Institute of Information Security

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Efficient Federated Learning for Gaze Estimation

Jonas Kittelberger

Course of Study:	Informatik
Examiner:	Prof. Dr. Ralf Küsters
Supervisor:	Mayar Elfares, M.Sc.
Commenced:	April 24, 2023
Completed:	October 24, 2023

Abstract

Gaze estimation is the task of deciding for given face images, in which direction people are looking. It is particularly useful for various applications including psychological analysis, authentication, and eye tracking in the context of virtual or augmented reality. To reduce the error of the predictions of gaze estimators, the training data should be collected from a large number of users to ensure the ability of the model to generalize correctly during the inference phase. However, the large data collection requirements conflict with privacy concerns. Building on existing federated learning approaches, this project aims to increase the efficiency of the training process. Hence, (i) we split the model into a part owned by the client and another part owned by a server. This results in strong data protection properties as well as model privacy. In addition, only a part of the model has to be stored and run by each client leading to decreasing computational effort for the typically substantially resource-constrained clients. (ii) We further train the gaze estimation model in an unsupervised fashion and (iii) prune the model weights to enhance the training efficiency. Furthermore, we extend our approach with several privacy-preserving techniques, e.g. Multi-Party Computation (MPC) and Differential Privacy (DP) mechanisms. We empirically demonstrate the effectiveness of these mechanisms with an implemented attack on our system. Our experiments show that our implemented system manages to predict gaze angles with an average deviation of less than 6.5 degrees from the actual angle in about 10 minutes and thus outperforms other privacy-preserving gaze estimators.

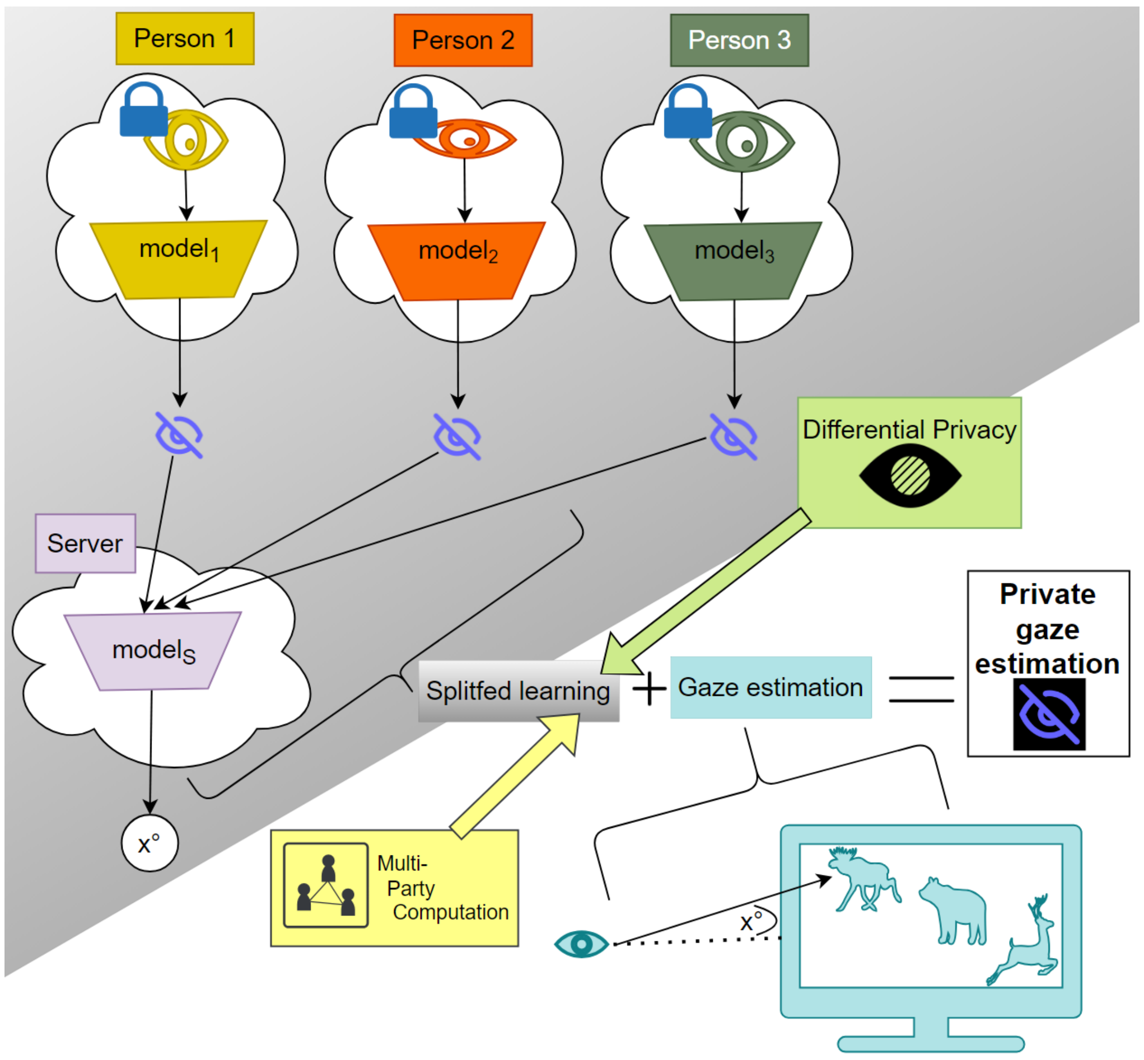


Figure 0.1: Graphical Abstract

Contents

List of Figures

List of Tables

Acronyms

CNN	Convolutional Neural Network. 13
DP	Differential Privacy. 14
DP-SGD	Differentially-Private Stochastic Gradient Descent. 26
FedAvg	Federated averaging of model updates. 13
FL	Federated Learning. 13
MPC	Multi-Party Computation. 14
SFL	Splitfed Learning. 14
SL	Split Learning. 13

1 Introduction

Since the line of sight and eye movements indicate human attention and cognition [2], gaze information can be utilized in various ways, e.g. in VR and AR as frequently encountered applications [25].

Besides the great number of useful applications, accurate learning-based gaze estimation is based on the fact that personal data from a large number of different people is available. In terms of privacy, the demand for protection of the individual highly sensitive face and eye images leads to a utility-privacy trade-off [2]. To comply with the mentioned requirements, we need dedicated privacy-preserving machine learning techniques that protect personal images while maintaining a high gaze estimation performance. We aim for only slightly increased gaze prediction errors compared to the case where no data protection techniques are applied.

Federated Learning (FL) is a common privacy-preserving machine learning approach that is suitable in our case where training data is not owned centrally [20]. Instead, multiple clients own private data and all data should be used for collaborative training of a model such as a Convolutional Neural Network (CNN) for gaze estimation. In one iteration of FL, each client obtains the current model from the server, trains the model for a given number of epochs on their local data and passes back their locally trained model to the server. The server aggregates the received local model parts (in a CNN: weights or gradients) by performing the Federated averaging of model updates (FedAvg), i.e. the server takes the element-wise average over the weights of the clients. Subsequently, the server could apply additional optimization methods as performed in Adaptive FL [7]. The server passes the averaged weights to the clients and the clients use them as instantiation for their next iteration. The whole process is repeated for a certain number of iterations or until a given desired accuracy is reached.

As pointed out in [7], the application of FL is useful if the data distribution is non-independent and identical (non-IID). This is definitely the case for gaze data where each client owns personal face or eye images. To perform supervised learning, the images need to be annotated with the desired output of the gaze estimator, namely the gaze angles. We use the MPIIGaze dataset [33] where face images and respective gaze angles were collected in a study and published for scientific use. Further information about the dataset is given in Section 2.1.1 and a small excerpt of the face images is shown in Figure 2.1.

Split Learning (SL) is another approach that secures personal data by performing training locally on each client. In each epoch, instead of training the whole network as in FL, the client only forwards the input until a certain layer (i.e. the split layer). The client passes the output of the split layer (i.e. the smashed data) to the main server that completes the forward propagation with its part of the network. The server starts the backpropagation until the split layer and passes the gradients to the client that completes the backpropagation.

Compared to FL, SL can keep a part of the model private from the clients (i.e. model privacy). In addition, a part of the forward- and backpropagation is outsourced to the main server, which decreases the computational effort as well as storage requirements of the typically resource-limited clients (e.g. mobile devices). SL is also applicable in the case of servers that own a repository containing multiple models that are implemented using the same input features. In this case, the same data obtained from the clients can be re-used to save communication and computational effort.

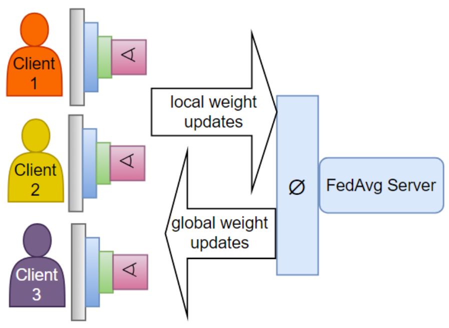
In this work, we apply Splitfed Learning (SFL) which is a combination of SL and FL as illustrated in Figure 1.1. With SFL, we aim to preserve the privacy of the gaze data and achieve model privacy as in SL and allow clients to execute local training epochs in parallel [27]. We improve the implemented models to minimize the error of gaze predictions on the MPIIGaze dataset and show how model pruning can lead to a more compact model representation and potentially reduced time for forwarding data samples through the network. In addition, we integrate Multi-Party Computation (MPC) techniques to obtain further privacy properties by keeping the forwarded input samples secret as well as allowing the weights of the clients and the server to be kept secret. We also use MPC techniques to carry out the computation of the FedAvg without revealing the weights of a single client. In addition, we apply Differential Privacy (DP) mechanisms to obfuscate model and smashed data and aim to avoid possible attacks that reconstruct the personal input images. Furthermore, we implement unsupervised gaze estimation, i.e. gaze estimator training without or with just a few gaze angle annotations, and we combine this with FL to preserve the privacy of client data.

In later comparisons with alternative learning methods, we prove that our implementation offers highly accurate gaze predictions. In addition, we give empirical evidence for the privacy properties of our systems with our own attacker implementation where we try to reconstruct the secret client images.

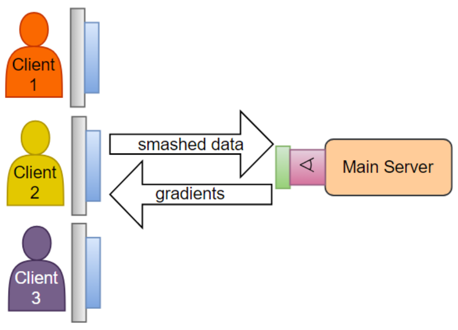
To the best of our knowledge, this project is the first work to apply and implement SFL on gaze estimation and constitutes a first step towards MPC-friendly ML training.

Our work is structured as follows:

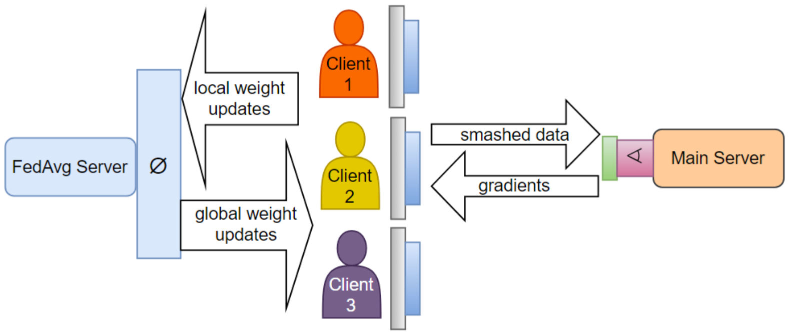
- Chapter 2 (Background) gives essential background information about gaze estimation and security measures (SFL, MPC and DP).
- In Chapter 3 (Methods), we explain the implementation and the required components of our system to perform SFL for privacy-preserving gaze estimation. We also present our own attacker implementation to empirically demonstrate the effectiveness of the privacy-preserving mechanisms.
- Chapter 4 (Experiments) presents the results to show the performance of our system. We compare our results (accuracies and runtimes) to results of other baseline methods.
- In Chapter 5 (Discussion), we reflect on the utility and privacy guarantees of our system.
- Chapter 6 (Related Work) presents an overview of the related methods for privacy-preserving gaze estimation in the literature.
- Chapter 7 (Conclusion and Outlook) highlights our main contribution and findings as well as possible future directions.



(a) FL



(b) SL



(c) SFL

Figure 1.1: Visualization of FL, SL and SFL. The model here consists of 4 layers shown with adjacent grey-blue-green-pink bars and a layer is owned by the client or main server next to it

2 Background

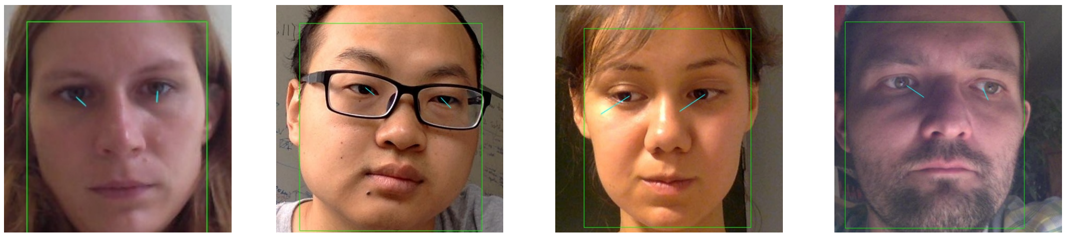
In this chapter, we introduce the gaze estimation task and the required components to perform accurate gaze estimation. We also provide existing techniques to preserve the privacy of clients and to protect their secret data.

2.1 Gaze estimation

Gaze estimation is the task of predicting the viewing direction of a given person as a gaze angle. This can be performed based on images, videos or live recordings to, for example, infer gaze information of users playing video games [25]. We first present our utilized dataset, then we show how a CNN is structured and which components it consists of so that we can use it as a gaze estimation model. Afterwards, we also describe existing approaches to learning gaze representations in an unsupervised fashion.

2.1.1 Gaze estimation dataset

The MPIIGaze dataset is a publicly available set of 213,659 images from 15 participants [33] that can be identified with their id (P00, P01, ..., P14). In the further course, we limit to a subset of 3000 images per participant which corresponds to the selected subset from Zhang et al. [33]. The setting for MPIIGaze is also called "in the wild" because it depicts the daily life recordings of participants in front of their personal devices. The dataset covers the great diversity of appearances, i.e. different races, genders and with or without glasses. All these are reasons why performing gaze estimation is a great challenge for this dataset. We mainly focus on the prediction of 2D gaze angles (yaw and pitch) exactly as in [33]. To provide a realistic evaluation of gaze estimation models, model training is done on 14 participants and the error is measured on the left-out participant. Therefore, the model has to infer the 2D gaze angles for a participant that was not considered during the training phase.



(a) Participant, ID 11 **(b)** Participant, ID 01 **(c)** Participant, ID 14 **(d)** Participant, ID 03

Figure 2.1: An excerpt of face images from the MPIIGaze dataset and visualization of the inferred gaze angles predicted by a trained model (shown with turquoise lines)

Zhang et al. [33] provide a pre-processing phase to transform face images into pre-processed eye images that are passed to the gaze estimator (a CNN) subsequently. In a nutshell, this pre-processing phase consists of detecting the face and its facial landmarks to create a facial shape model. Then normalization into a polar-coordinate angle space is applied, which results in a set of fixed-resolution 2D eye images. In addition to each image, a 2D head angle vector (also called: head pose) is obtained that can also be utilized by the model to predict the gaze angles. Figure 2.1 visualizes the output angles after applying the trained model from Zhang et al. [33] on four face images from the MPIIGaze dataset.

2.1.2 Gaze estimation model

The appearance of the shape of an eye is strongly dependent on the angle at which the eye is being viewed. This makes gaze estimation a difficult task and we need a model that is expressive enough to handle high variability in the appearance of the eyes [12]. Model-based methods first extract features (e.g. pupil centre and iris edges) and pass these features to the gaze estimation model [7, 33]. In contrast to this are the appearance-based methods that use the eye images directly as input and this has proven to handle images with varying quality and lighting conditions better and give predictions with higher accuracy [33].

Appearance-based methods require, besides a sufficiently large dataset for training, a model that is able to extract useful features from an image with potentially high dimensionality (many pixels) and that leads to the usage of a CNN as the model for appearance-based gaze estimation. A CNN is a network consisting of several specific layers that allow it to process multi-dimensional inputs such as 2D images. They usually consist of three types of layers: Convolutional layers, pooling layers and fully connected layers [22].

When forwarding an input image through the CNN, convolutional layers and pooling layers are typically applied in the beginning and should allow the CNN to reduce the input dimensionality and extract necessary features (depending on the task to be solved). These layers produce a gradually increasing number of 2D outputs of decreasing size (c_1 and c_2 in Figure 3.1). In a nutshell, the outputs of a convolutional layer (called feature maps or channels) are calculated by multiplying the input tensor with the weight matrices of the CNN that are called filters. Pooling layers perform downsampling of the given feature map, e.g. by taking the maximum value (max-pooling) from the values in a sliding window which moves over the input feature map.

After the input tensor passes convolutional and pooling layers, the tensor is flattened into a one-dimensional tensor and is passed to (usually around two) subsequent linear layers. For linear layers, each output element (neuron) is a linear combination of the whole input tensor and therefore they are also referred to as "fully connected layers" as each input neuron is connected to each output neuron by one weight from the weight matrix [22]. For the exact functionality of the layers, visualizations of the architecture and further utilized type of layers, we refer to [22].

The output of the last layer is the prediction of the CNN to the task that should be solved. This can be a classification task where membership to a certain class is checked, e.g. whether a dog or a cat is contained on a given input image. Another task is to predict numeric values (regression) and this is the case for gaze estimation when predicting continuous gaze angles [12].

To exploit speedup for parallelization, multiple inputs (one batch) are forwarded at once through the CNN. In the inference phase, the inputs are applied to an already trained model and the outputs are the predictions of the model for a given input, e.g. the estimated gazes in eye tracking applications. If the expected outputs (targets) are available as is the case for the MPIIGaze dataset, then they can be compared to the predicted outputs to calculate the error of the predictions and assess the performance of the trained model. To also get an estimate of how well a model can generalize, the inputs that were used to train the model have to be avoided for inference. For gaze estimation, this can be done by using a participant who was not considered during the training phase to compute the differences between predicted and expected angles (angle error).

In the training phase, the model should be improved iteratively to reduce the prediction errors. For each iteration (epoch), the training data is divided into batches that are forwarded successively through the network. For each forwarded batch, a loss function (often: mean squared error, [20]) is computed to measure the difference between predictions and targets. Then, the backpropagation phase is executed where the error is passed backward through the network in order to update the weights so that later outputs are pulled closer to the expected outputs. The weights are updated in the direction of the calculated gradients and the magnitude of the update is determined by a user-defined learning rate. During ongoing training, the learning rate is often decreased to allow for major updates in the beginning and more fine-tuning in the end. Using the popular machine learning framework Pytorch for python, the weights, learning rate and further training parameters are passed to an optimizer that takes care of the correct calculation of gradients and updates of the weights.

The presented way of training is known as supervised learning which means that targets are available for all input data and the targets are used during model training. On the other hand, there is unsupervised learning where training is performed with the absence of dedicated target outputs and we go into more detail about this now.

2.1.3 Unsupervised gaze representation learning

Collecting and annotating gaze data is complex and expensive [32]. It requires complex experimental setups and exact measurements. Measurement errors are made during the experiments by participants blinking or getting distracted. Also, the target angle for most datasets is computed visually which leads to noisy values [32].

The goal of unsupervised learning is to train a model even if no or only a few labels are available, i.e. with missing gaze angle annotations in case of a gaze estimation task. The observed neural networks for unsupervised learning, such as in [26, 32], consist of an encoder and a decoder. The simplified structure is shown in Figure 2.2. To train the Encoder-Decoder model, the input eye image is passed to the encoder which transforms it into a reduced and compact form that is called gaze representation [26, 32]. The decoder uses the input eye images as targets and is therefore trained to reconstruct the eyes from given gaze representations.

Encoder-Decoder training aims to obtain gaze representations that are as accurate as possible to allow the gaze to be predicted solely based on a given gaze representation and, depending on the approach, possibly a given head pose. A lightweight regressor (gaze estimator) consisting of only a few (usually linear) layers is trained to transform gaze representations into gaze angles. Target angle annotations are required to train this gaze estimator but it is trained only on a small amount of data (e.g. 100 samples). This is called few-shot gaze estimation [26, 32] or also n-shot gaze

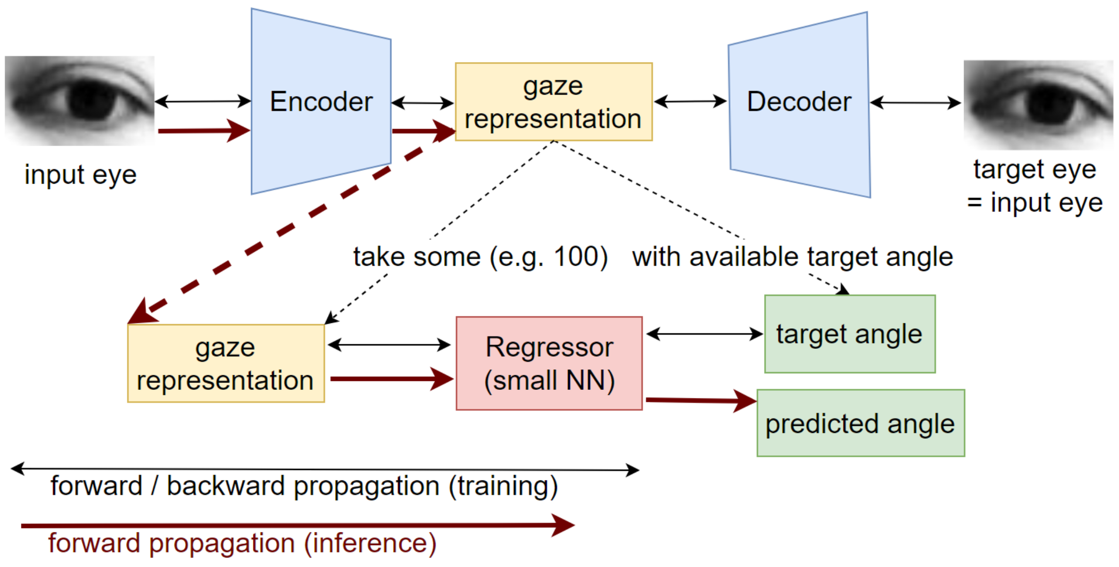


Figure 2.2: Training and inference using unsupervised gaze representation learning combined with few-shot gaze estimation

estimation when considering the exact amount of n samples with target annotation. Altogether, the Encoder-Decoder is trained on a large amount of data without gaze annotations and subsequently, only a small amount of annotated data is needed to train the gaze estimator.

In the further course, we focus on the implementation of Sun et al. [26] which the authors applied and tested, among others, on the MPIIGaze dataset. In their implementation, they perform 3D gaze estimation and therefore, in contrast to 2D gaze estimation, they consider one more gaze coordinate for the gaze predictions. This coordinate includes gaze depth information [16] that is, related to captured photographs, depending on the distance between the person and the recording device. However, the values from this coordinate differ little from each other for the MPIIGaze dataset which is related to the fact that pictured participants look at their personal laptop screen during common workday use on all images. In the following, we adopt the respective number of dimensions and perform 2D gaze estimation for supervised learning approaches and 3D gaze estimation for unsupervised learning to allow for a comparison with the underlying work in both cases.

Sun et al. [26] utilize a Cross-Encoder that encodes and decodes two images simultaneously. It once combines the encodings of two eye images that belong to the same eye from two different pictures (same eye, different gaze) as well as once an image of left and right eye from the same picture (different eye, similar gaze). The learned gaze representation consists of a gaze feature and an eye feature. The eye feature should be similar for images with (same eye, different gaze) while the gaze feature should be similar for (different eye, similar gaze) [26]. The authors show in their experiments, that the best gaze estimation results are obtained when passing only the gaze feature as input to the gaze estimator. Nevertheless, the training of the Cross-Encoder is influenced by both the fixed gaze feature dimension d_g as well as eye feature dimension d_e, which leads to different angle errors in [26] depending on the chosen dimensions. In contrast, the influence of the head pose is low as there are only slight differences in the results for MPIIGaze with and without considering the head pose as additional input to the gaze estimator.

For further details regarding the exact configuration of the Cross-Encoder network, we refer to [26]. In their experiments, the authors show that the gaze representation from the Cross-Encoder leads to lower gaze estimation errors compared to gaze representations from alternative unsupervised gaze representation learning methods.

2.2 Security measures

In this section, we introduce the concepts to protect the privacy of clients during training of the CNN. First, we show how SFL works, then we introduce MPC and DP that we utilize later to enhance the privacy guarantees.

2.2.1 Splitfed Learning

SFL, just like FL or SL, is an approach where multiple clients own private data that should be used to collaboratively train a model while the personal data should be kept secret. SL and SFL utilize the idea to outsource a part of the model (here: a CNN) from the client to a server (main server) that relieves the typically resource-constrained clients and is taking off part of the calculation [20].

The given model (CNN) gets divided into two parts. The part containing the input is owned by the clients and as each client owns different input data, their network weights differ after updating the weights using backpropagation. We call the client-side model the local client model, that can be accessed by the respective client. We now summarize the SFL training procedure as described in detail by Thapa et al. [27].

In parallel, each client forwards their input data through their local client model until the last layer (split layer). When a client is done with it, the client passes the output (called smashed data) as well as the overall targets (for gaze estimation: gaze angles) to the main server that finishes the forward propagation using its part of the model (server model). The main server starts the backpropagation until the split layer and sends the gradients back to the client that finishes the backpropagation and updates the weights of the local client model. This process is repeated for a certain number (local epochs) that we set to 1 as default in the following.

After all clients finish their local epoch(s), the server takes the weights that it computed for each client in its last local epoch, computes their averages (using FedAvg locally) and takes the averaged weights to update the server model. Note that during the local epochs, the server model gets updated while performing backpropagation. To avoid clients hindering each other during their local epochs, the main server holds one copy of the server model for each client and therefore, the server can handle client requests in parallel and there is no need for synchronization until the server performs FedAvg. This is the difference to the original Vanilla SL where a client has to wait for a previous client to finish its computation and they pass the updated weights to each other which leads to reduced parallelization [20].

Algorithm 2.1 Splitfed Learning

```
$n \leftarrow$ number-of-clients
$[M_0, M_1, \dots, M_{n-1}] \leftarrow [\emptyset, \emptyset, \dots, \emptyset]$    // Local client model weights, owned by each client
$[data_0, data_1, \dots, data_{n-1}] \leftarrow$ PREPROCESSIMAGES()    // $data_k$ is owned by client k
$M_S \leftarrow \emptyset$    // Server model weights, owned by the server
procedure MAIN($e_l$, $e_g$)    // Execute $e_l$ local epochs and $e_g$ global epochs of SFL
    RUN_THREAD(MainServer($e_l$))    // Run MainServer($e_l$) in another thread (non-blocking!)
    RUN_THREAD(Aggregator())    // Run Aggregator() in another thread (non-blocking!)
    $M_0, M_1, \dots, M_{n-1}, M_S \leftarrow$ INITWEIGHTS()
    for $e_g$ times do
        for each $k \in \{0, \dots, n-1\}$ do    // One iteration per client $k$, iterations run in parallel!
            TRAINCLIENT($k$, $e_l$)
        end for
    end for
    END_THREAD(MainServer($e_l$))    // End the MainServer thread
    END_THREAD(Aggregator())    // End the Aggregator thread
    return $M_0, M_S$    // Note: $M_0 = M_1 = \cdots = M_{n-1}$ (cause FedAvg is done in the end)
end procedure

procedure TRAINCLIENT($k$, $e_l$)    // Executed by client k
    for $e_l$ times do    // $e_l$ is the number of local epochs
        $smash \leftarrow$ FORWARD($M_k$, $data_k$)
        SEND($S$, $smash$)    // Pass smashed data to main server
        $smash\_gradients \leftarrow$ RECEIVE($S$)    // Wait for gradients from main server
        BACKPROP($smash\_gradients$)    // Backpropagation includes updating the weights $M_k$
    end for
    SEND($A$, $M_k$)    // Send updated weights to Aggregator
    $M_k \leftarrow$ RECEIVE($A$)    // Set weights to the average received from Aggregator
end procedure

procedure MAINSERVER($e_l$)
    $[S_0, S_1, \dots, S_{n-1}] \leftarrow [M_S, M_S, \dots, M_S]$
    for each $k \in \{0, \dots, n-1\}$ do    // One iteration per client $k$, iterations run in parallel!
        for $e_l$ times do    // $e_l$ is the number of local epochs
            $smash, target \leftarrow$ RECEIVE($k$)    // Wait for smashed data and target from client $k$
            FORWARD($S_k$, $smash$)
            $smash\_gradients \leftarrow$ BACKPROP($S_k$, $target$)
            SEND($k$, $smash\_gradients$)    // Send back gradients to client $k$
        end for
    end for
    $M_S \leftarrow$ FEDAVG($[S_0, S_1, \dots, S_{n-1}]$)    // internal FedAvg to update server-side model
    MAINSERVER($e_l$)    // jump back to the beginning of this procedure
end procedure
```

```
procedure AGGREGATOR()
    $[W_0, W_1, \ldots, W_{n-1}] \leftarrow [\emptyset, \emptyset, \ldots, \emptyset]$
    for each $k \in \{0, \ldots, n-1\}$ do
        $W_k \leftarrow$ RECEIVE($k$)    // Wait for model weights from client $k$
    end for
    $W \leftarrow$ FEDAVG($[W_0, W_1, \ldots, W_{n-1}]$)    // FedAvg to update client-side model
    for each $k \in \{0, \ldots, n-1\}$ do
        SEND($k$, $W$)    // Send averaged weights to all clients
    end for
    AGGREGATOR()    // jump back to the beginning of this procedure
end procedure

function FEDAVG($W_{all}$)    // $W_{all}$ is a 2D list (list of list of weights)
    assert $len(W_{all}) = n$    // length of $W_{all}$ equals number of clients
    assert $len(W_{all}[0]) = \cdots = len(W_{all}[n-1])$    // equal number of weights per client
    $W_{avg} \leftarrow W_{all}[0]$
    for each $i \in \{0, \ldots, len(W_{all}[0]) - 1\}$ do
        for each $k \in \{1, \ldots, n-1\}$ do
            $W_{avg}[i] \leftarrow W_{avg}[i] + W_{all}[k][i]$
        end for
        $W_{avg}[i] \leftarrow W_{avg}[i] \ / \ n$
    end for
    return $W_{avg}$
end function
```

After all SFL clients (independently!) completed their local epochs, they send their updated weights (or gradients) of the local client models to a FedAvg executor (aggregator) that can be either one of the clients, the main server or someone completely different. The aggregator averages the weights and obtains the "global client model" that is forwarded back to the clients. The clients use the global client model as weight initialization for their local client models to perform the next round of the whole procedure (global epoch). The FedAvg step is therefore responsible for synchronizing the model across multiple clients.

Our pseudocode in Algorithm 2.1 visualizes the rough process of SFL. We refer to [27] for more details including formulas for the calculation of weights and gradients. Also, the pseudocode does not consider the applied DP and MPC mechanisms to protect local client weights and smashed data that we discuss in the following sections.

Privacy guarantees of SFL: In SFL, clients keep their private data locally but, however, data from all clients is used for model training. Instead of the private data, clients only need to share smashed data and local models with other participants. In the further course, we discuss how this shared data can be protected to prevent possible conclusions about the underlying private client data.

SFL also enables model privacy, i.e. a crucial part of the model can be kept private from the clients [27] by splitting the model between the clients and the main server. The main server can thus be granted the exclusive use of the trained model.

2.2.2 Multi-Party Computation

MPC denotes the computation of a specific function that is executed by a set of parties without revealing anything except the result [14]. To do so, secret sharing is applied to the input values so that the parties have to collaborate if they want to open a secret value. Secret sharing relies on the concept of distributed information where some qualified sets of parties can collaboratively reconstruct the underlying information while a single party or some smaller (unqualified) set of parties cannot deduce anything about the information. A secretly shared value is commonly shown with brackets, i.e. $[a]_i$ is the share of the value a that is owned by party i. Since the performed operations of the parties on their respective shares are often identical, the index i is omitted in this case and $[a]$ denotes the share of value a.

Depending on the protocol, the qualified sets differ and up to a certain number of dishonest parties (not necessarily following the computation protocol and possibly trying to gain secret information) can be tolerated. A simple example from [14]: A number $x \in [0\,,\ M-1]$ is secretly shared among n parties where each party i owns a secret random value $x_i \in [0\,,\ M-1]$ and x is the sum of all shared values in $\mathbb{Z}_M$, i.e. $x = (\sum_i x_i) \mod M$. In this example, we can easily see that only the set of all parties is qualified to reconstruct x and a smaller subset cannot gain any information about x. Even if $n-1$ parties exchange their shares, then still any number $x \in \mathbb{Z}_M$ is possible with equal probability. In this way, MPC protocols can prevent malicious parties from opening a secret value.

In the following, we introduce the Mascot protocol which can keep the secrecy of values, also in case of a dishonest majority of parties [14]. The protocol is based on checking whether parties deviated from the protocol and in this case, the computation would abort without revealing information about the secret values. Mascot is one of the versions of the SPDZ protocol [14] where versions differ only in the offline phase which can be executed before the actual inputs are supplied (also: preprocessing phase).

In the offline phase, parties produce Beaver triples ($[a]$, $[b]$, $[a \cdot b]$) where a and b are random values. The Beaver protocol makes sure that all values from the triple are kept secret to all parties so the parties only own their private shares $[a]$, $[b]$ and $[a \cdot b]$ and, in case the threshold of tolerated dishonest parties is not exceeded, the parties cannot reveal the underlying values in the triples.

After inputs are supplied, the online phase with the actual computation of the result is executed. The computation of linear operations (additions of secret values or multiplying with known constants) can be performed locally and efficiently by the parties. In contrast, multiplications of ciphertexts are costly and require communication between the parties. Using Beaver triples, a significant part of the multiplication effort is moved to the offline phase, and during the online phase, multiplications of secret values $[x]$ and $[y]$ can be reduced to linear operations as follows:

$$[xy] = [(x + a - a) \cdot (y + b - b)] = (x + a) \cdot (y + b) - (x + a) \cdot [b] - (y + b) \cdot [a] + [ab]$$

Here it should be noted that $x + a$ and $y + b$ have to be revealed to perform the multiplication but this does not reveal information on x and y because a and b are random and secret values.

Damgård et al. [4] apply MPC to the prominent example where Danish banks try to select suitable customers from a set of Danish farmers. Therefore, the banks need to carefully select the right group of suitable farmers among this group of risk-prone customers with partially too much debt.

However, the selection requires confidential accounting and production data from a large number of farmers and to solve this issue, MPC is deployed and banks should learn nothing except for the needed final score of the farmers. The SPDZ protocol is used to do the computation which leads to the following issue: In [14], it is assumed that each client who supplies input and receives output also plays the role of a MPC party. But in the case of Danish banks, it is highly desirable to separate the roles of clients and parties as it should not be demanded from the banks to do the whole preprocessing phase and it may even be the case that the clients are unknown at preprocessing time [4].

Damgård et al. [4] solve this issue with the implementation of a protocol that can handle a disjoint set of clients and MPC parties. It allows clients to deliver secret input to the parties while parties only get to know their shares of the input value. We use the term secretly share for the delivery of secret values from clients to the parties. After the delivery of all inputs, the parties run the ordinary MPC protocol on their shared values. With another designed protocol for the output delivery [4], one or multiple chosen clients can receive opened output values after the MPC computation while the parties do not gain any information about these output values. Later, we show how we can make use of the input- and output delivery protocol for private gaze estimation.

Privacy guarantees of MPC: MPC allows to compute a function while obtaining the privacy of the inputs. The function is collaboratively calculated by several parties and the input privacy is maintained as long as a threshold of dishonest parties (given by the MPC protocol) is not exceeded. In addition, it is possible to blame malicious parties that deviate from the protocol so that the correctness of the computation can be ensured.

2.2.3 Differential Privacy

DP is a property defined for any randomized mechanism $M : D \to R$ with domain D and range R. A mechanism is (ϵ, δ)-DP if it fulfills:

$$\Pr[M(d) \in S] \leq e^{\epsilon} \Pr[M(d') \in S] + \delta.$$

for any subset of outputs $S \subseteq R$ and for two input sets $d \in D$ and $d' \in D$ that differ only in a single entry [1].

For “small” $\epsilon > 0$ and $\delta \in [0, 1]$, the definition guarantees that outputs are not sensitive to small changes in the input (output stability property) [1]. The standard example refers to a database where a change to individual records does not change the output distribution much [17]. Then it is not possible to draw conclusions from given output (aggregated information) to personal data (individual records) and therefore the personal data can be protected. The values ϵ, δ quantify the strength of the privacy [17] and smaller values lead to stronger privacy.

We can also directly apply the definition to classification tasks. Let D be the set of valid model inputs (e.g. for image recognition, an input contains the pixels of an image). Let R be the labels according to which an input needs to be classified. M is the model that performs the classification of the input. In this case, (ϵ, δ)-DP avoids that a small input perturbation (e.g. change of a pixel) to correctly classified input causes the model to produce another (wrong) classification [17]. This property is called robustness against adversarial examples and can be ensured up to a particular size of input perturbations [17].

Using DP as protection against adversarial examples is a frequently demanded property that is studied in many works, e.g. in [17]. However, gaze estimation is a regression task (not classification), and therefore our focus is not on the avoidance of adversarial examples.

Instead, we want to ensure the privacy of client data against a curious main server or aggregator in SFL. We refer to the (ϵ, δ)-DP definition in the following way: Let I denote the set of MPIIGaze images of size n and $D = \{S \subseteq I \mid |S| \in \{n-1, n\}\}$ so $d, d' \in D$ differ by exactly one missing image. Let R be the (infinite) set where each entry contains a possible instantiation of model weights. Here, M maps from a given set of images to the model weights that result from training the model on the given images (simplification here: order of images during training is not relevant). In this case, (ϵ, δ)-DP ensures that it is not possible to decide, based on the trained model, whether a given image was part of the training set or skipped during training. Thus, we can ensure that it is not possible to draw conclusions about the underlying training data for a given trained model. In the further course, we can use this (ϵ, δ)-DP property to measure the ensured privacy level for clients in SFL.

Several systems have been built to ensure DP for neural networks depending on the task to be solved [1]. We need sophisticated methods that preserve the utility of the neural network (high accuracies) and each SFL client needs to apply the method to protect their personal images. Accordingly, we need a method with acceptable computational requirements.

Privacy guarantees of DP: A DP mechanism, in general, ensures the privacy of individual records that are contained in a data aggregate. This prevents private information of the underlying individual records from being extractable from the data aggregate with respect to the ensured privacy level (ϵ, δ). For our application (SFL), DP guarantees the secrecy of the individual client data during model training.

Further, the composability theorem holds [1]: If the components of a mechanism are differentially private, then this also applies to the composition. This allows the modular design of a privacy-preserving mechanism. In particular for model training with Differentially-Private Stochastic Gradient Descent (DP-SGD) (see Section 3.3.2), the values (ϵ, δ) can be computed and accumulated at each data access to calculate the ensured amount of privacy for the overall model training [1].

3 Methods

In the following, we show how we split the CNN to perform supervised training of our gaze estimator with SFL. We also show how we can enhance training efficiency by performing unsupervised training and model pruning. Then, we show how to integrate defences to secure the personal client data and model. Afterwards, we show our attacker implementation that we use to demonstrate the empirical effectiveness of our defences. Finally, we analyze how to implement pruning and discuss the influence of pruning on runtime and memory requirements.

3.1 Network architecture

Based on previous work for gaze estimation [7, 33], we implement the gaze estimation model as CNN once as ResNet and once as LeNet. As shown for the LeNet in Figure 3.1, the normalized eye image is applied to one convolutional and one pooling layer in the client-side model part which results in the smashed data (c_1 in Figure 3.1). The main server continues forward propagation on its model part and after passing another convolutional and pooling layer, the output c_2 is obtained. Then, c_2 is flattened into a single channel and then passed to the linear layer which produces the output l_1. Afterwards, another linear layer is applied to obtain an output (l_2) with only two values, representing the predicted 2D gaze angle.

The extracted head pose (containing two values representing a 2D angle) is visualized by the box containing the smiley face in Figure 3.1 and because the head pose is a separate input consisting of only two values and it is not part of the image, it makes more sense to pass the head pose directly to the first linear layer, as also done in [33]. For our SFL architecture, it has to be considered that this linear layer of the CNN is owned by the main server. Therefore, we assume that the head pose information is passed from the client to the server and the head pose is not supposed to be kept secret. Nevertheless, we also refer to test cases in Chapter 4 where head pose information is not used as input for better privacy guarantees, i.e. the main server propagates solely the smashed data through its part of the network.

For the model that we split (see Figure 3.1), we first used the same structure and hyper-parameters from [33] to allow an accurate comparison to their results (Data Centre approach, see Section 4.1) as well as to the FL results from Elfares et al. [7]. Since SFL involves the FedAvg operation, model weights are modified differently than in conventional (non-privacy preserving) learning and an optimal solution on the composed network from [33] is not necessarily optimal for SFL. Therefore as a second step, we modified the hyper-parameters and structure of the model in order to decrease the error of the gaze predictions further.

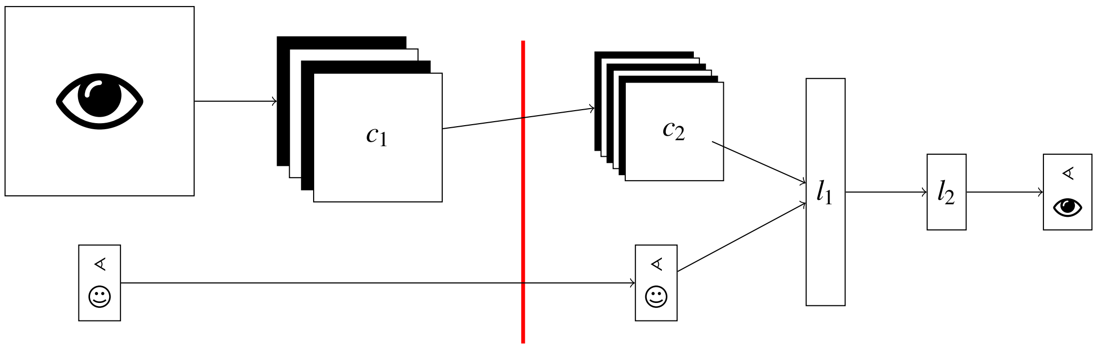


Figure 3.1: Split of the CNN (LeNet) where the clients own the layers on the left and the main server owns the layers on the right side of the red line

3.2 Unsupervised federated gaze representation learning

To avoid disclosing information from the annotations, we analyze how we can protect the personal client inputs in the case of unsupervised gaze representation learning. In unsupervised gaze representation learning methods, client images of all clients are collected first and then used to train the Encoder-Decoder in an unsupervised fashion. We apply FL to the Cross-Encoder in [26] so that each client trains the Encoder-Decoder locally with their personal eye images. After a chosen number of processed training samples or training epochs from each client, the FedAvg of client weights is computed by a dedicated server and passed back to the clients. Then, clients use the received average weights as instantiation for the next round of Encoder-Decoder training with their images and we continue this until a fixed number of rounds is done. For the training, we take the default configuration of Sun et al. [26]: gaze feature dimension $d_g = 12$, eye feature dimension $d_e = 32$ and we also consider the head pose as input to the gaze estimator.

After Encoder-Decoder training, 100-shot gaze estimation is applied to train the gaze estimator with two linear layers to learn the mapping from gaze features (output of the trained Encoder-Decoder) to gaze angles. For 100-shot gaze estimation, we also want to preserve the secrecy of the client images. To do so, we fix one client (with id 0) and train the gaze estimator on 100 images exclusively from this client. This (inexpensive) training can be carried out locally by the fixed client so that no images need to be passed on to other participants during this phase either.

Another possibility is to train the Encoder-Decoder with SFL (instead of FL) although we did not create an implementation for this case. It requires a more complex model split where the server owns the right part of the encoder and the left part of the decoder (considering Figure 2.2), i.e. the client owns input eyes and target eyes. We do not go into detail here and we did not implement Encoder-Decoder training with SFL.

Besides the facilitated data collection properties (Section 2.1.3), unsupervised gaze representation learning with FL or SFL has the positive effect that no more labels (gaze angles) have to be transmitted from clients to the server. Latter also applies to common (supervised) FL because the server gets solely access to the client model weights (for the FedAvg operation). Table 3.1 indicates for each method whether gaze angle annotations (labels) are required and whether they have to be shared with the server. Gaze angles could contain sensitive data that reveal information about the user's attention and thus, for example, reveal information about personal preferences.

Training method	FL	SFL	Unsupervised FL	Unsupervised SFL
Labels required?	for all samples	for all samples	a few ($\approx$ 100)	a few ($\approx$ 100)
Labels shared with server?	no	yes	no	no

Table 3.1: Comparison of ordinary FL/SFL with FL/SFL for unsupervised gaze representation learning. We show whether gaze annotations (labels) are required and whether they have to be shared with the server

It should be noted that both for unsupervised FL and SFL, MPC and DP mechanisms have to be integrated to ensure protection against data leakage attacks that are discussed in Section 5.1. In the following section, we apply MPC and DP to supervised SFL only.

3.3 Defences

For FL, clients transfer their updates to the aggregator and therefore the defences, such as PrivatEyes [6], focus on the protection of the individual model updates. For SL, on the other hand, smashed data is transferred from client to server which requires protection of smashed data to avoid the attacks (presented in Section 3.4). Since SFL is a combination of FL and SL, we need to provide protection against both attack scenarios and in this section, we analyze how we can use MPC and DP mechanisms for this purpose.

3.3.1 MPC for SFL

The main focus of SFL is on keeping the personal input images secret which leads to the idea to take advantage of MPC to strengthen our privacy properties. One challenge: When applying MPC, we do not want our SFL clients to act as MPC parties as well. The reason for this is that we have a potentially high number of clients that would result in (infeasible) MPC protocols of high complexity. In addition, we do not expect computational resources from our clients that suffice to execute the protocols. To solve this, we assume that another group of MPC parties can be found that is considered sufficiently trustworthy by the clients so that the tolerated threshold of dishonest parties is not exceeded (i.e. $n-1$ out of n parties could be malicious). We can use the protocols from [4] to allow our clients (including SFL server or aggregator) to pass input to the parties and parties do not get to know the secret values. After the MPC computation, clients can receive and reconstruct the output values.

MPC can be applied at different places and to different extents:

MPC for model training

First, we consider the case that MPC should be integrated during the training phase in SFL. We can make use of the architecture of SFL and clients can still perform forward propagation locally without usage of MPC. After forward propagation, clients secretly share their smashed data and MPC is used for forward and backward propagation within the server model on secretly shared network weights and biases. In this case, essentially all server operations are executed as MPC operations on secret data. This keeps the entire server model secret by existing only through the private shares of the MPC parties. Moreover, the secrecy of smashed data avoids possible attacks that harm privacy of the personal eye images (see Section 5.1).

MPC for inference

One more possibility is the application of MPC during the inference phase. Client(s) and server secretly share all parameters (weights and biases) of the trained model and the passed inputs are also secretly shared so that the MPC parties neither get to know the input images nor any parts of the trained model. In contrast to MPC for the training phase, there is no backpropagation phase needed which reduces the computational effort. Also depending on the use case, we may only want to predict a few gaze angles which results in a significant reduction of effort compared to the training set that would have to be forwarded in multiple epochs during training. So even if MPC for the whole training as described above results in too much effort, MPC for the inference phase remains a reasonable option. For the corresponding runtime measurements, we refer to Chapter 4.

For inference, there is also the option to skip the secret sharing of the model or part of it if it is not supposed to be kept secret from the parties and the parties still forward the secretly shared input images through the network. In this case, multiplications of model weights and input tensors can be performed locally by the parties because one of the factors (model weight) is known to all parties (not secretly shared). Therefore, no communication is required as in the case of multiplications of two secretly shared values. Without proof, we guess that communication effort is then limited to the secret sharing protocol for input images and the output delivery for gaze angles and therefore, inference can be carried out efficiently.

MPC for aggregation

Even if training and inference are performed in the original variant without using secret sharing for input data and model weights, MPC can be integrated very useful for the FedAvg operation during the training phase. Then, in order to perform FedAvg, clients secretly share their local weights and the MPC parties are solely responsible for adding up the weights and delivering the sums back to the clients. The clients can then obtain the federated average by simply dividing the sum by the overall number of clients. MPC for FedAvg is simple and efficient as it requires only additions of secretly shared values. Still, it gives additional useful privacy properties (see Section 5.1). More detailed empirical analyses for MPC for FedAvg (in the context of FL) are carried out by Elfares [6].

3.3.2 Differential Privacy for SFL

Thapa et al. [27] mention that it is necessary to protect the client data in SFL against a curious main server as well as against a curious aggregator. The main server could use the smashed data and the aggregator could utilize the local client model (if not protected with MPC) or the global client model to reconstruct individual client images.

To avoid this, we can use dedicated DP mechanisms for neural network training. These mechanisms commonly work by sampling and adding bounded random values (noise). Noise can be added to the gradients during backpropagation as done in the DP-SGD algorithm in [1]. Other approaches, such as [17], introduce a noise layer that adds noise to the tensors during forward propagation. If we combine both concepts, we can protect gradients as well as smashed data of clients during SFL and thus obtain privacy protection against aggregator and main server [27]. In the following, we illustrate the two methods in detail.

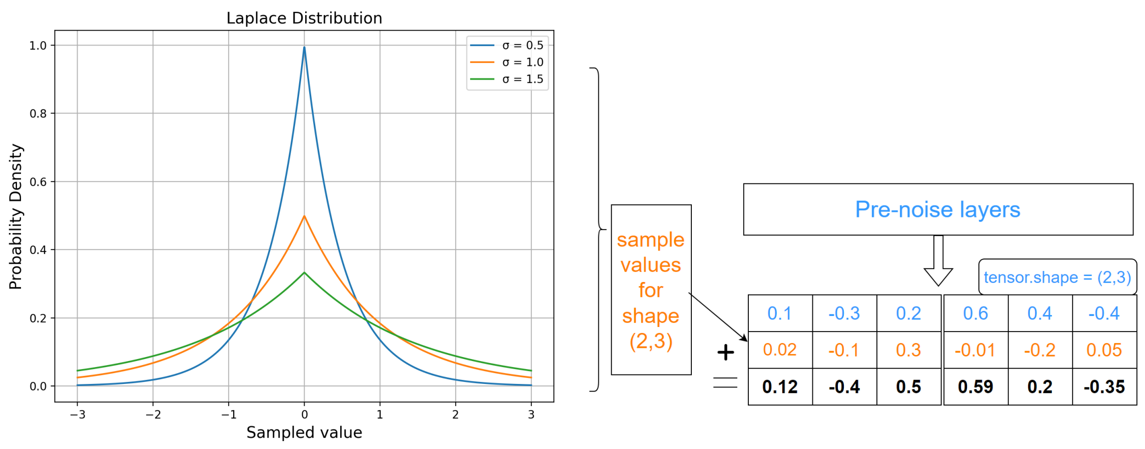
Differentially-Private Stochastic Gradient Descent (DP-SGD)

DP-SGD is based on adding noise to the gradients during the training phase. The amount of noise has to be large enough to provide a sufficient level of privacy but also small enough to not cause huge prediction errors. Therefore, the magnitude of noise has to be chosen carefully depending on the magnitude of the gradients. A simple attempt is to add noise to the final parameters after training is completed. But this approach is not useful because in this case, noise that is selected according to a worst-case analysis would destroy the utility of the model [1].

Instead, after each backpropagation process, gradients are clipped with a fixed value C to get a bound for their maximum value: $g_i \leftarrow g_i / \max\{1, \frac{\|g_i\|}{C}\}$ where g_i is the gradient for a certain local sample x_i [27].

This method comes with the main difficulty that the gradients have to be available per sample. To achieve this, we can set the batch size to 1 but this would result in high runtimes due to underutilization of GPUs [31]. Instead, we use the Opacus framework [31] that offers a performance-improved and batched computation for clipping gradients as well as the addition of noise subsequently. The amount of noise depends on C and the noise multiplier parameter that allows the user to increase or decrease the amount of noise (independent of the magnitude of gradients).

During SFL, clients can apply DP-SGD (with Opacus) to clip the gradients of their models and to add noise to their model updates. Then, Opacus provides obfuscation of local client models before they are passed to the aggregator (as required in Section 5.1). In addition, the framework can output the resulting ensured privacy level (the values ϵ and δ) against the distinction whether a specific image was used to train the model or not. Opacus also offers the option to pass the target privacy level (ϵ, δ) as parameters and then, it infers the noise multiplier and adds the required amount of noise.



(a) Laplace distribution for different σ **(b)** Addition of noise

Figure 3.2: Integration of Laplacian noise during forward propagation

PixelDP

After the first backpropagation phase, the updated gradients using DP-SGD also have an influence on subsequent forwarding phases and smashed data. Nevertheless, to avoid privacy leakage by a curious main server, we still need further designated protection (obfuscation) of the smashed data [27].

As realized by Thapa et al. [27], we use the defence mechanism PixelDP [17] to add noise during forward propagation in the client-side network. It is possible to place noise directly in the image, after the first layer or deeper in the network (if more than one layer exists). We add the noise after the first layer which is usually simple and standard for many neural networks.

We choose the magnitude of noise depending on the sensitivity of the pre-noise layers Δ, i.e. maximum change of output that can be produced by the pre-noise layers by a change in the input. A linear transformation or convolution can be modeled as matrix-vector multiplication where the matrix contains the weights of the network [17]. We have exactly one pre-noise layer and can calculate the sensitivity using matrix norms as illustrated by Lecuyer et al. [17].

For the noise, we sample random values from the Laplace distribution with mean zero and scale $s := \Delta \cdot l$ for sensitivity Δ and user parameter l that allows the user to control the amount of noise. The standard deviation $\sigma = \sqrt{2} \cdot s$ increases with increasing scale s and determines the magnitude of noise. With a larger standard deviation σ, it gets more likely that values are sampled further away from the mean (zero) and thus more noise gets added. Figure 3.2 shows the Laplace distribution (depending on σ) and visualizes an example noise addition to a tensor after passing the pre-noise layers. The blue tensor shows the output of the pre-noise layers with shape (2,3) and accordingly, 6 noise values get sampled (orange) and added to the blue tensor. Though we only show the range $[-3, 3]$ of the Laplace distribution, we want to clarify that the sampled values (given on the horizontal axis) are not bound to this range. But it is unlikely that much smaller or greater values get sampled because, as visible in the diagram, probabilities converge to 0 to the left and right side for all σ.

With PixelDP, Lecuyer et al. [17] aim to provide robustness to adversarial examples and for this case, they specify the (ϵ, δ)-DP values that are ensured depending on σ. PixelDP also gives us the required privacy for smashed data although we are skipping mathematical guarantees and instead rely on Thapa et al. [27] and select the user parameter l accordingly for our experiments (l^{-1} is equivalent to ϵ' in [27]!).

3.4 Attacking our system: Investigate privacy empirically

We implement an attacker that tries to re-construct eye images to empirically show the amount of information leakage still deducible from our defences. The popular model inversion attack aims to reconstruct data samples that were used to train a given target model [19]. In this attack, the adversary starts with noisy input samples that can be blurred eye images (no prerequisite to know actual client images) [6] and forwards them through the target model. Iteratively, the adversary uses backpropagation on the target model to optimize the input sample and after a certain number of rounds, the adversary outputs the optimized input sample (reconstructed image).

For our attack, we use a modification of the model inversion attack because we do not assume that the server has access to the target model (local client model). To carry out the attack, the server performs forward- and backward propagation as specified by Algorithm 2.1. In parallel, the server also trains an additional CNN (inverse network [13]) that should re-create eye images for given smashed data. To make training of the inverse network possible, we expect that the server knows a shadow dataset (eye image and smashed data-pairs of other clients but the server has no access to the eye images of the attacked client).

The structure of the attack is shown in Figure 3.3. The attacking main server trains the inverse CNN using the smashed data that is obtained in an epoch during the ordinary training process as well as using eye images of all clients except for p00 (Figure 3.3 skips visualization of p03,...,p14). We, therefore, assume that the eye images of clients p01, ..., p14 are leaked or these clients do not keep their images secret. The server trains the inverse CNN with smashed data as inputs and corresponding eye images as targets. Afterwards, the server tries to recreate secret eye images of the attacked client (p00) by forwarding their smashed data through the inverse CNN.

Our inverse CNN consists of convolutional and linear layers and we measure the loss value using the mean squared error. The outputs of the convolutional layer during training of the local client model (Model 2 in Section 4.1) is the smashed data which we take as input to the inverse CNN. The normalized eye images (after normalization, see Section 2.1.1) serve as inputs to train the local client model and at the same time, we use them as targets for the training of the inverse CNN.

We performed the attack on our SFL implementation once with and once without the addition of noise (PixelDP) in order to get an impression of which extent PixelDP protects privacy. We apply PixelDP with noise multiplier $l = 2.0$ (see Section 3.3.2). In both configurations (with and without PixelDP), we also apply DP-SGD with Opacus ($\epsilon = 0.5$, $\delta = 10^{-5}$). In Section 4.4.3, we give the outputs (reconstructed eye images) of the attacker.

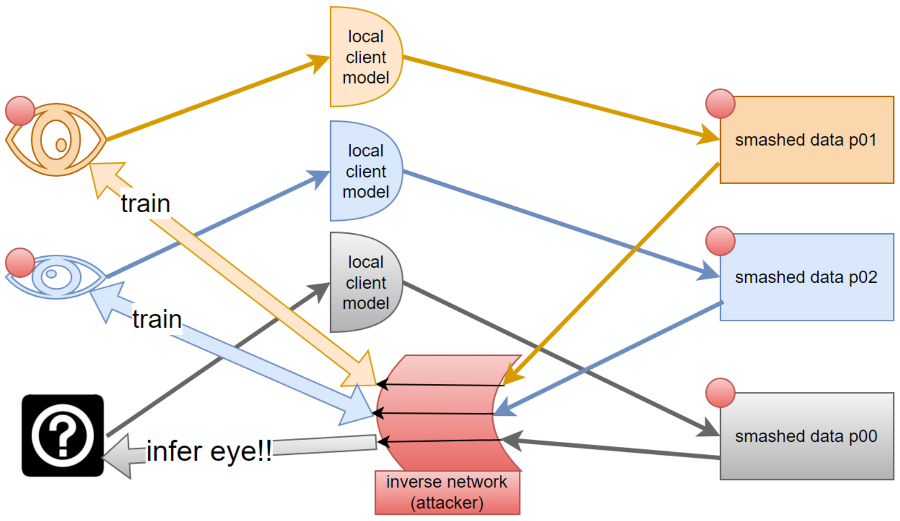


Figure 3.3: Structure of the attack where the attacker (main server) trains an inverse network using smashed data and eye images of p01 (orange) and p02 (blue) in order to infer eye images of p00 (grey) for given smashed data. Red circles indicate data to which the attacker has access

3.5 Model pruning

Now, we want to investigate how to decrease the model size and inference time with model pruning. Especially with MPC as described in Section 3.3.1, we discuss how we can use model pruning to reduce the overall computational effort.

Standard model pruning can be divided into local and global pruning [21]. For local pruning, a given layer is considered and a given amount of connections from this layer is removed (by setting the respective weight in the tensor to zero). We select the weights with the lowest absolute values to be pruned ($L1$ norm) by using the assumption that they have the least influence on the outcome. Global pruning works in the same way but across several layers, i.e. weights of several layers are all compared with each other to determine the weights to be pruned [21]. Although pruning can also be used during the training phase, we focus on applying pruning at the end of the training to reduce the inference time and the size of the trained model.

If the trained client or server model is not intended to be kept secret, then model pruning can be applied to SFL without restrictions in the same way as without privacy-preserving training. On the other side, in case there is no single party that owns both the final client and server model, then clients and server are limited to applying pruning separately on their part of the model. An idea to circumvent this restriction (in order to perform global pruning) is to apply an MPC protocol to compare secretly shared weights [18] to reveal solely the information of which weights are smallest and prune them subsequently. But this protocol would further increase complexity and runtime and therefore we skip the implementation. In the following, we apply local pruning which can be done directly by the respective model owner so that models can still be kept secret without requiring additional MPC protocols.

A sparse tensor format (removing pruned zero weights) is needed so that we can exploit sparseness in weight tensors and actually take advantage of pruned models. To do so, we use the standard sparse format provided by Pytorch which, instead of storing all values of the weight tensor (dense tensor), stores:

- All indices (from the dense tensor) that contain non-zero values. This is called a pruning mask.
- The corresponding non-zero values.
- Further metadata, e.g. the dimensions of the tensor.

According to the Pytorch documentation, pruning is still under development and at the moment, there exist several immature approaches that try to use sparse weight tensors to reduce the time for forwarding given input data. Essentially, the forwarding phase requires dot products of weight tensors with input tensors as illustrated later in this section. It is a challenge to use sparse tensors to calculate these products more efficiently, especially in comparison with advanced parallel implementations for dense weight tensors that are highly optimized for execution on GPUs. The observed runtimes of these implementations do not depend on the degree of sparseness. This is the reason why we could not create an (efficient) implementation for pruning to reduce the overall inference times. In Chapter 4, we instead focus on reducing the model size with pruning.

However, in case MPC is used for inference (Section 3.3.1), pruning remains a beneficial component to reduce the computational requirements. Machine learning protocols with MPC [14] show strongly increased runtimes for model training and inference (as we also see in Chapter 4) which leads to reduced usability. In these protocols, the overall runtime (for offline- and online phase together) strongly depends on the number of the costly ciphertext multiplications as they require communication between the parties (see Section 2.2.2). To exploit sparseness in weight tensors for MPC inference, we have to assume that the pruning mask is published (not secretly shared!) to the MPC parties by the respective model owner (client and server). Then, all parties know which weights are pruned and they can skip multiplications with pruned weights (zeros) which reduces the number of required ciphertext multiplications. This leads to decreasing computational effort as well as decreasing storage requirements by a reduced number of Beaver triples that are needed to be created and stored in the offline phase. In the following, we determine the amount of multiplications that we can save for inference on the LeNet model for a) pruned linear layers as well as b) pruned convolutional layers. We assume that a given amount (p% of the weights) of the respective layer gets pruned.

a) Linear (also called fully connected) layers apply a linear transformation to every input tensor x within a batch [22]. The output tensor y can be calculated as follows according to Pytorch documentation: $y = x \cdot A^T + b$ where b is a bias term and A is the matrix containing the weights. With this formula, it is clear that we can save p% of multiplications in case that p% of weights get pruned (so p% of entries in A are set to zero).

b) For convolutional layers, we again consider one single tensor within one batch. A single output pixel of one resulting feature map is obtained after multiplying the respective filter element-wise with the input. Let c_{in} be the number of input channels and (h_f, w_f) be the dimensions (height and width) of an applied filter so one filter is of size $c_{in} * h_f * w_f$. In other words, the calculation of one output pixel causes $c_{in} * h_f * w_f$ multiplications. An output pixel is part of a feature map with dimensions (h_m, w_m) and c_{out} feature maps are created. So in total, $h_m * w_m * c_{out}$ is the number of resulting pixels. We conclude that $c_{in} * h_f * w_f * h_m * w_m * c_{out}$ multiplications take place while forwarding a single tensor from one batch. The filters contain the $c_{in} * h_f * w_f$ weights each and all filters are multiplied in the same way with the input so it can be seen that again, we can save p% of the multiplications in case that p% of weights get pruned. For convolutional layers, there is one bias value added to each element in the output channel but again, the bias does not contribute to the number of multiplications.

Bias tensors are rather small (compared to weight tensors) and since they only require additions, they do not have a relevant influence on the overall runtime and storage requirements. To avoid that pruning on bias tensors leads to increased angle errors, we keep bias tensors unpruned (which is also the common case in other published implementations). In our experiments with pruning in the following chapter, we show how to prune weight tensors to decrease the model size while we aim to avoid significant increases in the angle errors.

4 Experiments

While we apply our privacy-preserving approach, we aim for a similar performance with the non-private baselines. Hence, we compare our approach with the results from the following learning methods that are also used as baselines in prior work by Elfares et al. [7]:

- **Data Centre Learning.** For this approach, all data is first collected and then used to train the model. Therefore, privacy-preserving techniques (e.g. SFL) are not needed and instead, the whole training with all input data can be performed locally on a single server as implemented in [33]. But in this case, there is no privacy protection and the Data Centre approach cannot be applied if participants are not willing to share their private face or eye images. With our implementation of SFL, it is possible to perform Data Centre Learning by just modifying the number of clients from 14 to 1.
- **Individual Learning.** In this case, only a single client trains the CNN locally using their own data. Therefore, the model is trained only with eye images of a single person which makes privacy-preserving model training easier since neither data nor model have to be exchanged between different participants during the training phase. With our implementation of SFL, we can also perform Individual Learning by setting the number of clients to 1 again and we only add the images of a single person to the training set of the client.
- **Adaptive FL.** This is a privacy-preserving learning method implemented in [7] that is based on FL and extended with a specific combination of optimizers that outperforms the original Vanilla FL by a mean error of 15.8% [7].

We first describe our setup and then do benchmarking by comparing our results with the other mentioned approaches.

4.1 Setup

To measure the runtimes, we use Google Colab[1] and ran the protocols utilizing the available GPUs if not mentioned differently (as in Table 4.1). We fix our CNN to the following structures and in the further course, we use this numbering to refer to the respective model:

1. We use the LeNet implementation from Zhang et al. [33] and split it after the first convolutional layer as visualized in Figure 3.1.

[1] https://colab.research.google.com/

2. We modify Model 1 to have 80 instead of 50 outgoing channels after the convolutional layer within the server model with the purpose that the higher model capacity increases the expressiveness of the model. In addition, we use the Adam optimizer available in the Pytorch library (instead of the SGD optimizer that is used for Model 1).
3. We run further experiments on the ResNet network as realized by Thapa et al. [27].

First, we give the results without applying MPC, DP or pruning techniques (Basic Model) and then we show the influence of each of these techniques on the performance of our system. In the further course, we examine the utility of individual components during the model training (ablation study): First, we present the results for the case that the head pose is no longer used as input to our model. We continue by increasing the number of local epochs as well as modifying the network split. We throughout measured the angle errors during the inference phase regarding the images of a person that was not considered (left-out) during the training phase. Then, we deviate from the SFL approach and present another concept (averaging the smashed data) along with the corresponding test results. Finally, we also give the test results for our implementation of unsupervised federated gaze representation learning.

4.2 Basic model training and inference

We first use Model 1 (see Section 4.1) for training and inference. We executed 10 epochs to train Model 1 with a runtime of approx. 10 minutes, with 3000 images of each participant except for the images of one given person (the left-out person as shown on the horizontal axis in Figure 4.1). The images of the left-out person were used to measure the angle errors during the subsequent inference phase. The mean angle errors for inference are visualized in the bar chart (Figure 4.1). For Data Centre Learning and Individual Learning, we used the same models and input parameters as for Model 1 (SFL). We adopted the results from Elfares et al. [7] to visualize the errors for Adaptive FL and we can see that the average error is about 17% larger than for our implementation of SFL. But we want to emphasize that due to potentially different optimizers and hyper-parameters (e.g. learning rate and number of epochs) from the two models, we cannot give a statement about which learning method makes more accurate predictions in general. Instead, we even justify in another section (Section 4.6.3) that weight updates in FL are equivalent to updates in SFL and therefore similar results can be achieved.

For Data Centre Learning, the errors are 7.5% lower on average than for SFL which shows that we could apply SFL while only causing slightly increased errors.

For Individual Learning, we used input data from the person with an incremented index for training, e.g. the first bar in the pillar chart (Figure 4.1) results from training on P01 and inference on P00. For the last bar, we used P00 for training and P14 for inference. Due to the decreased amount of training data, it took less than 2 minutes for each training (10 epochs) but this resulted in an increased average error of about 10% compared to Model 1. Individual Learning led to a high variance in the angle errors: While the errors for specific allocations (person for training, person for inference) for persons with many characteristic similarities (appearance, gender, with or without glasses) were about 6 degrees, the error was 16 degrees for another allocation (which is not covered by Figure 4.1). So we can see that there is the risk of very large errors during inference if the features are severely deviating from features of the training samples. As also shown for Adaptive FL [7], SFL

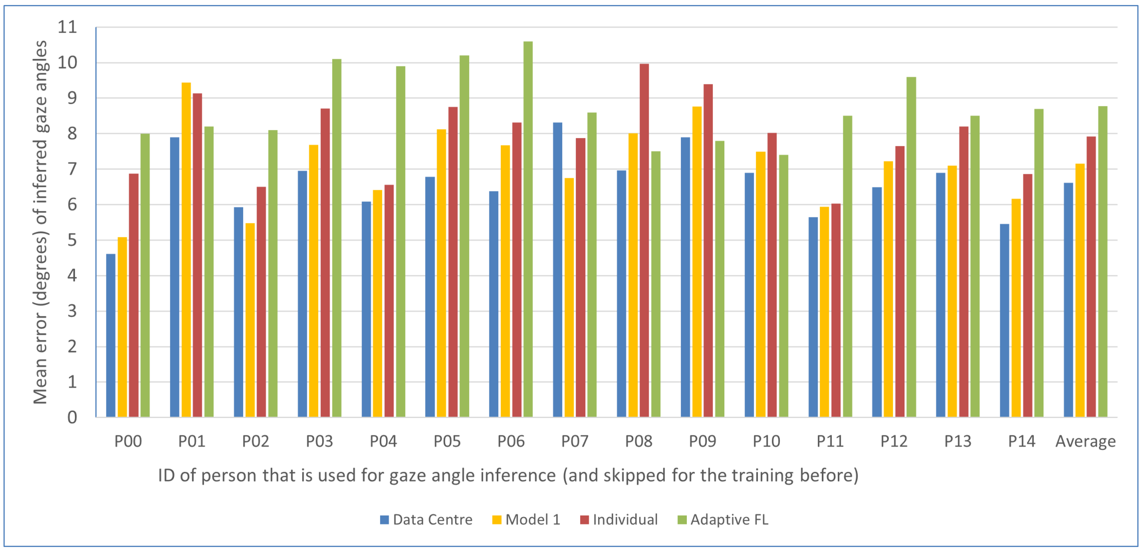


Figure 4.1: Comparison of prediction errors from Model 1 (SFL) with Data Centre Learning, Individual Learning and Adaptive FL

offers more robustness regarding this risk due to more diverse input data that is considered during the training phase. We can conclude that generalization is better with SFL than with Individual Learning.

We further used the modified structure and optimizer to run training for Data Centre, Individual Learning as well as Model 2 throughout for 10 epochs again. In Figure 4.2 we can see that our change to the network causes the average error to decrease clearly (10%) for Model 2 compared to Model 1. Due to the modified setup, also the Data Centre error decreases with 5% but this leaves only a small lead over SFL (Model 2). The greatest errors are now created by Individual Learning and they are in a range from a comparably small value (5 degrees) to an error more than twice as large (11.5 degrees). With these test results, we can confirm previous realizations that the errors for Individual Learning strongly deviate from each other. They depend on how the characteristics differ between samples for model training compared to samples for inference.

We also applied SFL to the ResNet-18, a significantly larger network that is used in [27] for the identification of skin lesions. We ran 10 epochs with the Adam optimizer using the same parameters as before. Again, we give our test results (Model 3) in Figure 4.3 along with the results for Data Centre Learning and Individual Learning on the ResNet and we also repeat the results from Model 2. We can see that the errors for Individual Learning have risen considerably but with SFL for ResNet, we further reduce the mean error slightly (3 % lower than for Model 2). Nevertheless, the training time has almost doubled to more than 18 minutes for the ResNet and therefore we use LeNet (Model 2) in the further course. Especially with MPC, we show that the runtime for forward and backward propagation is our bottleneck and therefore a small CNN is better suited.

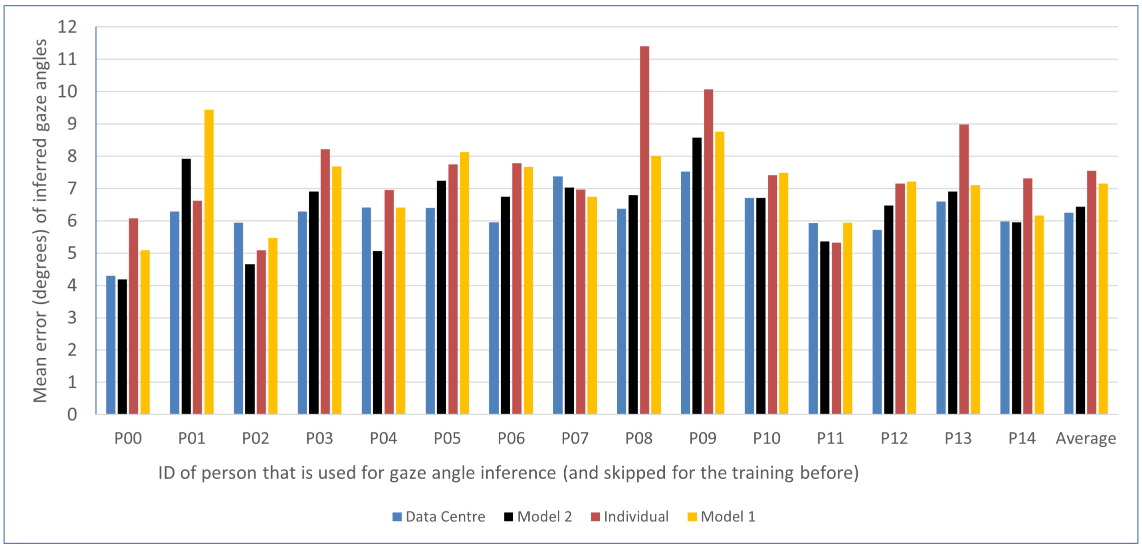


Figure 4.2: Comparison of prediction errors from Model 2 (SFL), Data Centre Learning, Individual Learning (all with modified model structure and optimizer) and Model 1 (with original setup, errors repeated from before)

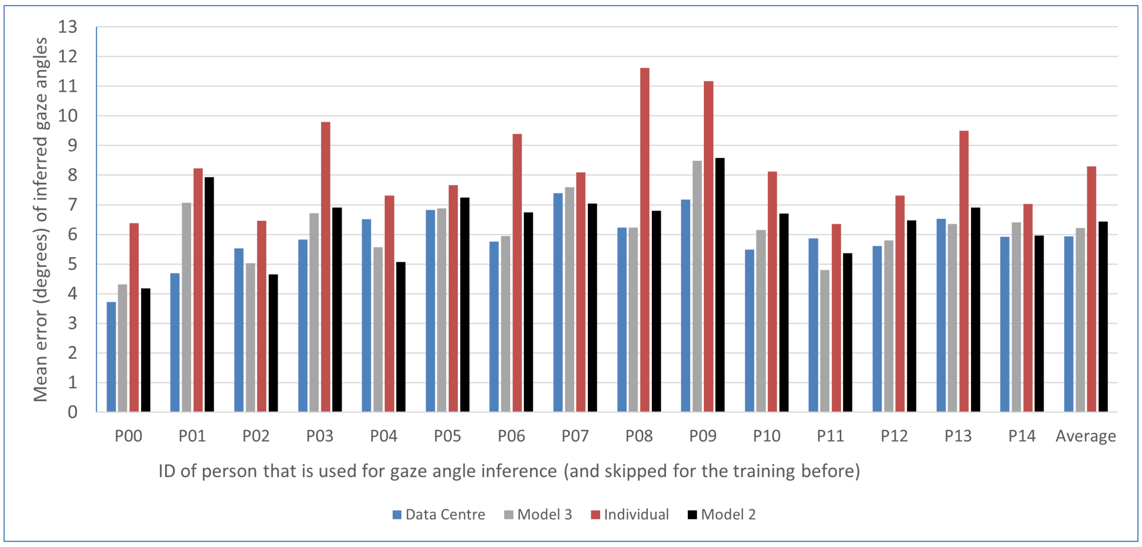


Figure 4.3: Comparison of prediction errors from Model 3 (SFL), Data Centre Learning, Individual Learning (all with ResNet model) and Model 2 (errors repeated from before)

4.3 Multi-Party Computation

In this section, we use the SPDZ protocol [14] for implementation and benchmarking of our configurations with MPC on the LeNet model. First, we show the results for MPC for the inference phase, then we refer to MPC for model training and finally, we show results for the secure aggregation with MPC.

Results: MPC for inference

Table 4.1 shows the computational requirements for forwarding secretly shared images through the LeNet with secretly shared weights and bias values. Here, we first restrict benchmarking to the online phase only and skip the offline phase in SPDZ by immediately using already created beaver triples. In Table 4.1, the number of required triples as well as integer opens are an upper-bound estimation given by the SPDZ program and we can see that both values are proportional to the number of forwarded images and they do not depend on the number of MPC parties.

Table 4.1 shows that the runtime for inference depends on the number of MPC parties, the number of forwarded images and whether we use GPUs as well. For 2 parties, we can see an almost linear increase in runtime with the number of images (approx. 53 seconds per image on a CPU and 30 seconds with GPUs). The linearity also holds for 3 parties and 4 parties with significantly increased inference time per image. The output gaze angles have been consistent with the results without MPC in all our experiments, we solely encounter a very small difference (smaller than 10^{-4}) which could be due to rounding within the MPC protocol.

Now, we want to analyze the computational requirements for the offline phase. In general, the number of required triples depends on the structure of the network and the size of the input (number of pixels). All observed implementations for MPC training and inference on a CNN[2] show the difficulty that they require an extensive offline phase where a large number of triples has to be created. Also for our offline phase for inference, a large amount of triples has to be generated (see Table 4.1). The actual experienced runtime for offline- and online phase together for inference of a single image was about 2:45h with just 2 MPC parties. Therefore, we can see that the runtime for the offline phase is significantly larger than the time for the online phase (54 seconds). On one hand, this confirms that the offline phase is an important component that can reduce computational effort during the online phase. On the other hand, even one image requires a very high pre-computation time which does not allow scalability of our system to a larger number of images.

Results: MPC for model training

With previous observations for inference, we can conclude that it is not applicable to perform the whole training with MPC due to too high computational effort: Let us assume $30s$ for forwarding a single image with 2 MPC clients. If no MPC is used for client-side layers (see Section 3.3.1), then it still takes more than $15s$ to forward one image because MPC is used for the server-side part of the LeNet which includes more layers with more weights. In total, we can conclude with one month

[2] `https://github.com/data61/MP-SPDZ`

MPC parties	Input	Device	Runtime	Triples	Opens
2 parties	1 image	CPU	54 s	17 799 758	104 212
		GPU	32 s		
	2 images	CPU	108 s	35 599 516	208 425
		GPU	62 s		
	3 images	CPU	165 s	53 399 329	312 640
		GPU	90 s		
	4 images	CPU	225 s	71 199 032	416 849
		GPU	115 s		
	5 images	CPU	264 s	88 998 845	521 064
		GPU	146 s		
3 parties	1 image	CPU	476 s		
		GPU	461 s		
	2 images	CPU	967 s		
		GPU	912 s		
	3 images	CPU	1455 s		
		GPU	1376 s		
4 parties	1 image	CPU	685 s		
	2 images	CPU	1362 s		
5 parties	1 image	CPU	893 s		

Table 4.1: Benchmarking of inference with MPC. The table shows observed runtimes for the online phase as well as an upper bound on required triples and integer opens depending on the configuration (number of parties, input images and used device)

as a lower bound for one epoch of the online phase only. Still, this number does not include the much more time-consuming offline phase as well as additional time for backpropagation with MPC. So even despite a complete MPC training would give us a lot of additional privacy properties (see Section 5.1), we cannot deliver an implementation that is suitable for practical use and we have to reduce the amount of required MPC operations.

Overall, we encountered high runtimes for inference with MPC and extremely high runtimes for training with MPC. This presents a significant challenge in maintaining the system's usability for the MPC implementation. Our limited usability of MPC underlines the need for the other privacy-preserving techniques for model training (e.g. SFL with PixelDP) so that our system can still ensure the secrecy of the input images.

However, there is a lot of work in progress with continuously more efficient protocols that may give more possibilities for training and inference with MPC. Model pruning can be applied (see Section 3.5) to additionally decrease the computational effort by reducing the high number of triples required for ciphertext multiplications.

Results: MPC for aggregation

Now, we limit the use of MPC to the secure aggregation (FedAvg operation). For this setup, we used the Mascot protocol as well as the protocol from Damgård et al. [4] for client I/O delivery. For the latter protocol, we used an existing MPC implementation for the credit rating of Danish

Setup	Runtime	Error p00	Error p01	Error p02	Error p03
No MPC	10 min	3.7274	7.9252	4.2507	6.2695
3 parties	21 min	3.6731	7.9376	4.2696	6.2593
4 parties	33 min	3.6731	7.9376	4.2696	6.2593

Table 4.2: Runtimes (offline- and online phase together) and inference angle errors with and without MPC for the FedAvg execution

banks from the SPDZ repository[2] and we applied the implementation to our use case. Then, it took us a little more than 20 minutes to execute 10 training epochs for SFL with 3 MPC parties on the LeNet model (offline- and online phase). This corresponds to an approximately doubled runtime due to the use of MPC. The output weights of the FedAvg operation throughout corresponded to the weights obtained without MPC except for minimal differences (all smaller than 10^{-5}) due to rounding. This leads to small differences in the final inference errors that are shown in Table 4.2 along with the runtimes. Here we have to mention that for all test cases, training was executed with Model 2 by leaving-out p01. Therefore, the presented inference errors on p00, p02 and p03 are lower than in Figure 4.2 as these persons were also part of the training set. This is only done once here as we want to emphasize the small differences of errors rather than the exact values here.

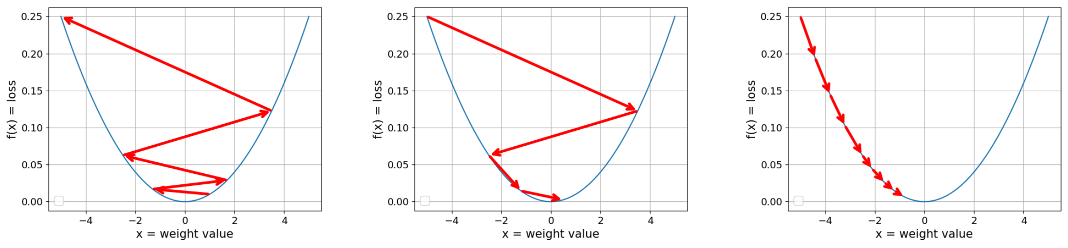
In summary, we cannot use MPC for an entire secure model training but we can use it efficiently for secure aggregation with acceptable runtime overhead and with almost equal angle errors. This makes MPC a useful additional component to increase the privacy guarantees as further discussed in Section 5.1.

4.4 Differential Privacy

In this section, we consider the DP mechanisms from Section 3.3.2. First, we describe a major difficulty that we have encountered when applying DP and which countermeasures we could apply to handle this issue. Afterwards, we give test results of our experiments for SFL with DP. Finally, we give the results of the attacker (Section 3.4) to demonstrate the effectiveness of our DP measures.

4.4.1 Dealing with large noise multipliers

For an increasing amount of noise (larger noise multiplier for DP-SGD resp. larger multiplier l for PixelDP), our program throws exceptions during the training phase. In particular, already $l \geq 0.1$ causes a vicious circle where a high loss value causes large weight updates which takes us even further away from a low-error-optimum. This issue is visualized in Figure 4.4a where weight changes are shown with red arrows and we can see that the optimum (here weight $x = 0$) is always overshooten quite far so that the loss increases even further. Note here that the figures are used to illustrate the problem but they reduce the problem to a single weight value on the x-axis whereas in reality, we have a multi-dimensional weight tensor and multiple weight updates happen in parallel.



(a) Learning rate too large **(b)** Learning rate appropriate **(c)** Learning rate too small

Figure 4.4: Development of the loss for different learning rates

The increasing loss in our program leads to the case where the loss value eventually got $\pm\infty$ or the large weights lead to sensitivity $\sigma = \infty$ and in both cases our training aborts and throws an exception. The value denoted with ∞ occurs in a python program after an overflow due to arithmetic operations with very large floating-point numbers. To avoid this erroneous case and still allow training with larger noise, the following strategies turned out to be useful (applied to the LeNet model):

- Decreasing the learning rate results in smaller weight updates and thus weights converge towards the low-error-optimum as shown in Figure 4.4b. But a smaller learning rate also slows down the convergence speed and we need more time for the overall training. Therefore, too small learning rates (as in Figure 4.4c) should be avoided as well. In any case, we can still not ensure an upper bound on model weights and sensitivity regardless of the chosen learning rate and our divergence problem (for increasing noise) persists.
- The normalization of client model weights turned out to be useful. As described by Lecuyer et al. [17], clients can normalize their weights in the pre-noise layer (divide by σ) every time a client has updated its local weights (i.e. after backpropagation). With this division, the sensitivity gets bound by 1 which also bounds the amount of noise that needs to be added. Now, $\sigma = \infty$ is not possible anymore and also the weights in the convolutional layer of client models are bound and do not diverge to $\pm\infty$ anymore. But a large amount of noise still causes exceptions because there is no restriction on model weights of the server yet and with large updates to server weights, the loss value can still reach $\pm\infty$.
- Our final step also restricts the weights of the server model by avoiding large weight updates. We now apply gradient clipping to the weights of the server model instances by scaling them down to a specific maximum norm (e.g. 1.0) after an update of a server model instance. So even in case of large loss values, the extent of the server model update is limited and we can now avoid divergence for both client and server model weights.

Of course, increased noise still leads to more inaccurate predictions (increased angle errors) and this relation cannot be avoided and leads to the fact that we should not add arbitrary large noise values. But in simple words: Using the presented measures, we force our model to keep searching for a good solution instead of giving up and terminating with exceptions. In the following course, we apply the latter two presented measures (bounding sensitivity and clipping gradients) for all test cases with included DP mechanisms (PixelDP or DP-SGD).

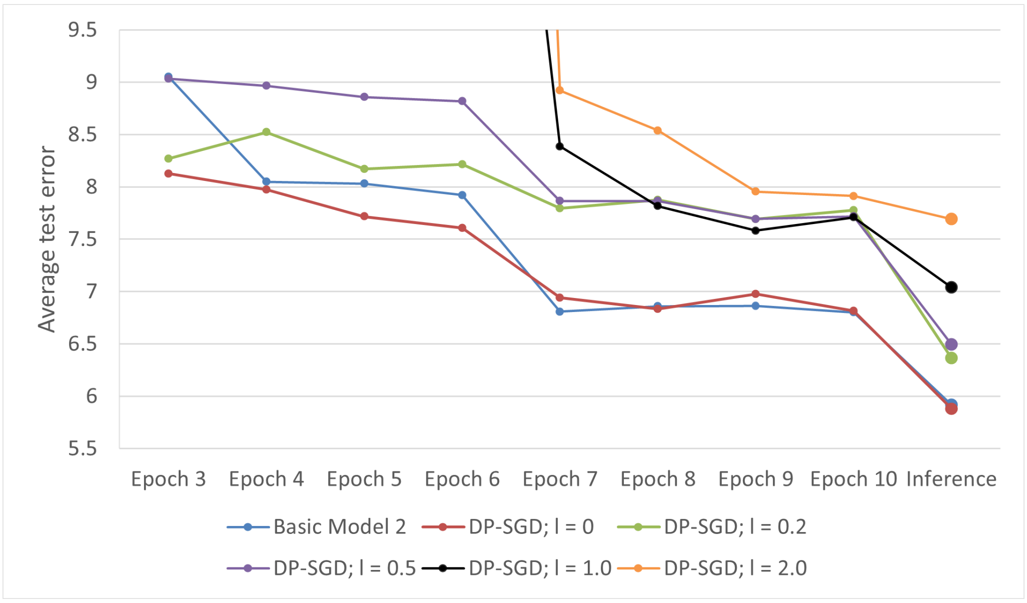


Figure 4.5: Average test error during training and inference phase for Basic Model 2 (no noise) compared to Model 2 with DP-SGD noise ($\epsilon = 0.5$, $\delta = 10^{-5}$) and varying amount of noise l for smashed data

4.4.2 Benchmarking with DP measures

In the following, we add noise using Opacus for DP-SGD as well as using PixelDP for smashed data. We determine the intensity of noise with the user parameters ϵ, δ (Opacus) and l (PixelDP) as stated in Section 3.3.2. To do so, we run configurations with $\epsilon = 0.5$, $\delta = 10^{-5}$ and varying $l \in \{0, 0.2, 0.5, 1.0, 2.0\}$ corresponding to the noise applied in [27]. For each of the configurations, we ran training and inference with Model 2 (as in Figure 4.2) but we limited ourselves to the first four cases (P00, P01, P02, P03) due to the large number of overall test cases. After each completed epoch during the training, we measure the test error (i.e. average error on the person that was left-out for training in the respective configuration), and subsequent to the training, we also measure the test error in the inference phase.

To do so, we measure the test error during the training for each client separately after their global epoch and just before performing FedAvg and we visualize the average error across the client-specific errors and across all four test cases (P00, P01, P02, P03) in Figure 4.5. Inference is, as before, executed on the model after performing FedAvg on client and server model instances. For inference, we also add noise to the smashed data with the same factor l as utilized during the previous training phase.

Due to the two applied DP techniques, the runtime for the training has approximately doubled to 22-23 min and this increase mainly results from sampling of noise for smashed data. Figure 4.5 shows that Opacus can obfuscate gradients without measurably increasing the training or inference errors as the error does not increase for DP-SGD (without noise for smashed data, i.e. $l = 0$) compared to the Basic Model. This is also caused by the applied measures from Section 4.4.1 (gradient clipping and bounded sensitivity) which do not lead to increased errors and we can even throughout recognize that the test errors converge slightly more straight for the red line. Only with lower ϵ, we could detect larger errors, e.g. the average inference error was about 6.4 for $\epsilon = 0.01$, $\delta = 10^{-5}$ and $l = 0$.

With added noise for the smashed data ($l > 0$), the test errors rise as clearly visible in Figure 4.5. Especially $l \geq 1$ leads to greatly increased errors for epochs 3-6 which are no longer fitting into the chosen range for our diagram. Nevertheless, we can see that the error converges to a similar level after 8-9 epochs as for $l = 0.2$ and $l = 0.5$. This is very surprising at first glance but we can explain it in the way, that the network learns how to get along with more noise and it makes the first convolutional layer strengthen meaningful features so that they "survive" the following obfuscation (noise addition). Nevertheless, we can recognize the rising inference error for $l \geq 1$. We suspect that this is because the client- and server model instances react with weight adjustments to get along with more noise but these useful adjustments are lost with the FedAvg operation to some (small) extent. This leads to similar test errors for epoch 10 (before FedAvg) but increased test errors for inference (after FedAvg). But still, for all our configurations, the average inference error is lower than the average test error of epoch 10. It can be deduced that the final FedAvg operation further decreases the errors and therefore this operation should always be executed to obtain a final CNN with minimal prediction errors.

4.4.3 Results of the attack

Now, we present the results of the attack (Section 3.4) to empirically show how our DP measures can enhance the privacy of the client data in SFL.

The results are given in Table 4.3. We fixed one normalized eye image of p02 and one normalized eye image of p00 and show them along with the reconstructed eyes that are output by the attacker (from the given corresponding smashed data). In addition, we give the average train loss (considering all p01,...,p14) as well as the average test loss on p00. Without noise ($l = 0$), the results clearly show that the attacker can reconstruct some features of the input eye images. In particular, the viewing direction (left/right) is throughout consistent with the original image (in Table 4.3: p02 looks to the left and p00 looks to the right). However, individual pixels remain, which obviously do not match. In terms of loss values, the attacker performed even better on p00 than on the training set. We guess that this is related to the fact that the predictions of Model 2 on p00 are also very accurate (see Figure 4.2).

With noise ($l = 2.0$), our attacker begins to always output the same eye image during inference (the neutral, forward-looking eye, see Table 4.3). The loss values have significantly increased and even on the training set, the output is throughout the neutral eye and therefore, we see that the attacker does not manage to draw any input-specific conclusions from the smashed data. We can conclude that the noise prevents our attacker from reconstructing input images.

In summary, we have empirically shown that PixelDP helps to protect the smashed data and makes it harder to draw conclusions about the input images. Nevertheless, we would like to emphasize that this empirical analysis does not guarantee that reconstruction attacks are generally avoided with $l = 2.0$ since we do not claim that our attacker fulfills certain requirements considering their abilities.

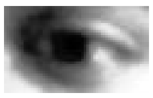
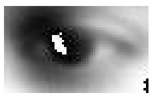
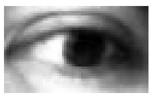
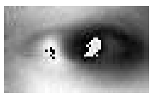
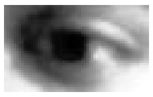
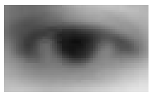
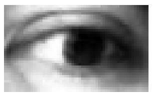
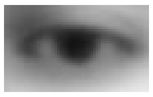
PixelDP noise?	Sample normalized eye (p02)	Reconstructed eye (p02)	avg. train loss	Sample normalized eye (p00)	Reconstructed eye (p00)	avg. test loss
No, $l = 0$			0.019			0.014
Yes, $l = 2.0$			0.049			0.043

Table 4.3: Performance of inverse CNN on p02 (in the training set!) and on p00 (left-out during training!) both with and without PixelDP noise

4.5 Model pruning

Now, we want to show the test results for model pruning and its influence on the model size and angle errors. We refer to Basic Model 2 again and we again execute 10 epochs of common model training (leaving-out p00), then we apply local pruning and afterwards, we store the resulting model in sparse tensor format. For the stored model, we measure the error for inference on p00 exactly as in Figure 4.2 (first bar in the chart).

First, we apply local pruning to all convolutional layers of the global client and server model with amount $p = 0.9$, i.e. we remove 90% of weights for each convolutional layer. The results are shown in Table 4.4. We can see that the pruned model parameters in the server model are only a small proportion (1.23 %) of all parameters because there remains a considerably larger number of unpruned weights in the linear layers. The client model reaches almost the amount p (90 %) of pruned model parameters and solely the unpruned bias tensors cause the slight difference between the number of zeros and the amount p.

Our original storage format (dense tensors) requires 11.1 MB for the server model and 3.13 KB for the client model. Due to the very small number of pruned weights in the server model, the sparse format requires 55.0 MB for the server model. The given amount p is also not enough to reduce the client model size. Moreover, the inference error increases dramatically and therefore we cannot benefit from pruning convolutional layers.

We repeat the same procedure for pruned linear layers instead (Table 4.5). Since the client model does not contain linear layers, it does not obtain pruned model parameters and the sparse format increases the model size from 3.13 KB to 19.6 KB. Therefore, the client model should be left in its original format (dense tensors) for minimal storage requirements. For the server model, however, the storage requirements decrease with increasing p and the size falls below the original value for dense tensors (11.1 MB). Since the angle errors increase only slightly, the utility of our gaze estimator is preserved.

Overall, we could show how to apply pruning to linear layers so that the model size decreases while the utility is preserved. This can be generalized to other model structures with contained linear layers. In this case, we can conclude that pruning helps to reduce the size of potentially large models

Amount p	Zeros server model	Zeros client model	Server model size	Client model size	Inference angle error
0.9	1.23 %	86.54 %	55.0 MB	3.82 KB	13.80

Table 4.4: Results with local pruning to convolutional layers in client model and server model

Amount p	Zeros server model	Zeros client model	Server model size	Client model size	Inference angle error
0	0.00 %	0.00 %	56.3 MB	19.6 KB	4.18
0.9	88.75 %	0.00 %	6.87 MB	19.6 KB	4.29
0.95	93.68 %	0.00 %	4.13 MB	19.6 KB	4.43
0.99	97.62 %	0.00 %	1.93 MB	19.6 KB	5.24
0.999	98.51 %	0.00 %	1.43 MB	19.6 KB	7.21

Table 4.5: Results with local pruning to linear layers in the server model

(many linear layers or linear layers with many weights) and therefore the storage requirements can be reduced. This is also an important feature if one device stores multiple models and thus the individual model sizes add up to a large total amount.

4.6 Ablation Study

In the following, we investigate the effect of the different inputs, parameters, and architecture choices on our approach.

4.6.1 Effect of the head pose processing

The head pose, which indicates the posture of the head as a 2D angle, is a useful information that is supposed to improve the accuracy of gaze angle predictions as shown in the results from Zhang et al. [33]. In previous test cases, we transferred the head pose to the server and the server used it as additional input to the final linear layer exactly as in the model in [33]. Now, we also give test results for Model 2 when not considering the head pose, i.e. performing gaze estimation solely with normalized eye images as input to the CNN. To do so, we take three configurations from Figure 4.5 and run training and inference in the same way but without considering the head pose.

The results are shown in Figure 4.6. As can be seen in the diagram, we could (surprisingly) not determine any deterioration in performance for our tests without utilized head pose information. Instead, the test results are similar, with the error sometimes being slightly lower and sometimes slightly higher than before. This shows that our model can also make accurate predictions that are based solely on the normalized eye images. However, it is not possible to transfer the statements from Zhang et al. [33] to SFL that the head pose makes the predictions more accurate. This may be since during one epoch of SFL training, different strategies are learned for the different client-specific instances of the server model on how to integrate the head pose information of a

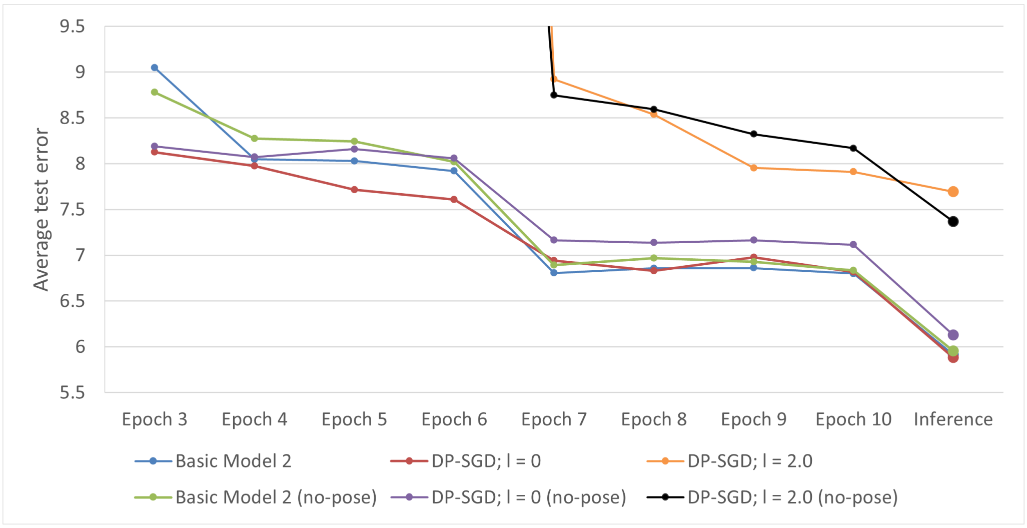


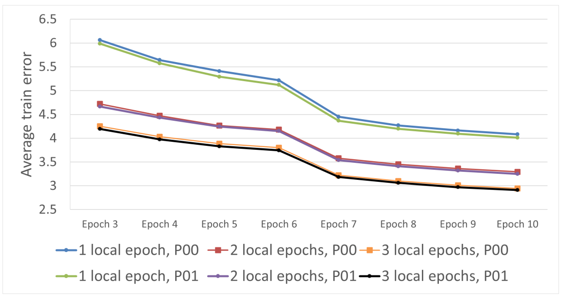
Figure 4.6: Average test error during training and inference phase for Model 2 with considered head pose (results repeated from benchmarking with DP) and without passing the head pose to the CNN (label: no-pose)

specific client. The learned strategies could be (partially) lost during subsequent FedAvg operations which leads to the fact that the head pose information does not cause any improvement for our test cases.

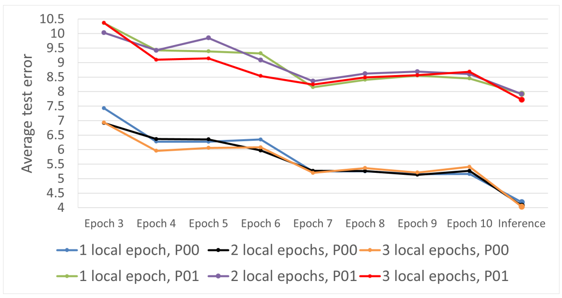
In general, it is clear that it depends on the underlying dataset whether accurate gaze estimation is possible without considering the head pose. For the Columbia dataset [24], participants had to turn their head horizontally at a given angle resulting in different head poses of fixed and known angles (0°, ±15°, ±30°). Due to the strongly varying head pose, there are varying eye shapes that are visible in the pictures [24]. By looking at the images, we can ascertain that turning the head with a certain angle leads to just this gaze angle if the participant does not counteract it by eye movement. Therefore, the known head poses are important to perform accurate gaze estimation for the Columbia dataset. On the other hand, since the participants for MPIIGaze look at their screen and no varying head pose is forced, we assume that this contributes to our ability to perform accurate gaze estimation even without passing the head pose to the CNN. The results in [26] (for their unsupervised learning method) comply with our statement: The gaze angle errors for Columbia are clearly smaller with considered head pose information (decreasing by 8% on average) while the errors for MPIIGaze differ only slightly (< 0.01% on average).

4.6.2 Effect of the number of local epochs

So far, we used one local epoch per client, i.e. the FedAvg for the client models and the server model instances is executed directly after each client passed their samples exactly once through the network. Now, we increase the number of local epochs while we keep the number of global epochs the same. The results are shown in Figure 4.7. The time for training was 10 min with one local epoch and 17 min and 23 min with two resp. three local epochs.



(a) Training



(b) Testing

Figure 4.7: Train and test errors for varying numbers of local epochs (once with P00 and once with P01 left-out during training)

Figure 4.7a shows that with an increasing number of local epochs, the average training error becomes considerably smaller but we cannot detect any significant change in test errors (Figure 4.7b). This is because during local epochs, the client and server model instances are not synchronized and the clients improve on the training set but this does not come along with better generalization. For better performance on unknown data samples, a sufficient number of global epochs is the key and the FedAvg operation is needed frequently to synchronize the models. The extreme case (only local epochs and no synchronization) would correspond to Individual Learning and, as already shown, leads to poorer performance.

4.6.3 Effect of modifying the network split

Now, we want to investigate how a different model split affects the accuracy of the predictions. We take the ResNet (Model 3) consisting of 18 layers which gives us the possibility to move layers from client to server and vice versa. To obtain Model 3a, we move one part of the end of the client model to the beginning of the server model and this part consists of multiple layers including two convolutional layers. For Model 3b, we do it the other way around and move the same part from the beginning of the server model to the client model. So in both cases, the composed model stays the same.

The results (Table 4.6) show that the errors only vary slightly. If we take a closer look at Algorithm 2.1 (without further DP mechanisms!) then it is possible to see that when varying the split, every forwarded sample still passes the same layers which leads to the same loss and weight updates. The weight updates also stay the same regardless of whether a layer is contained in the FedAvg operation of the client or the server as the FedAvg operation averages every weight separately. Differences in the results in Table 4.6 are due to random decisions (e.g. initialization) for different splits and due to specific implementation details of Pytorch optimizers as we use one optimizer for the client and one for the server.

We go one step further and consider the case where the split is such that the server owns no part of the model at all. Then the main server is no longer in use and the whole models are owned by clients and averaged by the aggregator which gives us exactly an implementation of FL and we realized this implementation for Model 2. Now, we have to expect that our previous observation also applies so that we get similar prediction errors with FL on Model 2 compared to SFL. Table 4.7

Model	Error P00	Error P01	Error P02	Error P03	Error P04	Error P05	Average
Model 3	4.312	7.069	5.024	6.723	5.566	6.883	5.929
Model 3a	4.246	6.897	5.051	6.979	5.938	6.519	5.938
Model 3b	4.449	7.310	5.203	6.747	5.402	6.413	5.921

Table 4.6: Comparison of prediction errors of Model 3 with Model 3a (one part moved from client to server) and Model 3b (one part moved from server to client) considering P00,...,P05

Method	Error P00	Error P01	Error P02	Error P03	Error P04	Error P05	Average
SFL	4.1829	7.9252	4.6522	6.9007	5.0672	7.2435	5.9953
FL	4.1803	7.8675	4.9357	6.9907	4.8748	7.1230	5.9953

Table 4.7: Comparison of prediction errors of FL and SFL using Model 2 considering P00,...,P05

shows the prediction errors for FL along with the repeated errors for SFL and, as expected, we obtain only slightly different prediction errors due to randomness and implementation details again. We can basically consider FL as a special case of SFL with one specific split so that the whole model belongs to the client. With FL, the runtime for the model training falls from about 10 to 8 minutes.

4.6.4 Effect of averaging the smashed data

The SFL approach, which we focus on in this work, is based on the averaging of model updates from different clients. In this section, we make an exception to this and we average the smashed data of different clients instead. For the modified model training, there is a single (immutable!) client model instance and each client owns a copy of this instance through which they forward their eye images. The model output (smashed data) and the associated label (target gaze) are throughout passed to the aggregator. The aggregator calculates the average of the smashed data and the average of the labels from different clients. Then, the aggregator passes both averages to the main server. The main server trains a single server model instance on the averaged smashed data as inputs and the averaged labels as targets. We use the original LeNet model (Model 2) for our client and server model.

In Table 4.8, we show the average inference errors after 10 training epochs without PixelDP noise ($l = 0$) as well as for training with PixelDP noise on the client model ($l = 0.2$ and $l = 0.5$). Here, we experienced larger angle errors on average than for SFL. Especially with the additional PixelDP noise, the errors vary strongly and they reach large values (up to 19.81 degrees). This is because the client model is a fixed transformation function and it cannot adapt to the noise with weight updates (in contrast to SFL). On the other hand, the training can be carried out efficiently (approx. 3:30 min without PixelDP and 7:00 min with PixelDP) and therefore, the overall training times are lower than for SFL.

An extensive privacy analysis still needs to be performed for this approach. The averaging of the smashed data definitely enhances the protection of individual smashed data from a reconstruction attack by the main server. Nevertheless, it has to be investigated how the individual smashed data can be protected from the aggregator, especially if the PixelDP protection is not practicable (due to

PixelDP noise	Error p00	Error p01	Error p02	Error p03	Error p04
l = 0	5.4755	7.7035	5.4961	8.2022	6.4427
l = 0.2	7.3614	8.5412	11.1186	10.07504	8.8992
l = 0.5	8.4981	8.6645	9.0516	19.8100	9.0610

Table 4.8: Prediction errors after training with smashed data averaging on Model 2 considering P00,...,P04

performance degradation). For the smashed data averaging, an MPC protocol could be useful again (exactly as for the aggregation of model updates in SFL). Then, the individual smashed data can be protected without influencing the performance.

Also regarding the performance, modifications of the utilized metadata and further variations of our concept should be investigated. For example, clients could sort their (input, label)-pairs regarding the labels (using a clustering method) so that the clients forward images with similar gazes at the same time. Then the resulting (smashed data average, label average) pairs could be even more suitable to train the server model which could lead to performance improvements.

4.6.5 Effect of the model training with unsupervised learning

Now, we want to give test results when applying FL to gaze representation learning as shown in Section 3.2. The Cross-Encoder from [26] is a comparably large network as the Encoder part, which is a ResNet-18, has already many layers that lead to high time requirements for training. Therefore we limit ourselves to just 3 epochs here and after each epoch, clients synchronize their weights with the FedAvg operation. Subsequently, we execute 5 training epochs for the gaze estimator with 100-shot learning using only the images of P00 (Individual Learning). As for supervised learning approaches, we fix one client that is left-out during training which includes both Encoder-Decoder as well as gaze estimator training here. After the 5 epochs of 100-shot learning, we measure the angle error of the gaze estimator regarding the images of the left-out client that were not used for training. The results are given in Table 4.9.

We can recognize that the angle errors increased significantly compared to previous supervised learning results. Therefore, we can see another trade-off here: The advantages of unsupervised FL (no gaze annotations needed) come along with reduced prediction accuracy.

In Table 4.10, we show the 3D angle errors for the MPIIGaze dataset when using gaze representations learned by alternative unsupervised gaze representation learning methods. The reported results are taken from [26] and we want to discuss the alternative methods roughly. ImageNet is another dataset that was used to train a ResNet-18 to learn gaze representations. The Auto-Encoder has the same architecture as the Cross-Encoder but it is trained on single images that are forwarded separately (instead of throughout combining encodings of two images). EFC (equal feature constraint) forces features to be equal by adding the L1 loss as an additional constraint to the Auto-Encoder. SimCLR [3] and BYOL [9] are two popular representation learning approaches that both belong to so-called Contrastive Learning methods [26].

Based on the reported results, we can see that our privacy-preserving approach is quite competitive with other unsupervised methods that do not preserve the privacy of gaze data. However, we can see that the integration of privacy-preserving FL results in significantly larger errors than the original

Left-out	P01	P02	P03	P04	P05	P06	P07	P08	P09	Average
Angle error	8.73	9.72	10.15	8.36	10.45	9.95	10.70	10.49	9.82	9.82

Table 4.9: Prediction errors (3D gaze angle errors) on the MPIIGaze dataset for unsupervised federated gaze representation learning

Method	ImageNet	Auto-Encoder	Auto-Encoder (EFC)	SimCLR	BYOL	Cross-Encoder	Cross-Encoder FL (ours)
Angle error	10.6	9.5	9.2	10.0	11.1	7.2[3]	9.82

Table 4.10: Prediction errors (3D gaze angle errors) on the MPIIGaze dataset for several unsupervised gaze representation learning techniques

Cross-Encoder implementation from [26]. Here we would like to point out that, based on our approach, experiments with additional adjustments and optimization techniques (e.g. Adaptive FL) can potentially lead to reduced prediction errors. With further work, it might be possible to get closer to the accuracies without privacy-preserving FL or SFL (Data Centre approach), exactly as we have managed in the case of supervised learning.

[3]The authors report slightly varying errors in the range 7.2-8.2 depending on the chosen feature dimensions (d_g,d_e)

5 Discussion

Now, we discuss the privacy guarantees that can be achieved depending on the utilized configuration (whether MPC and/or DP mechanisms are applied). Afterwards, we analyse the experienced runtimes for model training and we evaluate the various configurations in terms of their usability in practice.

5.1 Privacy

In this section, we evaluate to which extent desired privacy properties can be ensured by our system. We focus on the following essential properties that are also widely discussed in other current works for privacy-preserving gaze estimation: Model privacy, robustness to data leakage attacks, head pose and target angle privacy as well as robustness to model poisoning attacks.

Model privacy

Model privacy is a useful property that can be achieved with SFL [27]. The server can keep the trained server model secret from the clients and this gives SFL and SL an advantage over FL because in FL, clients throughout own the whole model during federated training. SL and SFL are therefore perfectly suitable if the main server is seeking exclusive ownership of the model.

The other way around, SFL also offers the possibility to hide the global client model from the main server to avoid a single owner of the whole model. To achieve this, the (low-cost) client model aggregation (FedAvg) can be performed locally by a client or a third party instead of the main server [27]. Nevertheless, the risk of a client that cooperates with the main server rises in large-scale systems with a large number of clients. All clients receive the global client model from the aggregator in each round and if the main server gets access to the global client model, then the server owns both parts (the whole model). However, the privacy of the local client models is still ensured if the aggregator is honest because the aggregator is the only participant that can access the local client models (besides the respective client itself).

If MPC is used for the aggregation, then model privacy is strengthened by completely protecting the local client models from external access. We can rely on the security properties provided by the MPC protocol which ensures the secrecy of local client models if the tolerated number of dishonest parties is not exceeded. Then, the SFL clients solely receive the output of the MPC protocol (the global client model), and nobody except for the respective client itself can access the local client model. The ensured secrecy of a local client model strengthens protection against data leakage attacks on this client as we discuss further in the following paragraph.

Robustness to data leakage attacks

An attacker might try to perform data leakage attacks [19] to gain personal information about the training data. In our case, this covers information about the persons depicted on the private face or eye images that were used to train the CNN.

First, we consider the case that the main server tries to perform attacks during the training phase in SFL. The main server can perform an attack using leaked data in the following cases:

a) The global client model gets revealed to the main server by a malicious client or the aggregator.

b) The local client models are not sufficiently obfuscated from the aggregator, such as using DP-SGD. Also, MPC for aggregation is not applied.

c) The smashed data is not sufficiently obfuscated, such as using PixelDP.

In all of these three cases, the main server could try to execute attacks with the leaked information. If only case a) occurs, i.e. clients' local models and smashed data are still sufficiently protected, then we claim that the privacy properties are still similar to the properties given by PrivatEyes [6]. The server can only utilize aggregated update information from the client models to perform a data leakage attack while individual client updates cannot be used.

But if case b) or c) occur, then the main server or the aggregator gains information about individual clients' updates or smashed data which harms the privacy protection:

Elfares [6] shows that data leakage attacks on FL have higher chances of success than on PrivatEyes due to known individual client updates. We guess that the protection from data leakage attacks for SFL in case b) corresponds to the protection given by FL because, in both methods, individual client weight updates get revealed to the server. With the leaked local client model (forwarded from the aggregator to the main server) in case b) and the ordinary ownership of the associated server model instance, the main server gets access to the whole model of this client. Therefore, the server has exactly the same information available as in FL where the whole model (or model update) is transferred to the server. This underlines the benefit of MPC for aggregation to avoid case b) in the same way as PrivatEyes can be utilized to protect individual client updates in FL with MPC.

In case c), the server could make use of attacks that are designed for the case that smashed data is not protected (Section 3.4). A malicious server can arbitrarily modify the gradients that are passed back to the client and thus move clients into desired states [23] where the server can reconstruct the secret input data. Therefore, the main server has a good chance to infer information about personal data if b) or c) are true, especially for smaller client networks (with a small number of nodes) [27].

Just like the main server, the aggregator could also get curious and start to execute data leakage attacks only using local client models if they are not sufficiently obfuscated. Therefore as protection against both a curious aggregator and a curious main server, it makes sense for the clients to apply DP-SGD or to use MPC for the FedAvg operation to avoid that any information about personal data can be inferred from the individual weight updates of the clients. Both MPC for FedAvg and DP-SGD have exclusive advantages over the other technique: With MPC, we can give strong guarantees regarding the secrecy of individual model updates without raising the angle errors. With DP mechanisms, the individual client updates get obfuscated (instead of hidden). In addition, the level of privacy (ϵ, δ) must be carefully determined so that the utility of the model is still preserved (acceptable angle errors). On the other hand, the usage of DP-SGD makes the global client model (after FedAvg) include DP obfuscation as well and thus offers improved protection if only case a) occurs. We do not go into detail here and we skip estimating the magnitude of this improved protection. But we can conclude that it makes sense to combine both techniques to achieve strengthened protection against data leakage by keeping individual client updates private with MPC and by using DP for additional obfuscation.

So far, we focused on the training phase and now we discuss the feasibility of data leakage attacks during the inference phase. We focus on the protection of the training data and we want to avoid that the training data can be deduced from the trained model. If we assume that there is just a single owner of the overall trained model, then solely this owner can execute white box attacks. This model owner can complicate black box attacks from other participants by limiting the number of permitted queries on the target model and by preventing others from performing arbitrary queries on the model.

Nevertheless, we implemented SFL based on the assumption that we cannot trust any single instance (Data Centre). But we can still rely on the fact that data leakage attacks on training data are more difficult during the inference phase than during model training: Elfares [6] gives evidence for this with the implemented data leakage attacks that throughout have lower chances of success on the final trained models than on given models at the beginning of the training. Especially the model inversion attack shows significantly better results on the models after the first training round than on the final trained model. As before, we strongly assume that these realizations can also be transferred to SFL due to the related weight updates during FL and SFL (see Section 4.6.3).

In addition, DP-SGD can defend against some attacks during the inference phase (e.g. membership inference) [19] and we expect that the same holds for PixelDP. Still, the increased protection for the training data with DP has to be weighed up against increasing angle errors during the inference phase.

Head pose and target angle privacy

The core of this work is the confidentiality of the clients' personal eye images. But, in addition, also the head pose can be considered as private information and the protection of this information by the clients might be desirable. For example, one could draw the direct conclusion that the underlying person has a certain problem or illness due to unusual or unchanging head poses. The head pose could also provide information about where a person's attention is directed. But, even more accurately, the labels (target gaze angles) contain information about personal attention. For MPIIGaze, a target angle provides information about the on-screen position that the depicted person is looking at [33]. This information can be used further, for example, to provide personalized advertisements to the user. FL offers the advantage that the whole model training is performed on the client and therefore no head poses or labels need to be transferred to the server. In Table 5.1, we give an overview of approaches that can be applied to protect head poses or gaze labels in SFL and we discuss these approaches now step by step.

In our proposed unsupervised learning approaches with SFL and FL (Section 3.2), the federated training of the Cross-Encoder does not utilize any head poses or gaze annotations. The subsequent 100-shot learning utilizes poses and gaze labels but it is performed locally by a single client (Individual Learning). Therefore, the privacy of gaze labels and head poses is preserved during the

Protection of	Unsupervised learning	Add one split	Training with MPC	Skip pose processing	Label obfuscation
head poses ?	yes	yes	yes	yes	no
gaze labels ?	yes	yes	yes	no	yes

Table 5.1: Comparison of different approaches for head pose and gaze label protection

entire model training in our unsupervised learning approaches. However, the enhanced protection of personal client data in both supervised Individual Learning and in our unsupervised learning approaches led to increased average angle errors compared to the standard (supervised) SFL.

Thapa et al. [27] propose another model structure for SFL by adding one more split to the model in SFL. Each client owns the first and the last few layers of the model and only the remaining middle layers are owned by the server. In this case, the client does not have to share its labels with the server. In addition, the client can pass the head pose directly to the final network layer and therefore the secrecy of the head pose is also preserved.

Alternatively, MPC can be applied during the training phase and during the inference phase to secure poses and labels with secret sharing. Except for the client that secretly shares the own values (poses and labels), no other participant gets any information about the underlying plaintext values during the MPC protocol. Therefore, the secrecy of head poses and gaze labels is preserved during both phases (training and inference) with MPC.

Both options (MPC as well as one more model split) are accompanied by drawbacks: MPC strongly increases the computational requirements and we could not create a feasible implementation for model training with MPC. Another model split, on the other hand, dramatically increases the computational and storage requirements of the clients. This is clearly shown by the different model sizes of the client model and the server model (Table 4.5). Despite the model size reduction due to pruning, the linear layers of the server model are still an order of magnitude larger. Therefore, the addition of one more split is not feasible if the required resources of the client should not increase.

Skipping the head pose processing (Section 4.6.1) is an obvious solution to completely hide all head pose information from the main server. We could see that this does not deteriorate the inference performance for the MPIIGaze data. As discussed in Section 4.6.1, the head pose information is generally more important if the gaze estimator is applied to images with strong head rotations (e.g. Columbia). Then, it should be decided carefully whether increases in errors should be accepted or whether the secrecy of the head pose is actually required.

There are recent approaches that protect the labels in SL by obfuscation. MALM (Multiple Activations and Labels Mix) is one approach by Xiao et al. [30] that addresses both label and smashed data obfuscation. Accordingly, MALM aims to protect client labels and, at the same time, also aims to prevent reconstruction attacks. However, the authors focus on classification tasks and it remains unclear whether the approach can also be applied to gaze estimation on the MPIIGaze dataset as the dataset contains continuous gaze annotations.

Robustness to model poisoning attacks

So far, we focused on the main server or the aggregator as an attacker during model training. Now, we assume that both the main server and the aggregator are honest and a few clients are malicious. These clients try to decrease the overall accuracy of the model predictions (model poisoning attack).

Khan et al. [15] investigate model poisoning attacks in SFL for various models. In their attacks, the malicious clients modify their local client models before passing them to the aggregator. Our implementation that we have shown so far does not defend against this attack as clients could ruin the utility of the model by passing arbitrary weight tensors to the aggregator.

The authors show two possible defences: Trimmed mean and median. They can be applied by the honest aggregator to avoid outliers in each dimension of the client updates. With these defences, the influence of the exceptional client updates on the global client model should be limited. This restricts the options of the malicious clients to increase the prediction errors.

The success of the adversaries against these defences is mainly dependent on the model split: The smaller the model, the less efficient the attack. The attacks show the highest efficiency in FL since the client owns the complete model [15]. As we also showed in Section 4.6.3, the choice of the cut layer does not influence the performance [15] but a small client model is more suitable to prevent model poisoning attacks. This should be considered if the avoidance of model poisoning attacks in SFL is desired.

Still, the influence of the two defences on the performance of our gaze estimator has to be investigated. In addition, it would be interesting to see how our approach can prevent model poisoning attacks that arise from modified smashed data by malicious clients [15]. We guess that we already have a beneficial mechanism to address this type of attack by clipping gradients on the server model (see Section 4.4.1) which limits the magnitude of individual updates. Possibly, the main server can also try to detect model poisoning attacks by checking the model outputs: If the model was already partially trained, then the output gaze angles should not have utopian values far outside of the range of realistic values. Otherwise, the affected client may be blamed for carrying out a model poisoning attack.

5.2 Runtime

In the following, we want to discuss the experienced runtimes for model training which we executed on Google Colab[1] GPUs. The fastest training method is Individual Learning with about 1:30 min for 10 epochs as it considers only images of a single client to train the model. Data Centre Learning, on the other hand, takes images from all clients into account and we get an overall runtime of almost 6 minutes for 10 epochs.

With SFL, we propagate exactly the same images as in Data Centre Learning but it requires additional privacy-preserving components (multiple client-specific nets, FedAvg), and therefore the entire training time increases to about 10 minutes. We implemented SFL with a single-thread program and we guess that the parallelization of the clients with multiple threads would decrease the overall runtime if enough GPUs are available to allow parallelized training. On the other hand, the introduction of multiple devices and external communication in a real-world setting would definitely increase the overall runtime although we have not implemented this setup.

Using MPC for the aggregation in SFL doubles the training time to a little more than 20 minutes (3 parties, offline- and online phase). Similarly, the integration of both DP mechanisms (PixelDP and DP-SGD) leads to a doubled runtime. All together (MPC for aggregation and both DP mechanisms) resulted in a total time of 35 min. Still, we can conclude that all configurations that we addressed so far allow model training within a reasonable time.

This looks different if MPC should be used for secret training and inference on secretly shared input data and model weights. As already demonstrated in Chapter 4, our implementation has very large runtime requirements and we could not create a running implementation for model training with MPC. Even the inference of one single input image has proven to be very time-consuming (2:45h for offline- and online phase together). Therefore, we suggest that the use of MPC should be limited to perform the aggregation during model training and, instead of MPC, PixelDP should be used to protect the smashed data from data leakage. We can definitely state that the complete integration of MPC for model training and inference with reasonable runtime remains a great challenge. With pruning, we could reduce the model size by setting weights in linear layers to zero. However, we could only reduce the storage requirements but the overall runtimes did not decrease. Still, an implementation of pruning for MPC training and inference remains to be carried out. We suspect that pruning can considerably decrease the runtime of the MPC protocols by skipping all multiplications with pruned weights.

6 Related Work

In this chapter, we want to discuss alternative approaches that are suitable to enable privacy-preserving gaze estimation. There is an immense number of existing methods to protect the privacy in eye tracking applications [2] and the used techniques are often similar, e.g. obfuscation and DP are important building blocks for many approaches (including ours). We have to admit that the focus of our work is limited to one part of the existing work, namely Decentralized Learning where the sensitive data is not distributed but kept locally instead [2]. FL and SL are two major Decentralized Learning techniques and with their modification and combination, a variety of related techniques have emerged in addition to SFL. We discuss a few of these alternative techniques now.

The article by Gupta and Raskar [10] presents the original (basic) SL version and their approach is also called Vanilla SL. The authors show that with their implementation, the same model is obtained as with the Data Centre training on a single entity (correctness property). Further advanced architectures that are built upon SL, such as SFL, drop this correctness property. The authors present a security proof to show that it is infeasible for a curious server to invert the model parameters of the client. But they did not consider more advanced attack setups, e.g. by Pasquini et al. [23] and therefore, they did not apply sufficient protective measures such as DP against reconstruction attacks. Considering the privacy protection, the SL architecture has a disadvantage compared to SFL: The next client receives the weight update from the previous client so it gets direct access to the gradients whereas, in SFL, only the aggregator gets access to the individual updates. And, as realized using MPC for the aggregation, client updates can be hidden completely and efficiently in SFL. It is not possible to directly adapt this MPC approach to SL.

Blind Learning is another Decentralized Learning approach that is closely related to SFL. It uses the same architecture as SFL but a different loss function which computes the average of the client losses [20]. Like SFL, Blind Learning also relaxes the synchronization requirements of SL by allowing parallel client model training. In a distributed system, this prevents clients from being idle as it is the case for the relay-based training in Vanilla SL.

Han et al. [11] relax synchronization requirements even further. Their approach is built upon SFL but clients can already calculate their loss function immediately after forward propagation on their local models. In this approach, the “local loss” can be calculated in an independent manner by the respective client or server which leads to a further increased level of possible parallelization: Clients do not have to wait until the main server continues forward propagation and passes back the gradients and instead, clients can directly perform backpropagation using their locally calculated loss. However, the authors focus on reduced communication load and training times but they do not analyze the ensured level of privacy. We suspect that the ordinary DP mechanisms can also be applied to protect gradients and smashed data and thus, it is possible to get a certain level of protection against reconstruction attacks.

In our work, we throughout focused on the SFLV1 version from Thapa et al. [27] and we used this version for our implementation of the gaze estimator training with SFL. The authors also suggest another version (SFLV2) which uses a single server model instance for forward- and backward propagation of smashed data from all clients. In SFLV2, the clients forward their smashed data sequentially through the server model and therefore there is no more FedAvg operation executed on the server side. Depending on the dataset, the authors show that this version can reach a higher accuracy than SFLV1. The "forgetting problem" of Vanilla SL, i.e. the sequential processing of clients that causes updates of previous clients to get lost to some extent, is solved in SFLV2 by randomizing the processing order across the clients [5].

Duan et al. [5] compare various further Decentralized Learning techniques that can be applied for a distributed Internet of Things (IoT) environment with a focus on scalability to a large number of clients. They mention the Cluster-based Parallel SL method (from [29]) where devices are separated into multiple clusters and in the first stage, training is executed within one cluster in the same way as in SFLV2. After a round of this intra-cluster-training, the updated model is used as initialization for the next cluster. Therefore, in the second stage, the inter-cluster-training works similarly to Vanilla SL except that each client consists of a cluster of user devices [5].

Due to the high number of recent research publications, we cannot give a complete picture of the current research. Decentralized Learning is the subject of current research worldwide and new methods are constantly being proposed and investigated. With varying architectures related to the number of available clients and servers, there are even more possibilities. The authors in [5] describe a procedure that is applicable in the case of several servers that take part in the model training. Each client is assigned to a dedicated server and this client and server train a model together. At the end of an epoch, the server-side models are aggregated with FedAvg. This approach is called FedSL and the detailed implementation is described in [28].

For the protection of the smashed data, Duan et al. [5] enumerate several techniques that can be applied instead or combined with our utilized noise-based defence (PixelDP). For SL and SFL, the decorrelation method is another prominent defence for the smashed data [5]. It adds another loss function that measures the distance correlation between the input and the smashed data. Minimizing this loss function helps to further decouple the smashed data from the input and therefore it strengthens the protection against reconstruction attacks [5]. The additional loss can be optimized in combination with the ordinary loss function which is responsible for reducing the prediction errors. As already mentioned, MALM is another proposed approach by Xiao et al. [30] that mixes the activation outputs (for us: smashed data) which also reduces the distance correlation with the secret client data that should be protected. There exist several further methods that aim to avoid attacks on the client data in SL and SFL. SplitGuard [8] is a method where clients observe their training process and try to detect when they are targeted by an attack from the server. The idea here: The clients introduce fake batches during the ordinary training phase and the obtained gradients for the fake batches should have a higher magnitude than gradients for regular batches [5]. Based on the numerous current works, we are sure that the development of further sophisticated defence measures will be proposed in the near future to offer strengthened privacy protection for the privacy-preserving gaze estimation task.

7 Conclusion and Outlook

In this work, we designed a system that enables the privacy-preserving training of a gaze estimator. We focused on the prediction of the 2D- and 3D gaze angles for given face images. To minimize the prediction errors of the gaze estimator, we could show that the model needs to be trained on a large set of face images from a variety of different people. With the implemented SFL method, we could preserve the privacy of the personal images during model training by splitting the model (a CNN) between multiple clients and one main server. The clients own the personal face images that are processed in a pre-processing phase first. The pre-processing results in normalized eye images which the clients forward through their part of the model and the main server completes the forward propagation on its part of the model. Another lightweight server (aggregator) synchronizes the local model updates from the clients by averaging their weights.

Overall, the clients transfer the intermediate model output (smashed data) to the main server in SFL and therefore there is no need for the clients to share their images with anyone else. Clients only need to store and process a small part of the model which leads to a reduction of the required client resources compared to FL. In addition, we enable model privacy, i.e. the server may be granted the exclusive ownership of the complete trained model.

In the course of this work, we presented several defences to protect the smashed data and the local client models from the main server and the aggregator, respectively. We showed how to apply DP measures to obfuscate the smashed data and the local client models in order to prevent the main server and the aggregator from gaining information about the personal client images (reconstruction attacks). With our implemented attacker, we were able to empirically prove the benefit of the applied DP measures. We also investigated the possible integration of MPC to enhance the ensured privacy guarantees. While the application of MPC for the entire model training was not feasible due to too high computational effort, the use of MPC for the aggregation turned out to be an efficient method to prevent reconstruction attacks based on the local client models.

We trained and tested our system on the MPIIGaze dataset which contains the face images along with the empirically determined gaze angle annotations from 15 different persons. The difficulty of the gaze estimation task for this dataset is that the images were taken during everyday laptop use and therefore the images varied greatly in brightness and quality. In addition, the dataset contains persons with different appearances (gender, race, with or without glasses) and we could verify that the gaze estimator needs to be trained on images from several different persons to increase its generalization abilities. Therefore, our implementation of SFL was perfectly suited to train the gaze estimator on the MPIIGaze dataset. The final trained model showed only small inference errors (less than 6.5 degrees on average) and thus we could outperform other privacy-preserving training methods. The Data Centre training (without privacy-preserving mechanisms) showed only slightly lower inference errors on average.

We further focused on different pruning methods to reduce the size of the model as well as the overall training time. With pruning, we could reduce the size of the server model from 11.1 MB (stored in dense format) to 1.93 MB (stored in sparse format) by pruning 99% of all weights in the linear layers. In this case, the measured angle error regarding one fixed person increased by only slightly more than one degree (4.18° to 5.24°). Furthermore, we investigated how to train the gaze estimator with unsupervised learning while still maintaining the privacy of the personal client images. However, we have to admit that our measured average angle error for unsupervised learning (9.82°) is significantly higher than for the supervised SFL configurations.

Outlook

We have multiple ideas for the extension of our system that we leave for possible future work. It would be interesting to see how our SFL implementation performs on further datasets (e.g. Columbia). In this case, the metadata and network structure (e.g. number of channels) might have to be adjusted to minimize the prediction errors. In addition, it can be attempted to modify and apply our implementation to other tasks than gaze estimation (e.g. a classification task).

There are also a multitude of possibilities to extend our implementation with further defence measures. All alternative Decentralized Learning methods that we mentioned in Chapter 6 can be applied to gaze estimation and it remains to be evaluated which prediction accuracies, training runtimes and privacy properties they offer in comparison with our SFL implementation. Likewise, our applied noise-based defence (PixelDP) can be compared with the mentioned alternative methods for smashed data protection (e.g. decorrelation).

With regard to pruning, we suspect that a sophisticated implementation for efficient input forwarding with sparse weight tensors can also lead to reduced inference times (in addition to our accomplished reduced storage requirements). Especially in combination with MPC (expensive ciphertext multiplications), we assume that pruning can reduce the inference time significantly. Moreover, we have not dealt that extensively with the unsupervised learning approaches and in this case, our system still has potential for improvement concerning lower angle errors and additional DP measures against reconstruction attacks.

We have throughout tested our implementation solely as one single-threaded program which we executed on Google Colab[1]. In practice, the application of SFL is useful when each client process is run on a separate computing unit (with exclusive memory access to its protected face images). Therefore, the deployment of our implementation to a distributed system (with external communication between the SFL participants) remains to be investigated. Then, we suspect that the overall training time is more heavily affected by the required amount of communication between the SFL participants.

Bibliography

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, L. Zhang. “Deep Learning with Differential Privacy”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2016. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145%2F2976749.2978318 (cit. on pp. 25, 26, 31).

[2] E. Bozkir, S. Özdel, M. Wang, B. David-John, H. Gao, K. Butler, E. Jain, E. Kasneci. *Eye-tracked Virtual Reality: A Comprehensive Survey on Methods and Privacy Challenges*. 2023. arXiv: 2305.14080 [cs.HC] (cit. on pp. 13, 61).

[3] T. Chen, S. Kornblith, M. Norouzi, G. Hinton. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III, A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 1597–1607. URL: https://proceedings.mlr.press/v119/chen20j.html (cit. on p. 52).

[4] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, T. Toft. “Confidential Benchmarking Based on Multiparty Computation”. In: *Financial Cryptography and Data Security*. Ed. by J. Grossklags, B. Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 169–187 (cit. on pp. 24, 25, 29, 42).

[5] Q. Duan, S. Hu, R. Deng, Z. Lu. “Combined Federated and Split Learning in Edge Computing for Ubiquitous Intelligence in Internet of Things: State-of-the-Art and Future Directions”. In: *Sensors* 22.16 (2022), p. 5983. DOI: 10.3390/s22165983. URL: https://doi.org/10.3390/s22165983 (cit. on p. 62).

[6] M. Elfares. “PrivatEyes: Appearance-based Gaze Estimation Using Federated Secure Multi-Party Computation”. Note: Unpublished submission was handed to me and kept secret by me. Citation approved. (cit. on pp. 29, 30, 33, 56, 57).

[7] M. Elfares, Z. Hu, P. Reisert, A. Bulling, R. Küsters. *Federated Learning for Appearance-based Gaze Estimation in the Wild*. Tech. rep. 2211.07330. arXiv, 2022. URL: https://arxiv.org/abs/2211.07330 (cit. on pp. 13, 18, 27, 37, 38).

[8] E. Erdogan, A. Küpçü, A. E. Cicek. “SplitGuard: Detecting and Mitigating Training-Hijacking Attacks in Split Learning”. In: *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. WPES’22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 125–137. ISBN: 9781450398732. DOI: 10.1145/3559613.3563198. URL: https://doi.org/10.1145/3559613.3563198 (cit. on p. 62).

[9] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu koray, R. Munos, M. Valko. "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 21271–21284. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf (cit. on p. 52).

[10] O. Gupta, R. Raskar. "Distributed learning of deep neural network over multiple agents". In: *Journal of Network and Computer Applications* 116 (2018), pp. 1–8. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.05.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804518301590 (cit. on p. 61).

[11] D.-J. Han, H. I. Bhatti, J. Lee, J. Moon. "Accelerating federated learning with split learning on locally generated losses". In: *ICML 2021 workshop on federated learning for user privacy and data confidentiality. ICML Board*. 2021 (cit. on p. 61).

[12] D. W. Hansen, Q. Ji. "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.3 (2010), pp. 478–500. DOI: 10.1109/TPAMI.2009.30 (cit. on p. 18).

[13] Z. He, T. Zhang, R. B. Lee. "Model Inversion Attacks against Collaborative Inference". In: *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC '19. San Juan, Puerto Rico, USA: Association for Computing Machinery, 2019, pp. 148–162. ISBN: 9781450376280. DOI: 10.1145/3359789.3359824. URL: https://doi.org/10.1145/3359789.3359824 (cit. on p. 33).

[14] M. Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. DOI: 10.1145/3372297.3417872. URL: https://doi.org/10.1145/3372297.3417872 (cit. on pp. 24, 25, 35, 41).

[15] M. A. Khan, V. Shejwalkar, A. Houmansadr, F. M. Anwar. "Security Analysis of SplitFed Learning". In: SenSys '22. Boston, Massachusetts: Association for Computing Machinery, 2023, pp. 987–993. ISBN: 9781450398862. DOI: 10.1145/3560905.3568302. URL: https://doi.org/10.1145/3560905.3568302 (cit. on p. 59).

[16] J. Ki, Y.-M. Kwon. "3D Gaze Estimation and Interaction". In: *2008 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*. 2008, pp. 373–376. DOI: 10.1109/3DTV.2008.4547886 (cit. on p. 20).

[17] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, S. Jana. "Certified Robustness to Adversarial Examples with Differential Privacy". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 656–672. DOI: 10.1109/SP.2019.00044 (cit. on pp. 25, 26, 31–33, 44).

[18] H. Lipmaa, T. Toft. "Secure Equality and Greater-Than Tests with Sublinear Online Complexity". In: ed. by F. V. V. Fomin, M. Kwiatkowska, D. Peleg. July 2013. ISBN: 978-3-642-39211-5. DOI: 10.1007/978-3-642-39212-2_56 (cit. on p. 34).

[19] Y. Liu, R. Wen, X. He, A. Salem, Z. Zhang, M. Backes, E. D. Cristofaro, M. Fritz, Y. Zhang. "ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA:

USENIX Association, Aug. 2022, pp. 4525–4542. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/liu-yugeng (cit. on pp. 33, 56, 57).

[20] H. Ludwig, N. Baracaldo, eds. *Federated Learning - A Comprehensive Overview of Methods and Applications*. Springer, 2022. ISBN: 978-3-030-96895-3. DOI: 10.1007/978-3-030-96896-0. URL: https://doi.org/10.1007/978-3-030-96896-0 (cit. on pp. 13, 19, 21, 61).

[21] A. K. Mishra, M. Chakraborty. "Does local pruning offer task-specific models to learn effectively ?" In: *Proceedings of the Student Research Workshop Associated with RANLP 2021*. Online: INCOMA Ltd., Sept. 2021, pp. 118–125. URL: https://aclanthology.org/2021.ranlp-srw.17 (cit. on p. 34).

[22] K. O'Shea, R. Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE] (cit. on pp. 18, 35).

[23] D. Pasquini, G. Ateniese, M. Bernaschi. "Unleashing the Tiger: Inference Attacks on Split Learning". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 2113–2129. ISBN: 9781450384544. DOI: 10.1145/3460120.3485259. URL: https://doi.org/10.1145/3460120.3485259 (cit. on pp. 56, 61).

[24] B. Smith, Q. Yin, S. Feiner, S. Nayar. "Gaze Locking: Passive Eye Contact Detection for Human?Object Interaction". In: *ACM Symposium on User Interface Software and Technology (UIST)*. Oct. 2013, pp. 271–280 (cit. on p. 49).

[25] J. D. Smith, T. C. N. Graham. "Use of Eye Movements for Video Game Control". In: ACE '06. Hollywood, California, USA: Association for Computing Machinery, 2006. ISBN: 1595933808. DOI: 10.1145/1178823.1178847. URL: https://doi.org/10.1145/1178823.1178847 (cit. on pp. 13, 17).

[26] Y. Sun, J. Zeng, S. Shan, X. Chen. "Cross-Encoder for Unsupervised Gaze Representation Learning". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 3682–3691. DOI: 10.1109/ICCV48922.2021.00368 (cit. on pp. 19–21, 28, 49, 52, 53).

[27] C. Thapa, M. A. P. Chamikara, S. Camtepe, L. Sun. "SplitFed: When Federated Learning Meets Split Learning". In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 8485–8493. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20825 (cit. on pp. 14, 21, 23, 31–33, 38, 39, 45, 55, 56, 58, 62).

[28] V. Turina, Z. Zhang, F. Esposito, I. Matta. "Combining Split and Federated Architectures for Efficiency and Privacy in Deep Learning". In: CoNEXT '20. Barcelona, Spain: Association for Computing Machinery, 2020, pp. 562–563. ISBN: 9781450379489. DOI: 10.1145/3386367.3431678. URL: https://doi.org/10.1145/3386367.3431678 (cit. on p. 62).

[29] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, W. Shi. "Split Learning Over Wireless Networks: Parallel Design and Resource Management". In: *IEEE Journal on Selected Areas in Communications* 41.4 (2023), pp. 1051–1066. DOI: 10.1109/JSAC.2023.3242704 (cit. on p. 62).

[30] D. Xiao, C. Yang, W. Wu. “Mixing Activations and Labels in Distributed Training for Split Learning”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.11 (2022), pp. 3165–3177. DOI: 10.1109/TPDS.2021.3139191 (cit. on pp. 58, 62).

[31] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, I. Mironov. *Opacus: User-Friendly Differential Privacy Library in PyTorch*. 2022. arXiv: 2109.12298 [cs.LG] (cit. on p. 31).

[32] Y. Yu, J. Odobez. “Unsupervised Representation Learning for Gaze Estimation”. In: *CoRR* abs/1911.06939 (2019). arXiv: 1911.06939. URL: http://arxiv.org/abs/1911.06939 (cit. on p. 19).

[33] X. Zhang, Y. Sugano, M. Fritz, A. Bulling. “MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 41.1 (2019), pp. 162–175. DOI: 10.1109/TPAMI.2017.2778103. URL: https://doi.org/10.1109/TPAMI.2017.2778103 (cit. on pp. 13, 17, 18, 27, 37, 48, 57).

All links were last followed on October 06, 2023.

A Zusammenfassung auf Deutsch

Gaze estimation bezeichnet die Abschätzung, wohin eine gegebene Person in einem bestimmten Moment blickt. Um die Blickrichtungen möglichst akkurat vorhersagen zu können, muss ein Modell auf möglichst vielen Bildern von einer Vielzahl an verschiedenen Personen trainiert werden. Das Sammeln dieser Bilder für ein auf einem Server zentral durchgeführtes Training (Data Centre) führt jedoch zu Datenschutzproblemen. In dieser Arbeit haben wir ein System entworfen, mit welchem ein Modell trainiert werden kann, wobei der Schutz der persönlichen Aufnahmen während des Trainings gewährleistet wird. Zur Implementierung unseres Systems verwenden wir das Splitfed Learning-Verfahren, bei welchem Clients und Server zusammen das Modell, ein konvolutionales neuronales Netz, trainieren. Wir erreichen durch das Splitfed-Verfahren, dass die während des Trainings zu schützenden Bilder nicht von den Clients an den Server übertragen werden. Ein Vorteil gegenüber alternativer Verfahren ist, dass die typscherweise in Ressourcen beschränkten Clients nur einen kleinen Teil des Modells trainieren müssen und der andere Teil auf den leistungsstärkeren Server ausgelagert wird. Außerdem führt das Splitfed-Verfahren zu Model Privacy, d.h. ein Teil des Modells kann vor den Clients geheim gehalten werden. Wir erweitern unseren Ansatz mit Multi-Party Computation (MPC) und Differential Privacy (DP)-Mechanismen, um die Sicherheit der zu schützenden persönlichen Aufnahmen zu verstärken. Mit einem eigens implementierten Angriff zeigen wir die Effektivität dieser Mechanismen. Desweiteren verwenden wir Pruning, um die Größe des Modells zu reduzieren und wir trainieren unser Modell zusätzlich mit Unsupervised Learning. Unsere durchgeführten Testergebnisse zeigen, dass die Vorhersagen unseres mit dem Splitfed-Verfahren trainierten Modells eine durchschnittliche Abweichung von etwas weniger als 6.5 Grad vom eigentlichen Blickrichtungs-Winkel betragen. Damit konnten wir eine erhöhte Genauigkeit im Vergleich zu bereits bestehenden Gaze-Estimation-Methoden erzielen, welche auch auf die Geheimhaltung der persönlichen Bilder abzielen. Das Training des Modells lässt sich effizient (innerhalb von 10 Minuten) durchführen.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature