 4.2, in the synchronous protocol of FL, if clients drop out during learning, it leads to severe degradation of the efficiency. Instead of waiting for the straggler or declaring the communication round as a failure, some work discusses the possibility of clients performing partial work. Li et al. [33] propose the framework *FedProx* which builds upon the original *FedAvg* framework [44]. It also selects a subset of devices at each round, performs local updates, and averages those to form a global update, but in addition, it tolerates partial work.

The main benefit of this solution is that when a communication round has hit a set timeout window, and clients are not done with their local epochs, the partial results they have obtained can still be aggregated into the global model. This leads to increased utilization of computing power, as partial

work will not be wasted as in ordinary synchronous FL, as well as the ability to not interrupt a round of learning due to stragglers not responding at all.

However, these benefits do not come without a price. Firstly, if there exist many stragglers within the network, it will perform poorly as the server then aggregates a lot of gradients from clients who have performed few local epochs. Secondly, it is difficult to know how much time to set for the timeout window or the number of local epochs for the clients to perform beforehand. Lastly, it is not trivial to implement partial work as it is dependent on a lot of factors of the dataset, data distribution, heterogeneity, and implementation details. For instance, the developer has to manually set a proximal term β, which has to be estimated through trial and error.

4.4 Free Riders

Similarly to the topic of clients dropping out, where clients do not contribute to learning by successfully delivering results, there are free riders. Free-riding is a familiar concept within the field of economics and philosophy [18, 4], and in a general context, may be defined as:

> A free rider, most broadly speaking, is someone who receives a benefit without contributing towards the cost of its production.

In regards to Information Technology (IT), it is commonly researched and explored within peer-to-peer systems. As those also rely on clients contributing together to form a functional system [36].

Plain free-riders are clients who return the same model parameters to the global server as they were initially served [17]. This is obviously a waste of resources as the network spends energy and time through communication rounds with clients that do not contribute to the overall learning. According to Sagduyu [55], the presence of free-riders potentially decreases global accuracy due to a lack of contribution to global model learning.

4.5 Ensamble Methods

To the experienced reader, the term "dropout" might be familiar within the context of ML. As discussed in section 3.1, overfitting may become a significant issue, as large networks may be slow, especially when combining predictions of several large neural networks. Dropout prevents overfitting and makes it possible to combine exponentially many different neural networks (approximately) by removing nodes temporarily during training. The term refers to dropping out both hidden and visible units in a neural network [67].

One of the main challenges within practical FL is that the client devices struggle with computation-intensive tasks due to resource constraints. One approach to tackle this issue is Federated Dropout (FedDrop), a FL scheme that builds upon the classing random model pruning scheme as already discussed. It reduces the communication overhead and devices' computation loads compared to traditional FL while also improving cases of overfitting [72].

Several works suggest improvements upon FedDrop. Bouacida et al. [8] proposes to tune the dropout rate based on feedback from the training loss. Horváth et al. [20] proposes a method called Ordered Dropout, where sub-models are selected in a nested fashion. This is meant to improve learning, as clients can train models in proportion to their computational resources.

4.6 Limitations of Related Work

Although much work has been conducted to improve the issues of stragglers and/or clients dropping out, they come with some limitations. This section will briefly discuss some of those limitations.

4.6.1 High Complexity

One of the main limitations of the related work is the high degree of complexity. The secure aggregation methods rely on the secret sharing of the random seeds used for mask generation at the user in order to be able to reconstruct and/or cancel the model updates of the clients who have dropped out. The complexity of such an approach also grows substantially when the number of dropped clients increases [66]. The complexity of implementing partial work from clients is one of the main disadvantages of that solution. It is untrivial how to efficiently utilize the partial work from clients, who might have a poor score due to few local epochs, against the results of high epoch clients. In addition, aggregation might be difficult due to the dataset, data distribution, heterogeneity, and implementation details. The solution also requires a timeout window to be set for the clients, which depends on many factors. These factors are client performance, network speeds, and task complexity. The developer also has to set a proximal term β, which has to be estimated through trial and error.

4.6.2 Performance

One of the main limitations of asynchronous FL is the trade-off between model performance and communications costs. The server might be the communication bottleneck as it has to aggregate a lot of models, making it a high server-to-client communication delay. In theory, making it asynchronous reduces the overall communication load on the server by allowing the clients to wait on the server, but it comes with a high price. Client updates will be postponed to later communication rounds, and it is not trivial to correctly use these updates in other communication rounds [83].

The performance when clients are performing partial work will also substantially degrade when there are a lot of clients not being able to finish within a given timeout window. This leads to a high degree of unfinished work being aggregated into the global model.

4.6.3 Non-Independent and Identically Distributed Data (Non-IID)

In real-world applications, the data is seldom IID when the clients may collect data from many different sources, through different tools, or only have access to partial/biased data. This will result in shifts/drifts in the distribution of samples or features to the clients. For all of the aforementioned related work, this might lead to poor performance and slower convergence [43] because when local models focus on different features or even feature representations, they might not complement each other well when being aggregated on the server. Furthermore, each client device will focus on maximizing its local optima, which may differ significantly from the global optima.

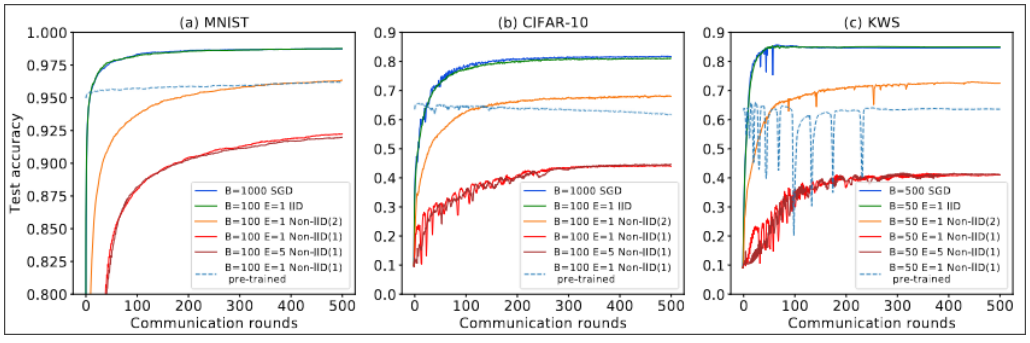


Figure 5: Federated Learning with Non-IID data [82]

McMahan et al. [44] states that the FedAvg strategy might perform well even in a Non-IID setting, but the work of Zhao et al. [82] shows a significant reduction of test accuracy in the case of skewed data. They show that this accuracy reduction can be explained by the weight divergence, which

can be quantified by the distribution over classes on each device and the population distribution. From Figure 5 it is apparent that the accuracy degrades close to 50% in the worst case as for the KWS dataset.

5 Research Design

This section will identify the relevant research questions (RQs) and the motivation for answering those.

5.1 Research Methodology

To correctly identify the problem, motivation, and solution while properly evaluating and demonstrating the solution's effectiveness, the Design Science Research Methodology (DSRM) by Peffers et al. [48] was utilized. An overview of the research design process can be seen in Figure 6.

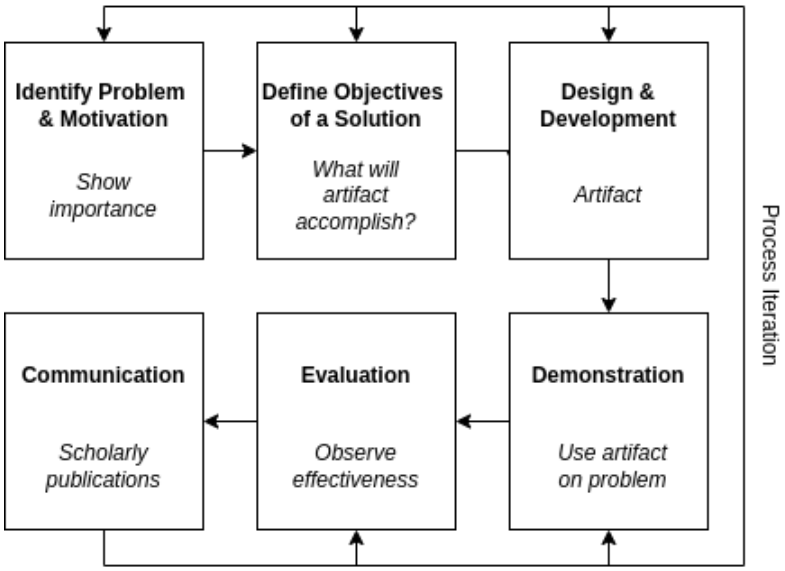


Figure 6: Overview of the research design process [48].

Peffers et al. [48] describes the DSRM as consisting of the following six stages:

Problem Identification and Motivation. As the problem definition will be used to create an artifact that effectively provides a solution, it is essential to properly identify the problem and motivate its relevancy and novelty. The problem will be identified and motivated in Section 5.2.

Objectives for a Solution. For the artifact to succeed, certain objectives must be in place. These objectives are inferred from the problem definition, and successfully reaching the objectives will lead to a better solution than the currently existing ones. The objectives for a solution are discussed in Section 5.3.

Design and Development. The third step is regarding the artifact's creation and its process. The activity includes desired functionality, architecture, and the actual creation of the artifact. The design and development will be discussed in Section 5.4.

Demonstration. The demonstration step uses the artifact to solve one or more instances of the problem. In practice, it may be through simulations, case studies, etc. For the artifact of this thesis, the demonstration of its effectiveness was through running FL simulations with the POC and is shown in Section 6.

Evaluation. The functionality of the evaluation step is to observe and measure the successfulness of the solution to the problem. There are several different types of possible evaluation methods depending on the qualitative/quantitative nature of the solution. The different evaluation metrics will be discussed in Section 5.5

Communication. The last step is to communicate the problem and its importance, the artifact,

its utility and novelty, the rigor of its design, and its effectiveness to researchers and relevant audiences. This will be discussed in Section 5.6.

5.2 Problem Identification and Motivation

Society has seen a large boom in the usage and research of Artificial Intelligence (AI) in the past decade. One of the largest contributions to this phenomenon is the amount of available data which can be gathered through IoT (Internet of Things) as well as the practical benefits of, for example, machine learning. However, amongst all this available information exists our private information, such as social security numbers, addresses, banking information, etc. Ordinary machine learning might require that some or all of this information be sent to a centralized server to create or update models.

This emerging concern about data privacy has introduced a new field within machine learning named Federated Learning. It is possible to train these models on the user's device and then synchronize the model to a central store Hu et al. [23]. While decentralized learning solves the difficulties of data privacy and sensitive information in an elegant way, it does come with some challenges. As these models are meant to be trained on smartphones and IoT devices, there is a high risk of these devices dropping out during training due to, i.e., bandwidth or availability constraints [61]. Similarly, there is the concept of free-riders introduced by Fraboni et al. [17], where free riders are clients who use the global model but do not contribute to learning. This results in those clients using valuable and limited resources and deteriorating the performance of the federated learning network by contributing fake or low-volume data [52].

Clients timing out is a large problem in FL as it may exacerbate the Non-IID problem as the data distribution among different rounds could vary greatly [60]. The issue is especially prominent within applications of FL where there are large differences in hardware specifications of the client devices. One such application is within the field of IoT, which consists of units such as robots, drone swarms, and low-cost computing devices (e.g., Raspberry Pi) that may have limited processing ability, low bandwidth, and power, or limited storage capacity [25]. The different possible steps which may lead to clients timing out can be divided into:

1. Dissemination time (e.g., dataset size)
2. Local computation time (e.g., HW restrictions)
3. Upload time (e.g., High latency, network congestion)

The dissemination time refers to the time spent initializing phase of the FL-network before learning. This possibly includes steps such as distributing datasets. The second step is the local computation time which is the time each client uses for training. The third step is the upload time which is the time spent on sharing model updates. IoT-devices may encounter problems within all of these steps due to the possible limitations already mentioned. For instance, due to the limited storage capacity, it may have issues during the dissemination step, or due to poor bandwidth/high latency, it may spend a lot of time communicating the model updates. It is also very likely that there will be large differences in the local computation time due to the differences in hardware specifications, allowing some devices to finish training much faster than other low-cost devices.

One of the challenges within the topic of clients timing out in FL is that it is difficult to know whether or not a client has timed out or if it is just spending a lot of time in any of the aforementioned steps. Thus, it is common to implement a timeout window where the server shuts down communication with clients who have not reported any results within the given time.

5.2.1 Current State of the Problem

A lot of work has been put into FL for IoT, but the state-of-the-art mainly focuses on two topics: solving the issue of heterogeneity and minimizing energy. Therefore, when viewing the state-of-

the-art solutions, the problem of clients timing out is still an issue that remains. Recent solutions include ways to neutralize the problem of Non-IID through domain generalization and group learning [79, 35]. As discussed in Section 4, asynchronous FL, secure aggregation methods, and resource-aware FL have been suggested to improve the issue of straggling clients.

To summarize the current state of the problem, although several solutions exist to the problem of clients timing out during FL, they do have several limitations. Firstly, the complexity is very high, making it challenging to utilize in new applications. For instance, with asynchronous FL, when clients are stuck many gradient updates behind due to perhaps slow computation, it is not trivial to aggregate those results into the global model as the information might not be relevant anymore. Secondly, the performance of the solutions halter, especially when a lot of clients do not manage to finish training within the timeout window. Furthermore, none of the state-of-the-art solutions for FL for IoT utilizes the existing solutions for solving the problem of clients timing out during learning, which means that at the time of writing, there does not exist a defacto solution to this problem.

5.2.2 Research Question

The RQ of this thesis is the following: **"How can we improve the communication success rate and minimize slow client outlier's impact on learning efficiency in Federated Learning with Independent and Identically Distributed (IID) data?"**.

The communication success rate, which will properly be defined in Section 5.5.4, is a highly important metric for FL as it directly measures the number of successful clients for each communication round. A low degree of success rate implies that clients are performing a lot of work that is not going to be aggregated into the global model, due to them, for instance, not finishing training within a given timeout window or connectivity issues. Likewise, a good success rate implies that a lot of clients manage to finish training and get their results aggregated into the global model.

There are many potential downsides to clients timing out during learning, and the most prominent ones are the high increase in communication costs and resource expenditure. The increased communications costs stem from the high degree of communication which is being transmitted between the clients and the server without bringing further results to the global model, while the resource expenditure from the local training performed by the clients. In the case of IoT devices, which are resource-constrained entities with limited computational power, storage capacities, and batteries [25], it is crucial to utilize these precious resources without waste. In addition, if the FL network has a high degree of Non-IID data, it will lead to accuracy degradation due to vital information being omitted during learning. In the case of IID data, the effects on the accuracy are not as prominent due to the results of one client being directly transferable to the others. Thus, improving the communication success rate directly corresponds to more successful clients each communication round and improves the communication costs and resource expenditure for the FL network.

Because these IoT devices might have widely different hardware specifications, and the developers do not necessarily know what these are beforehand, it might lead to slow client outliers - clients who are much slower than the others. In the presence of such clients, regular synchronous FL without a timeout window will spend much more time per communication round, as every client has to finish training before aggregating the results and iterating to the next one. Therefore, minimizing the impact of the slow client outliers on the efficiency is highly beneficial and especially relevant within the field of IoT.

Both the communication success rate and the slow client outliers are important aspects to study for IoT in FL, and the effectiveness of this thesis' artifact will be shown in Section 6.

5.3 Objectives for a Solution

There are several objectives for a solution in regards to improving FL through minimizing the effect of clients timing out. It already exists several attempts at solving this exact problem, but as discussed in Section 4.6, these do not come without a heavy price tag.

Therefore, in order to improve existing solutions and fill a missing void within the field of FL, the following list of objectives needs to be fulfilled for a successful and beneficial solution to exist:

1. Low complexity
2. Simplicity of integrating with existing solutions
3. Reduce the effect of clients dropping out

If the solution manages to have low complexity, it will be much easier for both academia and the industry to implement and utilize further in their work. One of the biggest issues with existing work such as asynchronous FL is the high degree of complexity which makes it really difficult to implement on new datasets and for different hardware. Similarly, the solution does not bring much value if it only works in an isolated environment with specific datasets or a single aggregation algorithm. It is important that the solution has to integrate with existing solutions. Lastly, there has to be a measurable improvement in minimizing the effect of clients dropping out. The solution cannot be considered successful without there being physical evidence that the solution in some way manages to improve learning when clients are timing out.

5.4 Design and Development

The artifact will be designed and created in the design and development stage. The artifact of this thesis will be the implemented code alongside the overall abstract solution, which benefits learning. The artifact development of this thesis was carried out in two iterations. In the first iteration, related work was explored and reviewed to understand the current state-of-the-art problem and to view the existing solutions. When it was clear that the problem was real and there existed a need for a solution, the rest of the first iteration was used to implement a bare-bones server-client architecture based on Federated Learning (FL). The second iteration revolved around creating the artifact and adapting it in an FL-environment.

5.5 Evaluation

In the evaluation stage, it is paramount that the effects of the artifact are both observed and measured [48]. The evaluation of design artifacts and design theories is a key activity in Design Science Research (DSR) as it provides valuable feedback for further development and assures the rigor of the research. Therefore, this thesis follows the steps provided by Venable et al. [70] in their Framework for Evaluation in DSR. The design process consists of the following four steps:

5.5.1 Explicate the goals of the evaluation

Venable et al. [70] lists four competing goals when evaluating a project: rigor, efficiency, ethics, and uncertainty/risk reduction.

Rigor: They explain that rigor has two senses. The first is that the artifact must cause an observed outcome and not some confounding independent variable (efficacy). The second is establishing that the artifact works in real-world situations (effectiveness). The created artifact of the thesis must cause an observed effect, but it will be difficult to prove its effectiveness in a real-world scenario. This is due to FL being an emerging technology and difficult for a single student to implement

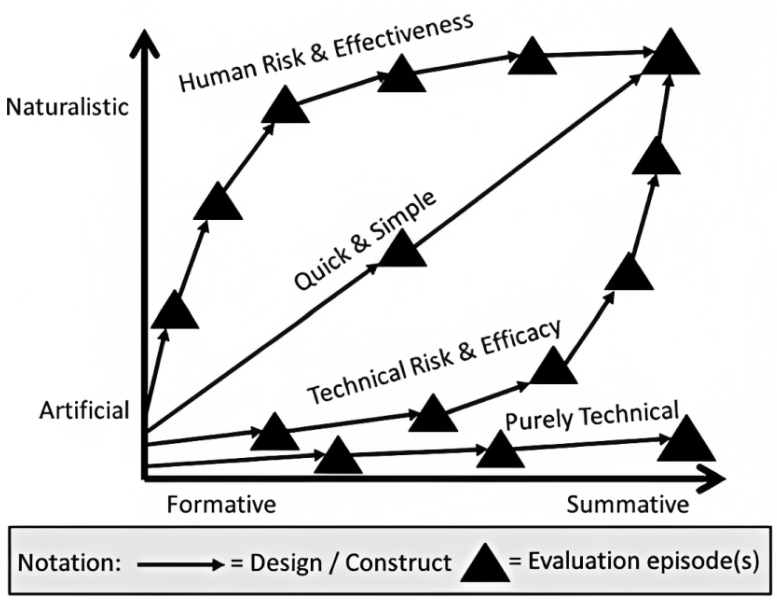


Figure 7: FEDS (Framework for Evaluation in Design Science) with evaluation strategies [70]

across a multitude of actual IoT devices. Therefore, the evaluation of the rigor was set to moderate due to this limitation.

Uncertainty and risk reduction: When design uncertainties are significant, it is key to reduce risks. These may be divided into human social risks or technical risks. Human social risks regard the artifact not fitting into the use case or social situations, while technical risks regard the artifact not being made to function. The thesis is highly technical and it is very important to avoid as many technical risks as possible. There was a high degree of design uncertainty when designing a dynamic timeout window for FL, making the evaluation of the uncertainty and risk reduction set to high.

Ethics: This step is especially important for safety critical systems and technologies and should address risks towards animals, people, organizations, or the public, including future generations. As the artifact is meant to improve FL for IoT-devices, it will not have any direct risks towards the aforementioned groups. Even though it might be used alongside already existing solutions which may be used in some unethical way, it was evaluated to be low.

Efficiency: This goal is created to balance the high-ordered goals against the resources available for evaluation (e.g., time and money). This thesis was conducted by a single student during one semester with virtually no budget. Thus, making efficiency very important in the artifact evaluation. By utilizing these four goals given by Venable et al. [70], this thesis was given the following priorities:

1. Rigor (moderate)
2. Efficiency (high)
3. Uncertainty and risk reduction (high)
4. Ethics (low)

5.5.2 Choose the evaluation strategy

The second step in FEDS is selecting one or more strategies for the evaluation. Figure 7 describe strategies and implies a decision about why, when, and how to evaluate. A textual representation can be seen in Appendix A. To summarize the different strategies:

The Quick & Simple Strategy. This strategy conducts little formative evaluation and progresses quickly with relatively few evaluation episodes. The main benefit of this strategy is the low cost and enables quick progression. It does, however, come with the drawback of being too quick in terms of various design risks. It is well suited when the project scope is small with a simple design and low risk.

Human Risk & Effectiveness Strategy. It emphasizes formative evaluations early in the process while progressing quickly. It is beneficial for scenarios where there are major design risks related to users and/or real-world scenarios where the utility is being evaluated. At the end of the strategy, the artifact is rigorously evaluated based on its effectiveness.

Techincal Risk & Efficacy Strategy. The strategy utilizes iterative evaluations early in the process in order to progressively move towards the goal. It is well suited for scenarios where the main design risk is technical and when the goal of the evaluation is to establish the utility of the artifact.

Purely Techincal Strategy. The purely technical evaluation strategy is used when the artifact is purely technical, without human users. It is comparable to the quick & simple strategy but focuses more on artificial evaluations throughout the process and is generally well-suited for applications that will not be deployed anytime soon.

This thesis utilizes the *Purely Technical Artifact* strategy as the challenges are completely technical as opposed to social. Furthermore, as the artifact is a POC, it will not be deployed to a real-life application anytime soon.

5.5.3 Design the individual evaluation episode(s)

The fourth step in FEDS is designing individual evaluation episode(s). This step is only possible after a strategy has been chosen and determined what properties to evaluate of the artifact.

The evaluation was conducted by one master's student after the artifact creation through simulation in a FL-environment.

Federated Learning Dynamic Timeout Window

The first experiment used the Modified National Institute of Standards and Technology (MNIST)-dataset [1] which contains a large database of handwritten digits commonly used in computer vision and deep learning. The database consists of 60,000 examples of 28x28 black-and-white images [2]. The second experiment used the Canadian Institute For Advanced Research (CIFAR-10)-dataset[2] which consists of 60,000 32x32 color images in 10 classes, with 6000 images per class [1]. An overview of the datasets with the corresponding samples, features, and machine learning tasks can be seen in Table 4.

Dataset	Machinelearning task	Samples	Features
MNIST	Classification	60,000	10
CIFAR-10	Classification	60,000	10

Table 4: Overview of datasets used in experiments

In addition to the aforementioned datasets, the simulations also required strategies to evaluate the performance of FedDyt. These strategies were chosen out of the pool of available strategies in the FL-framework, Flower, and a list of the chosen strategies together with the original paper can be seen in Table 5.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data
[2] https://www.tensorflow.org/api_docs/python/tf/keras/datasets/cifar10/load_data

Author	Title	Strategy
McMahan et al. [44]	Communication-Efficient Learning of Deep Networks from Decentralized Data	FedAvg
Hsu et al. [22]	Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification	FedAvgM
McMahan et al. [44]	Communication-Efficient Learning of Deep Networks from Decentralized Data	QFedAvg
Reddi et al. [53]	Adaptive Federated Optimization	FedOpt
Li et al. [33]	Federated Optimization in Heterogeneous Networks	FedProx
Reddi et al. [53]	Adaptive Federated Optimization	FedAdagrad
Reddi et al. [53]	Adaptive Federated Optimization	FedAdam
Reddi et al. [53]	Adaptive Federated Optimization	FedYogi

Table 5: Federated Learning strategies with corresponding papers

The pseudo-code for the Federated Dynamic Timeout Window (FedDyt) can be seen in Algorithm 2. After the server has set up the infrastructure and distributed the global model and datasets to the client (noted by initialize w_0), the learning happens in each round α. The intricacies of the aggregation strategies, weight updating, and server-client communication have been omitted. The main takeaway is that the number of successful clients, s, and the number of unsuccessful clients, f, together with the current timeout window, t, is sent to the *DynamicWindow*-function, and the result from that function is used to update the timeout window for the next communication round.

Algorithm 2 *Federated Dynamic Timeout Window.* s is the number of successful clients, f is the number of failed clients, and t is the timeout window.

Server executes:
initialize w_0
for each round $\alpha = 1, 2, \ldots, n$ **do**
  $t \leftarrow$ DynamicWindow(s, f, t)
end for
DynamicWindow(s, f, t) ▷ For each round α
$\rho \leftarrow \frac{s}{s+f}$
if $\rho <= 33\%$ **then**
  $t = t * 2$
else if $33\% < \rho <= 66\%$ **then**
  $t = t * 1.5$
else if $66\% < \rho <= 90\%$ **then**
  $t = t * 1.33$
  return t
end if

5.5.4 Definition of Communication Success Rate

As described in Section 5.2, a communication round in FL consists of three stages: dissemination, local computation, and uploading. If the server has implemented a timeout window to ensure that clients who are either unreasonably slow or have stagnated do not make a round continue forever if not all clients are done when the communication round hits the given window, there will be a portion of successful- and unsuccessful clients.

Therefore, for each communication round, the round success rate, ρ, is calculated. Let s be the number of successful clients and f be the number of unsuccessful clients.

$$\rho \rightarrow \frac{s}{s+f} \tag{2}$$

5.6 Communication

The last step is communicating the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences.

The problem and its importance were explained in Section 2.1, while its utility and novelty are described in Section 6.1. The discussion of the rigorousness of the design, as well as the relevance to researches and relevant audiences is described in Section 7.

6 Results

This section presents the design and implementation of the dynamic timeout window algorithm named Federated Dynamic Timeout Window (FedDyt) in Section 6.1. Then the experimental setup is described in Section 6.2, and lastly, the evaluation results are presented in Section 6.3.

6.1 Design and Implementation of the Dynamic Timeout Window

To achieve a successful dynamic timeout window algorithm in FL, it has to satisfy the following requirements:

1. Low complexity
2. Easy to integrate
3. Improve efficiency when clients time out

The proof-of-concept must also perform as well as the state-of-the-art solutions during ideal situations where clients are not timing/dropping out, as well as keep the benefits of Federated Learning (FL) such as maintaining data privacy.

6.1.1 Developer Interactions with FedDyt

One of the critical aspects of the artifact is the possibility of integrating it with existing solutions. Therefore it has to be adjustable to suit the needs of the individual FL scenarios of the developer, as well as be easy to integrate without increasing the complexity of their solution.

Adjust Parameters
The code was created in such a way that developers can easily adjust the inner parameters to suit their needs and requirements for performance. There are two main attributes to adjust. The first one is the lower, mid, and upper boundaries for when the solution should adjust the timeout window. The second one is the degree of incrementation of the timeout window when the conditions for a given boundary are met.

For each round of learning, the algorithm will find a round success rate, ρ, as shown in Equation 2 and compare it to the limits. By default, the algorithm contains the following boundaries:

- Low: $\rho \leq \frac{1}{3}$
- Medium: $\frac{1}{3} < \rho \leq \frac{2}{3}$
- High: $\frac{2}{3} < \rho \leq \frac{9}{10}$

Meaning that if the success rate of the clients is below 33%, it hits the conditions of the lower boundaries, and if it is between 66% and 90%, it will be within the high condition. For each of these conditions, the new timeout window is multiplied by a constant. The algorithm increases the timeout window by a greater value if there are a lot of unsuccessful clients that round, as the low boundary is met. Likewise, if there is a low degree of unsuccessful clients, the timeout window will be incremented to a lower extent as many clients managed to finish training within the given timeout window. The algorithm has the following defaults for adjusting the timeout window, t:

- Low: $t_{n+1} = t_n * 2$
- Medium: $t_{n+1} = t_n * 1.5$
- High: $t_{n+1} = t_n * 1.33$

- Else: $t_{n+1} = t_n$

If the success rate is above 90%, the timeout window will not be adjusted. The developer has a lot of possibilities in regard to tuning the algorithm to their needs. For instance, they might be satisfied with 70% of the clients returning successful results in any given round and can adjust the high interval to match that condition.

Implementation of Dynamic Timeout Window
In order for the dynamic timeout window code to work, it has to be properly integrated with the already existing solution on the server. This process is relatively straightforward and might be implemented like so: After the server has evaluated the model on a sample of available clients, the amount of successful and unsuccessful clients may be evaluated and sent to the dynamic timeout window algorithm to properly set the new timeout window. The algorithm then returns the updated timeout window which the developer can utilize to update the next communication round's timeout window.

6.1.2 Implementation Details

The entire artifact was implemented with the programming language Python, as it is the most common language within machine learning due to its simplicity and consistency. Instead of implementing the entirety of the Federated Learning (FL)-system from scratch, the friendly federated learning framework, Flower, was utilized. To quote Flower:

> Flower is a unified approach to federated learning, analytics, and evaluation. Federate any workload, any ML framework, and any programming language.[3]

The main benefit of the framework lies in that FL, evaluation and analytics require infrastructure to move machine learning models back and forth between the server and the clients, and Flower provides this infrastructure in an easy, scalable, and secure way. Furthermore, it provides several guides to configure and set up custom strategies, servers, clients, and client managers, as well as easy-to-read documentation.

The entirety of the solution was simulated by using Flower. In the context of the framework, it means that the whole FL system (a single server and a multitude of clients) was run on a single computer. It is not as simple in real-world FL systems for IoT where the client devices are separated on different devices. This implies that a lot of the characteristics are not present such as connectivity issues, network differences, and realistic hardware specifications.

There are two ways to set up the FL environment in Flower.

1. Create a server and client class. Spawn the server and then the number of clients through a batch script.
2. Utilize the built-in simulation function of the framework. It requires many parameters, such as the number of clients, resources, and client- and server configuration.

The different approaches both contain benefits and drawbacks. Approach 1) is the most realistic one as it creates all the instances of the client's devices and makes them wait for their turn to train. Meaning that if the system creates 100 instances and only one is being selected for each round of training, the other 99 are just waiting for their turn. Although this is how real-life FL is performed, it is really resource-heavy for a single computer. With approach 2), however, the clients are simulated through a function and are only executed once it is required to train. Thus, being much less demanding in terms of resources required to execute. Furthermore, approach 2) enables advanced real-time diagnostics through Ray Dashboard [4]. A screenshot of the dashboard can be seen in Figure 8.

[3] https://flower.dev/
[4] https://docs.ray.io/

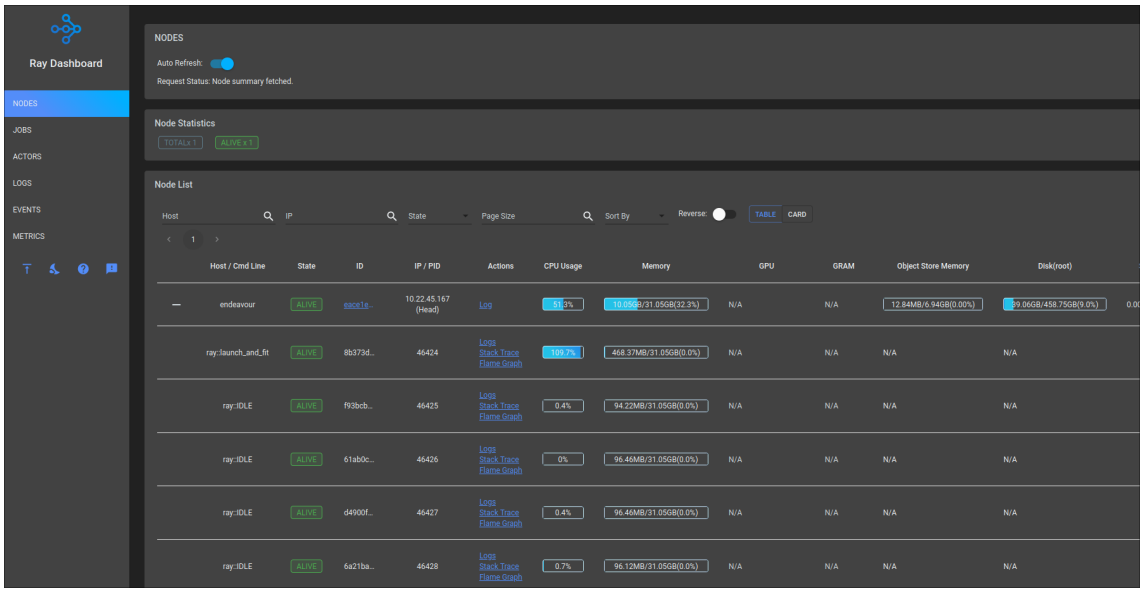


Figure 8: Ray Dashboard for Advanced Metrics

The dashboard requires configuration through Grafana and Prometheus, but once set up, it enables very useful metrics for viewing active clients and the number of resources being taken up by each client and in which state of training. In Figure 8, the first line corresponds to the application, and information about the current state, id, CPU Usage, and Memory is given by the dashboard. Furthermore, each client is registered as a node, and it is possible to view the status of the client through the "Host / Cmd Line" column. The figure shows four clients that are currently idle. The dashboard further gives the possibility to view logs and stack traces for each individual client and for separate jobs and events.

Based on the information above, approach number 2) was selected as it was highly beneficial to reduce the number of hardware resources needed for all the simulations to be executed on a single laptop computer, as well as the added benefits of a dashboard to follow the progress of each client when debugging execution times for clients timing out.

6.2 Experimental Setup

6.2.1 Environment

The experiments were run on a 64-bit single desktop computer (Lenovo Thinkpad X1 Carbon G10) with a 4.7GHz 12th Gen Intel Core i7-1260P, 32GB DDR4 RAM, running EndeavourOS with Hyprland on kernel 6.2.12-arch1-1. To run the experiments, Python was used with TensorFlow, Keras, and Flower. An overview of the software used can be seen in Table 6, and the hardware specifications can be seen in Table 7.

Table 6: Software used in the experimental setup

Software	Version	Usage
Linux Kernel	6.2.12-arch1-1	Operating System
Python	3.10	Implementation Language
TensorFlow	2.9.1	Machine Learning Framework
Keras	2.11.0	Machine Learning API
Flower	1.3.0	Federated Learning Framework
Matplotlib	3.7.1	Python plotting package

Table 7: Hardware used in the experimental setup

Hardware	Version	Additional Info
Laptop Model	ThinkPad X1 Carbon Gen 10	Architecture x86_64
Processor	12th Gen Intel Core i7-1260P	12-core (4-mt/8-st)
RAM	32GB DDR4	3200 MT/s
GPU	Intel Alder Lake-P Integrated Graphics	Driver i1915
Operating System	Endeavour	Hyprland as Wayland compositor

6.2.2 Server

The server has an integral role in FL as it is responsible for coordinating clients, resources and aggregating the model updates received from the clients.

For each round of learning, the server has several responsibilities. It has to train the model and replace the previous global model with the new and improved one. Then it has to evaluate the model based on the implementation of a given strategy (such as the strategies utilized in Table 5). Then it evaluates the model on a sample of the available clients.

Table 8: Overview of parameters

Parameter	Value
Number of clients	100
Fraction fit	0.1
Fraction evaluate	0.1
Min fit clients	3
Min available clients	10
Proximal mu	0.1
Initial parameters	{}

To validate the artifact's effectiveness, the algorithm was executed and validated with seven different strategies listed in Table 5. For the purpose of keeping the performance as equal as possible between the different strategies, each strategy was given the same parameters as far as possible. A list of the parameters can be seen in Table 8. The number of clients corresponds to the total amount of clients during the simulation. It was set to 100 to enable a large pool of clients to simulate a semi-realistic application scenario for IoT, where there generally are many such devices. The fraction fit, and fraction evaluate parameters correspond to the fraction of clients used during training and evaluation, respectively. They were set to 0.1 to select ten clients in each communication round. To ensure that there always were enough available clients, that parameter was set to ten. All the aggregation strategies used the same number of clients, fraction fit, faction evaluate, min fit clients, and min available clients, but the proximal mu was only required in the FedProx strategy. Li et al. [33] stated that finding the proximal mu may be difficult, but they provided a candidate set of {0.001, 0.01, 0.1, 1}. Because this thesis did not investigate the accuracy of the different strategies, 0.1 was chosen randomly without testing the potential gains of the others on the datasets in the experiments. The initial parameters were set to an empty configuration object, thus forcing the server to get the initial parameters from the clients.

By default, Flower does not give the accuracy from each round of training. Therefore, each strategy needed an aggregation function in order to get the communication round results. The framework stored the accuracy for each client from each round of training. A weighted average function was created to get the communication round accuracy by multiplying the accuracy of each client by the number of examples used and was then saved.

The server was also responsible for initializing the FL and distributing the datasets to the clients. An overview of the flow can be seen in Figure 9. During the initialization phase, the server checks whether or not the datasets have been downloaded and are situated in the cache. Otherwise, they are downloaded using the Keras API. Instead of giving each client the exact same replica of the dataset, the server partitioned the data into 100 sub-partitions (equal to the number of clients) and distributed the unique sub-partitions to each of the clients. As the data came from the same datasets, it ensured that each client received consistent data in terms of features and samples and in the exact same format. The only difference existed in the training data with corresponding classification labels. When the initialization phase was finished, and every client had received their

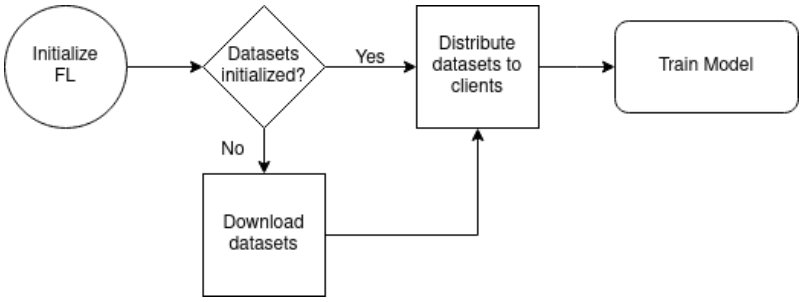


Figure 9: Flow of Data Initialization

individual dataset, the server started orchestrating the communication rounds and trained a global model by aggregating the individual results of the clients.

6.2.3 Clients

To simulate the time spent by clients through communication and computing, a timeout value was given to each client where they waited during the fitting process. To ensure different values, but not entirely random, these values were normally distributed with the Gaussian function, shown in Equation 3.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3}$$

The $f(x)$ corresponds to the probability density where σ is the standard deviation and μ is the mean. The function has its highest value at the mean and gets wider when the standard deviation increases. The main benefit of the equation lies in the increased probability of it returning values that are close to the mean.

From the experimental setup, there are 100 clients, where in each round, ten are sampled. To simulate differences in training time between the different clients, each was given a timeout during training following the normally distributed probability function as shown in Equation 3. Although the values of the training time are not important in this setup, the value has to be large enough to allow clients to finish training within a given timeout window. Through experiments, the timeout was set to two to allow for a quick training time. IoT-devices may have huge varieties in hardware specifications, leading to huge differences in performance in terms of learning time. Therefore, a large standard deviation time was set to one, a standard deviation of 50% of the experienced training time. Therefore, the sample training time of the clients resulted in being set to two seconds with a standard deviation of one second.

It would be beneficial to test how the default algorithms perform under challenging timeout windows. To find appropriate timeout values we construct a 90%-confidence interval with $\alpha = 0.1$ where $\bar{X} = 2$, $Z = 1.645$, $\sigma = 1$ and $n = 100$. Here $\bar{X}$ is the sample mean, Z is the z-score, σ is the standard deviation, and n is the number of clients. To construct the confidence interval, we need to find the lower and upper limits:

$$= \left[\bar{X} - Z \times \frac{\sigma}{\sqrt{n}}, \bar{X} + Z \times \frac{\sigma}{\sqrt{n}}\right]$$

$$= \left[2 - 1.645 \times \frac{1}{\sqrt{100}}, 2 + 1.645 \times \frac{1}{\sqrt{100}}\right]$$

$$= \left[1.836, 2.164\right]$$

We know that 90% of the clients will exide within the range 1.836 and 2.164. A visualization of the client distribution can be seen in Figure 10.

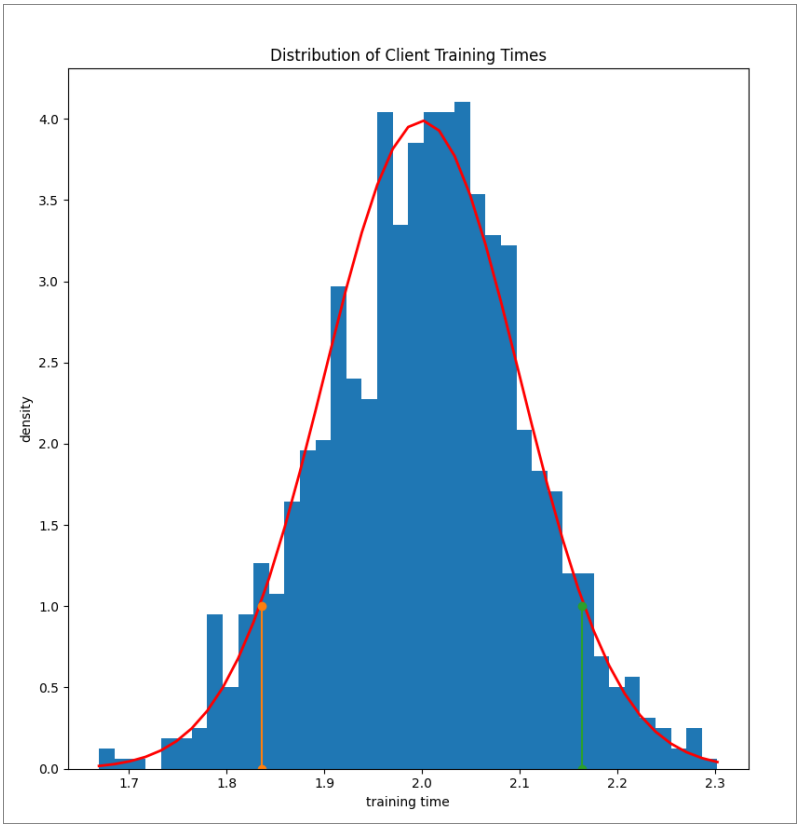


Figure 10: Client normal distribution with mean 2

Figure 10 shows a histogram of an example distribution of how the client training times were distributed during one of the simulations. This distribution was unique for each simulation but followed the same probability function and would therefore give very similar visual results every time. Each client's time was categorized into 40 bins and then divided by the total number of counts and the bin width to obtain the density. The corresponding probability density function of the clients was drawn in red, which shows the expected normally distributed result. The lower and upper limits of the 90%-confidence interval were also added to the figure. The lower limit is visualized with an orange line, while the upper limit is shown with a green line.

6.2.4 Machine Learning Models

The FedDyt-algorithm was used in two experiments to evaluate its compatibility with existing FL solutions and show its promise for future solutions. Keras's load function was used to load the relevant datasets before initializing the machine-learning models if these were not already initialized on the server and the clients.

***MNIST* Dataset Experiment**
This experiment used the *MNIST* dataset, which contains 10 features and 60,000 samples. Each sample consists of 28x28 black-and-white images separated into 10 classes consisting of handwritten digits. It is commonly used in FL and is an excellent dataset for testing basic classification functionality.

The *Sequential*[5] model from Keras was used with four layers where the first layer had input shape (28, 28), the second layer was relu-activation, the third level contained a dropout of 0.2, and the

[5] https://keras.io/guides/sequential_model

last layer was softmax-activation. The model was compiled with Adam as an optimizer with sparse categorical cross-entropy as the loss function.

***CIFAR-10* Dataset Experiment**
The second experiment used the *CIFAR-10* dataset, which also contains 10 features and 60,000 samples. Each sample consists of 32x32 color images in 10 classes, with 6000 images per class. It is similar to the *MNIST* dataset as it also classifies the model with ten possible outcomes.

The *EfficientNetB0*[6] architecture from Keras was used with the input shape as (32, 32,3), no initial weights, and 10 classes. This model was also compiled with Adam as an optimizer and with sparse categorical cross-entropy as the loss function.

6.2.5 Evaluation of Performance

To evaluate the performance of Federated Dynamic Timeout Window (FedDyt), two Evaluation Questions (EQs) were defined:

- **EQ1: How does the communication success rate of FedDyt compare to the state-of-the-art?**
- **EQ2: How does the efficiency of FedDyt compare to the state-of-the-art when there are slow client outliers present?**

The first EQ obtains information about how the artifact of this thesis, namely FedDyt, compares to the stat-of-the-art in terms of communication success rate. As discussed in Section 5.2.2, the communication success rate is a highly important metric as it directly measures the number of successful clients for each communication round, and if FedDyt manages to outperform the existing solutions it would be very beneficial. To measure the communication success rate it was necessary to obtain some basic metrics first. Firstly, there had to be a baseline for how the state-of-the-art performed under ideal conditions and where each client had unlimited time to finish training. Secondly, there had to be results of how the state-of-the-art performed under sub-ideal conditions. These conditions were described in Section 6.2.3, and gave insights into how FedDyt possibly improved upon existing solutions in those conditions. The result of EQ1 is shown in Section 6.3.3.

The second EQ gives insights into how FedDyt compares to the state-of-the-art in terms of efficiency when there are slow client outliers present in the FL-network. These client outliers are client devices that spend an abnormally long time during training, which in synchronous FL without a timeout window is highly consequential as described in Section 5.2.2. To investigate the efficiency of both FedDyt and the state-of-the-art, the experiments were run on the *MNIST* datasets where one percent of the clients were given a large timeout. The result of EQ2 is shown in Section 6.3.4

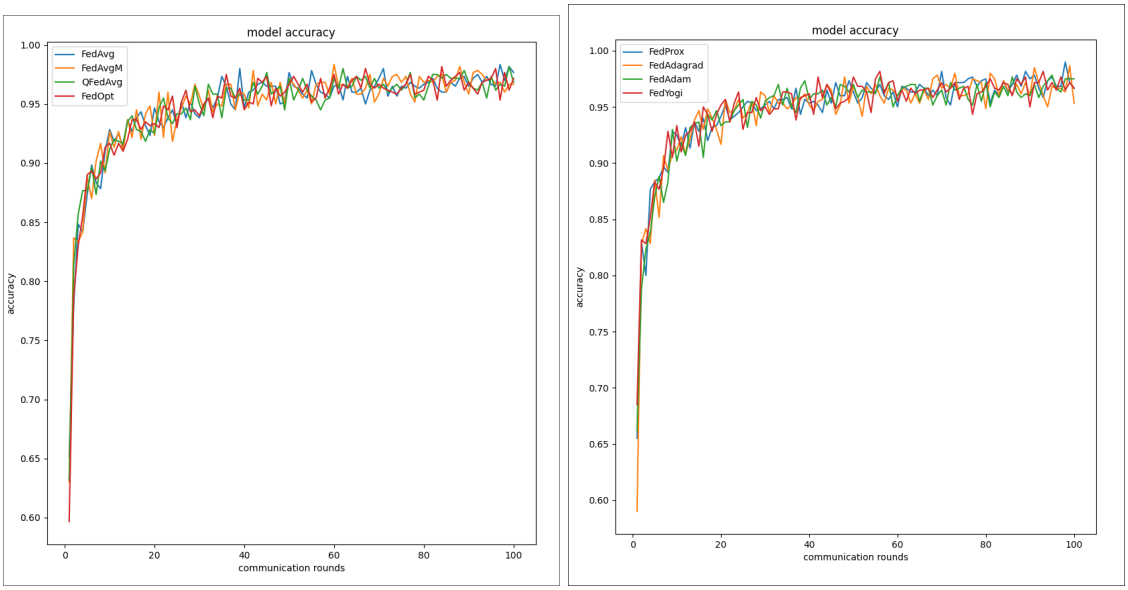
6.3 Experimental Results

6.3.1 Baseline

To obtain a baseline for optimal results, several FL-algorithms were run on the MNIST dataset with standardized parameters, no client timeout, and unlimited time in each round. The simulation consisted of 100 communication rounds between the clients and the server, where each client performed two local epochs. The server randomly selected ten clients within each round.

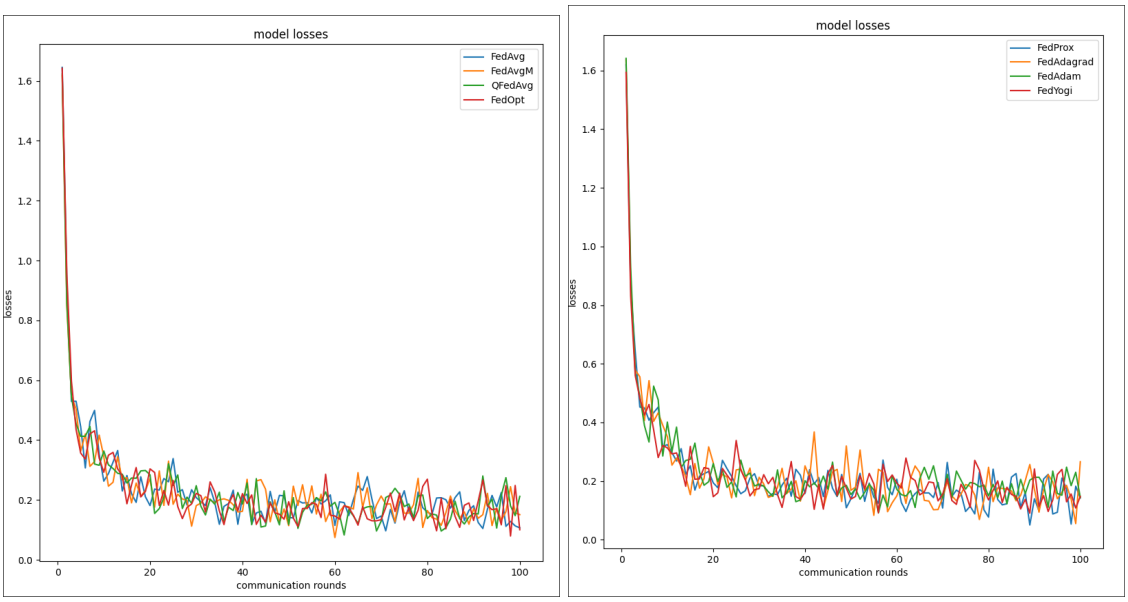
The accuracy of the baseline can be seen in Figure 11 and the corresponding losses in Figure 12. From the graphs, it is clear that the different strategies all converge towards a high accuracy of around 97% and a loss of around 0.15%. In a practical scenario, this means that the global model is able to correctly identify the handwritten digit 97 out of 100 times. In addition, it is clear that there are no significant gains from further communication rounds after round 60.

[6] https://keras.io/api/applications/efficientnet



(a) FedAvg, FedAvgM, QFedAvg, FedOpt **(b)** FedProx, FedAdagrad, FedAdam, FedYogi

Figure 11: Baseline accuracy for MNIST dataset



(a) FedAvg, FedAvgM, QFedAvg, FedOpt **(b)** FedProx, FedAdagrad, FedAdam, FedYogi

Figure 12: Baseline losses for MNIST dataset

6.3.2 Performance with Clients Timing Out

After knowing how several algorithms perform under ideal conditions and with unlimited time, it is worth investigating the performance when clients cannot finish training within a given time window. From the experimental setup, we know that the clients are normally distributed with a mean of two seconds. We can use the constructed confidence intervals to investigate the effect of a given percentage of clients within that given timeout window. From the obtained results from the baseline experiments, it was apparent that the algorithms stagnated at a high accuracy and low loss after around 60 communication rounds. To limit the total training time for each strategy, the rest of the results were run for 60 communication rounds.

Lower Bound Timeout Window
From Section 6.2.3, we know that by setting the timeout window to 1.836 seconds it will exclude approximately 95% of the clients due to them taking too long during the training phase. The results can be seen in Figure 13a.

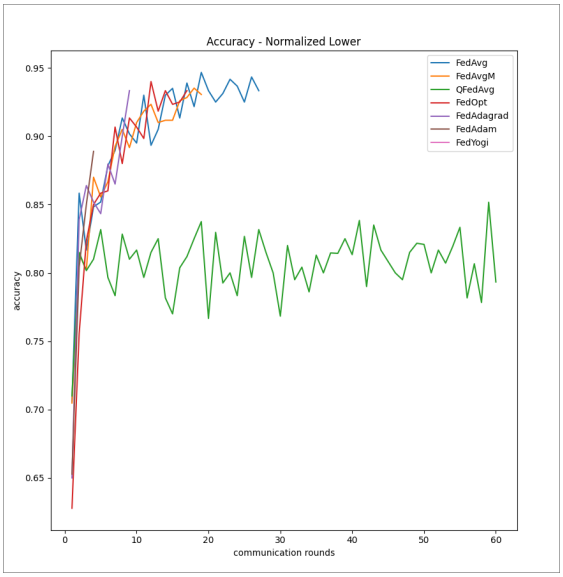
From the graph, it is clear that several disadvantages have been introduced to the overall efficiency of the learning network. The most prominent is the number of successful communication rounds. Out of the 60 communication rounds, the most successful rounds reported were 30 from the FedAvg strategy, and the least successful being FedAdam with only five. This means that the efficiency of the network in terms of successful rounds went from 100% down to the range $[8.3\%, 50\%]$. It is apparent that QFedAvg (the green line in Figure 13a) got stuck in the accuracy range $[76\%, 85\%]$, and also managed to complete all 60 communication rounds of learning. It is unknown why it got stuck in this accuracy loop without failing any communication rounds.

Mean Timeout Window
It was valuable to view how the algorithms performed when the timeout window was the same as the client distribution mean of two seconds. In theory, this would lead to approximately half of the clients finishing training within the given time window and the other half using too much time. The result can be seen in Figure 13b.

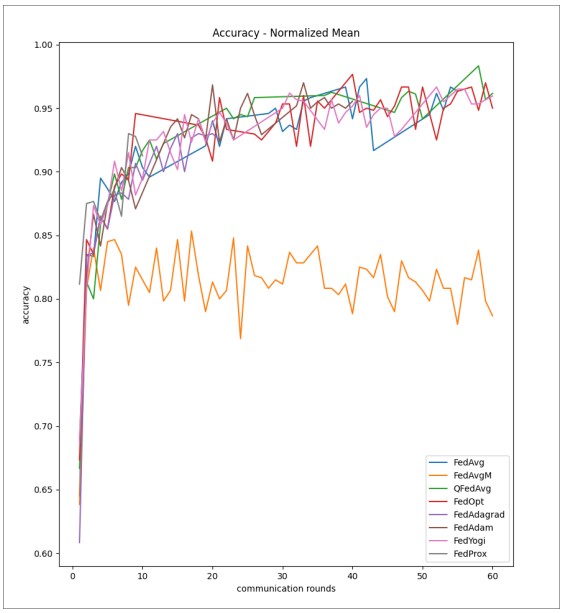
From the graph, it is apparent that in terms of successful communication rounds, there is a significant improvement from the results of the lower bound timeout window. First and foremost, five out of the seven strategies managed to have more than 90% successful communication rounds. While FedAdagrad and FedAdam only had 20 and 40 successful rounds, respectively. It is apparent that FedAvgM (the orange line in Figure 13b) got stuck in the accuracy range $[76\%, 85\%]$, and also managed to complete all 60 communication rounds of learning. This is the exact same range that happened with QFedAvg for the lower bound timeout window experiment.

Upper Bound Timeout Window
The higher bound was set to 2.164, as this number will include 95% of the available clients. The results can be seen in Figure 13c. The graph shows that almost all the strategies managed to complete close to all 60 communication rounds, with only a few failed ones. From the graph, it is apparent that QFedAvg and FedAvgM both did not gain much accuracy from each communication round, as a linear fit of their accuracy gives almost a constant accuracy. The best and worst accuracy range is similar to the outliers of the lower and mean timeout window experiments, which also included the same two FL-strategies.

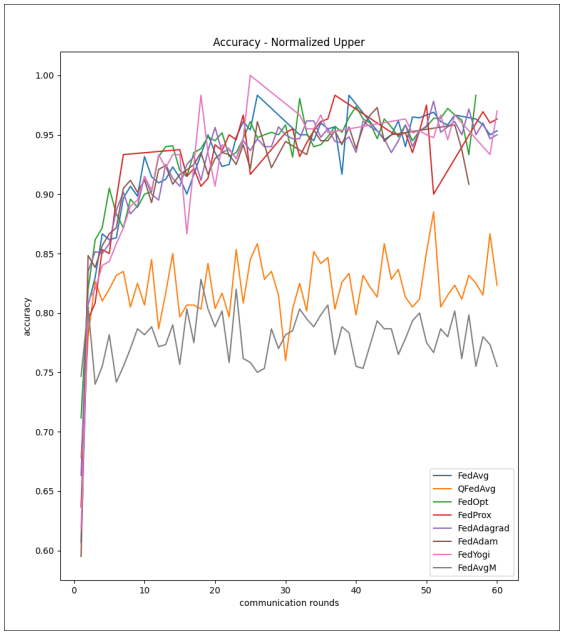
To summarize the findings for when clients are timing out. When a large number of clients are timing out, the efficiency of the FL-network substantially deteriorates as there is a very high degree of unsuccessful communication rounds between the server and the clients. In the experimental setup, the communication between the server and the clients is close to instantaneous, as the clients are run as processes on a single computer rather than distributed over large distances, as in a real-life scenario. The cost of communication in FL is very expensive [68] in the practical applications where the client devices are widely spread apart, such as for IoT. Therefore, when more than half of the communication rounds are unsuccessful, it is a lot of wasted bandwidth, energy, time, and resources spent on both the server and client devices. When a significant portion of the clients (more than half) manages to finish training, the results quickly improve. The server receives enough results each communication round to aggregate them into the global model successfully.



(a) Lower Bound Timeout Window



(b) Mean Timeout Window



(c) Upper Bound Timeout Window

Figure 13: Accuracy vs. Communication Rounds when Clients Time Out

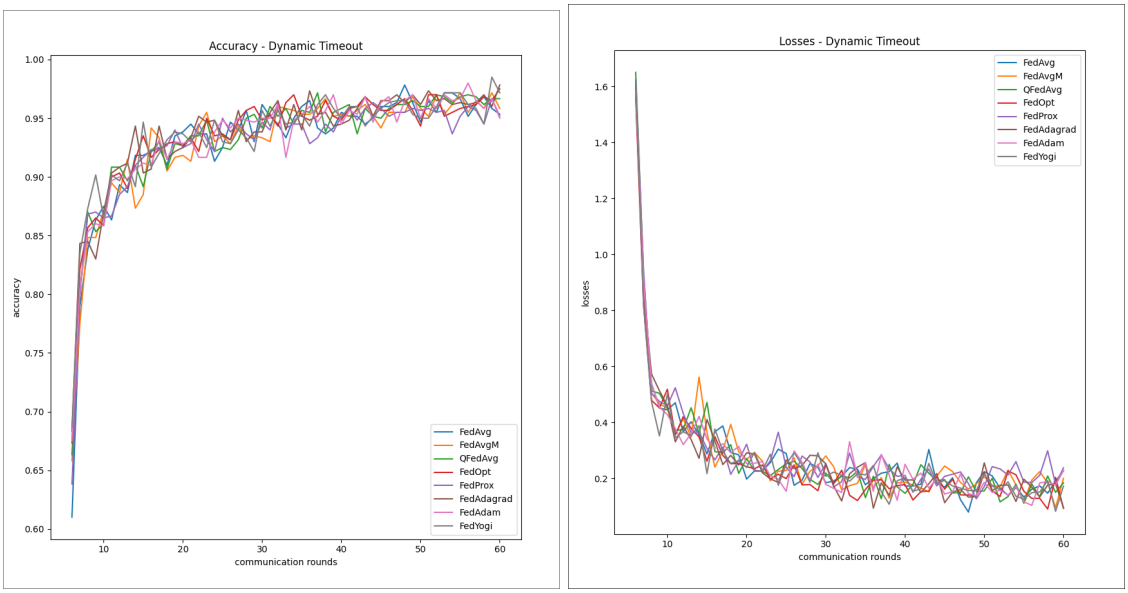
Table 9: Overview of results when clients are timing out. *s* - number of successful rounds.

Strategy	s - lower bound	s - mean	s - upper bound
FedAvg	27	55	60
FedAvgM	19	60	60
QFedAvg	60	60	60
FedOpt	17	60	57
FedProx	-	60	60
FedAdagrad	6	21	60
FedAdam	4	40	56
FedYogi	1	60	60

An overview of the number of successful rounds under the different timeout conditions *lower*, *mean* and *upper* can be seen in Table 9. From the table, it is clear that the strategies performed poorly in the lowest conditions when there were a lot of clients timing out. It significantly improved when the timeout window was set to match the expected timeout of the clients, as it resulted in a lot of clients being able to finish training, thus giving results to be aggregated to the global model. Finally, the number of successful rounds was close to 100% when the timeout window was set to include most of the clients for each round of learning.

6.3.3 How does the communication success rate of FedDyt compare to the state-of-the-art?

To induce a scenario where clients were bound to time out, the initial timeout window was set to 0.1s while the clients followed the same distribution as described in Section 6.2.3. For each round, the number of successful and unsuccessful clients was reported by the server during the evaluation phase of the communication round. After these values were reported, the number of successful and unsuccessful clients, together with the desired limits and adjustments, were sent to the FedDyt algorithm. The algorithm's output could then be used to dynamically update the timeout window for each communication round as described in Section 6.1.1. Applying FedDyt to the FL-network with the eight different synchronous aggregation strategies gave the corresponding accuracies and losses, which can be seen in Figure 14.



(a) Accuracy vs. Communication Rounds **(b)** Losses vs. Communication Rounds

Figure 14: Accuracy and Loss with Dynamic Timeout Window

There were significant improvements by utilizing FedDyt on clients with the same distribution as mentioned in Section 6.2.3. In the worst case, as few as only one or no communication rounds were successful when many clients were timing out. FedDyt managed to push the timeout window such that most of these clients managed to finish training before the communication between them and the server was shut down.

In practice, this led to more clients being able to contribute to each round of learning, which in turn led to a decrease in the relative cost of communication and an increase in overall fairness. As the experiments were IID, there were not a lot of accuracy gains since the clients all have data from the same dataset with the same features. If the experiments were IID, it probably would have led to accuracy gains since the aggregation from the different clients would have been more beneficial.

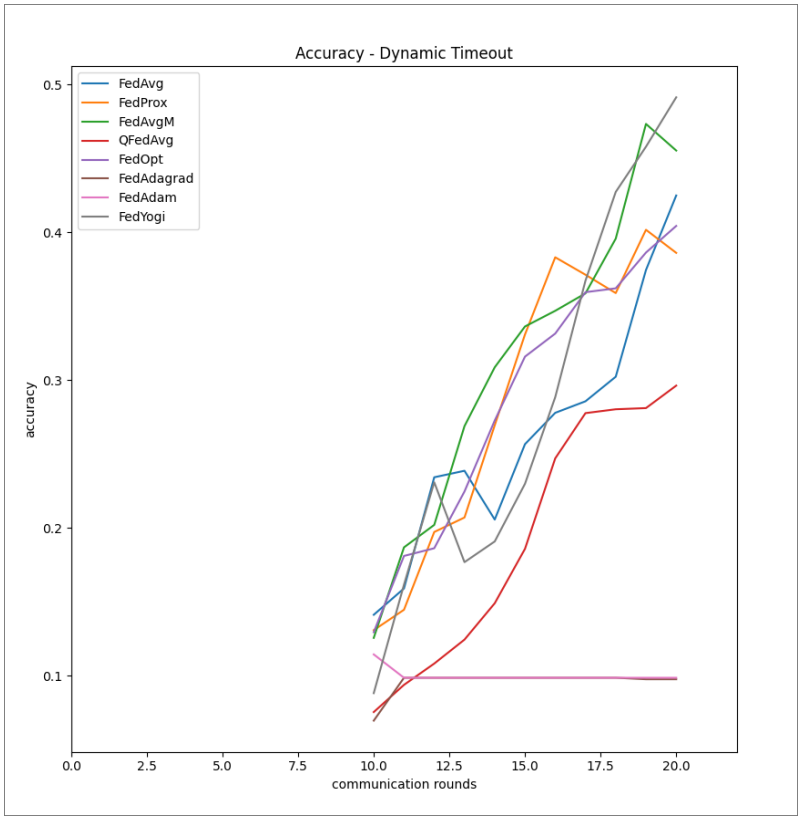


Figure 15: Accuracy vs Communication Rounds - CIFAR-10

To further validate the effectiveness of FedDyt, the algorithm was implemented alongside the *CIFAR-10* dataset. The experiment was run for twenty communication rounds on all aggregation strategies, and the results can be seen in Figure 15. The figure shows that FedDyt used the first ten communication rounds to dynamically find a timeout window which allowed the clients to finish training within the deadline before timing out. The accuracy of the global model kept increasing during the later rounds after finding an optimal timeout window for every strategy except FedAdam and FedAdagrad. It is unknown why their accuracy was stuck at 0.1 with only minor adjustments for each round of learning. However, FedDyt performed as expected on these aggregation strategies as it managed to find a proper timeout window for all of the strategies.

To summarize, the performance of the dynamic timeout window algorithm outperforms the state-of-the-art when there are clients that time out in the experimental setup. FedDyt used six communication rounds in the *MNIST* experiment and ten communication rounds in the *CIFAR-10* experiment to dynamically find a timeout window to allow clients to finish training. All of the strategies were able to perform to the same degree as when they were allowed to run with unlimited time under ideal conditions.

6.3.4 How does the efficiency of FedDyt compare to the state-of-the-art when there are slow client outliers present?

As shown in Section 6.3.3, the accuracy matches the state-of-the-art running under ideal conditions when implementing FedDyt alongside the various FL-strategies. However, it does not show the differences in the time spent learning especially in case when there exists outliers within the network. An outlier in this case meant a client who used a significantly longer time during learning. Thus making the entire network more inefficient as both the other clients and the server has to wait for that one client.

To illustrate this scenario, 1% of the clients were randomly given an additional 300 seconds during the learning step. The experiments were run ten times with ten communication rounds and then averaged in order to get a general result. The results can be seen in Figure 16.

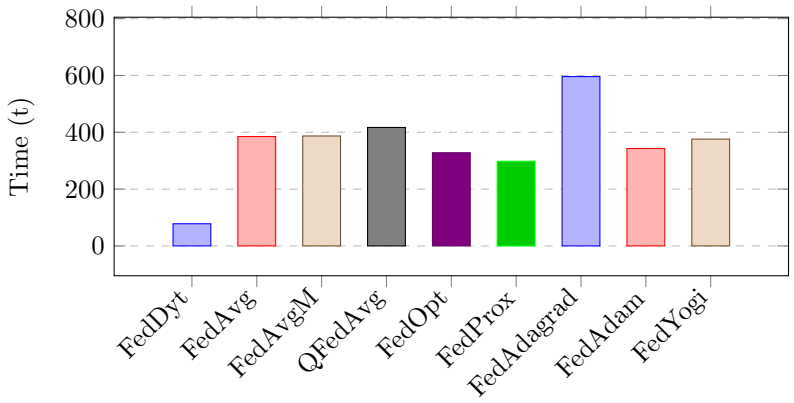


Figure 16: Efficiency of FedDyt compared to synchronous strategies with outliers

The reason that the synchronous protocols in the simulation cannot match FedDyt was that the global servers were waiting for the outlier to return the model update before all the updates were aggregated. From Figure 16 it is clear that in this experiment, the value of 300 seconds was almost added flat onto the value of FedDyt as the likelihood of the network having an outlier was 1%, and by selecting ten random clients out of a pool of 100 clients would mean that on average one round would be delayed by the outlier's amount. If, for some reason, there exists an outlier in each round of training, this would result in each round taking 300 seconds longer and resulting in ten rounds of learning being finished in a minimum of 3000 seconds.

The average of the eight synchronous protocols (FedAvg, FedAvgM, QFedAvg, FedOpt, FedProx, FedAdagrad, FedAdam, FedYogi) was 391 seconds, compared to the result of FedDyt which was 78 seconds. It is worth noting that FedAdagrad used 50% longer than the other synchronous protocols, and FedProx used 25% less time. This is due to the fact that the amount of slow client outliers were randomly assigned to 1% of the client population, and for each round of learning, 10% of the clients were randomly selected from the total amount of 100 clients. Therefore, during the aforementioned simulations, the slow client outliers appeared less and more frequently simply by chance. Thus, implementing FedDyt resulted in significantly improved results as it outperformed the state-of-the-art with a multiple of five in this experiment.

7 Discussion

This section will discuss the experimental results from Section 6, compare the results to related work, discuss implications to industry and academia, and address threats to validity.

7.1 Comparison to Related Work

Although FedDyt managed to answer the main RQ of this thesis **"How can we improve the communication success rate and minimize slow client outlier's impact on learning efficiency in Federated Learning with Independent and Identically Distributed (IID) data?"** by managing to dynamically set the timeout window to reduce the effect of clients timing out in regards to communication success rate and in terms of efficiency when facing slow client outliers.

However, this thesis was not the first effort put into researching and minimizing the effect of clients timing out in FL. It was the main topic for the related work, but this thesis and the related work differ mainly in regard to the abstraction layer and the overall complexity.

7.1.1 Abstraction Layer

FedDyt differed from the related work through its abstraction from existing and future solutions. It enables it to be implemented alongside other solutions. Whereas the related work is not possible to integrate with other solutions necessarily. For instance, asynchronous FL [11, 74, 71] alters the flow between the clients and the server, and it would not be possible to utilize synchronous aggregation protocols alongside it. Similarly, with secure aggregation protocols [5, 6, 7, 12, 28, 38, 66, 65], it would not be possible to combine it with existing solutions as it altered how the results from each client should be securely aggregated to form a global model.

7.1.2 Complexity

Another large difference between FedDyt and the related work is the difference in complexity between the solutions. Complexity is not, by default, a bad thing. However, when implementing large neural networks between a multitude of clients with many moving parts, adding to this complexity can quickly become difficult to maintain, troubleshoot and improve. Therefore, FedDyt outperforms the related work in terms of complexity, where the solution may easily be implemented alongside existing work.

7.2 Possible Applications

FL has seen a lot of promising domains for possible applications such as within recommender systems, smart cities and autonomous industry, Natural Language Processing (NLP), edge computing, and IoT. A complete figure of the taxonomy with corresponding subdomains can be seen in Appendix B. Within certain domains, such as healthcare, biomedical and industrial, the developers generally know the hardware specifications of the clients who are going to perform the learning.

The algorithm created and tested in this thesis, FedDyt, is very well suited for application domains where there are large variations between the clients in terms of either network speed or connectivity, hardware performance, or any factor which contributes to an increase the total time a client spends from receiving the initial model to returning the computed model update. As discussed in Section 4.2, when there are large differences in the time which the client uses, it leads to stragglers who are left behind and slows the entire communication round as the server is waiting for all the clients before updating the responds synchronously to the global model.

Therefore, a very well-suited application domain for FedDyt is the growing domain in IoT. This domain encompasses millions of devices that coexist with large heterogeneity both in terms of hardware specifications, network, and possible datasets. Some clients are guaranteed to be significantly slower than others due to these differences, making the algorithm very well-suited for this domain.

7.3 Implications to Industry

While Federated Dynamic Timeout Window (FedDyt) is not ready to be deployed to real-life applications in the industry, the artifact showed promising results through the experimental results. Before being deployed to the industry, it has to be tested in a more well-suited environment than a simulated FL network on a single computer. The simulated FL network lacks several of the properties of a real-world environment, such as encryption, transportation time, connection dropouts, and timeout (from hardware limitations instead of simulated), to name a few.

One of the main benefits of the artifact lies in the abstraction layer from other solutions. Instead of being a specific aggregation protocol or similar FL-specific algorithm, it is possible to implement alongside other protocols as a stand-alone improvement. This abstraction is highly beneficial to the industry as it is not dependent on their current or future infrastructure or solution. Furthermore, due to its low complexity, it would not require too much work to implement alongside its solution. Thus, making it relatively simple to test in their use cases and possibly remove if not bringing beneficial results.

Another valuable benefit of the artifact is that FedDyt finds a timeout window that meats the threshold the developer sets (if not choosing the default intervals). The industry can use this feature to find a timeout window for their environment and then utilize that window without further using the artifact.

7.4 Implications to Academia

The experimental results showed that successfully adjusting the timeout window such that enough clients managed to finish training led to a substantial increase in successful communication rounds of learning. The effect of this meant that the communication costs decreased as there were fewer communication rounds wasted rounds. Although this is beneficial in itself, the corresponding results in accuracy between the ideal scenario and the worst-case scenario were not too different. This is due to using IID data in the experiments, where each client has very similar data with identical features. Therefore, if a low percentage of clients finish training, those results will still be highly beneficial to the entire pool of clients as the results are directly transferable. Therefore research into the effects of FedDyt with Non-IID data should be conducted.

Although FedDyt was designed with IoT-devices in mind, as it is a field with large differences in hardware specification, further studies should be performed exploring the different application areas where these differences exist.

As mentioned in Section 7.3, more work has to be conducted to test the practical applications of FedDyt in a real-life scenario. Researchers should therefore implement and study the impact of dynamic timeout window in a realistic setup. For instance, by implementing the client devices on IoT devices and performing several aggregation strategies on a separate server.

7.5 Threats to Validity

For any research, it is paramount to threats to validity. If the research does not have high validity it corresponds that it does not produce results that correspond to actual properties and characteristics in the real-world. Section 7.5.1 explains the threats to internal validities, while Section 7.5.2 the threats to the external validities

7.5.1 Internal Validity

One assumption made during the experimental setup is that the clients are normally distributed. This assumption implies that the time spent to train a local model is symmetrically distributed with little to no skew. This assumption was made because the researcher could not find any scientific research on the distribution of IoT-devices. If the clients in a real-world scenario have another distribution, it will undoubtedly lead to different results. One possible scenario is that a large portion of the clients are much slower than the rest; perhaps they never even finish training. In this scenario, FedDyt will, for each communication round, dynamically increase the timeout window until a satisfactory amount of clients are included, which might be a considerable number or never happen.

The thesis utilized the Flower framework for conducting the experiments as it would require too much time to set up the entire real-world infrastructure. It certainly gave a lot of benefits in terms of getting the infrastructure up quickly to start testing several strategies. However, it might have brought a selection bias into the selection of FL aggregation strategies as all of the tested strategies were chosen out of the available pool included in the framework. All of these strategies were synchronous, which is relevant for most of the existing solutions, but it would be more appropriate to include other types of strategies, such as asynchronous. The framework's infrastructure was built with synchronous strategies in mind, and therefore not possible to utilize other types without building most of it from scratch. The student did therefore omit strategies other than synchronous ones.

All the research and experiments were conducted by a single student during one semester. Therefore it is a probability of biases and mistakes in all stages of the process. For instance, during the process of finding related work and existing solutions to the problem, information may have been overlooked or misinterpreted. A more robust approach would have been to utilize one or more researchers to conduct the same research in parallel and then validate the results.

7.5.2 External Validity

The experiments performed in this thesis use the *MNIST* and *CIFAR-10* datasets, which are great for machine learning classification. As these are divided into equal parts and distributed to the clients, it means that each client contains very similar data and an identical number of samples which implies that they are Independent and Identically Distributed (IID). In real-world IoT-applications, the datasets are seldom IID, and it is a large reason why most of the previous work focuses on solving the Non-IID issues in FL. A more realistic approach would be to find datasets that simulate these differences and distribute those to the clients. Nonetheless, as the main benefit of the artifact of this thesis lies in its ability to be implemented alongside existing and new strategies, it still brings novelty and possible benefits to future improvements, which improves upon the challenge.

This thesis also assumes that FL is a rising methodology within machine learning that will largely increase in the future. At the time of writing, when entering the search term "Federated Learning" into Google Scholar, about 17.000 results are being shown from the last year, which indicates that a lot of research and interest is being conveyed on the topic. Still, there are very few practical and successful applications in real life at the time of writing. One such application is Google's GBoard next word prediction algorithm [59]. Therefore, the value of the artifact and its novelty depends on how the future manages to adopt FL on a much larger scale.

8 Conclusion and Future Work

This thesis presented a novel algorithm for setting a dynamic timeout window in FL, named Federated Dynamic Timeout Window (FedDyt). The main benefit of the algorithm is that it finds an appropriate timeout window to be able to include clients without sacrificing efficiency in terms of time. A proof of concept was implemented to prove its effectiveness on eight existing strategies. The experimental results firstly found a baseline for how the strategies performed in an ideal scenario with unlimited time, then the efficiency when there were various degrees of clients timing out, and lastly compared it when applying FedDyt.

The experimental results showed that when applying the algorithm in scenarios where the state-of-the-art struggles due to client timing out, the efficiency of the network returns to the optimal accuracy only after spending a few rounds to find the timeout window. Furthermore, in the existence of slow client outliers, FedDyt drastically improved the efficiency compared to the state-of-the-art synchronous protocols without a timeout window. Instead of waiting for the slow outlier to finish training before aggregating the local result into the global model, FedDyt adjusted the timeout window to exclude the outliers, which resulted in an improvement in efficiency directly proportional to the time spent by the slow client.

Another benefit of FedDyt is its abstraction to other FL strategies and protocols while having a low complexity of implementation, making it possible to implement alongside other solutions efficiently. This makes it valuable for applications where the developers do not know how much time each client device spends during training and can utilize the dynamic timeout window to get a more efficient learning network. Furthermore, the developers do not need to use the dynamic timeout window on every run of the FL-network but run it once to find a suitable timeout window to be used on further experiments.

While the experimental results showed that applying FedDyt to existing algorithms is possible, there are still many possible future directions both for development and verification. Firstly, it would be very beneficial to view how it performs in a more realistic scenario with either Non-IID data or on actual distributed IoT-devices. Furthermore, already existing solutions claim that one of the limitations is having to set the timeout window manually for their clients. It would be beneficial to see how FedDyt performs in junction with those solutions [33].

The FedDyt-algorithm functions as a proof-of-concept, and there are a lot of uncertainties regarding the best limits and adjustments of the dynamic timeout window. For instance, a deeper convergence analysis of what the best adjustments towards an optimal upper limit within the normal distribution of clients would be highly beneficial; if normal distribution even is the correct distribution for IoT-devices.

Bibliography

[1] CIFAR-10 and CIFAR-100 datasets, . URL https://www.cs.toronto.edu/~kriz/cifar.html.

[2] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, . URL http://yann.lecun.com/exdb/mnist/.

[3] S. Abdulrahman, H. Tout, A. Mourad, and C. Talhi. FedMCCS: Multicriteria Client Selection Model for Optimal IoT Federated Learning. *IEEE Internet of Things Journal*, 8(6):4723–4735, Mar. 2021. ISSN 2327-4662. doi: 10.1109/JIOT.2020.3028742. Conference Name: IEEE Internet of Things Journal.

[4] W. J. Baumol. Welfare Economics and the Theory of the State. In C. K. Rowley and F. Schneider, editors, *The Encyclopedia of Public Choice*, pages 937–940. Springer US, Boston, MA, 2004. ISBN 978-0-306-47828-4. doi: 10.1007/978-0-306-47828-4_214. URL https://doi.org/10.1007/978-0-306-47828-4_214.

[5] C. Beguier, M. Andreux, and E. W. Tramel. Efficient Sparse Secure Aggregation for Federated Learning, Oct. 2021. URL http://arxiv.org/abs/2007.14861. arXiv:2007.14861 [cs, stat].

[6] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1253–1269, New York, NY, USA, Nov. 2020. Association for Computing Machinery. ISBN 978-1-4503-7089-9. doi: 10.1145/3372297.3417885. URL https://doi.org/10.1145/3372297.3417885.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1175–1191, New York, NY, USA, Oct. 2017. Association for Computing Machinery. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3133982. URL https://doi.org/10.1145/3133956.3133982.

[8] N. Bouacida, J. Hou, H. Zang, and X. Liu. Adaptive Federated Dropout: Improving Communication Efficiency and Generalization for Federated Learning, Nov. 2020. URL http://arxiv.org/abs/2011.04050. arXiv:2011.04050 [cs].

[9] M. Chen, B. Mao, and T. Ma. Efficient and Robust Asynchronous Federated Learning with Stragglers. Dec. 2019. URL https://openreview.net/forum?id=B1lL9grYDS.

[10] T. Chen, G. Giannakis, T. Sun, and W. Yin. LAG: Lazily Aggregated Gradient for Communication-Efficient Distributed Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/feecee9f1643651799ede2740927317a-Abstract.html.

[11] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, Dec. 2020. doi: 10.1109/BigData50022.2020.9378161.

[12] B. Choi, J.-y. Sohn, D.-J. Han, and J. Moon. Communication-Computation Efficient Secure Aggregation for Federated Learning, July 2021. URL http://arxiv.org/abs/2012.05433. arXiv:2012.05433 [cs, math].

[13] T. Coughlin. IoT trends to keep an eye on in 2023 and beyond | TechTarget. URL https://www.techtarget.com/iotagenda/opinion/IoT-trends-to-keep-an-eye-on.

[14] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258, Apr. 2020. ISSN 2095-2236. doi: 10.1007/s11704-019-8208-z. URL https://doi.org/10.1007/s11704-019-8208-z.

[15] I. El Naqa and M. J. Murphy. What Is Machine Learning? In I. El Naqa, R. Li, and M. J. Murphy, editors, *Machine Learning in Radiation Oncology: Theory and Applications*, pages 3–11. Springer International Publishing, Cham, 2015. ISBN 978-3-319-18305-3. doi: 10.1007/978-3-319-18305-3_1. URL https://doi.org/10.1007/978-3-319-18305-3_1.

[16] M. Feldman. *Computational Fairness: Preventing Machine-Learned Discrimination*. Thesis, 2015. URL https://scholarship.tricolib.brynmawr.edu/handle/10066/17628. Accepted: 2016-01-19T17:37:36Z.

[17] Y. Fraboni, R. Vidal, and M. Lorenzi. Free-rider Attacks on Model Aggregation in Federated Learning. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 1846–1854. PMLR, Mar. 2021. URL https://proceedings.mlr.press/v130/fraboni21a.html. ISSN: 2640-3498.

[18] R. Hardin and G. Cullity. The Free Rider Problem. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2020 edition, 2020. URL https://plato.stanford.edu/archives/win2020/entries/free-rider/.

[19] P. Harrington. *Machine Learning in Action*. Simon and Schuster, Apr. 2012. ISBN 978-1-63835-245-7. Google-Books-ID: XTozEAAAQBAJ.

[20] S. Horváth, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane. FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. In *Advances in Neural Information Processing Systems*, volume 34, pages 12876–12889. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/6aed000af86a084f9cb0264161e29dd3-Abstract.html.

[21] M. Hosseinzadeh, N. Hudson, S. Heshmati, and H. Khamfroush. Communication-Loss Trade-Off in Federated Learning: A Distributed Client Selection Algorithm. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6, Jan. 2022. doi: 10.1109/CCNC49033.2022.9700601. ISSN: 2331-9860.

[22] T.-M. H. Hsu, H. Qi, and M. Brown. Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification, Sept. 2019. URL http://arxiv.org/abs/1909.06335. arXiv:1909.06335 [cs, stat].

[23] C. Hu, J. Jiang, and Z. Wang. Decentralized Federated Learning: A Segmented Gossip Approach, Aug. 2019. URL http://arxiv.org/abs/1908.07782. arXiv:1908.07782 [cs, stat].

[24] T. Huang, W. Lin, W. Wu, L. He, K. Li, and A. Y. Zomaya. An Efficiency-Boosting Client Selection Scheme for Federated Learning With Fairness Guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1552–1564, July 2021. ISSN 1558-2183. doi: 10.1109/TPDS.2020.3040887. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[25] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini. A Survey on Federated Learning for Resource-Constrained IoT Devices. *IEEE Internet of Things Journal*, 9(1):1–24, Jan. 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2021.3095077. Conference Name: IEEE Internet of Things Journal.

[26] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire. SwiftAgg+: Achieving Asymptotically Optimal Communication Loads in Secure Aggregation for Federated Learning, Sept. 2022. URL http://arxiv.org/abs/2203.13060. arXiv:2203.13060 [cs, math].

[27] Y. Jee Cho, S. Gupta, G. Joshi, and O. Yağan. Bandit-based Communication-Efficient Client Selection Strategies for Federated Learning. In *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pages 1066–1069, Nov. 2020. doi: 10.1109/IEEECONF51394.2020.9443523. ISSN: 2576-2303.

[28] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran. FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning, Sept. 2020. URL http://arxiv.org/abs/2009.11248. arXiv:2009.11248 [cs, math, stat].

[29] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996. ISSN 1076-9757. doi: 10.1613/jair.301. URL https://www.jair.org/index.php/jair/article/view/10166.

[30] C. Kingsford and S. L. Salzberg. What are decision trees? *Nature Biotechnology*, 26(9):1011–1013, Sept. 2008. ISSN 1546-1696. doi: 10.1038/nbt0908-1011. URL https://www.nature.com/articles/nbt0908-1011. Number: 9 Publisher: Nature Publishing Group.

[31] H. Ko, J. Lee, S. Seo, S. Pack, and V. C. M. Leung. Joint Client Selection and Bandwidth Allocation Algorithm for Federated Learning. *IEEE Transactions on Mobile Computing*, pages 1–1, 2021. ISSN 1558-0660. doi: 10.1109/TMC.2021.3136611. Conference Name: IEEE Transactions on Mobile Computing.

[32] Q. Li, C. Thapa, L. Ong, Y. Zheng, H. Ma, S. A. Camtepe, A. Fu, and Y. Gao. Vertical Federated Learning: Taxonomies, Threats, and Prospects, Feb. 2023. URL http://arxiv.org/abs/2302.01550. arXiv:2302.01550 [cs].

[33] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated Optimization in Heterogeneous Networks, Apr. 2020. URL http://arxiv.org/abs/1812.06127. arXiv:1812.06127 [cs, stat].

[34] T. Li, M. Sanjabi, A. Beirami, and V. Smith. Fair Resource Allocation in Federated Learning, Feb. 2020. URL http://arxiv.org/abs/1905.10497. arXiv:1905.10497 [cs, stat].

[35] Z. Li, Y. He, H. Yu, J. Kang, X. Li, Z. Xu, and D. Niyato. Data Heterogeneity-Robust Federated Learning via Group Client Selection in Industrial IoT. *IEEE Internet of Things Journal*, 9(18):17844–17857, Sept. 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2022.3161943. Conference Name: IEEE Internet of Things Journal.

[36] J. Lin, M. Du, and J. Liu. Free-riders in Federated Learning: Attacks and Defenses, Nov. 2019. URL http://arxiv.org/abs/1911.12560. arXiv:1911.12560 [cs, stat].

[37] W. Lin, Y. Xu, B. Liu, D. Li, T. Huang, and F. Shi. Contribution-based Federated Learning client selection. *International Journal of Intelligent Systems*, 37(10):7235–7260, 2022. ISSN 1098-111X. doi: 10.1002/int.22879. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/int.22879. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.22879.

[38] Z. Liu, J. Guo, K.-Y. Lam, and J. Zhao. Efficient Dropout-resilient Aggregation for Privacy-preserving Machine Learning. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2022. ISSN 1556-6021. doi: 10.1109/TIFS.2022.3163592. Conference Name: IEEE Transactions on Information Forensics and Security.

[39] J. Lopez, R. Rios, F. Bao, and G. Wang. Evolving privacy: From sensors to the Internet of Things. *Future Generation Computer Systems*, 75:46–57, Oct. 2017. ISSN 0167-739X. doi: 10.1016/j.future.2017.04.045. URL https://www.sciencedirect.com/science/article/pii/S0167739X16306719.

[40] J. Ma, X. Sun, W. Xia, X. Wang, X. Chen, and H. Zhu. Client Selection Based on Label Quantity Information for Federated Learning. In *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, Sept. 2021. doi: 10.1109/PIMRC50174.2021.9569487. ISSN: 2166-9589.

[41] X. Ma, J. Zhu, Z. Lin, S. Chen, and Y. Qin. A state-of-the-art survey on solving non-IID data in Federated Learning. *Future Generation Computer Systems*, 135:244–258, Oct. 2022. ISSN 0167-739X. doi: 10.1016/j.future.2022.05.003. URL https://www.sciencedirect.com/science/article/pii/S0167739X22001686.

[42] T. S. Madhulatha. An Overview on Clustering Methods, May 2012. URL http://arxiv.org/abs/1205.1117. arXiv:1205.1117 [cs].

[43] V. S. Mai, R. J. La, and T. Zhang. Federated Learning with Server Learning: Enhancing Performance for Non-IID Data, Apr. 2023. URL http://arxiv.org/abs/2210.02614. arXiv:2210.02614 [cs, math].

[44] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, Apr. 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html. ISSN: 2640-3498.

[45] J. Mori, I. Teranishi, and R. Furukawa. Continual Horizontal Federated Learning for Heterogeneous Data. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2022. doi: 10.1109/IJCNN55064.2022.9892815. URL http://arxiv.org/abs/2203.02108. arXiv:2203.02108 [cs].

[46] T. Nishio and R. Yonetani. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, May 2019. doi: 10.1109/ICC.2019.8761315. ISSN: 1938-1883.

[47] W. S. Noble. What is a support vector machine? *Nature Biotechnology*, 24(12):1565–1567, Dec. 2006. ISSN 1546-1696. doi: 10.1038/nbt1206-1565. URL https://www.nature.com/articles/nbt1206-1565. Number: 12 Publisher: Nature Publishing Group.

[48] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A Design Science Research Methodology for Information Systems Research, 2008. URL https://www.tandfonline.com/doi/epdf/10.2753/MIS0742-1222240302?needAccess=true&role=button.

[49] P. Pinyoanuntapong, W. H. Huff, M. Lee, C. Chen, and P. Wang. Toward Scalable and Robust AIoT via Decentralized Federated Learning. *IEEE Internet of Things Magazine*, 5 (1):30–35, Mar. 2022. ISSN 2576-3180, 2576-3199. doi: 10.1109/IOTM.006.2100216. URL https://ieeexplore.ieee.org/document/9773089/.

[50] Prayitno, C.-R. Shyu, K. T. Putra, H.-C. Chen, Y.-Y. Tsai, K. S. M. T. Hossain, W. Jiang, and Z.-Y. Shae. A Systematic Review of Federated Learning in the Healthcare Area: From the Perspective of Data Properties and Applications. *Applied Sciences*, 11(23):11191, Jan. 2021. ISSN 2076-3417. doi: 10.3390/app112311191. URL https://www.mdpi.com/2076-3417/11/23/11191. Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[51] Z. Qu, R. Duan, L. Chen, J. Xu, Z. Lu, and Y. Liu. Context-Aware Online Client Selection for Hierarchical Federated Learning. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4353–4367, Dec. 2022. ISSN 1558-2183. doi: 10.1109/TPDS.2022.3186960. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[52] S. Rai, A. Kumari, and D. K. Prasad. Client Selection in Federated Learning under Imperfections in Environment. *AI*, 3(1):124–145, Mar. 2022. ISSN 2673-2688. doi: 10.3390/ai3010008. URL https://www.mdpi.com/2673-2688/3/1/8. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[53] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan. Adaptive Federated Optimization, Sept. 2021. URL http://arxiv.org/abs/2003.00295. arXiv:2003.00295 [cs, math, stat] version: 5.

[54] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. ISSN 1939-1471. doi: 10.1037/h0042519. Place: US Publisher: American Psychological Association.

[55] Y. E. Sagduyu. Free-Rider Games for Federated Learning with Selfish Clients in NextG Wireless Networks. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 365–370, Oct. 2022. doi: 10.1109/CNS56114.2022.9947274.

[56] S. Saha and T. Ahmad. Federated Transfer Learning: concept and applications, Mar. 2021. URL http://arxiv.org/abs/2010.15561. arXiv:2010.15561 [cs].

[57] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959. ISSN 0018-8646. doi: 10.1147/rd.33.0210. Conference Name: IBM Journal of Research and Development.

[58] S. Schlögl, C. Postulka, R. Bernsteiner, and C. Ploder. Artificial Intelligence Tool Penetration in Business: Adoption, Challenges and Fears. In L. Uden, I.-H. Ting, and J. M. Corchado, editors, *Knowledge Management in Organizations*, Communications in Computer and Information Science, pages 259–270, Cham, 2019. Springer International Publishing. ISBN 978-3-030-21451-7. doi: 10.1007/978-3-030-21451-7_22.

[59] M. Shaheen, M. S. Farooq, T. Umer, and B.-S. Kim. Applications of Federated Learning; Taxonomy, Challenges, and Research Trends. *Electronics*, 11(4):670, Jan. 2022. ISSN 2079-9292. doi: 10.3390/electronics11040670. URL https://www.mdpi.com/2079-9292/11/4/670. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[60] J. Shao, Y. Sun, S. Li, and J. Zhang. DReS-FL: Dropout-Resilient Secure Federated Learning for Non-IID Clients via Secret Data Sharing, Oct. 2022. URL http://arxiv.org/abs/2210.02680. arXiv:2210.02680 [cs].

[61] F. Shi, C. Hu, W. Lin, L. Fan, T. Huang, and W. Wu. VFedCS: Optimizing Client Selection for Volatile Federated Learning. *IEEE Internet of Things Journal*, pages 1–1, 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2022.3195073. Conference Name: IEEE Internet of Things Journal.

[62] P. P. Shinde and S. Shah. A Review of Machine Learning and Deep Learning Applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, Aug. 2018. doi: 10.1109/ICCUBEA.2018.8697857.

[63] M. Silverio-Fernández, S. Renukappa, and S. Suresh. What is a smart device? - a conceptualisation within the paradigm of the internet of things. *Visualization in Engineering*, 6(1):3, May 2018. ISSN 2213-7459. doi: 10.1186/s40327-018-0063-8. URL https://doi.org/10.1186/s40327-018-0063-8.

[64] C. Smestad and J. Li. A Systematic Literature Review on Client Selection in Federated Learning. June 2022. doi: https://doi.org/10.1145/3593434.3593438.

[65] J. So, B. Güler, and A. S. Avestimehr. Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, Mar. 2021. ISSN 2641-8770. doi: 10.1109/JSAIT.2021.3054610. Conference Name: IEEE Journal on Selected Areas in Information Theory.

[66] J. So, C. J. Nolet, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Guler, and S. Avestimehr. LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning. *Proceedings of Machine Learning and Systems*, 4:694–720, Apr. 2022. URL https://proceedings.mlsys.org/paper/2022/hash/d2ddea18f00665ce8623e36bd4e3c7c5-Abstract.html.

[67] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. 2014.

[68] X. Tan, W. C. Ng, W. Y. B. Lim, Z. Xiong, D. Niyato, and H. Yu. Reputation-Aware Federated Learning Client Selection based on Stochastic Integer Programming. *IEEE Transactions on Big Data*, pages 1–12, 2022. ISSN 2332-7790. doi: 10.1109/TBDATA.2022.3191332. Conference Name: IEEE Transactions on Big Data.

[69] J. E. van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, Feb. 2020. ISSN 1573-0565. doi: 10.1007/s10994-019-05855-6. URL https://doi.org/10.1007/s10994-019-05855-6.

[70] J. Venable, J. Pries-Heje, and R. Baskerville. FEDS: a Framework for Evaluation in Design Science Research. *European Journal of Information Systems*, 25(1):77–89, Jan. 2016. ISSN 1476-9344. doi: 10.1057/ejis.2014.36. URL https://doi.org/10.1057/ejis.2014.36.

[71] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen. AsyncFedED: Asynchronous Federated Learning with Euclidean Distance based Adaptive Weight Aggregation, June 2022. URL http://arxiv.org/abs/2205.13797. arXiv:2205.13797 [cs].

[72] D. Wen, K.-J. Jeon, and K. Huang. Federated Dropout—A Simple Approach for Enabling Federated Learning on Resource Constrained Devices. *IEEE Wireless Communications Letters*, 11(5):923–927, May 2022. ISSN 2162-2345. doi: 10.1109/LWC.2022.3149783. Conference Name: IEEE Wireless Communications Letters.

[73] Z. Wu, Q. Li, and B. He. Practical Vertical Federated Learning with Unsupervised Representation Learning. *IEEE Transactions on Signal Processing*, 70:1–16, 2022. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2021.3129364. URL http://arxiv.org/abs/2208.10278. arXiv:2208.10278 [cs].

[74] C. Xie, S. Koyejo, and I. Gupta. Asynchronous Federated Optimization, Dec. 2020. URL http://arxiv.org/abs/1903.03934. arXiv:1903.03934 [cs].

[75] J. Xu and H. Wang. Client Selection and Bandwidth Allocation in Wireless Federated Learning Networks: A Long-Term Perspective. *IEEE Transactions on Wireless Communications*, 20 (2):1188–1200, Feb. 2021. ISSN 1558-2248. doi: 10.1109/TWC.2020.3031503. Conference Name: IEEE Transactions on Wireless Communications.

[76] X. Ying. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 1168(2):022022, Feb. 2019. ISSN 1742-6596. doi: 10.1088/1742-6596/1168/2/022022. URL https://dx.doi.org/10.1088/1742-6596/1168/2/022022. Publisher: IOP Publishing.

[77] L. Yu, R. Albelaihi, X. Sun, N. Ansari, and M. Devetsikiotis. Jointly Optimizing Client Selection and Resource Management in Wireless Federated Learning for Internet of Things. *IEEE Internet of Things Journal*, 9(6):4385–4395, Mar. 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2021.3103715. Conference Name: IEEE Internet of Things Journal.

[78] Q. Zeng, Y. Du, K. Huang, and K. K. Leung. Energy-Efficient Radio Resource Allocation for Federated Edge Learning. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, June 2020. doi: 10.1109/ICCWorkshops49005.2020.9145118. ISSN: 2474-9133.

[79] L. Zhang, X. Lei, Y. Shi, H. Huang, and C. Chen. Federated Learning for IoT Devices with Domain Generalization. *IEEE Internet of Things Journal*, pages 1–1, 2023. ISSN 2327-4662. doi: 10.1109/JIOT.2023.3234977. Conference Name: IEEE Internet of Things Journal.

[80] Q. Zhang, B. Gu, C. Deng, and H. Huang. Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating, Mar. 2021. URL http://arxiv.org/abs/2103.00958. arXiv:2103.00958 [cs].

[81] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang. Client Selection for Federated Learning With Non-IID Data in Mobile Edge Computing. *IEEE Access*, 9:24462–24474, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3056919. Conference Name: IEEE Access.

[82] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated Learning with Non-IID Data. 2018. doi: 10.48550/arXiv.1806.00582. URL http://arxiv.org/abs/1806.00582. arXiv:1806.00582 [cs, stat].

[83] Z. Zhou, H. Chen, K. Li, F. Hu, B. Yan, J. Cheng, X. Wei, B. Liu, X. Li, F. Chen, and Y. Sui. A Novel Optimized Asynchronous Federated Learning Framework, Nov. 2021. URL http://arxiv.org/abs/2111.09487. arXiv:2111.09487 [cs].

[84] H. Zhu, J. Kuang, M. Yang, and H. Qian. Client Selection with Staleness Compensation in Asynchronous Federated Learning. *IEEE Transactions on Vehicular Technology*, pages 1–6, 2022. ISSN 1939-9359. doi: 10.1109/TVT.2022.3220809. Conference Name: IEEE Transactions on Vehicular Technology.

Appendix

A Circumstances for selecting evaluation strategy

DSR evaluation strategies	**Circumstance selection criteria**
Quick & Simple	If small and simple construction of design, with low social and technical risk and uncertainty
Human Risk & Effectiveness	If the major design risk is social or user oriented and/or If it is relatively cheap to evaluate with real users in their real context and/or If a critical goal of the evaluation is to rigorously establish that the utility/benefit will continue in real situations and over the long run
Technical Risk & Efficacy	If the major design risk is technically oriented and/or If it is prohibitively expensive to evaluate with real users and real systems in the real setting and/or If a critical goal of the evaluation is to rigorously establish that the utility/benefit is due to the artifact, not something else
Purely Technical Artefact	If artifact is purely technical (no social aspects) or artifact use will be well in future and not today

B Applications of Federated Learning

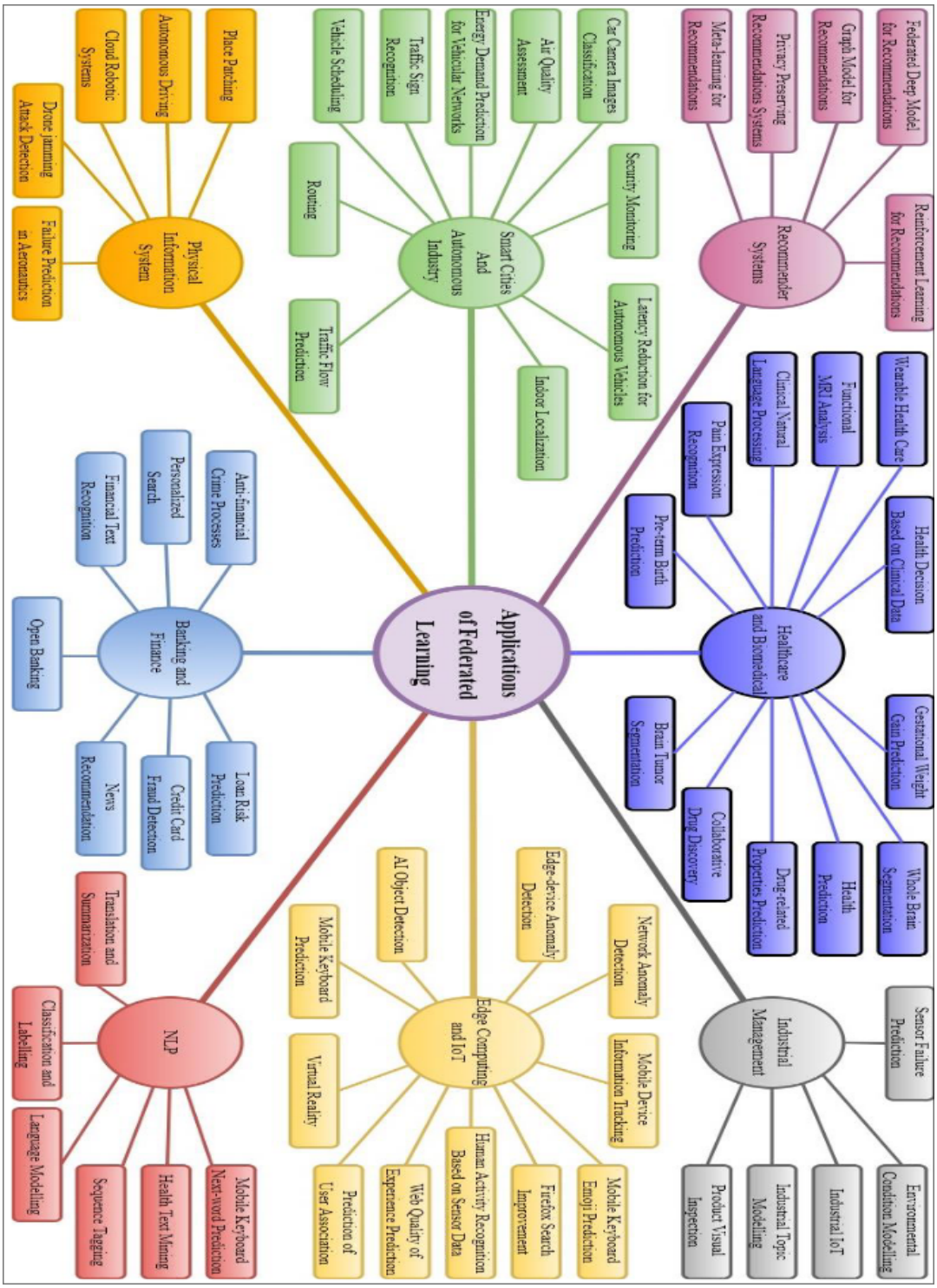
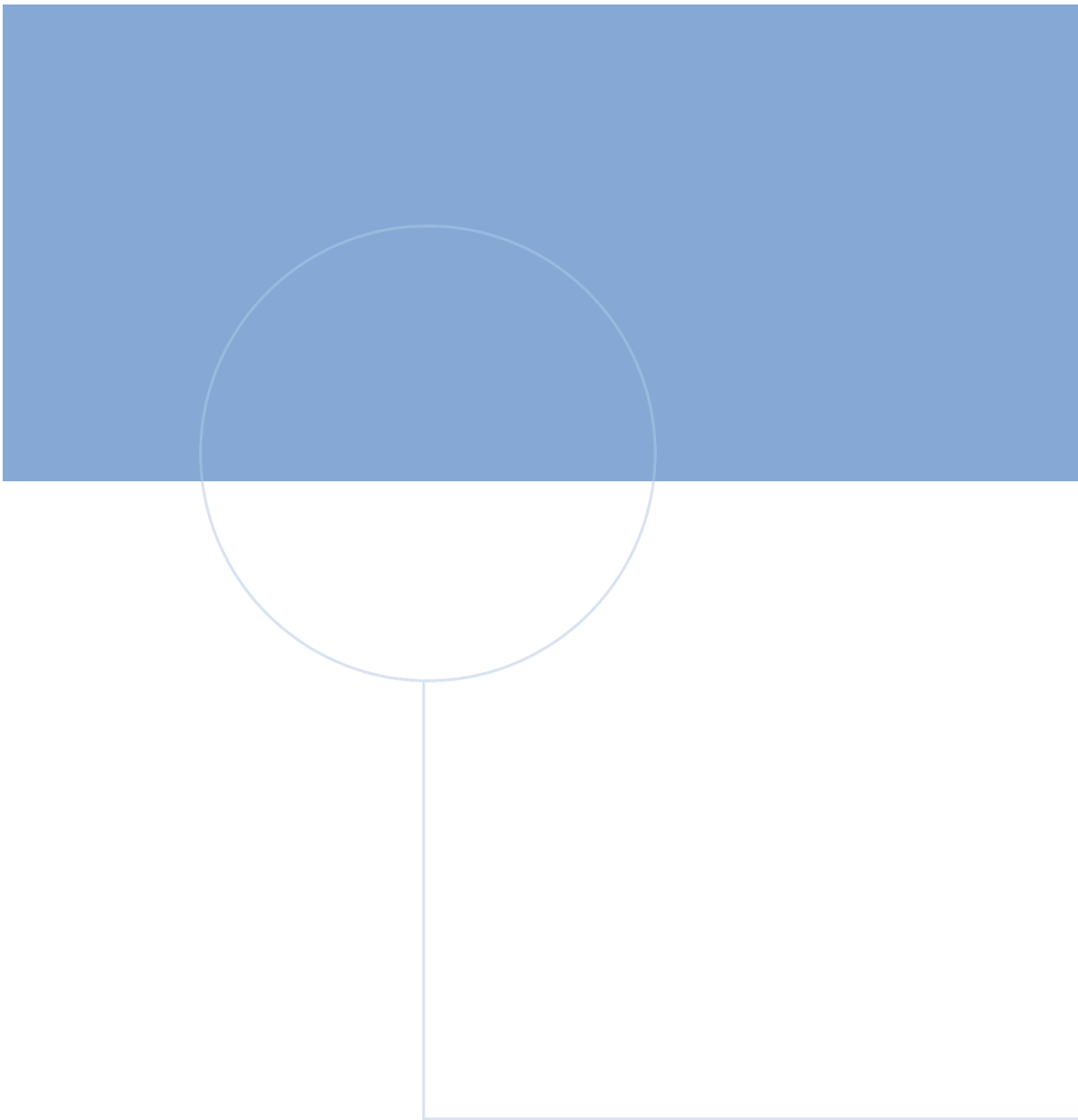


Figure 17: Taxonomy for applications of federated learning across different domains and sub-domain [59]



NTNU

Norwegian University of
Science and Technology