Renaissance Capstone Project AY2020/21

(Smart Platform Infrastructure

Research on Integrative Technology (SPIRITS))

Privacy-Preserving Deep Learning

Submitted by:

1. Yik Jia Ler (U1721233K)

2. Tan Zhen Yuan (U1721017F)

3. Zaw Maw Htun (U1722506E)

Supervisor: Prof Lam Kwok Yan

Co-supervisor: Dr Sourav Sen Gupta

A Renaissance Capstone Project report presented to the Nanyang Technological University

in partial fulfilment of the requirements of the degree of

Bachelor of Engineering

2020/21

Acknowledgements

Dr. Sourav is extremely helpful and proactive in sharing his wealth of knowledge on the subject matter. He guides us along every step of our implementation, while providing constructive feedback regarding the dataset, the optimisers used, and how the concepts expounded in this paper are translated to code, just to name a few. Not only that, he shares with us invaluable insights to the findings we produce, which sharpens our concept and allows us to think more critically throughout the course of our implementation.

Prof Lam Kwok Yan is the main supervisor to our project. Together with Assoc Prof Tan Lay Poh and Prof Lalit Goel, we would like to thank all three professors for giving us invaluable feedback during our interim presentation. Their feedback has allowed us to shift the needle in our paradigm and evaluate our content more critically from differing perspectives. In particular, we were able to better appreciate our understanding of our subject matter by catering to peoples with different understanding of our material. Especially in the case of the fledgling technologies which we are working with, literature is too often catered to people already knowledgeable of the subject matter. In working towards integrating such technologies with greater acceptance and less fear, it is imperative for us to be able to explain and break down complex concepts into edible chunks. We have authored our paper with this in mind; you, as the reader, will be able to find the significance of different concepts and the links between these concepts. We hope this will help you in your appreciation of the subject matter.

We are also grateful for the Renaissance Engineering Programme (REP) as well as the Nanyang Technological University, Singapore, for making this project possible in the first place. In particular, REP has allowed us the rare (and very fine) opportunity for us to work together and apply a multidisciplinary approach in our endeavours. The REP Office staff has been delightfully helpful in their coordination between us undergraduate students and the school administrative bodies.

We would also like to thank everyone who has helped us along the way, in ways both big and small and in ways both related to the subject matter and not. We have omitted a whole list of these amazing people due to the lack of space, but always know that we are more grateful than words can ever express.

Chapter 1

Introduction

This chapter seeks to provide context for the underlying principles required for this study, as well as the motivations and objectives behind this study.

The capabilities of machine learning (ML) models largely depend on the information it is fed with to build and improve the model, and one of the best ways to attain said required information is amongst the public. With the plethora of data and information available on the Internet of Things (IoT), and with the burgeoning number of mobile devices being used worldwide [1], usage of artificial intelligence becomes even more prevalent. ML, and the subset of it: deep learning (DL), has performed numerous miraculous feats in various fields such as image classification [2, 3], speech recognition [4], language detection [5], facial recognition [6] and artificial intelligence in games such as the board game Go [7, 8, 9].

In the following sections, we will briefly expound on topics which will build the foundation on what our experiment entails: machine learning and subsequently deep learning, federated learning (FL), and differential privacy (DP) with a brief introduction on the other types of methods commonly used for enhancing privacy in the process of training models.

1.1 Machine Learning

Machine Learning [Jung, A. 2020, 13] can be seen as a combination of three components: data, model, and loss function. The model, or hypothesis space, is what allows the data to be translated into what is usually a hypothesized output. A key challenge in ML is to find an accurate predictor h (in the hypothesis space) to predict the output accurately from the input, and one of the ways to circumvent this is to apply a weight factor to all predictors h and search for a sufficiently accurate model across all these weight factors.

And the method of determining when it would be sufficient is to have a loss function, defined as $L: X \times Y \times H \rightarrow R$ where $L((x, y), h)$ is the loss incurred by using one (set of) predictors within the hypothesis space to predict the output y. In most ML scenarios which take into account numeric data or output, the loss function that is commonly used would be the squared error loss, as shown by Equation 1.1.

$$L\,((x, y), h) :=\, (y\, -\, h\,(x))^2 \tag{1.1}$$

A subset of Artificial Intelligence (AI), Machine Learning's key purpose [12] is to automate processes or tasks by learning specific patterns from provided data. This moves humans away from the reliance of traditional computing paradigm which requires data and a program (manually programmed by humans) to provide an output, to one that uses data and currently available output data to produce a program to produce future outputs, as shown in Figure 1.1.

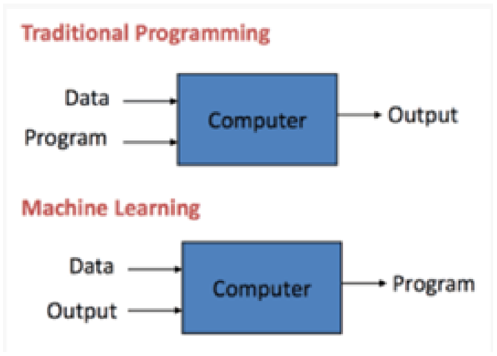


Fig 1.1: Traditional programming paradigm versus machine learning paradigm.

Given the convenience of this, ML has been applied onto multiple fields [15]:

- Finance: Banks can determine customers' portfolio offerings based on certain data they have over the customers. This may help with determining a customer's likelihood to default on credit and hence determine their credit ratings for purposes of loaning money. [14]

- Healthcare: Hospitals can predict hospital admission at the emergency department (ED) triage. In a study done by Woo S.H. et al., [16] to utilize patient history combined with information provided at the ED triage to predict hospital admission, they found that ML can robustly predict it and the addition of patient history improved the results tremendously, indicating the importance of additional relevant data as ways to improve ML accuracy.

- Web search: Google's PageRank system which is one of the key features on its highly recognized search system utilizes ML heavily in determining the most relevant links to show based on search entries by users, by utilizing data from past users such as the links that were most clicked on following a search, or from a specific website.

The numerous types of machine learning used commonly are as listed [12]:

- Reliance on human supervision: Supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning

- Learning from incremental data samples: Batch learning, online learning
- Generalization method of data samples: Instance based learning, model based learning

For the purpose of this paper, we will not go further in details for these types mentioned as it is no longer as relevant to what we aim to study in federated learning incorporated with differential privacy.

1.2 Deep learning

Diving slightly deeper into ML, deep learning is a subset of machine learning. The terms ML and DL have become so commonplace that they might seem to be used interchangeably, but they hold some differences. [18] Robins, M. puts it as such: deep learning is just a special type of machine learning. [19] Figure 1.2 shows this in a pictural format for clearer understanding.

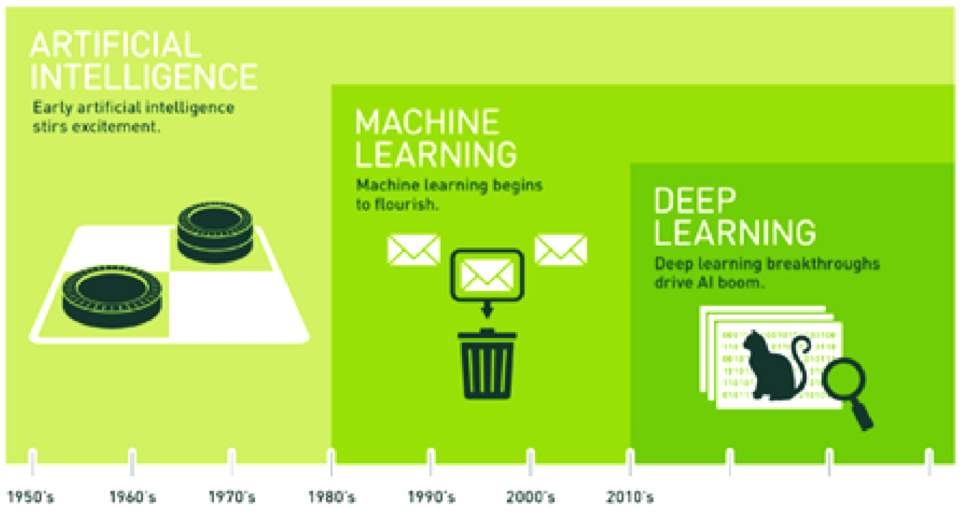

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Fig 1.2: DL as a subset of ML which in turn is a subset of AI. Taken from: [25]

An example to illustrate the difference between ML and DL in facial recognition shown in Figure 1.3. Simply put, the classic ML model would require humans to input the features that are important in face detection, such as the distance between eyes, circumference of face, nose width, etc., before being able to process them and come up with a working model. However, even we as humans may miss out certain important features that may be required in the model training. On the other hand, the DL model would be able to take in the data and process it without any form of human intervention and determine (on its own) the important and relevant aspects to look out for in facial recognition. Something that becomes obvious as well is that DL will then require a much more extensive database than ML, given that the patterns are left to the model to configure and analyze.

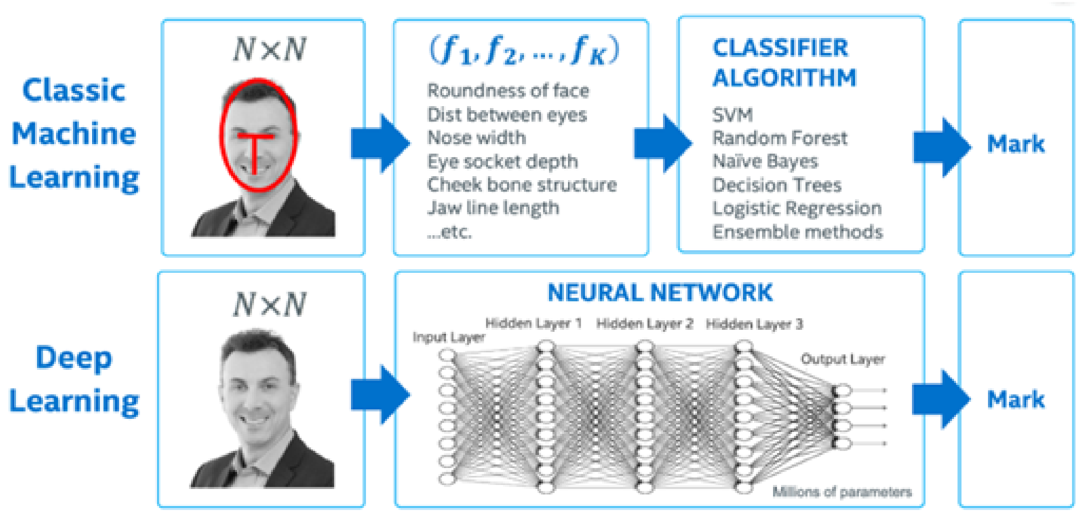


Fig 1.3: Difference in training between classic ML and DL. Taken from: [19]

Commonly used DL methods for image classification models which we will work on in our experiment would be the convolutional neural network (CNN), [54] which divides the layers of artificial neurons into multiple levels, with each layer fulfilling different purposes at each stage. One key benefit of using the CNN is the added simplicity it provides to the model during the training phase as it heavily reduces the number of parameters being computed across the layers.

Robins talks about how some types of machine learning relies on human input, to define the features to be deemed as relevant within a dataset in order to predict an output. An example he provided was in the case of determining the price of a home up for sale, some factors to consider would be the number of bedrooms, size of home, and accessibility of education in the district. This is also known as “feature engineering”. The factor that differentiates DL from the other forms of ML is the ability to determine these relevant features on its own, through multiple trial and errors within its training phase, using parameterized functions such as affine transformation or simple nonlinear functions. [21] It utilizes its own artificial neural network, or deep neural network in DL’s case, where artificial neurons are arranged in layers and each one of them are functions that take in inputs from the previous layers and produces an output for the subsequent layer, or a predicted output if it is the final layer. This can be seen in Figure 1.3 where there are multiple layers of neurons and each node in a layer is linked to the nodes in a previous layer and the subsequent layers. The links would be the weights that each of these nodes carry to the subsequent layer’s nodes.

All these nodes are given a certain value based on the weight of the previous nodes multiplied by the values they provide, after being subjected to either a sigmoid function or the Rectified Linear

Units function (ReLU). [20] Mathematically, the ReLU function is as such: *y = max(0, x)*. Visually, the sigmoid and ReLU functions are shown below in Figure 1.4(a) and 1.4(b) respectively.

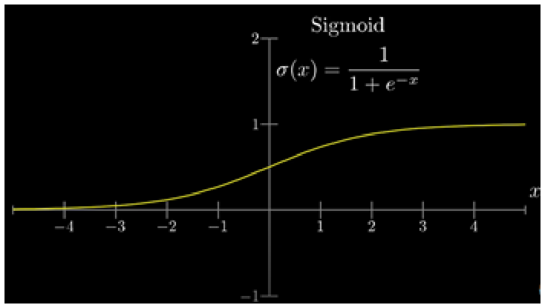


Fig 1.4(a): Sigmoid function graphically. Taken from: [23]

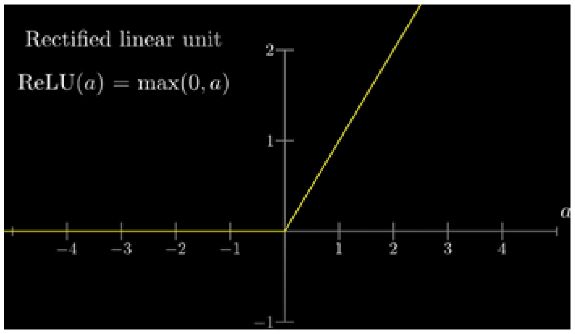


Fig 1.4(b): ReLU function graphically. Taken from: [23]

These are also known as activation functions since the value must exceed a certain threshold before being assigned a significant value (or a non-zero value for positive values in ReLU's case). This is similar to a step function. There are also other forms of activator functions such as the *tanh* function. The ReLU function is much more commonly used in DL and especially with the convolutional neural network (CNN) method due to the efficiency it has over the sigmoid function [23] in training a model, given its simpler mathematical operations. [22] And because it is more sparsely activated since it is likely for a given node to not activate at all considering that it is zero for all negative inputs. [20] ReLUs are also faster than functions like tanh as they have non-saturating linearity f(x) = max(0,x). [53] For DL, in determining the weights of each neuron, multiple methods can be used as optimizers, such as the stochastic gradient descent (SGD), limited-memory BFGS (L-BFGS), and Conjugate Gradient (CG) with line search. [24]

1.3 Federated learning

One key problem associated with ML is the privacy concerns that are derived from the massive storage of data ML models use to train themselves in a centralized location [26]. Given the proliferation of skilled hackers on such models seeking to attain sensitive information on individuals or organizations, these parties will no longer be as willing to provide their sensitive information up for advancements in technology.

For example, given that more ATMs are using Facial recognition software [27], if hackers are able to obtain such data of customers from a hack onto a centralized system containing all the data, they would be able to gain access to multiple other platforms/devices/systems which utilizes facial recognition [28]. However, given that most of the important data nowadays are kept in user's own mobile devices, there may no longer be a need to store the information in a centralized system. ML scientists and researchers have come up with a way to train their models separately on the user's own devices instead of siphoning said data from users' phones. All the models from individual devices then get aggregated together to produce a new and improved model to be sent out to the devices once again for training. This process is called Federated Learning (FL). [29]

FL is a distributed ML approach which enables training on a large dataset collection in a decentralized manner. While central ML "brings the data to the code" during the training of the model, FL is unique in that it "brings the code to the data". This allows it to circumvent the concerns over privacy, ownership, and locality of data. A few benefits of FL are keeping sensitive data "local" to the users' devices, and by sending the models to the users for training, it reduces the need to log data into the data center. Furthermore, using constantly updated information from the real world from users on mobile devices proves much more useful and relevant to training a ML model [26].

The way it works is that there will be a global model stored on the Central Server and a task script which will include the hyper-parameters to begin with. These are then downloaded onto the client devices locally. [30] Figure 1.5 shows a visual representation of what occurs over the course of multiple rounds of training the global model through FL. Relevant data to the training of a model is stored locally and used to train the global model that has been downloaded into the device seamlessly. An optimizer, such as the stochastic gradient descent or gaussian gradient descent, is used to adjust the parameters on the global model after each round of data collection (locally). This is followed up by uploading the updated local model parameters or gradients back to the Central Server. After aggregating the input from the multiple users' local devices, the global model is updated, and a new version is distributed to the devices once again to repeat this process and this process repeats until there is a little to no change in the updating of the global model.

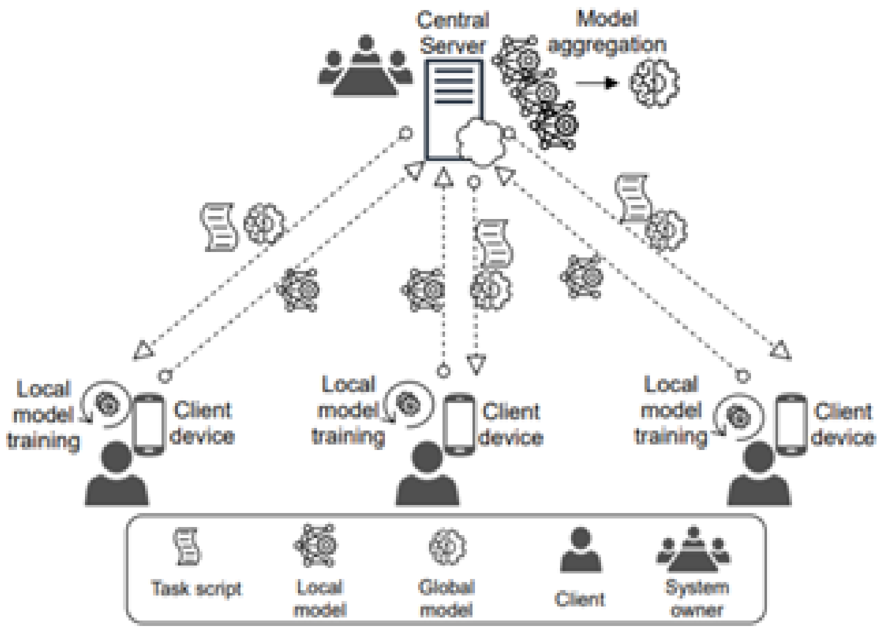


Fig 1.5: Federated learning with Central Server and multiple clients. Taken from: [30]

A common example showing FL working would be the GBoard on Android [31]. The global model is updated using data about how the users interact with the suggestive texts shown on the keyboard when a search query is made, as well as other information that the model may find patterns in (as described in earlier sections). After multiple rounds of updating a global model across all the devices, more accurately predictive suggestions will pop up for users on their GBoard's query suggestion model, to the point where it can even suggest responses to messages being sent in from other parties to the user.

However, it is worth taking note that while federated learning helps to mitigate the risks of an attack on a datacenter, it neglects other forms of attacks, one of which is de-anonymization. While what is constantly sent through the uplinks and downlinks are the models, and not sensitive data, it may be possible for attackers to obtain sensitive information about individuals by de- & re-identification processes. [32, 33]

In a study of model inversion attacks of linear classifiers in personalized medicine done by Fredrikson et al. [34], it was shown that adversarial attacks on a ML model can be abused to obtain sensitive personal genomic data about the parties who were involved in the training of the ML model. [35] Another study by Shokri et al. [36] also displayed similar findings where a membership inference attack was able to determine whether a given record was part of the training data used in the model. [36] One classic case of linkage attack is the case of data scientists being able to de-anonymize data provided by Netflix during a challenge it organized back in 2006. [39] In the data provided by Netflix which came from over 480,000 users, there were over 100 million ratings for over 17,000 movies. Netflix anonymized the data by removing the users' names from the ratings, but the data scientists were able to cancel this out by matching the data it had to data from IMDb.

Hence, it becomes apparent there needs to be more done to preserve the privacy of sensitive data and mitigate risks associated with attackers on ML models.

1.4 Privacy-preserving methods: Cryptographic tools

As aforementioned, attacks on DL or FL models are not rare occurrences. Methods to deal with these forms of attacks have been discussed previously [33, 37, 38], but most of them speak of cryptographic tools such as differential privacy (DP), Homomorphic Encryption (HE), and (Secure) Multi-party computation (MPC). We will discuss briefly on each of these tools, but the tool that our work focused on was DP and the ways it can be incorporated into FL while training a model to classify images.

1.4.1 Differential Privacy

Proposed by Dwork et al., [40] the concept of differential privacy (DP) aims to allow models to be trained and updated solely based on statistical information of the population without extensive reliance on individual data. [41] DP works on the concept of "statistical", where a few individuals' data should not affect the overall outcome of the results, the algorithm of the model in this case. This is largely drawn from the fact that if an individual's data is able to drastically affect the results, it means that attackers will be able to easily obtain data on said individual using the changes in the results.

DP bases itself on a randomized function M, where the function will provide some degree of privacy budget, ε, based on the following equation:

$$\Pr[\mathrm{M}(D) \in S] \leq \exp(\varepsilon) \Pr[\mathrm{M}(D') \in S] \tag{1.2}$$

The privacy budget is what determines the level of privacy provided to a training model, and to achieve this function M, there is noise added to the function that correlates the query response f to an existing piece of data D, as shown by the following equation:

$$\mathrm{M}(D) = f(D) + \mathrm{n} \tag{1.3}$$

where n is the noise added. “n” is then usually determined based on a Gaussian distribution or Laplacian distribution. The smaller ε is, the better privacy is preserved up to the limit of $\varepsilon < 1/n$. This method allows the model to obtain training datasets that are slightly adjusted such that attackers will not be able to obtain any reliable information due to the addition of noise.

DP is largely accepted as the industry standard tool for preserving privacy and is practiced by large companies such as Google [42], Apple [43], Microsoft [44], and Facebook [45]. However, a key issue that is consistently brought up with this tool is the fact that accuracy of a model is affected by the extent to which privacy is given i.e., the more privacy is given, the less accurate the results will be.

1.4.2 Homomorphic Encryption (HE)

The basis of HE lies in the simple concept of only allowing encrypted data to be worked on. This means that only the model which holds the key to decrypting the data will know what the data is actually supposed to be. It allows simple levels of computation to be done on ciphertexts, and can be written as such:

$$E(a) * E(b) = E(a*b)$$

where E represents the encrypted value of a, b and a+b, converting these values to ciphertext, and the ‘*’ symbol represents a mathematical operation. The partial HE (PHE) is only able to support certain operations [41], such as simple addition or multiplication. An example of additive HE is the Paillier cryptosystem [46] and an example of multiplicative HE is the unpadded RSA cryptosystem [47]. The fully homomorphic encryption (FHE) is able to support all operations. However, Liu X. et al. [48] have shown that HE’s implementation on DL models are still considered highly inefficient as it is only able to support integer-type data or high complexity in terms of computing. Although some companies utilize this to preserve privacy such as WeBank, it is largely unused due to the extensive computational power required compared to DP.

1.4.3 Multi-party Computation

A Multi-party Computation (MPC) relies heavily on a group of participants creating a conjoint function together using their own personal data, such that each individual is unaware of the responses from the other participants [49]. This stems from the need to protect a participant’s data amongst untrusted participants.

An example in the way MPC works would be to consider 3 participants in a dataset with salaries A, B and C respectively. If the purpose of the algorithm were to simply find out the average salary amongst these 3 participants, while ensuring that each participant does not reveal his true salary, what MPC does is to allow the participants' own salary to be broken up into 3 elements (which when added gives the actual salary of that individual), where each participant keeps one of those 3 elements of their own and takes one element each from the other two participants. These 3 new elements are then combined, and the resulting pseudo-salary is used to compute the average.

A key problem with this tool is that it requires extremely high computational overhead, especially if the number of participants gets too big (which is usually the case for DL models, given the sheer amount of data it needs to appropriately train a model).

1.2 Motivations

This thesis addresses the issue of privacy preservation in ML solutions in a rapidly advancing digital society with a huge pool of information. The proliferation of mobile phones alone as the primary computing device for many people has brought about a wide range of data especially with the equipping of a multitude of sensors, such as cameras, microphones, and GPS. [1] However, this blessing does not come without a price. Crowdsourced information may contain sensitive data about individuals or organizations and this brings about concerns of privacy breaches due to attackers or hackers, who seek to utilize the public data for malicious intents. [50]

There are growing concerns from all sectors of society. Regulators are increasingly putting in place data protection laws to protect users on the IoT from such harm; the Personal Protection Data Act (PDPA) is one such set of law found in Singapore. Consumers have shown greater aversion towards mishandling of data privacy as well; 84% of customers will abandon an online purchase upon realising a website is insecure. [51] At the same time, or consequently, companies are increasingly recognising the importance of training their ML models on decentralised data to keep up with these evolving landscape. [52]

As such, we looked into ways of incorporating DP into FL models to preserve the privacy of individuals or organisations and protect sensitive data. However, one of the key issues with incorporating DP, given its noise-adding nature to the actual data before the model is updated, is that there will be some degree of inaccuracies in the model, compared to if the model was trained on data directly fed from the users. As such, our thesis looks into how we can juggle between accuracy versus privacy and draw relations between these two factors.

Chapter 2

Technical background

Throughout this chapter, as is common if not standard with studies in the area of FL, we hold the assumption that stakeholders involved in FL are honest-but-curious.

That is, the different stakeholders execute the protocol exactly as specified in their roles (honest), but they may attempt to obtain more information than they are privy to by analysing the transcripts and running queries on the data (curious). In addition, they will not attempt to bypass the protocol by, for example, sending untrue data.

2.1 Federated Learning Architectures

Compared to centralised ML, FL has the advantage of granting better data privacy (see Section 1.1.3 in Chapter 1) due to its architecture, by training a centralised model on decentralised data. With this being said, as a large-scale distributed system, developers employing FL models suffer from designing the model architectures in two ways:

1. More complicated interactions between the hardware components in the FL system than in traditional ML systems (termed as 'nodes');

2. Having to balance trade-offs between user privacy and software quality attributes, in particular accuracy and communication costs. [57, 58]

While other forms of FL models exist [70], the focus of our report is on traditional centralised FL models which we employed in our experiment. One such example is the decentralised FL model, illustrated in diagram (a) of Fig 2.1.

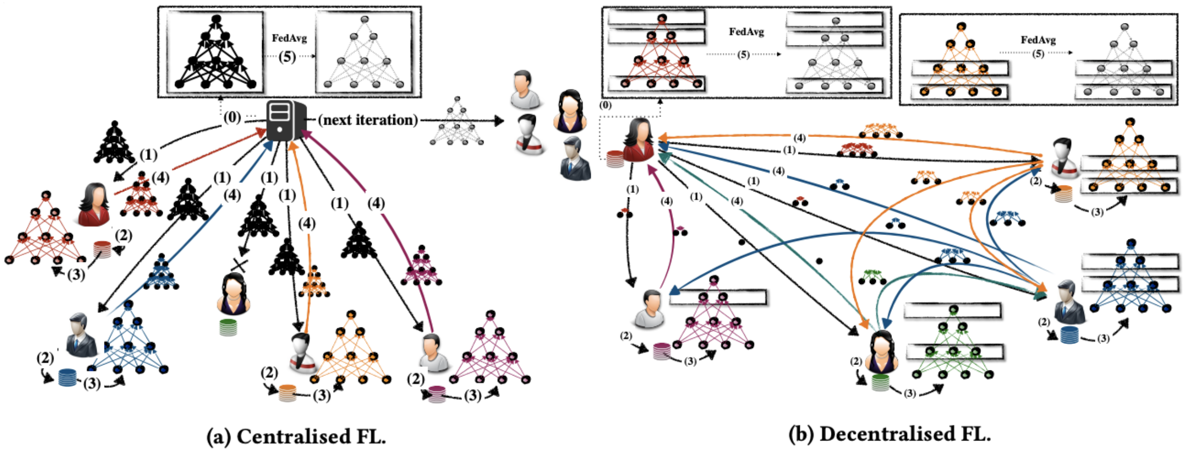


Fig 2.1: A comparison between centralised FL and decentralised FL showing the differences in interaction of the FL stakeholders and overall architecture. In centralised FL (a), the Central Server plays a central role in training the model. In decentralised FL (b), on the other hand, there is no need for a Central Server to coordinate training. Taken from: [70]

We explore in greater detail stakeholders in FL (Section 2.1.1), design challenges (Section 2.1.2), and three different data distributions - horizontal and vertical FL (Section 2.1.3).

2.1.1 Stakeholders in Federated Learning

There are two types of nodes in a FL system - the Central Server and the client devices. The main components in FL server architectures make up the FL population, and are as follows:

1. Learning Coordinator;
2. Master Aggregators and Aggregators; and
3. Selectors.

Coordinators and Selectors are persistent (long-lived) actors, whereas Aggregators are ephemeral (short-lived) actors. [55, 56]

The ways these components interact with each other are illustrated below in Fig 2.2. While our research focuses on the Aggregators and Coordinators, the relationship of the two classes with the client device is included in the illustration for the reader to better appreciate the relevance of FL in the everyday life of a layman.

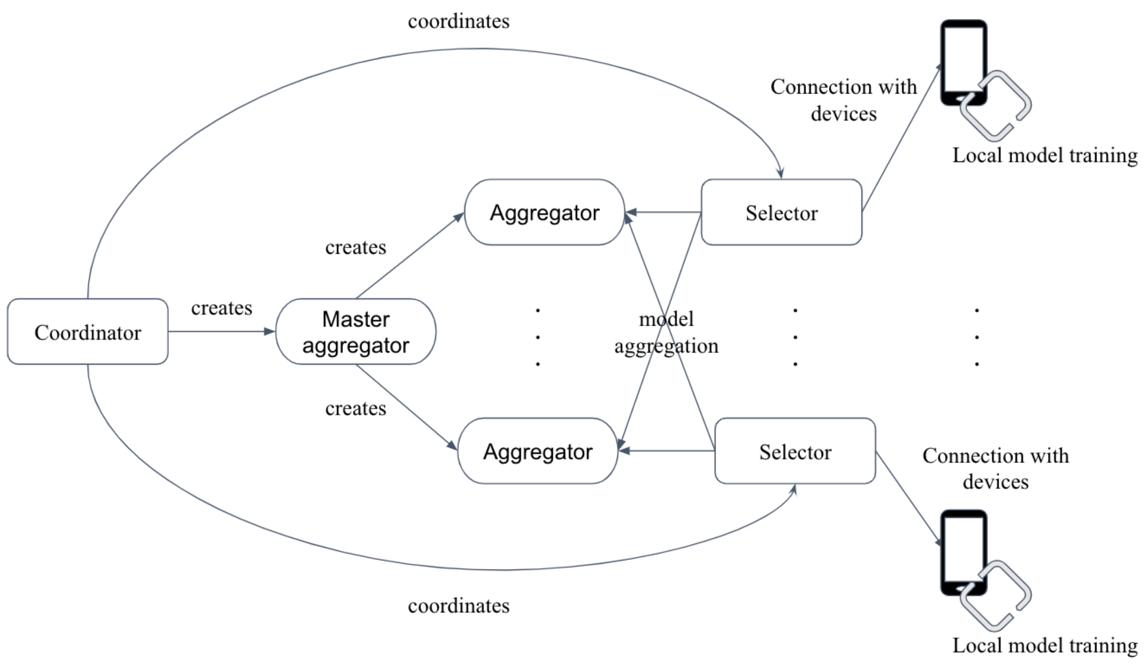


Fig 2.2: Relationship of components in FL systems.

The term ‘Coordinator’ is sometimes used interchangeably with ‘Central Server’, but we would like to make a distinction for our paper - our use of ‘Central Server’ refers to Coordinator, Master Aggregator, and Aggregators rather than the Coordinator alone. Conversely, the client side refers to Selectors and client devices.

The Central Server is responsible for model aggregation and evaluation, whereas the client side is responsible for data collection, data preprocessing, feature engineering, model training, and inference. [57]

Finally, it is worth noting that client devices and users are different in that client devices are a subset of users that are selected for participation in model training.

2.1.1.1 Learning Coordinator

The FL process is started by the Learning Coordinators, which are top-level stakeholders that are critical in their role in initiating global synchronisation and orchestrating lock-step execution of rounds in a FL population of devices. [55] Coordinators are typically authoritative third-party stakeholders - relative to the Selectors - which are trusted by all stakeholders in the FL system. [60]

The role of the Coordinator in an FL system can be broken down into five steps [70]:

1. Determining the type of model to be trained on client devices by Selectors using identical loss function and optimisation algorithm, such as Adam optimisation (FedAdam) [72] and stochastic gradient descent (SGD) [71];

2. Registering interested Selectors and coordinating directly with them (see Section 2.1.1.3 for more details);

3. Randomly sampling from a subset of suitable client devices to tackle specific FL problems [61];

4. Passing on updated values of global model parameters to Selectors, to be then passed on to client devices; and

5. Aggregating model result contributions to update values of global model parameters for use in subsequent training round(s).

In centralised FL systems, Coordinators are responsible for initialising the first global model by either using random model parameters and/or gradient [68, 69] or pre-training the global model with a self-generated dataset or small subset of client devices.

In a FL system, the Coordinator registers an address with Aggregators and Selectors in a shared locking service and owns the FL population. [55] Physically, these locking services can be hosted on local machines [66], cloud servers [64], and mobile edge computing platforms [65] among many other options. [63] It preserves reachability with the FL population and receives information about the number of devices connected to Selectors. This information is used to decide on the number of devices each Selector will select for receiving a training model. [61] The Coordinator directly coordinates with Selectors in such a manner and gives differing instructions depending on the tasks scheduled. They also create Master Aggregators to administer the rounds of FL tasks.

The Learning Coordinator in our experiments is in the form of a cloud server, which will be explored in Section 4.3.2 of Chapter 4.

2.1.1.2 Aggregators

Aggregators are server-based components, and is the most mentioned component of its class. [63] There are two classes of Aggregators in a FL system - Master Aggregators and the broader class of Aggregators.

After the execution of rounds of FL tasks, the trained models from user devices are aggregated locally by Aggregators to form a consensus change to the shared model in the Central Server, as illustrated in Fig 2.3. [62]

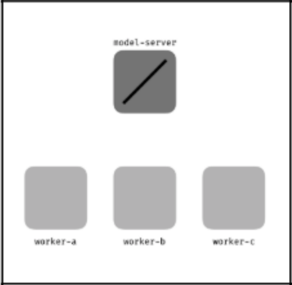
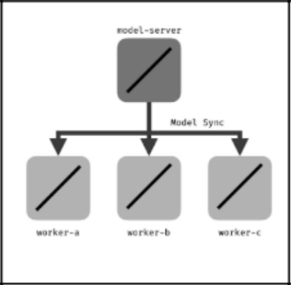
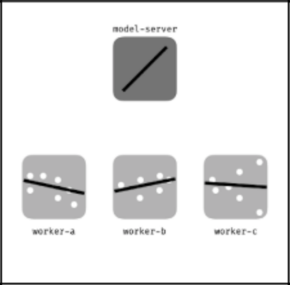
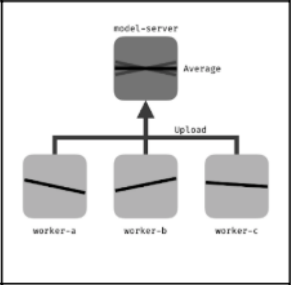
Step 1	Step 2	Step 3	Step 4
			
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Fig 2.3: A simplified explanation of the FL process. Aggregators are responsible for aggregating the trained model results from client devices in Step 4. Taken from: [67]

In addition to being able to perform the same roles as Aggregators, Master Aggregators are unique in their ability to create Aggregators to delegate the administration of rounds to, according to the number of connected devices and update size. [59]

Information from each round is only written to persistent storage after being fully aggregated by the Master Aggregator. All stakeholders otherwise keep their state in short-lived memory. This is known as in-memory aggregation. Compared to distributed storage, this approach not only improves scalability because it removes latency which would have been incurred but also removes logs thereby eliminating targeted attacks within the data centre on persistent logs of per-device updates. [55]

Working along with the Coordinator to function as the Central Server, the Central Server also hosts, among many possibilities, software components responsible for encryption/decryption mechanisms for model encryption and decryption for privacy preservation [73], and resource management mechanisms to optimise resource consumption, such as moment accountants and privacy budgets, known as epsilon (Ɛ). [74]

2.1.1.3 Selectors

Selectors are primarily responsible for selecting device connections linked to them for forwarding to receive a training model. As such, they directly coordinate with the Coordinator and receive information regarding the number of client devices needed for a FL population before making local decisions on whether to accept participation from each client device.

After the creation of the Master Aggregator and Aggregators under it, the Coordinator sends out instructions for Selectors to forward a subset of its connected devices to Aggregators for the Coordinator to designate FL tasks to client devices irrespective of the number of available client devices.

2.1.2 Design Challenges in Federated Learning

As a large-scale distributed system fundamentally, FL presents greater architectural challenges than conventional ML [75] due to the two considerations mentioned in Section 2.1. The main general design challenges are as follows. [56]

Loss in accuracy and generality

Client devices produce non-IID (independent and identically distributed) data that is used in training models. The nature of non-IID data means that global models generated will lose out on accuracy by up to 55% [76] and generality. The quality of FL models is highly dependent on local data distributions from selectors; when local data distributions show significant variance between different selectors, FL pales in comparison to conventional ML. While conventional ML centralises and randomises data, the inherent nature of privacy preservation of FL means such techniques are unsuitable as the Central Server does not have access to raw data from clients.

High communication costs

High communication costs are a central and recurring focus of concern for FL during the process of producing quality global models, since multiple update iterations before convergence can be achieved. [57] This necessitates multiple rounds of communication between the stakeholders to facilitate the exchange of local model updates, resulting in communication costs being widely considered the primary constraint. [77]

This constraint is especially pronounced for cross-device FL, which involves massive numbers of client devices. Some clients may therefore be unable to participate as a result of bandwidth limitations. [57]

Quality of model is dependent on capabilities of client devices

Limited resources on the end of client devices could lead to the inability to execute the multiple rounds of training and communications necessary to ensure the quality of the training models and, consequently, the global model.

Increased difficulty in maintaining model provenance and system reliability

Increasing the number of clients helps to enhance the quality and accuracy of models, but makes it more difficult to govern the process of iterations during learning and to maintain model provenance and system reliability. [57]

2.1.3 Data Distribution in Federated Learning

Data distribution in FL differs from conventional ML, and can be categorised into horizontal and vertical FL. This will be explored in the following subsections, Sections 2.1.3.1 and 2.1.3.2.

Before moving on to explore the two data distribution methods, it is beneficial to first understand the disparities in FL and conventional ML to better appreciate how these distribution methods work differently in both models of learning.

The differences are as follows. [59]

- While FL aims to preserve privacy of its end users, the objective behind ML is to accelerate training speed.
- In FL, we cannot ascertain data distribution of any client devices; this is how privacy of end users is preserved. On the other hand, ML allows for the arbitrary allocation of subsets of learning data from client devices.
- Unlike conventional ML, the number of client devices in FL is often massive and in the magnitude of millions. [59, 62] To complicate things, these client devices do not necessarily have consistent capabilities or stable server connections (see Section 2.1.2). This is usually alleviated by stringent selections of participating client devices, such as having as requirements connection to WiFi and being on charge to prevent interruptions related to device availability. [62]

Although FL evolved from conventional ML and is hence also widely categorised according to its data distribution method, FL deviates from ML in that it does not have access to a whole complete dataset due to the differences mentioned above. Consequently, it is more difficult to describe data distribution in FL, compared to ML.

While horizontal FL is used when datasets share a common feature space but different ID space, vertical FL is used when datasets share a common sample ID space but different feature space. [78]

2.1.3.1 Horizontal Federated Learning

Horizontal FL is the data distribution method used in our experiment.

Horizontal FL is distinct in that “the whole dataset is horizontally partitioned over data samples and allocated to clients” [59], such that the separate data generated on different clients share common features, as illustrated in Fig 2.4.

To further illustrate, we can take an instance in the finance sector for assessing credit score - different banks collect identical information such as names, spending habits, and debts (feature space) for different customers (sample space).

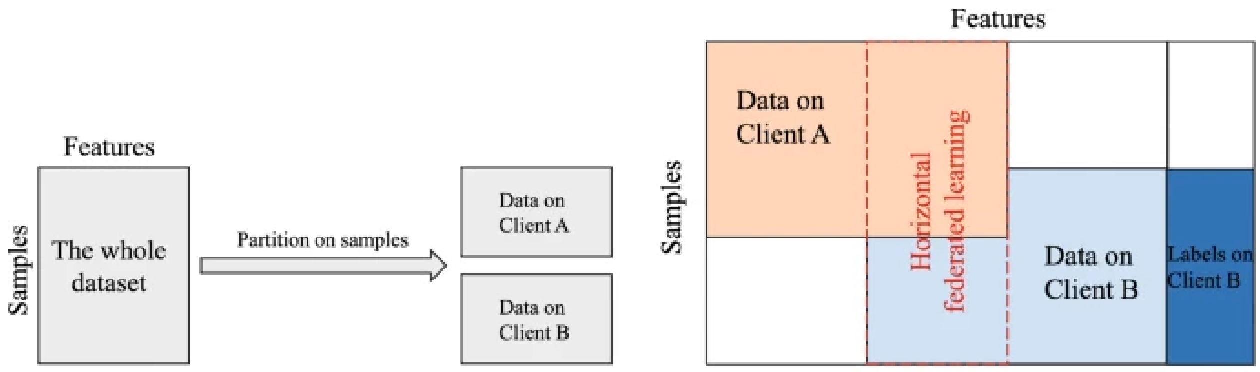


Fig 2.4: Comparison between conventional ML (left) and horizontal FL (right). Note how data on Client A shares a common feature space with Client B, but both datasets do not share a sample space. Taken from: [59]

There are three main differences between conventional ML and horizontal FL, as follows.

Different data types

While data used in horizontal FL may be non-IID, the data used in conventional ML is typically IID. This difference in data type can be attributed to the difference in purpose of both methods of training models - it is acceptable practice for subsets of client data to be manually allocated to enhance convergence in conventional ML but not for FL (see Section 2.1.2).

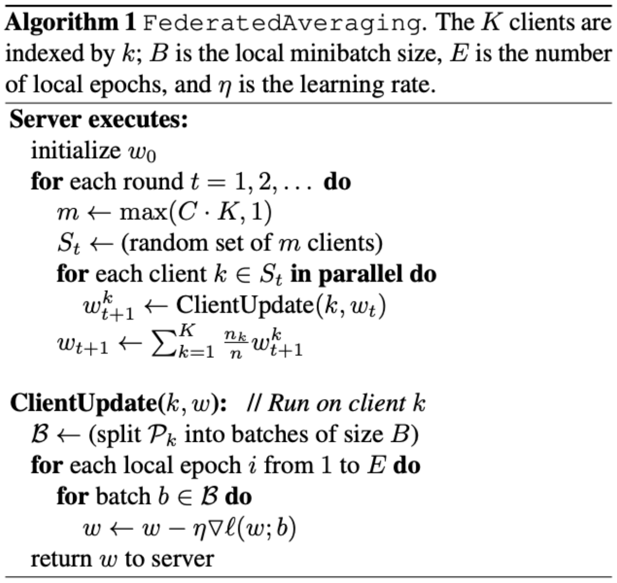
Difference in connectivity

While horizontal FL always has large connected clients and therefore a high rate of connectivity, conventional ML usually has a small number of such nodes, since it would otherwise face the challenge of information bottleneck from network latency from iterative parameter mixing. [79, 59]

Differences in global model update mechanism

While conventional ML typically engages in synchronised global model update whenever local gradients of local populations are computed, horizontal FL does not employ such a model update mechanism due to the consequent high communication costs that it will not be able to handle. [80]

One common horizontal FL algorithm is FederatedAveraging, also abbreviated as FedAvg. The pseudocode of FedAvg can be explored in Fig 2.5, and can be compared to the general steps of horizontal FL algorithms explored below.



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
      $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate$(k, w)$:   // Run on client $k$
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
  for each local epoch $i$ from 1 to $E$ do
    for batch $b \in \mathcal{B}$ do
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return $w$ to server
```

Fig 2.5: Pseudocode for the FederatedAveraging algorithm used commonly in implementations of horizontal FL. Taken from: [58]

The five main steps of horizontal FL algorithms are as follows:

1. The Coordinator initialises global model parameters. The global model is downloaded to participating client devices chosen and accepted by the Selectors (see Section 2.1.1 for more details on the roles of stakeholders in general FL architecture).

2. The connected device clients train the downloaded global model using its own data over multiple training epochs. The trained model weights or gradients are communicated (uploaded) to the Aggregators. Due to differences in computational resources between client devices (see Section 2.1.2), the Aggregators often receive uploads from the different clients at different times.

3. The Aggregators securely aggregate the received uploads without learning anything about any participants. This can be done synchronously or asynchronously, depending on the algorithm. These are then further aggregated by the Master Aggregator before being communicated to the Coordinator.

4. The Central Server sends back the aggregated results to the clients. Participating client devices update their models with the decrypted weights.

5. Steps 2 to 4 are repeated until convergence point is reached.

It is worth noting that the algorithm used in our experiment aggregates uploads synchronously in Step 3 of the outline above.

To further promote security to ensure the privacy of clients during the training process, it might be a priority for clients to prevent the Central Server from inferring data from client devices. In such a scenario, secure multiparty computation (SMC) can be implemented. [81] This is done by partitioning a secret key into multiple key shards, with each client holding only one shard. In order for decryption, any actor will need at least *t* number of shards, where *t* is the threshold value. However, this imposes a large communication cost, as the encrypted model weights will need to be communicated (as is required in HE) between the Central Server and at least *t* clients for aggregation.

2.1.3.2 Vertical Federated Learning

Vertical FL differs from horizontal FL in that it shares a common sample space across all clients, rather than a feature space, as illustrated in Fig 2.6. Converse to horizontal FL, vertical FL is unique in that it ’vertically’ partitions the training data.

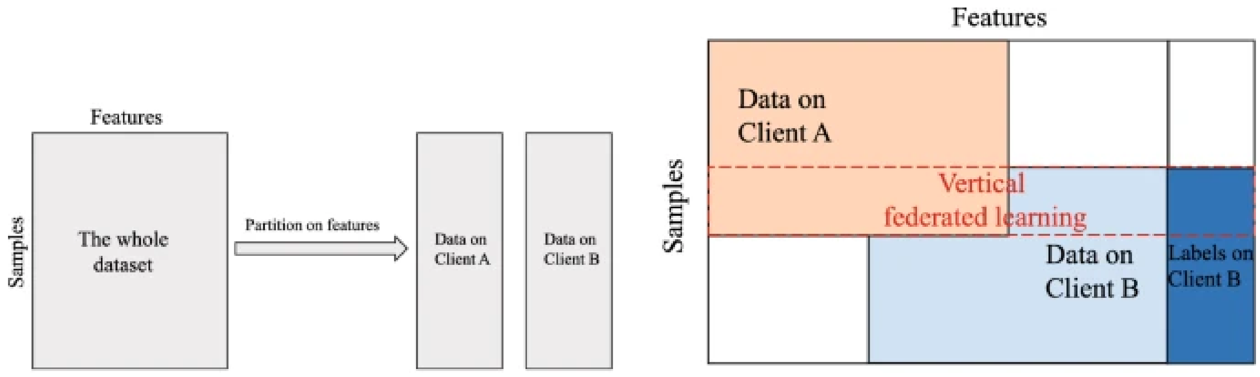


Fig 2.6: Comparison between conventional ML (left) and vertical FL (right). Note how data on Client A shares a common sample space with Client B, but both datasets do not share a feature space. Taken from: [59]

To use a parallel example from Section 2.1.3.1 to further illustrate this structure, we can imagine different product teams in banks selling different products to the same customers (sample space). However, the type of information collected may differ; for instance, the product team in charge of insurance may collect information on the overall health of customers whereas this information is not collected by hedge fund managers.

In order to only expose users in the overlapping sample spaces of clients in the training of the model, encrypted entity alignment is frequently used without revealing data about any clients outside the overlapping space. [84, 86]

Like horizontal FL, SMC can also be used in this data distribution method to provide formal proof of privacy preservation. [85, 87] After convergence, clients only hold the updated model weights relating to their own features.

A more important contrast between horizontal and vertical FL is that the Coordinator in vertical FL calculates the total loss or collecting features, instead of aggregating uploaded weights from clients. This makes it possible for vertical FL to be conducted between client-parties without the need for a Coordinator. [84, 85]

2.2 Differential Privacy

'Differential Privacy describes a promise, made by a data holder, or curator, to a data subject, and the promise is like this:

"You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available".'

Cynthia Dwork, leading scientist in DP.

DP is a technique that is widely adopted for the purpose of privacy preservation without having to incur significant communication costs found in the implementation of additive HE with SMC. The overall concept of DP has been covered in Section 1.1.4.1, and the following subsections will cover mechanisms and bounds of DP.

There are two main forms of DP - local DP and global DP. While utilising global DP results in greater accuracy than local DP, local DP is preferred in applications where there is a lack of trust towards the Central Server. This grants local DP the beauty of enabling participants to ensure their own privacy without having to depend on the Central Server to do so by adding the necessary noise and then sending the randomised response to the Central Server, thereby guarding against information leakage. [88, 93, 94] This will be further explored in Section 3.1 in Chapter 3.

Simply put, the underlying concept of DP algorithms rely on adding random noise to client data (the ground truth) and consequently add imprecision and noise to data, making it difficult for adversarial parties to breach privacy while at the same time providing utility from the data as a whole.

2.2.1 Mechanisms

In this section, we will cover the following basic construction of DP mechanisms:

- Randomised response;
- Laplace mechanism; and
- Gaussian mechanism.

DP is used in both horizontal and vertical FL. To reduce communication costs in horizontal FL, DP is often implemented for privacy preservation due to the substantial savings in communication costs which it introduces. [82] Gaussian and Laplacian noise can be added to mask the trained model from clients before uploading them to the Central Server. [83, 84] On the other hand, DP is used only in asynchronous vertical FL for privacy preservation purposes.

2.2.1.1 Randomised Response

Queries are performed by aggregating all responses from client devices into a single result, whereas microdata is the release of datasets with individual responses (after processing) attributable to each device client. It should be noted that randomised response is only used for queries and not for microdata, as doing otherwise would violate the requirement of DP by showing participation of a participant. [89, 90]

The mechanism of randomised response is broken down as follows.

For a counting query q: $\chi \rightarrow \{0,1\}$ with a dataset $x \in \chi^n$, for each row x_i,

$$y_i = \begin{cases} q(x_i) \text{ with probability } \frac{1+\varepsilon}{2} \\ \neg q(x_i) \text{ with probability } \frac{1-\varepsilon}{2} \end{cases}$$

and

$$\mathcal{M}(x_1, \ldots, x_n) = (y_1, \ldots y_n)$$

If $x \sim x'$ are datasets that differ on the i-th row, their output contributions do not equate only if $q(x_i) \neq q(x_i')$ so that their output contribution in the i-th component differ, which can be expressed as $y_i \neq y_i'$.

This gives us the relationship as follows:

$$\frac{\Pr[y_i = q(x_i)]}{\Pr[y_i' = q(x_i)]} = \frac{\frac{1+\varepsilon}{2}}{\frac{1-\varepsilon}{2}} = e^{O(\varepsilon)}$$

This gives the following:

$$\Pr[y_i = q(x_i')] \leq \Pr[y_i' = q(x_i')]$$

As such, we can deduce that randomised response is O(ε)-differentially private. We can then estimate, with high confidence interval, the value of counting query q(x) with the randomised result by Chernoff Bound.

$$\left| \frac{1}{n} \sum_i \frac{1}{\varepsilon} \cdot \left(y_i - \frac{(1-\varepsilon)}{2} \right) - q(x) \right| \leq O\left(\frac{1}{\sqrt{n} \cdot \varepsilon} \right)$$

As n → ∞, we can estimate the average with increasing accuracy.

2.2.1.2 Laplace Mechanism

The Laplace mechanism works by drawing noise from the continuous Laplace distribution illustrated in Fig 2.7. It can be represented with the equation as such:

$$\mathcal{M}_{\text{Lap}}(x, f, \epsilon) = f(x) + \text{Lap}\left(\mu = 0, b = \frac{\Delta f}{\epsilon} \right)$$

where μ is the expectation (mean) of the Laplace distribution and b is the distribution's scale parameter.

Generally, a small value of privacy budget ε will correspond to a large level of noise added and hence, better privacy preservation at the cost of having a greater degree of uncertainty of the original input (accuracy). The converse is true as well. This relationship will be further explored in Section 3.1.1 in Chapter 3.

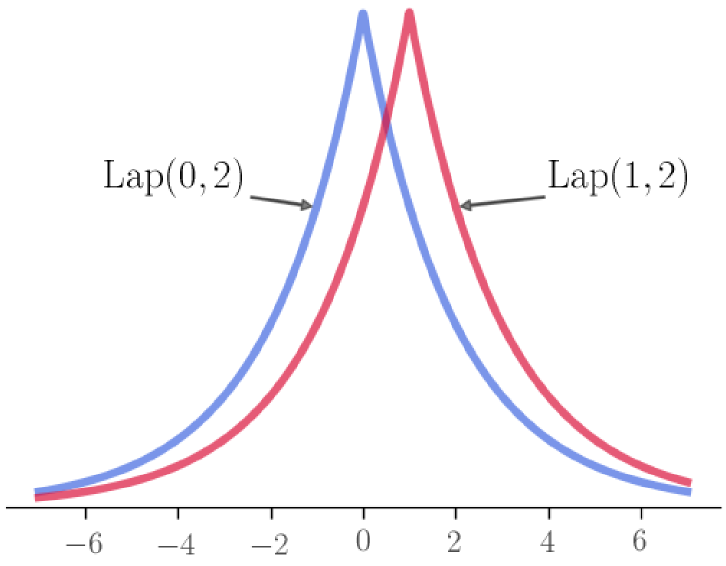


Fig 2.7: Illustration of the Laplace distribution used in Laplace mechanism with mean μ=0 and standard deviation of 2. Compare this distribution against the Gaussian mechanism in Fig 2.8. Taken from: [96]

The Laplace mechanism can be broken down as follows. [91]

Let q be a counting query, such that

$$\mathcal{M}(x) = q(x) + \eta$$

where η is the noise added.

Given that for $x \sim x'$, we obtain the equation $|q(x) - q(x')| \leq 1/n$, which implies that η of magnitude $1/(\varepsilon n)$ is enough for $\mathcal{M}(x)$ and $\mathcal{M}(x')$ to be ε-indistinguishable.

Defining the Laplace distribution Lap(σ) with density at $z \propto e^{-|z|/\sigma}$, we get a ratio of densities:

$$\frac{\text{density of Lap}(1/\varepsilon n) \text{ at } z + 1/n}{\text{density of Lap}(1/\varepsilon n) \text{ at } z} = \mathrm{e}^{1/(n\sigma)} = \mathrm{e}^{-\varepsilon}$$

for $z \geq 0$ and $\sigma = 1/(\varepsilon n)$. [88]

So, for a query $q : \chi^n \rightarrow R$ with bound B and $\varepsilon > 0$, the Laplace mechanism $\mathcal{M}_{q,B}$ over a data universe χ takes a dataset $x \in \chi^n$ and generates output:

$$\mathcal{M}_{q,B}(x) = q(x) + \text{Lap}(B / \varepsilon)$$

This grants two properties relating to the Laplace mechanism $\mathcal{M}_{q,B}$ as follows [88]:

1. If $B \geq GS_q$, the Laplace mechanism $\mathcal{M}_{q,B}$ is ε-differentially private; and
2. For all $x \in \chi^n$ and for $\beta > 0$, we obtain $\Pr[|Mq,B(x) - q(x)| > (B/\varepsilon) \cdot \ln(1/\beta)] \leq \beta$.

2.2.1.3 Gaussian Mechanism

Our experiment utilised the Gaussian mechanism in our implementation of DP.

A parallel to the Laplace mechanism, the Gaussian mechanism works by drawing noise from the continuous Gaussian distribution which is illustrated in Fig 2.8. The variance of the distribution is calibrated according to sensitivity and privacy parameters, which will be further explored in Section 3.1 in Chapter 3.

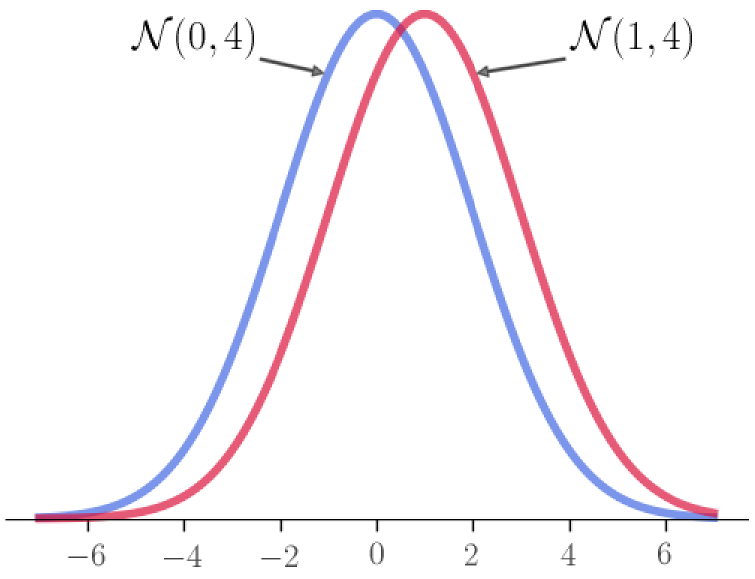


Fig 2.8: Illustration of the Gaussian distribution used in Gaussian mechanism with mean μ=0 and standard deviation of 2. Compare this distribution against the Laplace mechanism in Fig 2.7. Taken from: [96]

For all $\varepsilon \in (0,1)$ and $\boldsymbol{\delta} \in (0,1)$, the Gaussian mechanism provides $(\varepsilon,\boldsymbol{\delta})$-differential privacy and is represented as such [92]:

$$\mathcal{M}_{\text{Gauss}}(x, f, \epsilon, \delta) = f(x) + \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{2\ln(1.25/\delta) \cdot (\Delta f)^2}{\epsilon^2}\right)$$

It is noteworthy that differing from the Laplace mechanism, the Gaussian mechanism $\mathcal{M}(x)$ is not capable of satisfying the criteria for $(\varepsilon,\boldsymbol{\delta})$-differential privacy only for $\varepsilon > 1$, outside its range

(0,1). Randomly sampling a fraction q of the data, rather than the entire dataset, will result in the (ε,$\boldsymbol{\delta}$)-differentially private mechanism $\mathscr{M}$(x) satisfyingthe criteria for (qε,q$\boldsymbol{\delta}$)-differential privacy.

Additionally, the Gaussian mechanism differs from the Laplace mechanism in that in the tail of its privacy loss distribution, the Gaussian mechanism density "changes by an unbounded multiplicative factor over intervals of fixed width". [88]

Chapter 3

Privacy Preserving Federated Learning

3.1 Differential Privacy in Federated Learning Architecture

DP-based based FL typically works on balancing accuracy and the preservation of client-side privacy. The relationship of accuracy and privacy, as well as ways to measure them, will be further explored in the subsection below, Section 3.1.1. As discussed in Section 2.2 of Chapter 2, the addition of noise via DP is an integral part of FL to prevent the disclosure of client data to adversaries.

The way DP interacts with FL will be explored in greater details in this Section.

DP by centralised/local models

DP can be characterised by its use in two different FL model types - centralised models and local models. The main difference in the utilisation of DP between the two models lies in two main factors:

1. The presence of a centralised figure; and

2. The stage at which DP is implemented.

The centralised model

In the centralised model, data from clients in its raw form is provided to a centralised figure. Typically in FL, this comes in the form of uploads of the raw data from participating client devices to Selectors and then to the Aggregators in the Central Server. In the centralised model, DP is typically implemented by the Central Server. The DP implementor, with full access to clients' raw data, then applies DP to obfuscate the data before releasing it. DP on trained data is conducted primarily for statistical approaches and calculations rather than for the purpose of iterations in training.

With this, we can better understand the math behind this model. [93]

We assume a (ε,δ)-DP requirement for both uplink and downlink channels. Uplink-wise, clipping will ensure that for training parameters, from the i-th client among N clients, w_i without perturbation:

$$\|w_i\| \leq C$$

where C is the clipping threshold for bounding w_i. The concept of clipping is explored in greater detail in Section 3.3.

Assuming equal batch size in local training and number of training samples, we can take the local training process in the i-th client to be expressed as:

$$s_{\mathrm{U}}^{\mathcal{D}_i} \triangleq \mathbf{w}_i = \arg\min_{\mathbf{w}} F_i(\mathbf{w}, \mathcal{D}_i)$$
$$= \frac{1}{|\mathcal{D}_i|} \sum_{j=1}^{|\mathcal{D}_i|} \arg\min_{\mathbf{w}} F_i(\mathbf{w}, \mathcal{D}_{i,j})$$

where D_i is the i-th client's database and $D_{i,j}$ is the j-th sample in D_i. Following which, the sensitivity of $s_U^{D_i}$ can be taken to be:

$$\Delta s_{\mathrm{U}}^{\mathcal{D}_i} = \max_{\mathcal{D}_i, \mathcal{D}_i'} \|s_{\mathrm{U}}^{\mathcal{D}_i} - s_{\mathrm{U}}^{\mathcal{D}_i'}\|$$
$$= \max_{\mathcal{D}_i, \mathcal{D}_i'} \left\| \frac{1}{|\mathcal{D}_i|} \sum_{j=1}^{|\mathcal{D}_i|} \arg\min_{\mathbf{w}} F_i(\mathbf{w}, \mathcal{D}_{i,j}) - \frac{1}{|\mathcal{D}_i'|} \sum_{j=1}^{|\mathcal{D}_i'|} \arg\min_{\mathbf{w}} F_i(\mathbf{w}, \mathcal{D}_{i,j}') \right\| = \frac{2C}{|\mathcal{D}_i|}$$

where D_i' and D_i are adjacent datasets with the same size and one differing sample. So, we can infer that for all i:

$$\Delta s_{\mathrm{U}} \triangleq \max \left\{ \Delta s_{\mathrm{U}}^{\mathcal{D}_i} \right\}$$

To ensure (ε,δ)-DP is observed for uplink in one exposure and across all participating clients, the noise scale, represented by the standard deviation of the additive Gaussian noise in the Gaussian mechanism, is:

$$\sigma_U = c\Delta s_U/\varepsilon$$

Downlink-wise, the aggregation for D_i is expressed as for N number of participating clients:

$$s_{\mathrm{D}}^{\mathcal{D}_i} \triangleq \mathbf{w} = p_1\mathbf{w}_1 + \ldots + p_i\mathbf{w}_i + \ldots + p_N\mathbf{w}_N$$

where w is the updated aggregated weights to be downloaded by clients.

Although DP was originally studied comprehensively for use in the centralised model, the success of this model can be compromised by internal and external threats. Insider threats may come from within the centralised figure stemming from a myriad of factors running the gamut from internal fraud to pure human error; external threats typically take the form of data breaches. [94, 95]

Despite these risks, the centralised model remains a viable model that is widely recognised and preferred for its lower communication costs compared to the local model. The central model could be more potent with greater integrity within the central figure and tighter security against external threats. Our experiment utilises the centralised model, by implementing DP during the compilation of the models in the Cloud, which falls under the class of Learning Coordinator.

The local model

In contrast, the local model implements DP at the client-side from participating client devices on their own for obfuscation before releasing the data to stakeholders in the FL architecture. As a result, this model does not suffer from data breaches or require the presence of a centralised figure. The implementation of DP in the local model is termed local DP and has been studied extensively and even been adopted in industry. Because there is no distinction in the sensitivity of personal data in local DP, there have been studies to increase data utility by distinguishing between the sensitivity of different data types and answers, such as [94] by Murakami et al.

Local DP is typically conducted by adding noise to the weights at the client-side before the uploading of data to the Aggregators, so that the data received by the Central Server is already obfuscated even before aggregation. This is termed noising (the addition of noise, whether by Laplace or Gaussian mechanism) before model aggregation FL (NbAFL), which is covered extensively by Kang et al in [93]. The algorithm of NbAFL can be understood with Fig 3.1 below, which illustrates the pseudocode. We will not be covering the algorithm in detail, however, as this is beyond the scope of our project; the inclusion is for the interested reader.

```
Algorithm 1: Noising before Aggregation FL
Data: $T$, $\mathbf{w}^{(0)}$, $\mu$, $\epsilon$ and $\delta$
Initialization: $t = 1$ and $\mathbf{w}_i^{(0)} = \mathbf{w}^{(0)}, \forall i$
while $t \leq T$ do
    Local training process:
    while $\mathcal{C}_i \in \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N\}$ do
        Update the local parameters $\mathbf{w}_i^{(t)}$ as
        $\mathbf{w}_i^{(t)} = \arg\min_{\mathbf{w}_i} \left(F_i(\mathbf{w}_i) + \frac{\mu}{2}\|\mathbf{w}_i - \mathbf{w}^{(t-1)}\|^2\right)$
        Clip the local parameters
        $\mathbf{w}_i^{(t)} = \mathbf{w}_i^{(t)} / \max\left(1, \frac{\|\mathbf{w}_i^{(t)}\|}{C}\right)$
        Add noise and upload parameters
        $\widetilde{\mathbf{w}}_i^{(t)} = \mathbf{w}_i^{(t)} + \mathbf{n}_i^{(t)}$
    Model aggregating process:
    Update the global parameters $\mathbf{w}^{(t)}$ as
        $\mathbf{w}^{(t)} = \sum_{i=1}^{N} p_i \widetilde{\mathbf{w}}_i^{(t)}$
    The server broadcasts global noised parameters
        $\widetilde{\mathbf{w}}^{(t)} = \mathbf{w}^{(t)} + \mathbf{n}_{\mathrm{D}}^{(t)}$
    Local testing process:
    while $\mathcal{C}_i \in \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N\}$ do
        Test the aggregating parameters $\widetilde{\mathbf{w}}^{(t)}$ using local
          dataset
    $t \leftarrow t + 1$
Result: $\widetilde{\mathbf{w}}^{(T)}$
```

Fig 3.1: Pseudocode for the algorithm for NbAFL under local DP. Taken from: [93]

NbAFL is able to satisfy the criteria of DP under distinct protection levels through the adjustments of the different variances of added noise. It has the following key properties, as follows [93]:

1. Convergence performance is negatively correlated to privacy preservation - an improvement in convergence performance will necessarily lead to lower privacy protection levels, with the converse being true;

2. For a fixed privacy protection level, the number of participating client devices is positively correlated to the convergence performance, with the converse once again holding; and

3. There exists an optimal number N of communication rounds, the number of times aggregation takes place, such that convergence performance no longer improves but privacy protection levels continue to get compromised beyond N.

3.1.1 Personally Identifiable Information (PII)

Personally Identifiable Information (PII) is defined as any data that, alone or when complemented with other data, could be used to identify a specific individual. [97] PII might come in the form of unsuspecting forms - it is not limited to uniquely identifying factors such as names, identification numbers, or e-mail addresses; rather, it could also include indicators such as addresses and birthdays, or as Narayanan et al. has shown, even includes reviews of shows on Netflix.

In [98], Narayanan et al. demonstrated that the application of their deanonymization methodology to the Netflix Prize dataset [99], a dataset containing anonymous movie ratings by 500,000 Netflix subscribers, allowed them to identify individual users by comparing matches in the data sets to film reviews on the Internet Movie Database (IMDb). This also allowed Narayanan et al. to trace the identified users' "apparent political preferences and other potentially sensitive information" and was explored in Section 1.3 of Chapter 1.

In a similar vein, Sweeney was also able to identify medical information of a governor William Weld in an anonymously published data by linking PII in the dataset to publicly available data. [97]

Both of these demonstrate how a linkage attack works on attacking privacy.

Careful consideration must be made to balance the utility that can be obtained from data and the privacy of participating individuals. DP prevents this when it is applied in FL, so that PII of participants from the client-side is not revealed; this works by adding random noise with rigorous mathematical measures, such as the Laplace and Gaussian mechanism covered in Section 2.2.1 in Chapter 2. Consequently, the dataset, before and after the addition of differentially additive-noise, is statistically indistinguishable with a sufficiently large dataset; no specific participating individual can be identified singly. However, as pointed out multiple times in this paper, privacy and accuracy (and therefore utility of data) is negatively correlated. Excessive additive noise and inappropriate randomness will noticeably reduce the reliability and utility of the dataset.

FL works best when the trained models capture common trends and patterns in the data, rather than specific data. In the case of PII that is specific to a certain individual due to the rarity of the data, the contribution of the rare data is limited (explored in the following Section 3.1.2) and additive noise is added to obfuscate it as well. [61, 100] This is, interestingly, closely intimate to the defence of adversarial attempts to poison the dataset through the skewing of data and/or model, termed data poisoning and model poisoning respectively. [101] These attempts to poison the global model introduce seemingly rare data as well. As a result, defences against these poisoning attacks also filter genuinely rare data that could be used as PII to identify said particular individual. Fung et al. discusses a defence against sybil-based adversarial attacks by excluding the attacker's sufficiently different updates from the model during model averaging. [102, 103]

3.1.2 Epsilon (ε) and Delta (*δ*) in Differential Privacy Equation

Referring to Equation 1.2 in Chapter 1, a FL model using such an equation is said to satisfy ε-differential privacy. The equation is derived from the following:

$$\log \frac{\Pr[\mathrm{M}(D) \in S]}{\Pr[\mathrm{M}(D') \in S]} \leq \varepsilon \tag{3.1}$$

Adjacent datasets refer to a pair of datasets which only defer by the presence of one individual data entry. For example, in the case of an image classification dataset, we can say that *D* and *D'* are adjacent datasets if both of them contain all the same image-classification pair data except for one pair, which is present in one and absent in the other. In Equation 3.1, *D* and *D*' refer to two adjacent datasets with outputs f(*D*) and f(*D*'). As shown in Equation 1.3, M(*D*) is the adjusted output following the addition of a noise determined usually by a Gaussian normal distribution that is used to train the model.

As an illustration of how Equation 3.1 works, we look at the following example in Fig 3.2: a training of a model on a medical dataset D and whether a specific patient (X) will develop cancer. Dataset D is the default dataset which is used to train the initial model (Case 1). Dataset D' is one which includes Bob's data in the model training phase, which then "predicts" Bob's likelihood of developing cancer using some input.

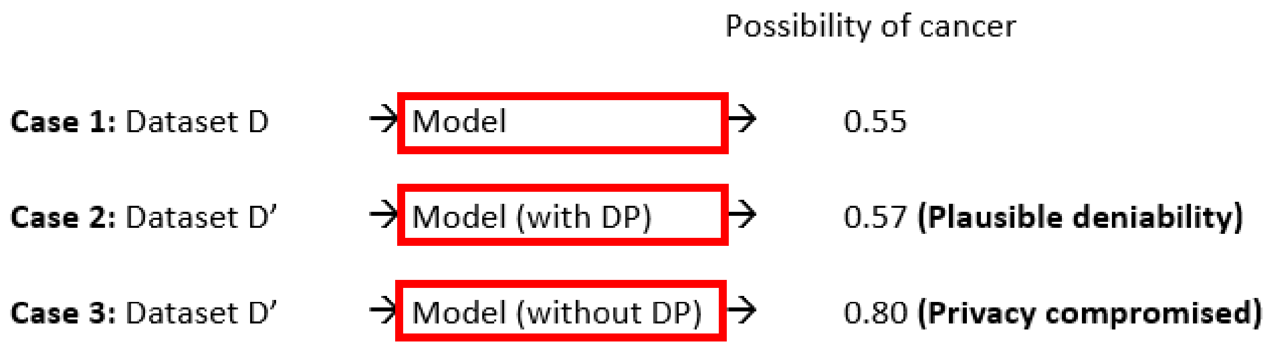


Fig 3.2: 3 cases to illustrate ε-differential privacy

In case 2, the increment in the possibility of cancer is only slightly higher than case 1, and Bob could claim that the model was able to predict with a higher confidence rate due to an increased amount of data used for training, hence claiming "plausible deniability". However, in case 3, if Bob's data causes the model to predict at a much higher success rate, it would show with a high likelihood that Bob does indeed have cancer and this would impede on his privacy. Equation 3.1

would help to prevent such a situation from occurring. The privacy budget, ε, would determine how much this prediction of the possibility of cancer can defer by given adjacent datasets. The relation is as such: a lower privacy budget would mean a lower difference between predictions for models trained using adjacent datasets.

However, such an equation is extremely restrictive, and would be difficult to implement in a FL model. As such, Dwork et al. [40] introduced the use of delta, δ, as a means to provide for some leeway for privacy failure. δ determines the probability of DP failing i.e. adjacent pairs of datasets may display very drastic differences in output e.g. case 1 and case 3.

$$\Pr[M(D) \in S] \leq \exp(\varepsilon) \Pr[M(D') \in S] + \boldsymbol{\delta} \tag{3.2}$$

A FL model is said to be (ε,δ)-differential private if it fulfils Equation 3.2. A ε-differentially private model is equivalent to a (ε, 0)-differentially private model.

To further enhance the privacy of a FL model, we can also apply the privacy amplification theorem. This theorem utilises random sampling and states that if we randomly sample a fraction *q* of the data, rather than the entire dataset, then a (ε,δ)-differentially private mechanism becomes (*q*ε,*q*δ)-differential private, making it more private.

Relation between epsilon and accuracy of FL model

Equation 3.1 would show that if both models predicted the same outcome i.e. the additional dataset did not affect the model at all, it means that privacy is fully conserved since it would be impossible to tell whether the additional dataset was used in the training. However, in order to achieve a very low ε level, it usually means that the noise added to the dataset prior to training has to be significantly high (refer to Equation 1.3), such that the additional dataset in *D'* does not hold much weight in the model training. As such, the accuracy of the model will be negatively affected since the noise is a completely random factor that does not help in improvements to the model.

3.2 Optimisers in Federated Learning

In the implementation of DP on FL, there are different points at which DP mechanisms can be used on the model. While it may be intuitive to treat the final parameters of the model after training with noise in order to preserve the (ε,δ)-differential privacy, one may not be able to appropriately gauge the dependence of such parameters on the training data. By randomly adding noise to them, using noise levels generated based on the worst-case scenario, it may erode the usefulness of the model. Instead, it may be wiser to perform interference during the training phase of the model [21].

In this section, we will first explore some optimisers that are commonly used in training of DL models.

3.2.1 Stochastic Gradient Descent

The idea behind SGD derives from gradient descent (GD), where a set of parameters of the artificial neurons in a FL model is updated consistently in an iterative manner during training to obtain a local minima for an error function.

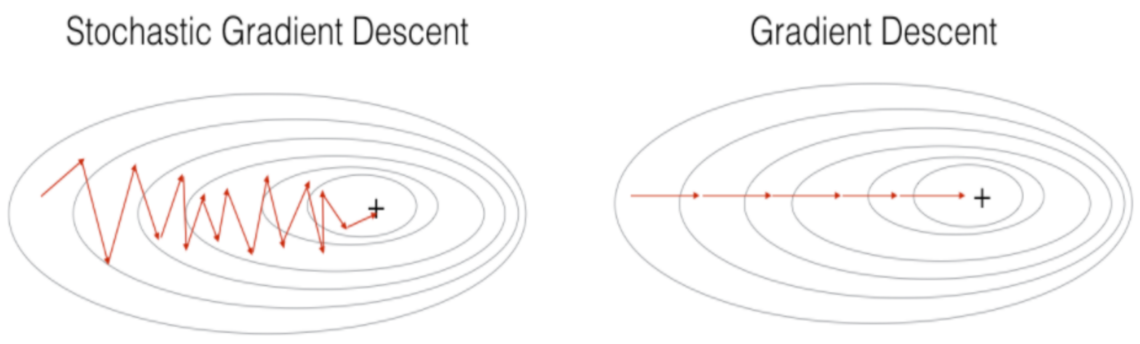


Fig 3.3: Parameters convergence for SGD and GD. Taken from: [105]

For GD, all the data in the training dataset is required to be fully run before the parameters are updated in a singular iteration. Given that FL models require an extensively large dataset to train, using GD may be highly inefficient and take too long since each iteration of parameter updating has to go through the entire training set. As such, the SGD allows for one of the training samples from the entire training dataset to conduct the update of a parameter during one iteration. Fig 3.3 demonstrates how SGD will usually result in more steps before reaching the optimized value, given that it is being constantly updated through each training sample, which may result in some very unusual steps rather than a much more direct approach as shown for the case of GD in Fig 3.3.

If a subset of the training samples are used instead, it is known as a Mini Batch SGD (MB-SGD). This is shown in Fig 3.4, where the MB-SGD will allow for less steps to reach convergence to a local minimum, but the duration of training may not necessarily be shorter due to the time it takes for each iteration (step). Our experiment utilises MB-SGD.

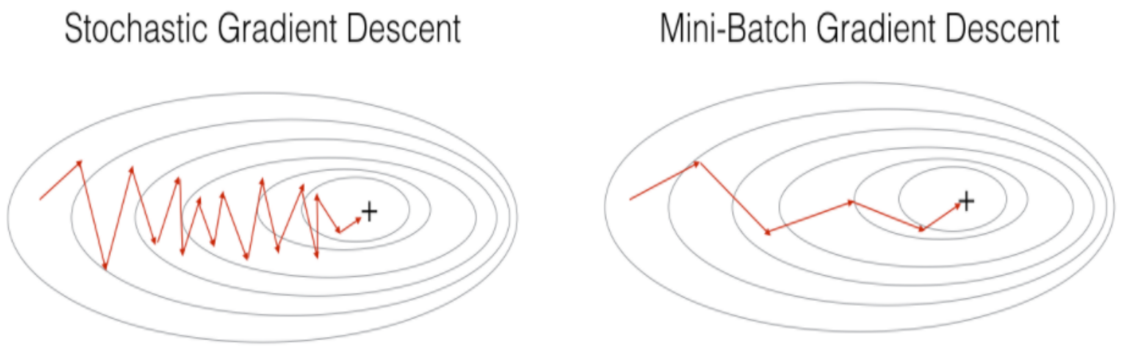


Fig 3.4: Parameters convergence for SGD and MB-SGD. Taken from: [105]

Utilizing SGD allows for parameters to be updated at a much quicker pace, since the model accuracy improves immediately after running the first training sample rather than the entire training dataset. This also helps in allowing SGD to converge much faster than GD, at the expense of the error function not being fully minimized. However, this is acceptable in most cases since the parameter values gotten through SGD will be a good enough approximation.

3.2.2 Adam optimization algorithm

Adam is an optimization algorithm that can serve as an improvement, in some situations, to the classical SGD method to attain optimized parameters, utilizing adaptive estimation of first-order and second-order moments. Kingma, D. et al. [104] mentioned some of the benefits from Adam would include:

- Simple to implement and incorporate into model
- Computationally efficient with little memory requirements
- Usable for non-stationary objectives
- Relevant for solving problems with noisy and/or sparse gradients
- Little tuning required for hyperparameters due to their intuitive interpretation
- Suitable for problems with large dataset and/or parameters

The SGD utilises a single learning rate throughout all its iterations of parameter updating, being kept constant for each parameter and separately adapted as the training occurs. Adam combines the benefits that two other branches of SGD brings about: from the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [104]. The AdaGrad creates per-parameter learning rates to aid with problems involving sparse gradients such as computer vision and natural language, while the RMSProp adapts the learning rate based on how fast the gradients are changing in the most recent updates to the training which allows it to outperform others in a noisy environment.

RMSProp only makes its estimation of the learning rates based on the first moments (the mean). The key improvement that Adam makes over RMSProp is the additional use of the second moments of gradient, which is the uncentered variance. In the code shown in Fig 3.5, beta_1 and beta_2 refer to the exponential decay rate of the first and second moments averages. These values are usually set close to 1 by default which causes the bias in the moments estimate to be close to zero initially. This bias is corrected by calculating the biased-corrected estimates based on the biased estimates that are calculated. The epsilon parameter shown in Fig 3.5 is to prevent divisions by zero during the usage of Adam.

```
tf.keras.optimizers.Adam(
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False,
    name='Adam', **kwargs
)
```

Fig 3.5: Code for implementing Adam optimizer. Taken from: [106]

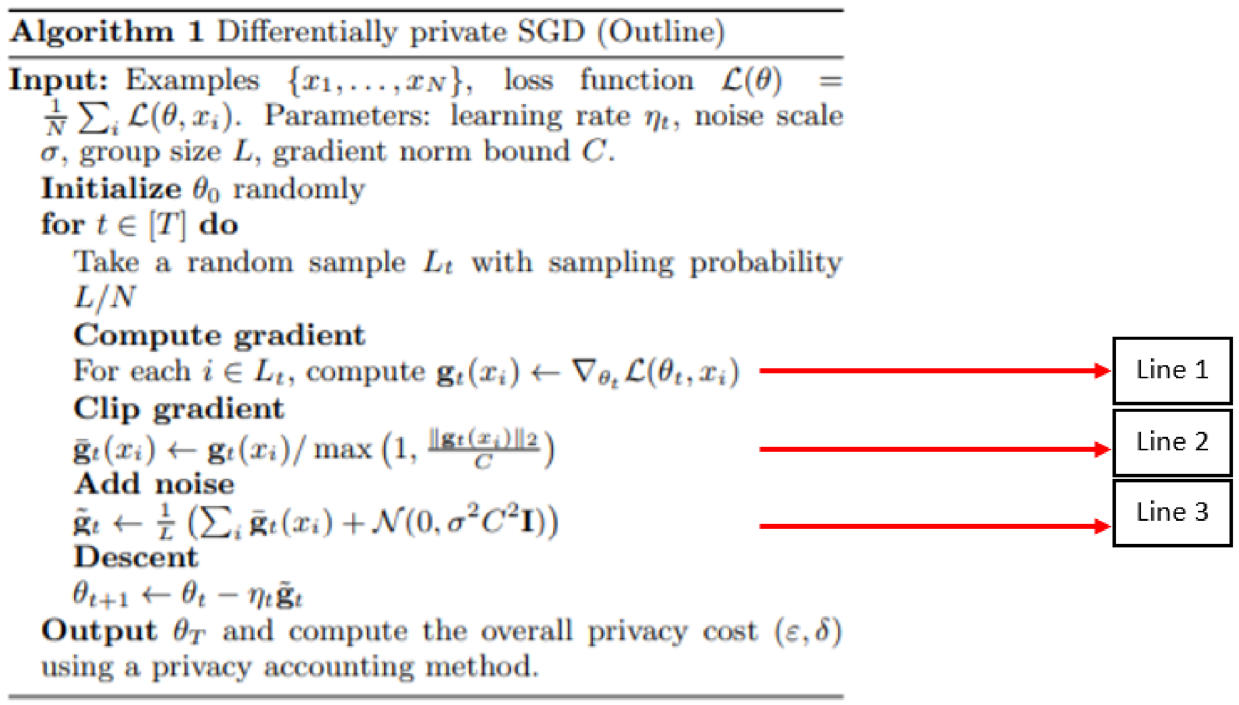
During the initial testing phase of our experiments, we attempted at using Adam as an option for our optimizer, but later switched it for another form of SGD.

3.3 Differentially private SGD (DP-SGD)

The aim of this algorithm is to limit privacy loss per parameter update. This means that DP will be incorporated into FL through the optimizers that will be used to update the parameters.

Fig 3.6 shows the pseudo-code for running this in the FL model. In line 1, we see what would be expected from a classical SGD, where the gradient is computed. However, there is a process of gradient clipping or norm clipping as shown in line 2. This is necessary for implementing DP as the computation of gradient does not take into account an outlier in the dataset i.e. there is no bound on the size of the gradients, which may impact size of the update on the model parameter significantly. As previously explained in Chapter 3.1.1, if the variation of the model is too big given the difference of one training sample in the dataset, attackers may be easily able to de-anonymize the data, hence leaking private information. How line 2 works is to put a bound on the gradient that can be used in an update, which is done through the use of a clipping threshold, C. If the gradient computed is smaller than C i.e. $||g|| < C$, then g is preserved, whereas if $||g|| > C$, then the gradient will be scaled down to the max l2 norm of C and C will be used for the model training in that step. This limits the amount of information we learn from any sample.

Line 3 goes on to add the noise after this norm clipping, where the noise is generated based on a Gaussian distribution with a standard deviation proportional to C*σ. These 2 parameters can then be tuned to obtain the ε and δ required for differential privacy guarantee for each step of gradient descent.



```
Algorithm 1 Differentially private SGD (Outline)
Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate $\eta_t$, noise scale $\sigma$, group size $L$, gradient norm bound $C$.
  Initialize $\theta_0$ randomly
  for $t \in [T]$ do
    Take a random sample $L_t$ with sampling probability $L/N$
    Compute gradient
    For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$
    Clip gradient
    $\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$
    Add noise
    $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
    Descent
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
  Output $\theta_T$ and compute the overall privacy cost $(\varepsilon, \delta)$ using a privacy accounting method.
```

Fig 3.6 Pseudo-code for DP-SGD. Taken from: [21]

3.3.1 Moments accounting

However, these steps were only meant to explain one step in the entire training phase. In order to calculate the overall privacy cost for running the training, we take a look at a very basic way to compute the privacy budget. By the privacy amplification theorem, since DP-SGD is done in mini-batch with sampling with probability of q=L/N, where L is the lot size and N is the size of the input dataset, the privacy budget at each stage would be (qε, qδ)-differentially private. If we were to run the experiment over T number of steps, the overall privacy budget would be (Tqε, Tqδ)-differentially private. However, this is considered to be quite a loose bound and it is possible to create tighter privacy bounds (hence higher privacy) by using methods like the strong composition theorem or moments accounting. The moments accounting theorem is far superior to the strong composition theorem and works on the basis that the privacy loss being calculated is a random variable on its own, with a probability distribution that has a long tail (values that are beyond the ε value. Looking at a Gaussian normal distribution, we can see that over 95% of outcomes fall within 2 standard deviations of the mean. In order to achieve tighter privacy bounds, higher moments can be used, with the first moment being the mean and the second moment being the variance/standard deviation. What the moment accountant does is to keep track of the bounds associated with each of the moments throughout the training and picks the highest bound, choosing from the first moment up to the 31st moment for practical reasons.

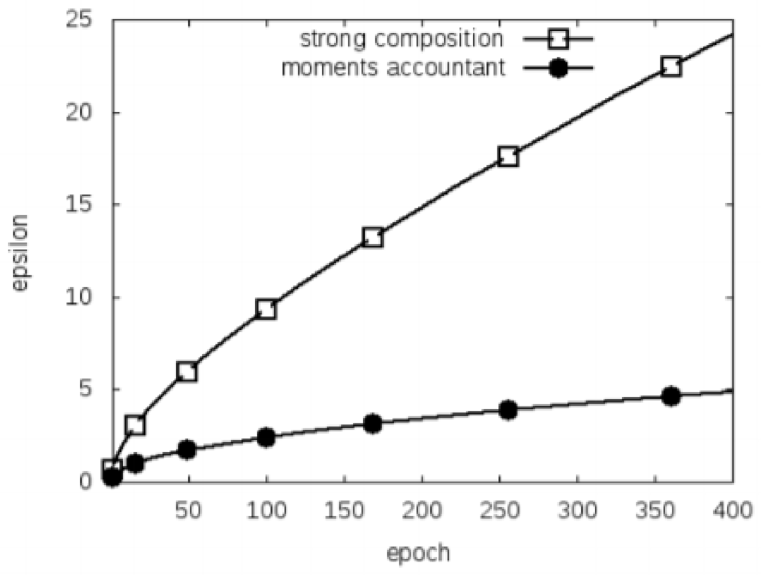


Fig 3.7: Difference between privacy budget achievable using moments accountant and strong composition. Taken from: [21]

As seen from Fig 3.7, we can see that moments accountant theorem outperforms even the strong composition theorem which was supposed to be an improvement to the basic idea of adding privacy manually through the privacy amplification theorem. Abadi, M. et al. showed that using moments accounting, the model only needed a privacy budget of 2.55, versus previous guarantees of 24.22. Having a tighter upper bound on the privacy budget means there is no longer heavy over-estimation of privacy loss of the algorithm, so the algorithm can be run longer or reduce the noise (hence improving accuracy of the model as previously discussed).

Chapter 4

Implementations, Experiments & Observations

4.1 Objectives

The main objective of our experiment is to explore the effects of the Federated Learning architecture, as well as differential privacy on the accuracy, more specifically, the validation accuracy of a trained model. Along the way, we would determine the relationship between privacy and the nature of noise added, more specifically, the relationship between Standard Deviation (stddev) and epsilon (Ɛ).

4.2 Dataset

We have chosen the MNIST (Modified National Institute of Standards and Technology) Dataset [107] as our training and testing sources, due to its robustness, widespread-usage and reliability. This dataset was created by combining the data points from NIST's original datasets, and samples from American high school students [108]. It consists of a large number of handwritten digits, from 0-9. An example of how the dataset looks like is shown below in Figure 4.1.

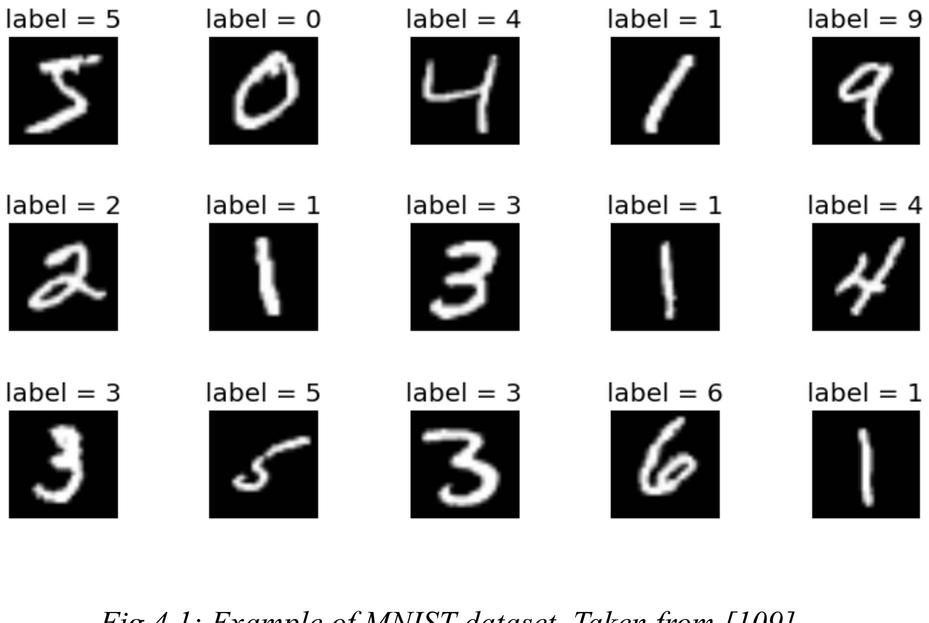


Fig 4.1: Example of MNIST dataset. Taken from [109]

More specifically, the API endpoint provided by Tensorflow [110] supplies us with 60 000 training images, and 10 000 testing images. Note that the train-test split as governed by the proportion of the datasets is about 86:14, which is around the industry standard of a 80:20 train-test split [111].

I would like to point out that our project centres around a Proof of Concept. As such, we believe that the MNIST dataset would be the perfect baseline for us to explore the development of a locally-trained deep-learning model, into a shared federated structure, and lastly after incorporating privacy elements into the sandbox.

We have decided to utilise only half of the datasets provided, i.e. 30 000 training images, and 5 000 testing images. In order to produce statistically-significant results, it is imperative to repeat the procedures in a robust fashion, which we would be elaborating more in the later sections. Moreover, as this project seeks to establish experimentally a relationship between privacy and accuracy, we felt that it is not critical for our model to provide the optimal accuracy values and reach convergence.

4.3 Architecture

4.3.1 Model

The model that we would be utilising throughout our experiment is a convolutional neural network (CNN) with 7 layers, implemented using Tensorflow Keras [112]. Our model consists of these following layers:

- Convolutional layers, using RELu as the non-linear activation function,
- Max Pooling layers, for reducing samples for future layers,
- Flattening layers, for removing all dimensions except for one onto the targeted data structures,
- Dense/Fully-connected layers, also using RELu as the non-linear activation function.

The implementation of the instantiation of the model would be shown in Section 4.3.2.3.

4.3.1.1 Optimisers

It is important to note that the nature of the model (DP or non-DP) would depend on the type of optimiser passed to the model upon compilation.

We would be using a standard Stochastic Gradient Descent (SGD) Optimiser from Tensorflow [113] for our control non-differentially-private setup. The workings of a SGD training process is explained in Section 3.2.1. We decided to utilise SGD, in favour of other optimisers, e.g. Adam and Adagrad, as we felt that, although slower, SGD treads more carefully towards the convergence point, given the reduction in size of the dataset the model is training on.

To keep the comparison as similar as possible, we would be utilising the differentially-private Gaussian SGD optimiser, provided by Tensorflow Privacy [114]. The workings of a differentially-private optimiser is explained in Section 3.3, i.e. the clipping of gradients and the addition of noise. The mechanism of Gaussian Noise is explained in Section 2.2.1.3.

The SGD optimisers Tensorflow and Tensorflow Privacy provide are in fact considered to be Mini Batch GD optimisers, since both provide the parameter of 'batch size'. In our experiment, the batch size trained on is not 1, but instead 125, which would be mentioned later in Section 4.3.2.2.

4.3.2 Federated Learning Architecture

As explained above in Section 2.1.3.1 of Chapter 2, we would be implementing a horizontal Federated Learning architecture. There are 3 main stakeholders defined in our implementation, which would be explained more in-depth in the following subsection.

4.3.2.1 Definitions

Cloud

The Cloud acts as the server and hub. It is responsible for consolidating the trained weights, as well as creating, updating and maintaining the updated version of the model after every round, which we termed it as Generation. It would be touched upon further below.

Traditionally, clients trained on their local data, in which these data have no contact with each other, not even with the Cloud. This is coherent to the purpose and definition of a Federated Learning architecture. In our setup, the Cloud is also responsible for retrieving the training and test sets and parsing them in accordance to the number of Clients in the ecosystem.

Clients

The Clients are the main players in the training of a specific version of the model received from the Cloud, determined by the number of Generations elapsed. The Clients would not have contact with each other, but only with the Cloud and Aggregator.

Aggregator

The job of the Aggregator is to, as the name suggests, aggregate the trained weights received from the Clients. Our implementation works closely with the traditional aspect of the aggregation of weights as depicted in the pseudocode in Fig 2.5 in Section 2.1.3.1 - Simple averaging of the trained weights down to each individual layer. In the future, should there be a need for differential-private elements to be incorporated into the aggregation of trained weights, it could be added to the aggregation function, without the knowledge of the Cloud or Clients, since the Aggregator is decoupled from the Cloud and Clients.

Generations

We believe that it is imperative to fully clarify the various concepts we have introduced in our experiments, as they possess some elements that deviate from the traditional definitions. The pseudocode in Fig 2.5 utilises the concept of 'Round'. We believe that the word 'Generation' perfectly illustrates the notion of the evolution of a model in training, and the complete flow of data within the ecosystem and back to the starting point. The specifics of a Generation and its elapse would be explored in Section 4.3.2.3.

Epochs

The traditional sense of an epoch is maintained in our implementation, and it should not be confused with the concept of a 'Generation'. An epoch is defined as the number of times the training dataset is passed through the layers of the entire CNN model [115], and in our implementation, it refers to the number of cycles the local data is trained locally by each Client in a single Generation.

4.3.2.2 Hyperparameters and Package Versions

To lay the foundation down for the following few sections on the specifics of our implementation, we would like to list the hyperparameters set throughout our experiment. We would be covering the DP hyperparameters in Section 4.4.2.

- Number of Clients: 2
- Size of Training Dataset: 30 000 (15 000 per Client)
- Number of Generations: 3

- Learning Rate: 0.1
- Batch Size: 125

As an introduction to the experiments that we have carried out, which would be explained below in Section 4.4, we would be varying the number of Epochs run per Generation, namely setting to 3 and 10.

The versions of the Tensorflow packages are listed below.

- Tensorflow Package Version: 1.15.4
- Tensorflow Privacy Package Version: 0.5.1

4.3.2.3 Flow of Information

In this subsection, we would be explaining thoroughly our implementation of a Horizontal Federated Learning architecture, together with an overview on the flow of information throughout the architecture.

We would like to draw parallels to the 5 steps a Horizontal Federated Learning architecture should be, as illustrated in Section 2.1.3.1 of Chapter 2, with our own 9-step flowchart.

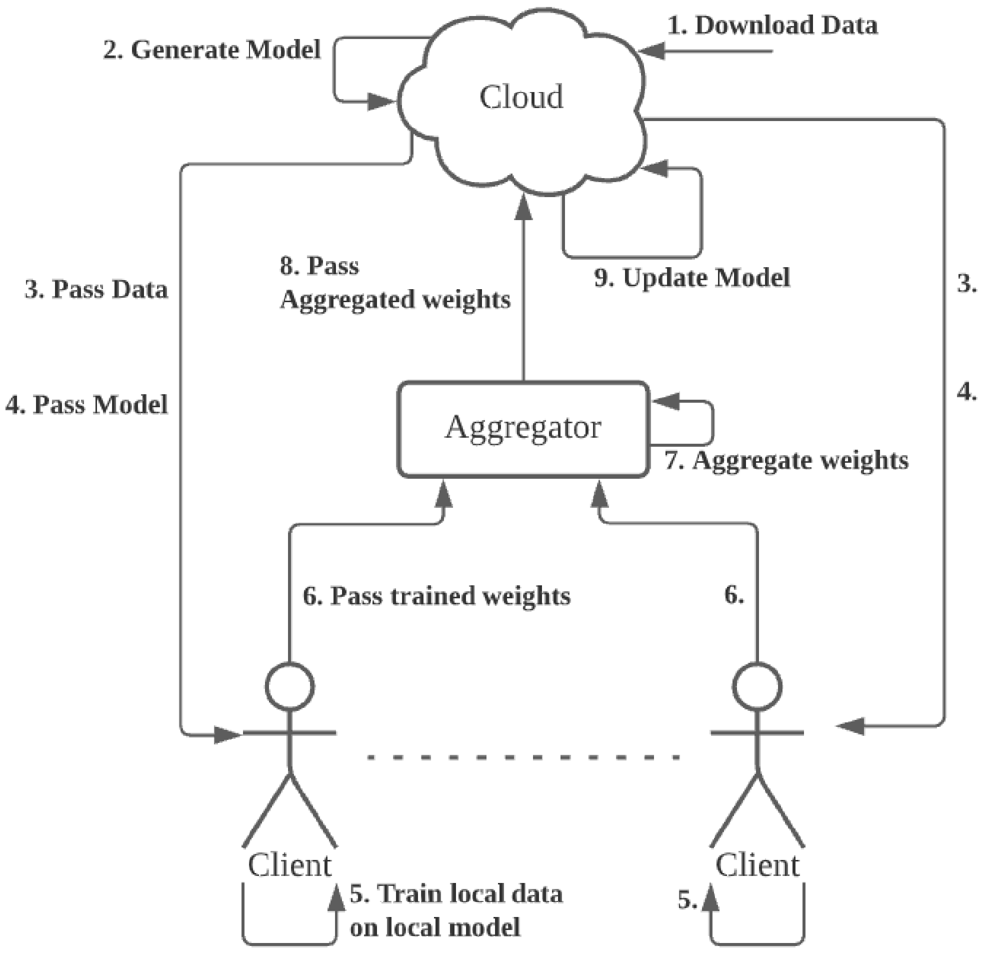
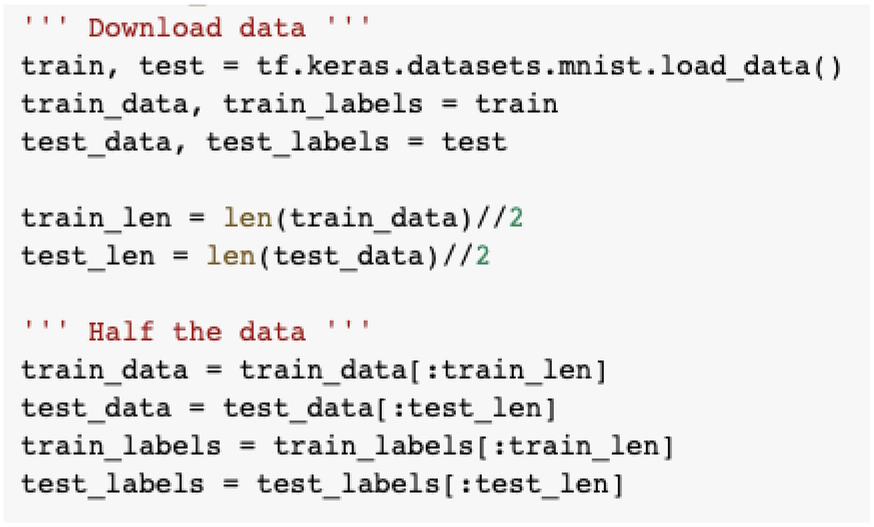


Fig 4.2: Horizontal FL Architecture implemented

As shown in the figure above, below explained are the detailed steps taken:

1. The Cloud downloads the training and testing datasets from the API endpoint provided by Tensorflow. The Cloud also segments half of the each dataset to be utilised in our experiment, since there are 2 Clients in our setup.



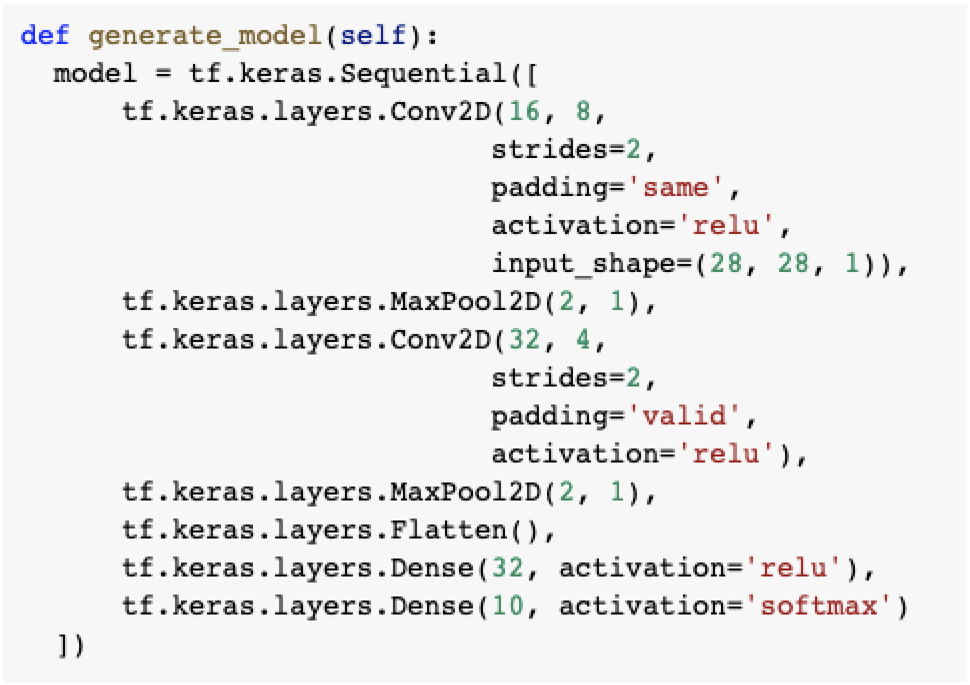
```python
''' Download data '''
train, test = tf.keras.datasets.mnist.load_data()
train_data, train_labels = train
test_data, test_labels = test

train_len = len(train_data)//2
test_len = len(test_data)//2

''' Half the data '''
train_data = train_data[:train_len]
test_data = test_data[:test_len]
train_labels = train_labels[:train_len]
test_labels = test_labels[:test_len]
```

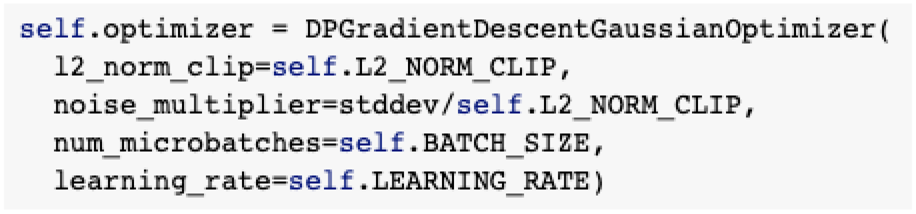
Fig 4.3: Code Segment for Downloading Data

2. The Cloud then generates the CNN model with the appropriate layers. It then compiles the model with the target optimiser (DP or non-DP).



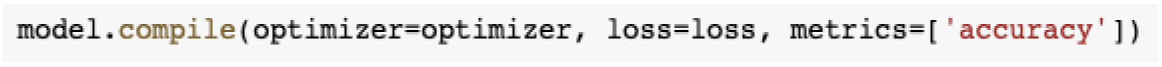
```python
def generate_model(self):
  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(16, 8,
                             strides=2,
                             padding='same',
                             activation='relu',
                             input_shape=(28, 28, 1)),
      tf.keras.layers.MaxPool2D(2, 1),
      tf.keras.layers.Conv2D(32, 4,
                             strides=2,
                             padding='valid',
                             activation='relu'),
      tf.keras.layers.MaxPool2D(2, 1),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(32, activation='relu'),
      tf.keras.layers.Dense(10, activation='softmax')
  ])
```

Fig 4.4: Code Segment for Generating Model



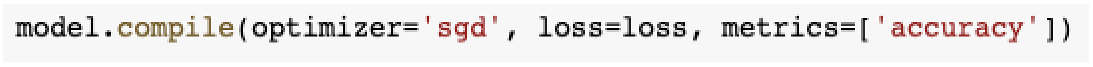
```python
self.optimizer = DPGradientDescentGaussianOptimizer(
  l2_norm_clip=self.L2_NORM_CLIP,
  noise_multiplier=stddev/self.L2_NORM_CLIP,
  num_microbatches=self.BATCH_SIZE,
  learning_rate=self.LEARNING_RATE)
```

Fig 4.5: Code Segment for Differentially-Private Optimiser



```python
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

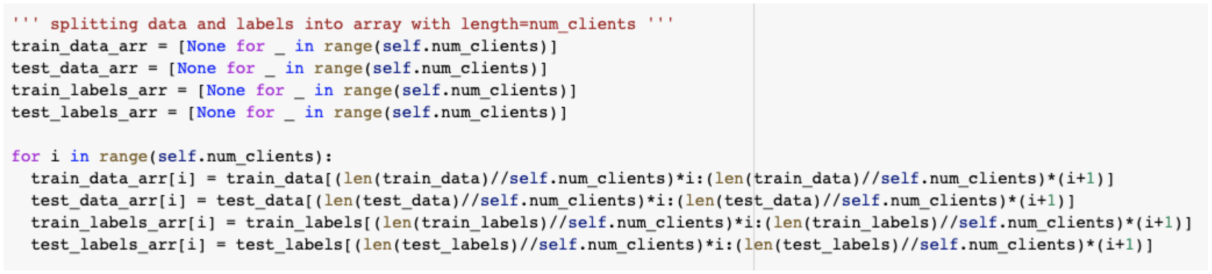
Fig 4.6: Code Segment for Compilation of Model with DP Optimiser



```python
model.compile(optimizer='sgd', loss=loss, metrics=['accuracy'])
```

Fig 4.7: Code Segment for Compilation of Model with DP Optimiser

3. Next, the Cloud parses the datasets according to the number of Clients and sends the respective datasets to the Clients. In our case, it would be 2 Clients, and the first Client receives the first half of each dataset, and the second Client is responsible for the second half.

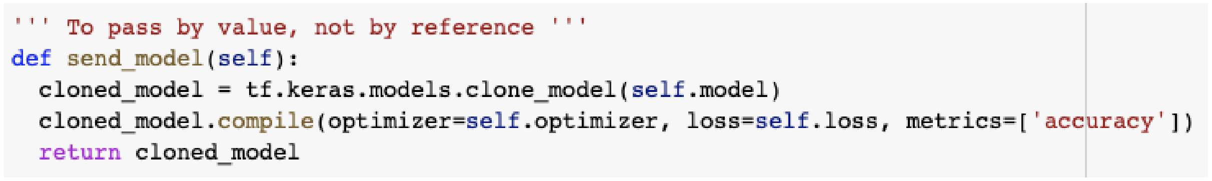


```python
''' splitting data and labels into array with length=num_clients '''
train_data_arr = [None for _ in range(self.num_clients)]
test_data_arr = [None for _ in range(self.num_clients)]
train_labels_arr = [None for _ in range(self.num_clients)]
test_labels_arr = [None for _ in range(self.num_clients)]

for i in range(self.num_clients):
  train_data_arr[i] = train_data[(len(train_data)//self.num_clients)*i:(len(train_data)//self.num_clients)*(i+1)]
  test_data_arr[i] = test_data[(len(test_data)//self.num_clients)*i:(len(test_data)//self.num_clients)*(i+1)]
  train_labels_arr[i] = train_labels[(len(train_labels)//self.num_clients)*i:(len(train_labels)//self.num_clients)*(i+1)]
  test_labels_arr[i] = test_labels[(len(test_labels)//self.num_clients)*i:(len(test_labels)//self.num_clients)*(i+1)]
```

Fig 4.7: Code Segment for Halving Datasets

4. The Cloud now passes a copy of the current model to the Clients



```python
''' To pass by value, not by reference '''
def send_model(self):
  cloned_model = tf.keras.models.clone_model(self.model)
  cloned_model.compile(optimizer=self.optimizer, loss=self.loss, metrics=['accuracy'])
  return cloned_model
```

Fig 4.8: Code Segment for Passing Copy of Model to Clients

5. The Clients train on the copy of the current model with the received local training dataset individually. At this stage, should the optimiser of the model be differentially-private,

clipping would be done to the trained weights, and noise would be added to weights. These concepts are explained in Section 3.3. These occur at every epoch.

```
''' fit the model with local training data '''
def train(self, epochs, batch_size):
  self.model.fit(self.train_data, self.train_labels, validation_data=(self.test_data, self.test_labels), epochs=epochs, batch_size = batch_size)
```

Fig 4.9: Code Segment for Calling of Training Method

6. After all 3 epochs have elapsed, the Clients pass the trained weights to the Aggregator. The local model and datasets do not leave the Clients.

```
''' pass trained weights to aggregator '''
aggregator.get_weights_from_clients(client_idx=client.idx, weights=client.get_weights_from_model())
```

Fig 4.10: Code Segment for Passing Trained Weights to Aggregator

7. The Aggregator waits for the trained weights from Clients in a synchronous fashion, i.e. until all of the trained weights for the current Generation is received. The Aggregator would then proceed on with the aggregation. As mentioned in Section 4.3.2.1, the Aggregator performs a simple averaging of weights on each layer of the model in our implementation. We would like to point out that in a differentially-private setup, the trained weights would be differentially-private and the Aggregator should not, in principle, easily infer the PII from a specific client, i.e. perform an inference attack.

```
def aggregate_weights(self):
  ''' Populate return list with empty numpy arrays of appropriate size '''
  new_weights = []
  for layer in self.weights_from_clients[0]:
    new_weights.append(np.zeros(shape=layer.shape))

  ''' Add value of weights from all clients at each layer '''
  for client_weights in self.weights_from_clients:
    for i, w in enumerate(client_weights):
      new_weights[i] += w

  ''' Find average value of weights at each layer '''
  for layer in new_weights:
    layer /= self.num_clients

  return new_weights
```

Fig 4.11: Code Segment for Aggregation of Weights

8. After aggregation, the Aggregator passes the aggregated weights back to the Cloud.

```
def send_weights_to_cloud(self, cloud):
  aggregated_weights = self.aggregate_weights()
  cloud.receive_and_save_weights(aggregated_weights)
```

Fig 4.12: Code Segment for Passing Aggregated Weights to Cloud

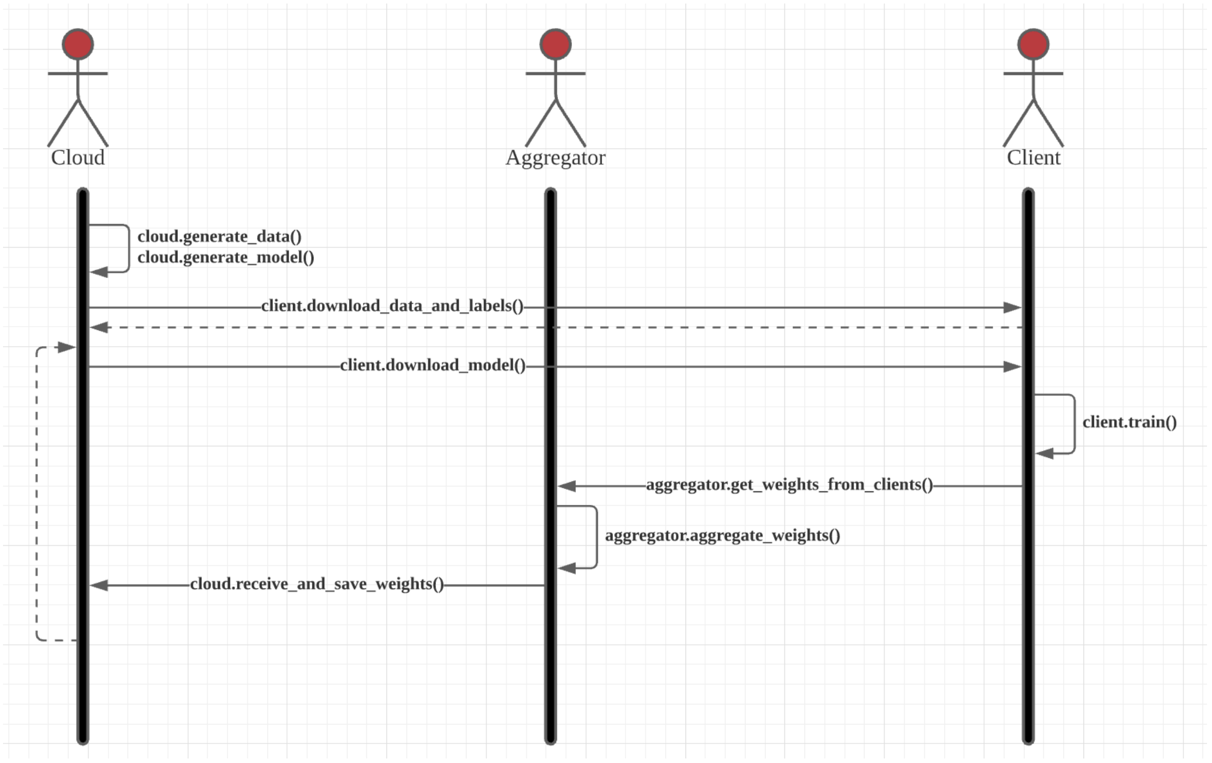
9. The Cloud then updates its model, i.e. the global model with the aggregated weights

```
''' receive new weights from aggregator and set to model '''
def receive_and_save_weights(self, weights):
  self.model.set_weights(weights)
```

Fig 4.13: Code Segment for Updating Global Model with Aggregated Weights

A Generation is defined to be the process as described from Step 4 to 9. As such, steps 4 to 9 are run 2 more times.

The UML diagram below depicts the logic flow between the 3 parties, with the method utilised clearly defined.



Cloud
Aggregator
Client
cloud.generate_data()
cloud.generate_model()
client.download_data_and_labels()
client.download_model()
client.train()
aggregator.get_weights_from_clients()
aggregator.aggregate_weights()
cloud.receive_and_save_weights()

Fig 4.14: UML Diagram of Data Flow

4.4 Findings

To reiterate, we would be repeating the run as described in Section 4.3.2.3 for the values of Standard Deviation, which would be mentioned in Section 4.4.2. To achieve statistical significance, we would be also repeating the run 20 times for each of the Standard Deviation values tested on.

Also, as mentioned briefly in Section 4.3.2.2, our experiments would be repeated twice, for the two different values of the Number of Epochs per Generation, namely 3 and 10.

4.4.1 Differential Privacy hyperparameters

We would like to first state the rest of the hyperparameters required for our differentially-privacy implementation.

- L2 normalisation clipping: 1.0
- Delta (δ): 1.0 x 10^{-5}

We chose this L2 normalisation clipping value as firstly, it is used in the tutorial provided by Tensorflow Privacy [116], and secondly, it is more convenient for calculation. It is explained in the tutorial that the concept of 'noise multiplier', which is used in the calculation of epsilon, is defined as the 'Ratio of the standard deviation to the clipping norm'.

The delta value utilised in the calculation is in accordance to the sample values provided by the method from Tensorflow Privacy [117]. This method applies the RDP accountant [118, 119] to estimate the privacy budget of an iterated Sampled Gaussian Mechanism. Not only that, the delta value utilised conforms to the standard of being the inverse to the size of the dataset used [120]. Delta values such as 1.0 x 10^{-5} or less should not compromise utility. [120]

$$\delta = \frac{1}{100\,000} < \frac{1}{30\,000}$$

4.4.2 Establishment of Relationship

Before we continue on with our findings, we would like to first establish the relationship between Standard Deviation and epsilon, i.e. the degree of randomness of the noise added, and metric of privacy loss. By definition, the smaller the epsilon value, more noise has been added to the weights. [119] With this relationship established, with a lower epsilon value, there would be a lower privacy loss, more privacy is preserved [120], and hence resulting in a lower final accuracy value of the trained model. To summarise, Standard Deviation has an inverse relationship with epsilon. Both experiments with the different epoch values are able to portray the similar relationship, by using the calculator method provided by Tensorflow Privacy [117]. This association is demonstrated in the figures below:

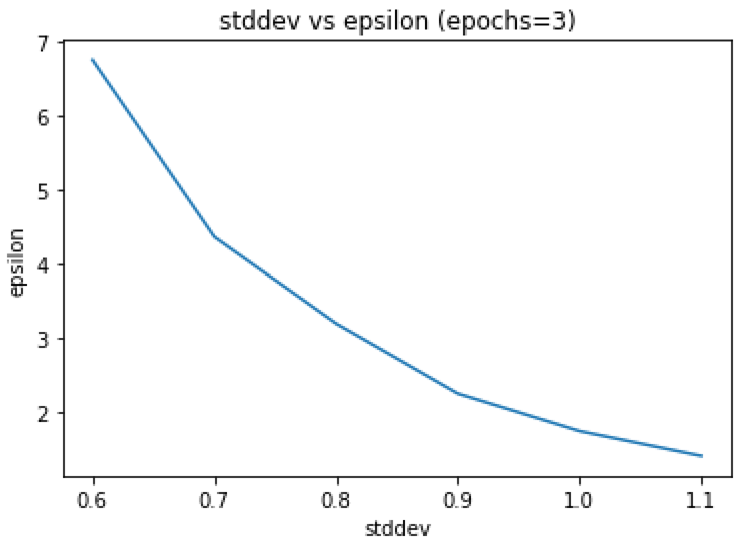


Fig 4.15: Relationship between Standard Deviation & Epsilon for 3 epochs per Generation

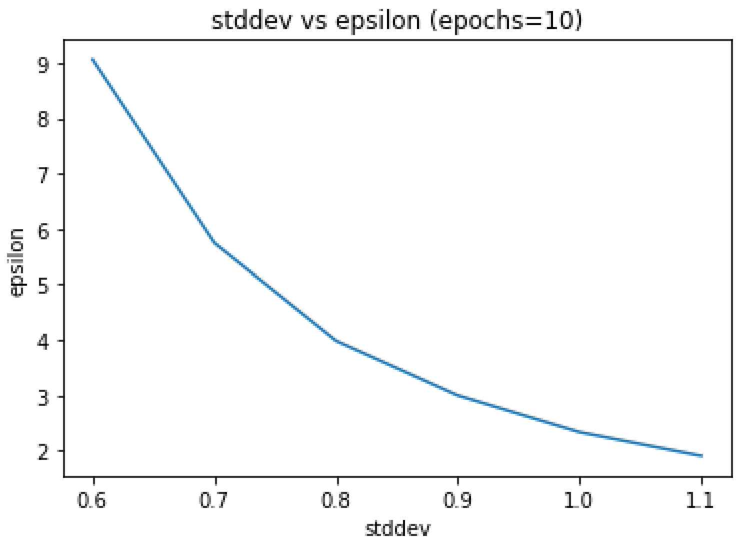


Fig 4.16: Relationship between Standard Deviation & Epsilon for Model with 10 epochs per Generation

The parameters required (size of dataset and batch size) are listed above in Section 4.3.2.2.

We have decided on the values of epsilon, as a result of the values of Standard Deviation selected, to be within the range of 1-10. Cross-referencing to the experiment explained in this paper [121], values of epsilon considered were from 1-10. In our experiment, the values of Standard Deviation are determined to be 0.6, 0.7, 0.8, 0.9, 1.0, 1.1.

4.4.3 Differences in overall Accuracy

4.4.3.1 Results, Findings & Motivation for difference in Epoch values

In this subsection, we would be exploring deeper the effects of Differential Privacy have on accuracy. The reason as to why we have decided to carry out the experiment twice, with different epoch values, is to showcase our findings of the number of epochs run per Generation not having a significant impact on establishing the relationship between Standard Deviation and epsilon. More specifically, our experiments do not display an obvious relationship as mentioned, and we carry out the experiment twice to cement our hypothesis, which would be discussed in detail below.

The validation accuracy values we have gathered amongst the 2 clients after the training processes for the respective Standard Deviation values, averaged amongst 20 rounds of training, are shown in the diagrams below. The distribution of the accuracy values gathered over 20 rounds of training for each epoch value are plotted using boxplots to showcase the degree of the noisy nature of the trained weights, due to effects from differential privacy. We have also included the averaged accuracy value achieved from a non-differentially-private model, to juxtapose with the results found as a result of differential privacy.

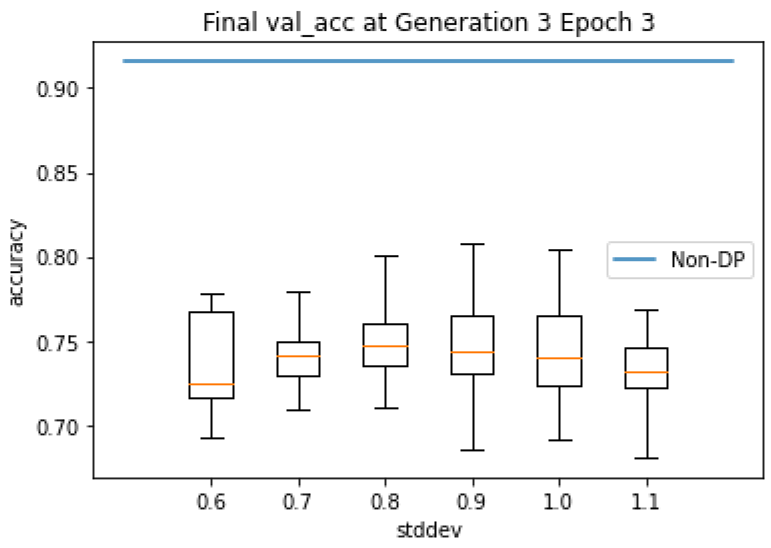


Fig 4.17: Final Validation Accuracies for Model with 3 epochs per Generation

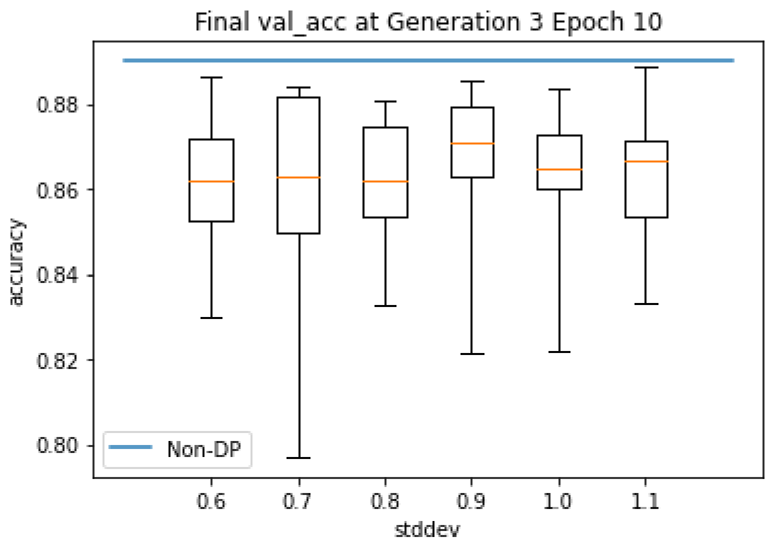


Fig 4.18: Final Validation Accuracies for Model with 10 epochs per Generation

It is evident that the accuracy of a non-differentially-private model is higher than that of a differentially-private model, which is coherent with our understanding and research. It can also be determined that the overall accuracy values for all models trained with 10 epochs per Generation are higher than that trained only with 3 epochs per Generation. Naturally, with a higher degree of training engenders a larger accuracy value, especially in the context of a differentially-private architecture, where aggregation affects accuracy.

Curiously, the averaged accuracy value for the non-differentially-private model, trained with 10 epochs per Generation, is lower than that of the non-differentially-private model, trained with 3 epochs per Generation. This is due to a certain degree of overfitting in the more trained model, which led to a slight drop in accuracy value, which in fact since we are recording the validation accuracy value, as training progresses. This is the reason why we chose 10 epochs per Generation for the second experiment, as a larger number of epochs per Generation set would result in a larger degree of overfitting. This would skew the nature of our model away from being a control setup, hence muddling our analyses.

However, from the boxplots shown, it is indeterminate in establishing a relationship between Standard Deviation and accuracy. Both the distributions and quartile ranges for all of the Standard Deviation values tested on are similar.

Looking closely, we can see a slight decrease in the values of the distribution for the Standard Deviation values of 0.9, 1.0 and 1.1, in the model trained with 3 epochs per Generation, seen in Fig 4.17. This phenomenon agrees with our hypothesis and research. However, the drop in values is not large enough to establish a concrete conclusion, and this relationship does not extrapolate to the Standard Deviation values of 0.6, 0.7 and 0.8 in the same Figure, as well as the boxplots in Fig 4.18.

4.4.3.2 Explanation for difference in Epoch values

At this juncture, we would like to delve deeper into our justification for carrying out the experiment twice, training with a different epoch value.

After concluding the training process and analyses for 3 epochs per Generation, we surmise that one of the plausible reasons to account for this relationship explained in the previous subsection, or the lack thereof, could be due to the small number of epochs run per Generation. Perhaps the models, after the conclusion of the training processes, were still a distance away from their respective true convergence points. As such, with the limited number of epochs ran, the accuracy values retrieved could not differentiate the differentially-private models with different Standard Deviation values from each other well.

This effect is exacerbated especially due to the aggregation of local weights, and hence the significant difference between the aggregated weights and the local weights from the same Generation. This disparity retracted each local model away from its respective convergence point (governed by the local training data) after every Generation, accentuating the non-convergence and similarity of accuracy values. We would explore the phenomenon further in the next section, Section 4.4.4.

As such, we have repeated the experiment, keeping all the hyperparameters constant, but varying the number of epochs run per Generation, and of course the Standard Deviation values. What we have found, whose results plotted in Fig 4.18, display the similar lack of relationship. We hypothesize that the relationship could in fact be determined by varying the other hyperparameters (batch size, delta value), or even widening the spread of Standard Deviation values tested on.

However, regarding expanding the range of Standard Deviation values, and hence the noise multiplier, we would like to argue that the larger the Standard Deviation values, the smaller the difference it would make on the privacy budget. This could be clearly interpreted from Fig 4.15 & Fig 4.16 . Epsilon decreases less than proportionately as Standard Deviation increases. Hence, widening the spread of Standard Deviation would not have much effect on epsilon.

The other option is to consider smaller values of Standard Deviation, as that could affect the epsilon values to a larger extent. Still, we must bear in mind that the purpose of this experiment is to introduce noise via the differential-private mechanism to enforce privacy. The smaller the standard deviation of the noise added, the closer the noise values are to the median, as such making the noise added more predictable, the weights more easily inferred and hence privacy less enforced. Inference attacks could in fact be possible, should the values of Standard Deviation set be too low.

Through these discussions, perhaps varying the other hyperparameters is the only viable solution. Nevertheless, it is imperative to bear in mind to always maintain the range of epsilon, of 1-10, when deciding to experiment with the other hyperparameters.

4.4.4 Differences in Accuracy Drops between Generations

In this subsection, we would be exploring in greater detail the effects of Federated Learning have on accuracy, more specifically, the effects of the aggregation of trained weights, and the disparity of the aggregated weights with the locally trained weights, on the drop in accuracy between Generations. The figures below showcase our analyses between Generation 1 and 2 (i.e. Generation 1 Epoch 3 or 10 with Generation 2 Epoch 1), as well as Generation 2 and 3 (i.e. Generation 2 Epoch 3 or 10 with Generation 3 Epoch 1).

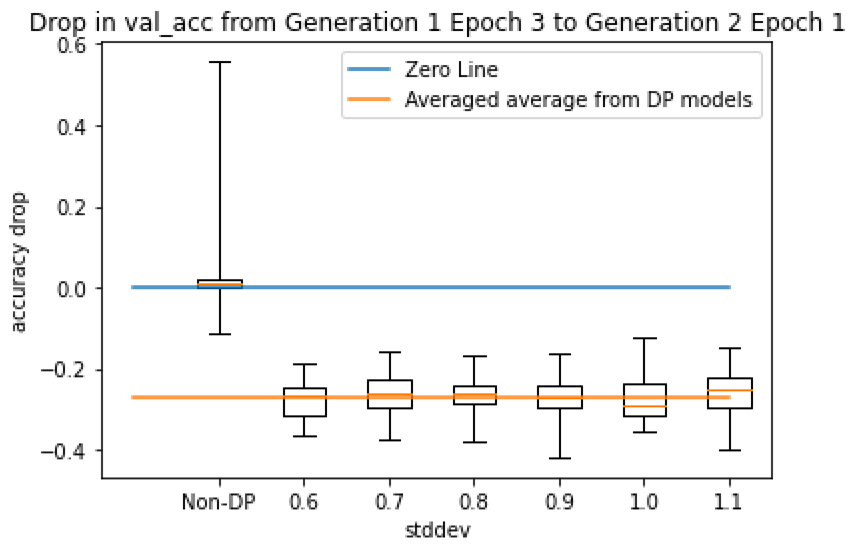


Fig 4.19: Validation Accuracies Drop from Generation 1 Epoch 3 to Generation 2 Epoch 1 for Model 3 epochs per Generation

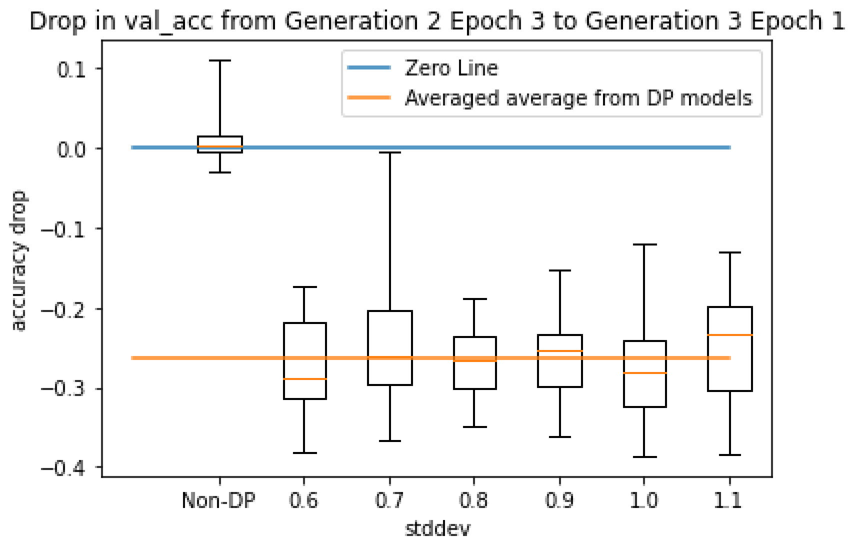


Fig 4.20: Validation Accuracies Drop from Generation 2 Epoch 3 to Generation 3 Epoch 1 for Model 3 epochs per Generation

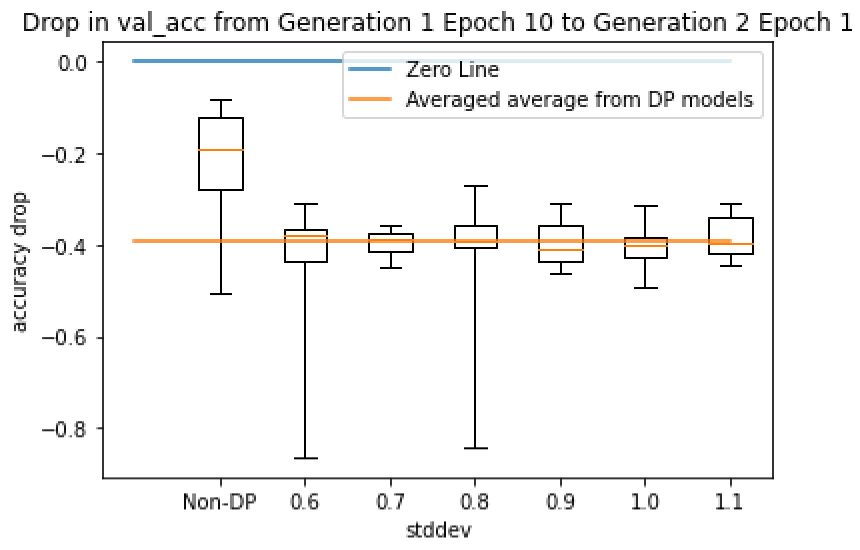


Fig 4.21: Validation Accuracies Drop from Generation 1 Epoch 10 to Generation 2 Epoch 1 for Model 10 epochs per Generation

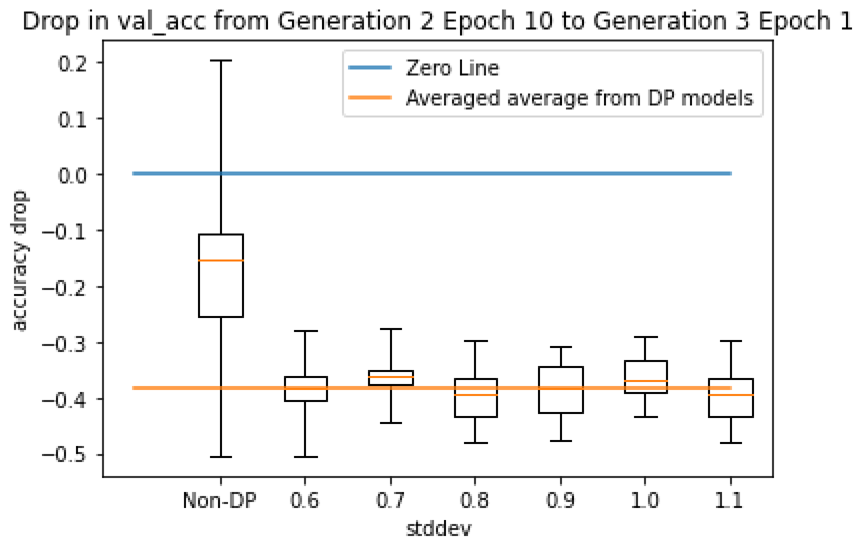


Fig 4.22: Validation Accuracies Drop from Generation 2 Epoch 10 to Generation 3 Epoch 1 for Model 10 epochs per Generation

The diagrams contain boxplots of all of the differentially-private models tested on, as well as a non-differentially-private model per setup for comparison and as a control setup. The Zero Line, which depicts no accuracy drop, is drawn for clarity.

It is apparent that all of the differentially-private models in all scenarios depict a definite drop in accuracy between Generations, without any signs of outliers and non-conforming data points. In comparison, the non-differentially-private models mostly depict all possibilities - a decrease, non-changing and even an increase in accuracy values. One possible explanation as to why the accuracy values between the last epoch of the previous Generation and the first epoch of the current Generation do not have any change, or even increase, could be due to the fact that the accuracy gained from the training done in the first epoch of the current Generation offsets the discrepancies that arise as a result of aggregation, by the Federated Learning mechanism. The additional training done after one single epoch in the differentially-private models is insufficient to counteract the sheer degree of polarity between the aggregate weights and locally trained weights, especially since the local weights from each Client are so different from each other due to random noise added by the differential privacy mechanism.

Interestingly, what we could also infer from the findings is that the number of epochs run per Generation has a slight effect on the accuracy drop between Generation of the models. The models that are trained more, experienced a larger drop in accuracy. If we were to focus on the differentially-private models, the drop in accuracy for the better-trained models experienced approximately a 50% increase, in comparison to the models from the first experiment. To explain this phenomenon, we could argue that the accuracy drop from the aggregation impacted the better-trained models to a larger extent, which in fact, are actually only better-trained local models on local data. In the grand scheme of things, although training with 10 epochs per Generation produces a higher overall accuracy value, than that with only 3 epochs per Generation; In a Federated Learning architecture, the drop in the larger accuracy values produced by each Client at the end of each Generation to the larger accuracy value of the same Client after the first epoch at the start of the next Generation is shown to be larger than that of the poorer-trained models. This suggests that perhaps both the larger drop and smaller drop of accuracy values might land in the same threshold.

And this postulate could actually be proven in Figure and below.

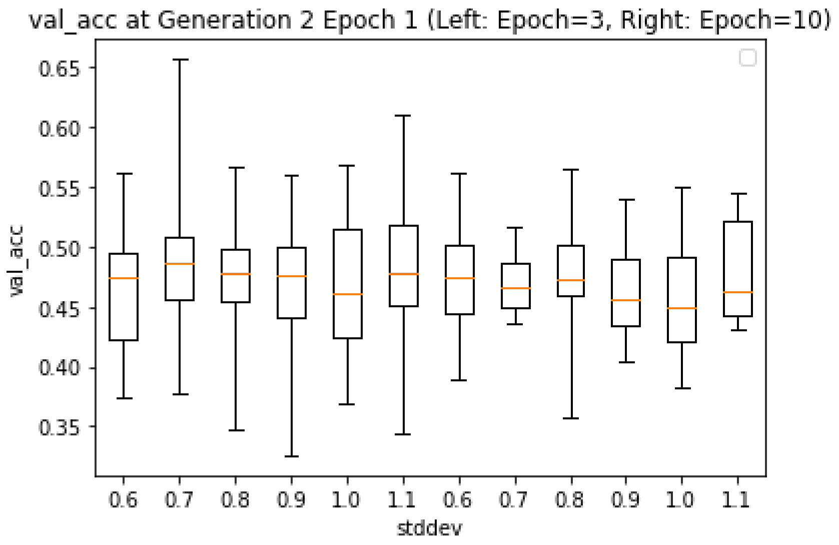


Fig 4.23: Validation Accuracies at Generation 2 Epoch 1 for Models 3 epochs per Generation on the Left & 10 epochs per Generation on the Right

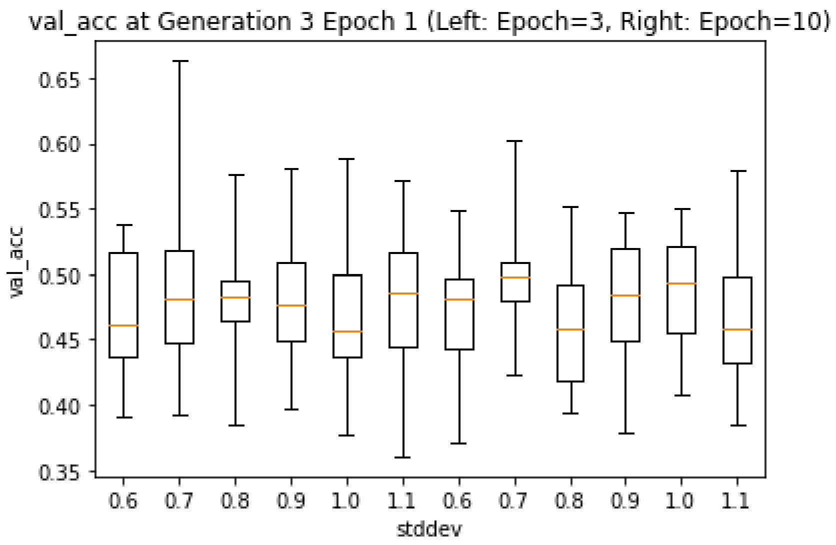


Fig 4.24: Validation Accuracies at Generation 3 Epoch 1 for Models 3 epochs per Generation on the Left & 10 epochs per Generation on the Right

Our analyses thus far show that the accuracy values of the better-trained models at the end of each Generation are higher than that of the poorer-trained models, naturally. The drop to the similar accuracy threshold is then as a result, greater than of the poorer-trained models, as proven.

Amongst the differentially-private models for both values of epochs, we are unable to establish a direct correlation between Standard Deviation and accuracy drops between Generations. We still have to take into account that Standard Deviation affects the degree of randomness of noise added to the weights while training, so hence a larger Standard Deviation should technically create a larger disparity between aggregated and local weights. However, as established in Section 4.4.3, the increase in Standard Deviation does not impact accuracy achieved, at least as shown in our experiments. This could be extrapolated to the accuracy dropped as determined.

Chapter 5

Project Schedule

5.1 AY 2020 Semester 1

Milestones (Sem 1 – Weekly basis)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Discuss overall scope of project	🟩												
Research on CNN/federated learning and differential privacy		🟩	🟩										
Explore CNN models for implementation			🟩	🟩									
Choose a CNN model and implement					🟩								
Incorporate FL into model						🟩	🟩						
Train a model using differential privacy imported from "tensorflow-privacy" library								🟩	🟩	🟩	🟩		
Preparation of interim presentation											🟩	🟩	🟩

Fig 5.1: Gantt Chart of Semester 1.

Semester 1 is defined mostly as an exploratory phase. The crux of our learning processes involve us gaining as much knowledge as we can from our co-supervisor Dr. Sourav, and then doing our own research for additional insights. Not only that, we started on our experimental process in 3 phases:

1. Look for a suitable dataset and implement a basic CNN model.
2. Implementing a Horizontal Federated Learning architecture, expanding on the structure completed in Step 1.
3. Incorporating Differential Privacy into the project in Step 2, and observing the effects of both FL and DP have on accuracy, without much in-depth analyses.

The above-mentioned steps are laid out in chronological order in the Gantt Chart, depicted in Fig 5.1.

5.2 AY 2020 Semester 2

Milestones (Sem 2 – Weekly basis)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Research on how optimizer (DP and non-DP) works internally	🟩	🟩											
Research on statistics behind DP		🟩	🟩										
Draw up experiment structure			🟩	🟩	🟩								
Deep dive into research on technical content			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
Training of models and result analysis				🟩	🟩	🟩	🟩	🟩	🟩	🟩			
Drawing up content for Final Report							🟩	🟩	🟩	🟩			
Preparation for Final Presentation											🟩	🟩	🟩

Figure 5.2: Gantt chart of Semester 2.

Semester 2 is defined by the following processes:

1. Translating things and learnings from Semester 1 into practice;
2. Exploring topics depth-wise to develop a more sophisticated understanding of subject matters; and
3. Designing and executing the experimentation process.

All three are difficult to divorce from each other and often required interdependency between the different processes. Despite the differences between the three, a large common factor stands out between all three - the fact that we had to build upon the foundations set in Semester 1 for all the accumulated work in Semester 2 to be possible at all.

Our team faced a huge time pressure from the start of Semester 2 due to the sheer magnitude of the project as well as the unique challenge by our team composition; our team comprises two (of three) persons from backgrounds and specialisations not related to computer science or engineering.

While Semester 1 was enough for us to explore and form a foundation of our understanding of the different concepts relating to Federated Learning and Differential Privacy, Semester 2 required rigorous understanding of our concepts relative to current literature to better device the internal components of our project, such as the Architecture and relationships of different parameters. All of these culminated in the findings and results of our experiments, and are covered extensively throughout Chapter 4.

> *"If I only had an hour to chop down a tree, I would spend the first 45 minutes sharpening my axe."*
>
> *– Abraham Lincoln.*

As with every large project, we prioritised planning, setting goals, and further strengthening our project foundations in the early stages. This can be observed in our deliberate focus on research and drawing up plans between Week 1 and 10 (see Figure 5.2).

From week 3 on, however, we started putting into action our plans, and implemented the PDCA (plan-do-check-act) cycle for continuous improvement. This allowed us to quickly compare our objectives and evaluate our experiment to determine areas we succeeded in and areas to be improved on. Naturally, this is then translated into actions which got us started on the next improvement iteration.

Throughout the 10-week time period, we were also diligent in consulting Dr. Sourav to capitalise on his expertise and verified our understanding with him.

Milestones (Sem 2 – Weekly basis)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Research on how optimizer works internally	🟩	🟩											
Research on statistics behind dp		🟩	🟩										
Implement a dp optimizer from scratch				🟩	🟩	🟩							
Incorporate dp optimizer into model						🟩	🟩	🟩					
Preparation for final presentation									🟩	🟩	🟩		
Poster and final report preparation												🟩	🟩

Figure 5.3: Outdated Gantt chart of Semester 2.

There was a slight change in focus of our project during our extensive research and experimentation process. Initially, we planned to place our attention on Differential Privacy optimisers and implementing our personalised optimiser from scratch, as shown in the Gantt Chart in Fig 5.3. After intensive discussions with Dr. Sourav, we had come to a conclusion that we should steer the direction of our project, primarily to conduct in-depth analyses between accuracy, with the Federated Learning and Differential Privacy concepts, which are core notions of our project. Nevertheless, heavy emphasis is still given to the research of Differential Privacy optimisers, as such a concept is ultimately relevant to our understanding, and to the crux of our project in the first place.

Chapter 6

Conclusion & Future Developments

6.1 Conclusion

In this paper, we focused on how Federated Learning and Differential Privacy together function as a complementary set of privacy-preserving technology. We visited existing research developments in privacy-preserving Deep Learning applications on structured and unstructured data and designed a proof-of-concept platform for the same, in the form of a CNN for MNIST dataset handwritten digits hosted on the Cloud.

6.1.1 Outlook

Although FL is relatively new, there is strong evidence to suggest a growing interest and attention towards it. We hold the opinion that Federated Learning has a place in collaborative Machine Learning-based applications while preserving the privacy of end-users. This is increasingly important in the era where simultaneously, everything and everyone is progressively connected and there are growing concerns (whether founded or unfounded) about the place of Machine Learning and privacy. Machine Learning has strong potential to be a force for good, and the privacy preservation techniques of Federated Learning and Differential Privacy explored in this paper will help to pave the way for a more harmonious integration of Machine Learning in the life of the everyday layman.

6.1.2 Experiments & Findings

Based on the findings and analyses discussed in Section 4, we have to admit that our experiment data are not entirely able to depict an obvious relationship between Standard Deviation and epsilon. However, this is not to say that we are unable to establish this particular relationship in the first place, and this could be shown from the calculator method we have utilised from Tensorflow Privacy, shown in Section 4.4.1.

We have conducted our experiment run twice, with different epoch values per Generation, to showcase the ineffectiveness of increasing epoch values on affecting the relationship between Standard Deviation and epsilon. Essentially, in our experiment, increasing to 10 epochs per Generation from 3, does not help us in producing the said relationship. We surmise that by changing other hyperparameters, we could potentially produce a clearer result.

Through our experiment, we have come across very interesting insights relating to the main concepts expounded in this paper, i.e. Federated Learning, Differential Privacy and elements of Deep Learning, i.e. accuracy.

First off, we are able to establish the effect of differential privacy on accuracy, namely by adding noise to our trained weights result in an overall decrease in trained accuracy, shown in Section 4.4.3.

Also, we have also provided evidence for the accuracy drops between aggregation and update of weights, i.e. between Generations, as a result of implementing a Federated Learning architecture. This is explained in Section 4.4.4. Not only that, we have compounded the two concepts and showcased the combined effects on the accuracy drop, shown in the same section.

Not only that, by carrying out the experiment again with a different epoch value, we are successful in showing a larger accuracy drop between Generations, also explained in Section 4.4.4. We have also gone on to produce evidence of the accuracy drop to a similar threshold value for all situations, further proving the fact that a larger locally trained accuracy for a larger epoch run presented itself with a larger accuracy drop, as described in the same section.

All in all, though we are unable to produce pronounced evidence of the postulated relationship between Standard Deviation and epsilon, we are however able to learn much more throughout our experiments, discovering new insights and translating them into visual representations.

6.2 Future Developments

As discussed in our concluding summary, Section 6.1, as well as in Section 4.4.3.2, some of the experimentations we could develop further would be to utilise different hyperparameters, while maintaining the acceptable range of epsilon, i.e. 1-10, as expounded in Section 4.4.1, to determined more clearly the relationship between Standard Deviation and epsilon. Perhaps we could consider utilising the entire datasets instead of halving them, as one of the changes to the hyperparameters.

Some of the improvements that we have considered, and that could have been implemented, would be to utilise more efficient models like Recurrent Neural Networks (RNN) and Long-Short-Term Memory, for more efficient training and for faster convergence. Not only that, we have considered using a faster form of differentially-private optimiser, as expounded in this paper [122], which is termed 'JL projections'. However, we believe that this is out of the scope of this project, as this project mainly analyses Federated Learning architectures and Differential

Privacy, in the aspect of affecting accuracy, while taking into account privacy budgets. Speed is therefore not of a concern.

Regarding Federated Learning architectures, perhaps we could explore more and implement Vertical Federated Learning structures, as introduced in Section 2.1.3.2, and Federated Transfer Learning, which has been excluded from discussions within the paper. Going deep into the differences in the various types of Federated Learning architectures, however, is out of the scope of this project. Also, regarding the data distribution amongst the Clients, we have considered having the Clients train their local models on data not received from the centralised pool of data from the Cloud, but with their own related set of data. This way, privacy could be enforced at its core essence, but disallowing even the central server, i.e. the Cloud, from ever knowing, or inferring, the local dataset each Client is training their local model on.

References

1. Anderson, M., “U.S. Technology Device Ownership 2015,” 2020. Available: https://www.pewresearch.org/internet/2015/10/29/technology-device-ownership-2015

2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” 2012. [Online]. Available: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

3. C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594

4. G. Hinton et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82-97, Nov. 2012, doi: 10.1109/MSP.2012.2205597.

5. O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, “Grammar as a Foreign Language,” *arXiv.org*, 09-Jun-2015. [Online]. Available: https://arxiv.org/abs/1412.7449.

6. Y. Sun, D. Liang, X. Wang, and X. Tang, “DeepID3: Face Recognition with Very Deep Neural Networks,” *arXiv.org*, 03-Feb-2015. [Online]. Available: https://arxiv.org/abs/1502.00873.

7. C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, “Move Evaluation in Go Using Deep Convolutional Neural Networks,” *arXiv.org*, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1412.6564.

8. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature News*, 27-Jan-2016. [Online]. Available: https://www.nature.com/articles/nature16961.

9. O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature News*, 30-Oct-2019. [Online]. Available: https://www.nature.com/articles/s41586-019-1724-z.

10. “Dogs vs. Cats,” *Kaggle*. [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats.

11. N. Pinto, D. D. Cox, and J. J. DiCarlo, “Why is Real-World Visual Object Recognition Hard?,” *PLOS Computational Biology*. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371%2Fjournal.pcbi.0040027#:~:text=Visual%20object%20recognition%20is%20an,(e.g.%2C%20%5B1%5D).

12. Sarkar D., Bali R., Sharma T., “Machine Learning Basics. In: Practical Machine Learning with Python. Apress, Berkeley, CA. ,” 2018. Available: https://doi.org/10.1007/978-1-4842-3207-1_1

13. A. Jung, “Machine Learning: Basic Principles,” *arXiv.org*, 21-Oct-2020. [Online]. Available: https://arxiv.org/abs/1805.05052.

14. Ramki Gaddipati @gramki and Ramki Gaddipati @gramki, “How AI and machine learning are changing the banking industry,” *cnbctv18.com*. [Online]. Available: https://www.cnbctv18.com/views/how-ai-and-machine-learning-are-changing-the-banking-industry-7190961.htm.

15. J. Brownlee, “Basic Concepts in Machine Learning,” *Machine Learning Mastery*, 14-Aug-2020. [Online]. Available: https://machinelearningmastery.com/basic-concepts-in-machine-learning/.

16. W. S. Hong, A. D. Haimovich, and R. A. Taylor, “Predicting hospital admission at emergency department triage using machine learning,” *PLOS ONE*. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0201016.

17. N. Kumar, “What is the difference between Machine Learning and Deep Learning,” *Medium*, 18-Oct-2017. [Online]. Available: https://medium.com/@Say2neeraj/what-is-the-difference-between-machine-learning-and-deep-learning-5795e4415be9.

18. G. Leone, “The Best Explanation: Machine Learning vs Deep Learning,” *Medium*, 23-Jun-2017. [Online]. Available: https://medium.com/@GabriellaLeone/the-best-explanation-machine-learning-vs-deep-learning-d5c123405b11#:~:text=Let's%20start%20with%20simple%20definitions,determine%20a%20high%20level%20meaning.

19. “Data Center Solutions, IoT, and PC Innovation,” *Intel*. [Online]. Available: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html.

20. D. Liu, “A Practical Guide to ReLU,” *Medium*, 30-Nov-2017. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7#:~:text=ReLU%20sta

nds%20for%20rectified%20linear,max(0%2C%20x).&text=ReLU%20is%20the%20most%20commonly,usually%20a%20good%20first%20choice.

21. M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep Learning with Differential Privacy,” *arXiv.org*, 24-Oct-2016. [Online]. Available: https://arxiv.org/abs/1607.00133.

22. A. S. V, “Understanding Activation Functions in Neural Networks,” *Medium*, 30-Mar-2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

23. “But what is a Neural Network? | Deep learning, chapter 1,” *YouTube*, 05-Oct-2017. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk&ab_channel=3Blue1Brown.

24. Q. V. Le, J. Ngiam, A. Y. Ng, B. Prochnow, A. Lahiri, and A. Coates, “On Optimization Methods for Deep Learning,” 2011. [Online]. Available: https://icml.cc/2011/papers/210_icmlpaper.pdf.

25. J. 29 and M. Copeland, “The Difference Between AI, Machine Learning, and Deep Learning?: NVIDIA Blog,” *The Official NVIDIA Blog*, 06-Nov-2019. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

26. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” *arXiv.org*, 28-Feb-2017. [Online]. Available: https://arxiv.org/abs/1602.05629.

27. “Macau's ATMs Are Using Facial Recognition to Help Follow the Money,” *Bloomberg.com*. [Online]. Available: https://www.bloomberg.com/news/articles/2017-06-28/macau-atms-need-face-time-before-payout-to-help-follow-the-money.

28. A. Polyakov, “AI and ML Security 101,” *Medium*, 22-Sep-2020. [Online]. Available: https://towardsdatascience.com/ai-and-ml-security-101-6af8026675ff.

29. “Federated Learning: Collaborative Machine Learning without Centralized Training Data,” *Google AI Blog*, 06-Apr-2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

30. S. K. Lo, Q. Lu, L. Zhu, H.-young Paik, X. Xu, and C. Wang, “Architectural Patterns for the Design of Federated Learning Systems,” *arXiv.org*, 07-Jan-2021. [Online]. Available: https://arxiv.org/abs/2101.02373.

31. R. Lee, “Gboard, now available for Android,” *Google*, 16-Dec-2016. [Online]. Available: https://blog.google/products/search/gboard-now-on-android/.

32. A. Dorschel, “Blog: Data Privacy in Machine Learning,” *Luminovo*, 08-May-2020. [Online]. Available: https://luminovo.ai/blog/data-privacy-in-machine-learning.

33. A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, “Protection Against Reconstruction and Its Applications in Private Federated Learning,” *arXiv.org*, 03-Jun-2019. [Online]. Available: https://arxiv.org/abs/1812.00984.

34. Fredrikson M;Lantz E;Jha S;Lin S;Page D;Ristenpart T; “Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing,” *Proceedings of the ... USENIX Security Symposium. UNIX Security Symposium*. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/27077138/.

35. M. F. C. M. University, M. Fredrikson, C. M. University, Somesh Jha University of Wisconsin - Madison, S. Jha, U. of W.- Madison, T. R. C. Tech, T. Ristenpart, C. Tech, C. S. University, P. University, U. of California, “Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures,” *Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures | Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 01-Oct-2015. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2810103.2813677.

36. R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership Inference Attacks against Machine Learning Models,” *arXiv.org*, 31-Mar-2017. [Online]. Available: https://arxiv.org/abs/1610.05820.

37. Y. Liu, D. Dachman-Soled, and A. Srivastava, “Mitigating Reverse Engineering Attacks on Deep Neural Networks,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8839491.

38. M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep Learning with Differential Privacy,” *arXiv.org*, 24-Oct-2016. [Online]. Available: https://arxiv.org/abs/1607.00133.

39. A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," 2008 IEEE Symposium on Security and Privacy (sp 2008), Oakland, CA, USA, 2008, pp. 111-125, doi: 10.1109/SP.2008.33.

40. C. Dwork and A. Roth, “The Algorithmic Foundations of Differential Privacy,” 2014. [Online]. Available: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf.

41. “Privacy and Security Issues in Deep Learning: A Survey,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/9294026.

42. “Enabling developers and organizations to use differential privacy,” *Google Developers Blog*, 05-Sep-2019. [Online]. Available: https://developers.googleblog.com/2019/09/enabling-developers-and-organizations.html.

43. “Differential Privacy,” *Apple*. [Online]. Available: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

44. “How differential privacy enhances Microsoft's privacy and security tools: SmartNoise Early Adopter Acceleration Program Launched,” *Microsoft On the Issues*, 12-Feb-2021. [Online]. Available: https://blogs.microsoft.com/on-the-issues/2020/12/10/differential-privacy-smartnoise-early-a dopter-acceleration-program/.

45. B. C. Nayak and C. Nayak, “New privacy-protected Facebook data for independent research on social media's impact on democracy,” *Facebook Research*, 24-Feb-2020. [Online]. Available: https://research.fb.com/blog/2020/02/new-privacy-protected-facebook-data-for-independent-r esearch-on-social-medias-impact-on-democracy/.

46. P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern J. (eds) Advances in Cryptology — EUROCRYPT ’99.,” *EUROCRYPT 1999. Lecture Notes in Computer Science, vol 1592. Springer, Berlin, Heidelberg.* 1999. https://doi.org/10.1007/3-540-48910-X_16

47. Rivest, Ronald & Shamir, Adi & Adleman, Leonard. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.” *Commun. ACM. 21. 120-126.* 1978. 10.1145/357980.358017.

48. “Privacy-Preserving Outsourced Calculation Toolkit in the Cloud,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/8318625.

49. “What is Secure Multiparty Computation,” *inpher*. [Online]. Available: https://www.inpher.io/technology/what-is-secure-multiparty-computation.

50. “Automatically Detecting Malicious Sensitive Data Usage in Android Applications,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/8488632.

51. Admin, “Three Reasons ECommerce Platforms and Website Builders Should Integrate SSL,” *GlobalSign GMO Internet, Inc.*, 06-Feb-2020. [Online]. Available: https://www.globalsign.com/en/blog/why-integrate-ssl.

52. “Data Privacy Rules Are Changing. How Can Marketers Keep Up?,” *Harvard Business Review*, 01-Feb-2021. [Online]. Available: https://hbr.org/2020/08/data-privacy-rules-are-changing-how-can-marketers-keep-up.

53. Vinod Nair Department of Computer Science, V. Nair, D. of C. Science, Geoffrey E. Hinton Department of Computer Science, G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” *Rectified linear units improve restricted boltzmann machines | Proceedings of the 27th International Conference on International Conference on Machine*

Learning, 01-Jun-2010. [Online]. Available: https://dl.acm.org/doi/10.5555/3104322.3104425.

54. S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way,” *Medium*, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

55. K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, “Towards Federated Learning at Scale: System Design,” *arXiv.org*, 22-Mar-2019. [Online]. Available: https://arxiv.org/abs/1902.01046.

56. S. K. Lo, Q. Lu, L. Zhu, H.-young Paik, X. Xu, and C. Wang, “Architectural Patterns for the Design of Federated Learning Systems,” *arXiv.org*, 07-Jan-2021. [Online]. Available: https://arxiv.org/abs/2101.02373.

57. S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, “A Systematic Literature Review on Federated Machine Learning: From A Software Engineering Perspective,” *arXiv.org*, 04-Dec-2020. [Online]. Available: https://arxiv.org/abs/2007.11354.

58. H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” *arXiv.org*, 28-Feb-2017. [Online]. Available: https://arxiv.org/abs/1602.05629.

59. H. Zhu, H. Zhang, and Y. Jin, “From federated learning to federated neural architecture search: a survey,” *Complex & Intelligent Systems*, 04-Jan-2021. [Online]. Available: https://link.springer.com/article/10.1007/s40747-020-00247-z.

60. S. Yang, B. Ren, X. Zhou, and L. Liu, “Parallel Distributed Logistic Regression for Vertical Federated Learning without Third-Party Coordinator,” *arXiv.org*, 22-Nov-2019. [Online]. Available: https://arxiv.org/abs/1911.09824.

61. *Google*. [Online]. Available: https://federated.withgoogle.com/.

62. “Federated Learning: Collaborative Machine Learning without Centralized Training Data,” *Google AI Blog*, 06-Apr-2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

63. S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, “A Systematic Literature Review on Federated Machine Learning: From A Software Engineering Perspective,” *arXiv.org*, 04-Dec-2020. [Online]. Available: https://arxiv.org/abs/2007.11354.

64. “The Cost of a Cloud: Research Problems in Data Center Networks,” *Microsoft*. [Online]. Available:

https://www.microsoft.com/en-us/research/wp-content/uploads/2009/01/p68-v39n1o-greenberg.pdf.

65. W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *arXiv.org*, 28-Feb-2020. [Online]. Available: https://arxiv.org/abs/1909.11875.

66. C. G. Evans, B. Lambert, and A. Wood, "Lagrangian mean curvature flow with boundary," *arXiv.org*, 12-Nov-2019. [Online]. Available: https://arxiv.org/abs/1911.04977.

67. "Federated learning," *Wikipedia*, 12-Mar-2021. [Online]. Available: https://en.wikipedia.org/wiki/Federated_learning#/media/File:Federated_learning_process_central_case.png.

68. Mingzhe Chen, Zhaohui Yang, Walid S58, Changchuan Yin, H Vincent Poor, and Shuguang Cui. 2019. A joint learning and communications framework for federated learning over wireless networks. arXiv preprint arXiv:1909.07972, 2019.

69. M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu. 2019. Efficient and Privacy-enhanced Federated Learning for Industrial Artificial Intelligence. IEEE Transactions on Industrial Informatics, 2019.

70. C. Pappas, D. Chatzopoulos, S. Lalis, and M. Vavalis, "IPLS : A Framework for Decentralized Federated Learning," *arXiv.org*, 06-Jan-2021. [Online]. Available: https://arxiv.org/abs/2101.01901.

71. Herbert Robbins and Sutton Monro. "A stochastic approximation method. The annals of mathematical statistics (1951)", 400–407, 1951.

72. J. Mills, J. Hu, and G. Min, "User-Oriented Multi-Task Federated Deep Learning for Mobile Edge Computing," *arXiv.org*, 17-Jul-2020. [Online]. Available: https://arxiv.org/abs/2007.09236.

73. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, 72on Segal, and Karn Seth. "Practical Secure Aggregation for Privacy-Preserving Machine Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.," *Association for Computing Machinery, Dallas, Texas, USA, 1175–1191*. 2017.

74. T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L. Wang. "Efficient Training Management for Mobile CrowdMachine Learning: A Deep Reinforcement Learning Approach." *IEEE Wireless Communications Letters 8, 5.* 2019.

75. Lo, S.K., Lu, Q., Wang, C., Paik, H.Y., Zhu, L., “A systematic literature review on federated machine learning: From a software engineering perspective.,” *accepted by ACM Comput. Surv., minor revision.* 2021.

76. Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated Learning with Non-IID Data,” *arXiv.org*, 02-Jun-2018. [Online]. Available: https://arxiv.org/abs/1806.00582.

77. R. Doku, D. B. Rawat, and C. Liu., “Towards Federated Learning Approach to Determine Data Relevance in Big Data.,” *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. 2019.

78. Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. “A Communication Efficient Vertical Federated Learning Framework.,” *arXiv preprint arXiv:1912.11187.* 2019.

79. McDonald R, Hall K, Mann G, “Distributed training strategies for the structured perceptron.” *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics, 456–464.* 2010.

80. Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated Machine Learning: Concept and Applications,” *arXiv.org*, 13-Feb-2019. [Online]. Available: https://arxiv.org/abs/1902.04885.

81. Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, Ramage D, Segal A, Seth K. “Practical secure aggregation for privacy preserving machine learning.,” *Cryptology ePrint Archive, Report 2017/281,* 2017. Available: https://eprint.iacr.org/2017/281

82. Dwork C., “Differential privacy: a survey of results.,” *International conference on theory and applications of models of computation. Springer, 1–19.* 2008.

83. Shokri R, Shmatikov V. “Privacy-preserving deep learning.,” *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 1310–1321.* 2015.

84. Yang S, Ren B, Zhou X, Liu L., “Parallel distributed logistic regression for vertical federated learning without third-party coordinator.,” *arXiv preprint arXiv:1911.09824.* 2019.

85. Liu Y, Kang Y, Xing C, Chen T, Yang Q, “A secure federated transfer learning framework.” *IEEE Intell Syst.* 2020.

86. Gang Liang and Sudarshan S Chawathe. “Privacy-preserving inter-database operations.,” *International Conference on Intelligence and Security Informatics. Springer, 66–82.* 2004

87. O. Goldreich, S. Micali, and A. Wigderson. “How to Play ANY Mental Game.” *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC ’87). ACM, New York, NY, USA, 218–229. https://doi.org/10. 1145/28395.28420*. 1987

88. S. Vadhan, *The Complexity of Differential Privacy*. [Online]. Available: https://salil.seas.harvard.edu/publications/complexity-differential-privacy.

89. Dwork, Cynthia., "A firm foundation for private data analysis." *Communications of the ACM 54.1 (2011): 86–95.* 2011

90. Bambauer, Jane, Krishnamurty Muralidhar, and Rathindra Sarathy. "Fool's gold: an illustrated critique of differential privacy." *Vand. J. Ent. & Tech. L. 16 (2013): 701.* 2013.

91. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. "Calibrating noise to sensitivity in private data analysis." *Journal of Privacy and Confidentiality, 7(3), 2016. To appear. Preliminary version in Proc. TCC '06*. 2016.

92. Dwork, Cynthia; Roth. "The Algorithmic Foundations of Differential Privacy.," *Foundations and Trends in Theoretical Computer Science. 9 (3–4): 211–407. doi:10.1561/0400000042. ISSN 1551-305X.* 2013.

93. K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farhad, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated Learning with Differential Privacy: Algorithms and Performance Analysis," *arXiv.org*, 08-Nov-2019. [Online]. Available: https://arxiv.org/abs/1911.00222.

94. T. M. AIST, T. Murakami, Aist, A. I. S. T. V. Profile, Y. K. AIST, Y. Kawamoto, U. of Pennsylvania, U. of Florida, "Utility-optimized local differential privacy mechanisms for distribution estimation," *Utility-optimized local differential privacy mechanisms for distribution estimation | Proceedings of the 28th USENIX Conference on Security Symposium*, 01-Aug-2019. [Online]. Available: https://dl.acm.org/doi/10.5555/3361338.3361468.

95. "Data Breaches Increase 40 Percent in 2016, Finds New Report from Identity Theft Resource Center and CyberScout." Available: http://www.idtheftcenter.org/2016databreaches.html, 2017.

96."File:Gaussian mechanism.png," 19-Apr-2019. [Online]. Available: https://en.wikipedia.org/w/index.php?curid=60513211.

97. S. Asseffa and B. Seleshi, "A Case Study on Differential Privacy," *DIVA*, 22-Jun-2017. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1113852&dswid=5687.

98. A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," *IEEE Xplore*, 18-May-2008. [Online]. Available: https://ieeexplore.ieee.org/document/4531148?reload=true.

99."The Netflix Prize Rules." [Online]. Available: https://www.netflixprize.com/assets/rules.pdf. [Accessed: 27-Mar-2021].

100. N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, “Privacy Preservation in Federated Learning: An insightful survey from the GDPR Perspective,” *arXiv.org*, 18-Mar-2021. [Online]. Available: https://arxiv.org/abs/2011.05411.

101. Kelvin, “Introduction to Federated Learning and Challenges,” *Medium*, 22-Nov-2020. [Online]. Available: https://towardsdatascience.com/introduction-to-federated-learning-and-challenges-ea7e02f260ca.

102. C. Fung, C. J. M. Yoon, and I. Beschastnikh, “Mitigating Sybils in Federated Learning Poisoning,” arXiv.org, p. arXiv:1808.04866, Aug. 2018.

103. C. Briggs, Z. Fan, and P. Andras, “A Review of Privacy-preserving Federated Learning for the Internet-of-Things,” *arXiv.org*, 08-Sep-2020. [Online]. Available: https://arxiv.org/abs/2004.11794.

104. D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv.org*, 30-Jan-2017. [Online]. Available: https://arxiv.org/abs/1412.6980.

105. F. Berhane, “Optimization Methods,” *datascience-enthusiast*. [Online]. Available: https://datascience-enthusiast.com/DL/Optimization_methods.html.

106. “tf.keras.optimizers.Adam : TensorFlow Core v2.4.1,” *TensorFlow*, 21-Feb-2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam.

107. “THE MNIST DATABASE,” MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

108. “FWRITR - NIST.” [Online]. Available: https://www.nist.gov/system/files/documents/srd/nistsd19.pdf.

109. A. Panchal, “Improving accuracy on MNIST using Data Augmentation,” *Medium*, 08-Feb-2021. [Online]. Available: https://towardsdatascience.com/improving-accuracy-on-mnist-using-data-augmentation-b5c38eb5a903.

110. “Module: tf.keras.datasets.mnist : TensorFlow Core v2.4.1,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist.

111. T. D. Detective, “Finally: Why We Use an 80/20 Split for Training and Test Data Plus an Alternative Method (Oh Yes...),” *Medium*, 31-Jan-2020. [Online]. Available: https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d.

112. “Module: tf.keras : TensorFlow Core v2.4.1,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras.

113. “tf.compat.v1.train.GradientDescentOptimizer : TensorFlow Core v2.4.1,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer.

114. Tensorflow, “tensorflow/privacy,” *GitHub*. [Online]. Available: https://github.com/tensorflow/privacy/blob/a579cc4afcfa0420fef37841aedf91d22b3eef48/tensorflow_privacy/privacy/optimizers/dp_optimizer.py.

115. DeepAI, “Epoch,” *DeepAI*, 17-May-2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/epoch.

116. Tensorflow, “tensorflow/privacy,” *GitHub*, 18-Jul-2020. [Online]. Available: https://github.com/tensorflow/privacy/blob/master/tutorials/mnist_dpsgd_tutorial.py.

117. Tensorflow, “tensorflow/privacy,” *GitHub*. [Online]. Available: https://github.com/tensorflow/privacy/blob/master/tensorflow_privacy/privacy/analysis/compute_dp_sgd_privacy.py.

118. Tensorflow, “tensorflow/privacy,” *GitHub*, 30-Dec-2020. [Online]. Available: https://github.com/tensorflow/privacy/blob/master/tensorflow_privacy/privacy/analysis/rdp_accountant.py.

119. I. Mironov, “Renyi Differential Privacy,” *arXiv.org*, 25-Aug-2017. [Online]. Available: https://arxiv.org/abs/1702.07476.

120. “What are good epsilon (ε) and delta (δ) values in differential privacy?,” *Gretel*. [Online]. Available: https://gretel.ai/gretel-synthetics-faqs/what-are-good-epsilon-e-and-delta-d-values-in-differential-privacy.

121. M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep Learning with Differential Privacy,” *arXiv.org*, 24-Oct-2016. [Online]. Available: https://arxiv.org/abs/1607.00133.

122. “FAST DIFFERENTIALLY PRIVATE-SGD VIA JL PROJECTIONS,” *OpenReview*, 05-Mar-2021. [Online]. Available: https://openreview.net/forum?id=0Jr4rjA6glk.