REal-time data monitoring for **S**hared, **A**daptive, **M**ulti-domain and **P**ersonalised prediction and decision making for **L**ong-term Pulmonary care **E**cosystems

D3.2: Privacy-preserving designs of ML

Dissemination level:	PU
Document type:	Report
Version:	1.0
Date:	29.04.2024

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 965315. This result reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Document Details

Reference No.	965315
Project title	**RE-SAMPLE - RE**al-time data monitoring for **S**hared, **A**daptive, **M**ulti-domain and **P**ersonalised prediction and decision making for **L**ong-term Pulmonary care **E**cosystems
Title of deliverable	Privacy-Preserving Design of ML
Due date deliverable	30/04/2024
Work Package	WP3
Document type	Report
Dissemination Level	PU: Public
Approved by	Coordinator
Author(s)	Florian Hahn (UT), Federico Mazzone (UT)
Reviewer(s)	César Mediavilla Martínez (ATOS); Costas Lambrinoudakis, Christos Kalloniatis and Thrasyvoulos Giannakopoulos (UPRC)
Total No. of pages	42

Partners

Participant No	**Participant organisation name (country)**	**Participant abbreviation**
1 (Coordinator)	University of Twente (NL)	UT
2	Foundation Medisch Spectrum Twente (NL)	MST
3	University of Piraeus Research Center (GR)	UPRC
4	Foundation Tartu University Hospital (EE)	TUK
5	Foundation University Polyclinic Agostino Gemelli IRCCS (IT)	GEM
6	European Hospital and Healthcare Federation (BE)	HOPE
7	German Research Center for Artificial Intelligence GMBH (DE)	DFKI
8	ATOS IT Solutions and Services Iberia SL (ES)	ATOS
9	Roessingh Research and Development BV (NL)	RRD
10	Innovation Sprint (BE)	iSPRINT

Abstract

This deliverable describes the privacy risks for the federated training of a predictive model in RE-SAMPLE and presents a privacy-preserving solution to this problem. The solution utilises homomorphic encryption to conceal a subset of the model's parameters, while allowing the training procedure to proceed obliviously from potential adversaries. The proposed approach leads to a trade-off between the efficiency of the training process, in terms of local computations and communication overhead, and the privacy of the training data, experimentally measured against privacy attacks. The proposed approach is implemented as a prototype and assessed on public datasets against state-of-the-art privacy attacks.

Contents

List of Figures

List of Tables

Symbols, definitions, abbreviations, and acronyms

API	Application Programming Interface
CCC	Complex Chronic Condition
CKKS	Chen, Kim, Kim, Song
CNN	Convolutional Neural Network
D	Deliverable
DP	Differential Privacy
FHE	Fully Homomorphic Encryption
FL	Federated Learning
HE	Homomorphic Encryption
MSE	Mean Squared Error
MHE	Multiparty Homomorphic Encryption
ML	Machine Learning
MPC	Multiparty Computation
PPML	Privacy-Preserving Machine Learning
SIMD	Single Instruction, Multiple Data
SGD	Stochastic Gradient Descent
SL	Split Learning
WP	Work Package

1. Introduction

Deliverable D3.2 “Privacy-preserving designs of ML” is part of WP3 “Personalised prediction and modelling of CCC exacerbations” and is linked with Task 3.1 “Training of a predictive model” and Task 4.4 “Implementation and experimental evaluation of privacy-preserving machine learning model training and prediction”.

Specifically, we analyse the privacy risks associated with the use of federated learning to train a joint predictive model on distributed data. Information about the training data of a model can be retrieved by having access to its prediction functionality or to the model itself, posing a risk to the privacy of the patients involved in the RE-SAMPLE project. We show that Federated Learning (FL) alone is not enough to fully prevent adversaries from carrying out such privacy attacks. Adversaries may attempt to corrupt one or more pilot sites and try to retrieve information about the other sites’ patients. Defences against this kind of attacks exist, but they all suffer from downsides (severe utility loss of the model, poor scalability with the number of training parties in terms of communication overhead, or high computational cost that would made the training impractical). Here, we propose a new solution which can be flexibly tuned to trade-off privacy for efficiency, while keeping the utility of the model unaffected.

The output of this deliverable consists of a detailed description of the proposed solution, and a strategy to select its hyperparameters for the aforementioned trade-off. These outputs will be fed as input to D4.5 “Proof-of-concept privacy-preserving ML and data aggregation” and D4.6 “Report on the final parameter selection” within the work package WP4.

The deliverable starts by summarising its objectives in Section 2, and by providing some background notions on privacy-preserving machine learning in Section 3. We use Section 4 to provide the reader with a basic understanding of the notion of privacy in machine learning, describing different privacy attacks (membership inference, model inversion, and property inference), and focusing in particular on the federated setting. We show how current solutions for privacy-preserving federated learning can turn out to be impractical, focusing on solutions based on encrypting the model in Section 5. These approaches provide privacy at the cost of training efficiency. In Section 6, we show how we improve on these solutions for achieving better efficiency at the cost of little privacy loss. Further, we outline the compatibility of our solution with existing solutions based on differential privacy – as also envisioned in the RE-SAMPLE project.

2. Objectives

The objective of this deliverable is to present the privacy-preserving design for the machine learning (ML) components of the RE-SAMPLE project. It serves as the first part of a series of three deliverables, outlined as follows:

- **D3.2 Privacy-preserving design of ML (this deliverable)**: Provides an understanding of privacy-preserving machine learning, privacy concepts, existing risks, related work, and available mitigations. Describes the proposed solution for private training and inference of ML models in RE-SAMPLE, including high-level descriptions of algorithms and protocols, privacy evaluation, security proof, and benchmarking of prototypes on publicly available datasets. Aims to highlight the threats, the need for the proposed solution, and the advantages and disadvantages in terms of privacy and utility.
- **D4.5: Proof-of-concept privacy-preserving ML and data aggregation**: Describes the implementation of the solution within the RE-SAMPLE framework, detailing the Application Programming Interface (API) calls and the interactions among the hospitals and the coordinating server.
- **D4.6: Report on the final parameter selection**: Focuses on the finalization and optimization of hyperparameters of the solution within the RE-SAMPLE framework, by finding a suitable trade-off between privacy and efficiency.

3. Background

In this section we provide some basic notions and definitions.

3.1 Differential Privacy

Differential Privacy (DP) [1] is a rigorous privacy concept that provides strong guarantees for protecting individual data when performing statistical analyses. It ensures that the presence or absence of any single data point in the dataset does not significantly impact the outcome of the analysis, thus safeguarding the privacy of individuals. The idea is that given a dataset D and a statistical mechanism f you want to compute on D, just enough tailor crafted noise is injected into the dataset or the mechanism to ensure that individual privacy is preserved while still allowing meaningful analysis to be conducted. Unlike traditional approaches such as k-anonymity, DP provides privacy guarantees regardless of adversaries' prior knowledge.

The epsilon-delta framework is central to DP. It quantifies privacy guarantees using two parameters:

- epsilon (ϵ), which quantifies the privacy loss incurred by an individual's participation in a dataset, and
- delta (δ), which represents the allowable probability of a privacy breach.

Mathematically, a statistical mechanism f is (ϵ, δ)-differentially private if, for all possible pairs of neighbouring datasets D and D' (i.e., datasets that differ by the presence or absence of a single individual's data point), and for all possible outcomes S of the computation:

$$\Pr[f(D) \in S] \leq e^{\epsilon} \Pr[f(D') \in S] + \delta$$

In simpler terms, this formula states that the probability of obtaining a certain output remains relatively stable regardless of whether a specific individual's data is included or excluded.

Choosing an appropriate ϵ value is crucial and challenging in practice. A smaller ϵ implies stronger privacy guarantees, but it may also lead to significant noise being added to the computation, potentially affecting its accuracy. Conversely, a larger ϵ allows for more accurate computations but provides weaker privacy protection. The challenge lies in finding the right balance between privacy and utility. Empirical evaluations and sensitivity analyses are often conducted to iteratively adjust ϵ based on the specific requirements and constraints of the application.

3.2 Multiparty Computation

Multiparty Computation (MPC) is a cryptographic technique that enables multiple parties $P_1, \dots, P_N$ to jointly compute a function f over their inputs $x_1, \dots, x_N$ while keeping those inputs private. Essentially, MPC allows computation on sensitive data without revealing the data itself, ensuring privacy and confidentiality in collaborative settings. It relies on cryptographic protocols to distribute the computation across multiple parties in such a way that each party cannot learn more than what they can infer from their own inputs and the result of the computation. This is achieved through a combination of encryption, secret-sharing schemes, and secure computation protocols.

The MPC framework proves the security of its protocols depending on the view and capabilities a potential adversary is assumed to have. The adversary is usually supposed to be able to corrupt up to t parties, for a given $t < N$, and obtain their view on the protocol, i.e., their internal computations and the messages they exchange with the other parties. If the adversary is also allowed to tamper with the protocol, that is to force the corrupted parties to deviate from the given instructions and send arbitrarily crafted messages, then we are modelling a *malicious* or *active* adversary. Otherwise, we are modelling a *semi-honest* or *passive* adversary.

Theoretically, it is possible to create a multiparty protocol to securely compute any given computable function f. However, the challenge is to come up with a protocol that is efficient. MPC protocols turn out to be expensive to run, mostly in terms of communication complexity. Their communication overhead does not usually scale well with an increasing number of participants (i.e., N). Especially in the malicious setting, where MPC protocols need to incorporate heavy cryptographic techniques such as zero-knowledge proofs and commitment schemes to ensure security against active adversaries.

3.3 Homomorphic Encryption

Homomorphic Encryption (HE) is a type of encryption that allows performing computations on encrypted data without decrypting it first. It enables different kind of operations to be executed directly on ciphertexts, resulting in encrypted results that correspond to the outcomes of the same operations on the plaintext data. This property allows for computation on sensitive information without exposing it. More formally, there is an evaluation functionality Eval such that given an n-ary function f and messages $m_1, \dots, m_n$

$$\text{Dec}(\text{Eval}(f, Enc(m_1), \dots, Enc(m_n))) = f(m_1, \dots, m_n)$$

where Enc and Dec are the encryption and decryption functionality for a given key, respectively.

HE can be used to outsource computations to an untrusted third party by providing it with the encryption of the input of these computations. The third party will then apply the homomorphic evaluation on the encrypted input to obliviously compute the encrypted output, which is then returned to the original querier who can decrypt it. In this scenario, the querier is the one setting up the crypto scheme, generating the encryption and decryption keys, in addition to an evaluation key, which is needed by the third party to perform the homomorphic evaluation. Using these delegation-based computations is quite challenging in practice for mainly two reasons. The first one is the high computational cost of performing operations under encryption. The second one is that not all functions are (directly) supported for homomorphic evaluation.

HE schemes are classified depending on what kind of functions (usually arithmetic circuits composed of additions and multiplications, or Boolean circuits composed of ANDs and XORs) they support for the homomorphic evaluation.

- Partial Homomorphic Encryption schemes allow for circuits of any depth but consisting of only one kind of operation (either only additions/XORs or only multiplications/ANDs).
- Somewhat Homomorphic Encryption schemes support any number of one operation, and a limited number of the other operation (this bound is fixed by the scheme).
- Levelled Homomorphic Encryption schemes support both operations but up to a given number of times (this bound can be modified within the scheme, usually at the cost of bigger ciphertexts and higher computational overhead).
- Fully Homomorphic Encryption (FHE) schemes allow for any number of both operations. FHE schemes are very useful due to their flexibility, but also very expensive from a computational point of view.

FHE schemes are often built upon Levelled schemes, enhanced with a bootstrapping functionality. Bootstrapping enables the “refreshing” of ciphertexts once the number of allowed operations on them is over. This process involves homomorphically evaluating an approximation of the decryption circuit on the ciphertext to be refreshed. Through this mechanism, the ciphertext is decrypted and re-encrypted in an oblivious manner, renewing its usability for further computations.

3.4 Multilayer Perceptrons (MLPs)

Multilayer Perceptrons (MLPs), also known as fully-connected or dense networks, are the simplest kind of feedforward neural networks, where each neuron in one layer is connected to every neuron in the next layer. Due to their structure, the parameters of these models can be represented by matrices, and in some cases, an additive bias parameter is included for added flexibility. Given an MLP with L layers and $i \in \{1, \dots, L\}$, we denote by w_i and b_i the weight matrix and bias vector between layer $i-1$ and layer i, respectively. We denote by l_i the output of layer i, that is $l_i = \phi_i(l_{i-1}w_i + b_i)$, where $l_0 = x$ is the input of the model and ϕ_i is the so-called activation function used to incorporate non-linearity in the model. And we denote the intermediate linear application output as $u_i = l_{i-1}w_i + b_i$. With a little abuse of notation, we will sometimes refer to the weight and bias w_i, b_i as to the parameters of the layer they allow to transition to, namely layer i. The model is then a parameterized function $f(x; w_i, b_i)$, whose output is l_L.

The model is trained by minimising the empirical risk with respect to a given loss function. For supervised learning, we assume to have a training dataset D of labeled examples (x, y), where x is the feature vector and y is the ground-truth label. Given a model $f(x; w_i, b_i)$, the goal is to optimize the model's parameters by minimising some loss function $L(f(x),y)$. This is usually done through Gradient Descent (GD) techniques, namely computing the gradient of the loss function and moving towards the negative direction of the gradient. To estimate the gradient of L with respect to the model parameters w_i, b_i, feedforward and backpropagation are used. During feedforward, an input data x is propagated layer by layer through the network, computing all the u_i and l_i. The output prediction l_L is then compared to the actual label y using the chosen loss function L to compute the loss value. The backpropagation algorithm then calculates the gradients of the loss function with respect to the model's parameters. When the loss function L, the model f, and the example (x,y) are clear from the context, we will write ∇w_i and ∇b_i in place of $\nabla_{w_i} L(f(x),y)$ and $\nabla_{b_i} L(f(x),y)$, respectively. Below is a schematic representation of the computations performed during one step of the training process.

$$\begin{tikzcd}
\begin{aligned} u_1 &= l_0 w_1 + b_1 \\ l_1 &= \phi_1(u_1) \end{aligned} \arrow[r] & \cdots \arrow[r] & \begin{aligned} u_L &= l_{L-1} w_L + b_L \\ l_L &= \phi_L(u_L) \end{aligned} \arrow[d] \\
l_0 = x \arrow[u] & & \begin{aligned} &L(l_L, y) \\ e_L &= \partial L / \partial l_L \end{aligned} \arrow[d] \\
\begin{aligned} \nabla b_1 &= e_1 \phi_1'(u_1) \\ \nabla w_1 &= e_1 \phi_1'(u_1) l_0^T \\ (e_0 &= e_1 \phi_1'(u_1) w_1^T) \end{aligned} & \cdots \arrow[l] & \begin{aligned} \nabla b_L &= e_L \phi_L'(u_L) \\ \nabla w_L &= e_L \phi_L'(u_L) l_{L-1}^T \\ e_{L-1} &= e_L \phi_L'(u_L) w_L^T \end{aligned} \arrow[l]
\end{tikzcd}$$

This step is repeated for a batch of examples B, and the resulting gradients are averaged to get a better approximation of the actual loss gradient on the real population. The parameters are then updated following the negative direction of the gradient, by a step size proportional to a learning rate $\eta > 0$:

$$w_i \leftarrow w_i - \frac{\eta}{|B|} \sum_{(x,y)\in B} \nabla_{w_i} L(f(x),y)$$
$$b_i \leftarrow b_i - \frac{\eta}{|B|} \sum_{(x,y)\in B} \nabla_{b_i} L(f(x),y).$$

This iterative process of feeding the data forward, computing the loss, and updating the model's parameters continues until convergence, or for a fixed number of iterations.

4. Privacy-Preserving Machine Learning (PPML)

Privacy-preserving machine learning (PPML) is the branch of ML that aims to develop techniques and methodologies to perform model training and inference while safeguarding the privacy of the training data and/or of the model itself. Such techniques should mitigate the danger coming from privacy attacks, while keeping the utility of the model and the efficiency of the training and inference as high as possible.

4.1 Privacy Attacks

In general, a privacy attack is a technique designed to extract information about the data that a particular model has been trained on. In this section, we provide a high-level description and classification of privacy attacks based on the information available to the adversary and its goal.

4.1.1 Threat Model

In this section, we briefly discuss the threat model under which PPML works and the kind of adversaries it considers [2].

4.1.1.1 Black-Box vs. White-Box

PPML literature classifies privacy attacks depending on the adversary's view of the model, making a distinction between black-box attacks and white-box attacks. In black-box attacks, the adversary can only access the model's output for arbitrarily chosen inputs but lacks information about model parameters. While in white-box attacks, the adversary has full access to the model's architecture, parameters, and hyperparameters used during training. This enables them to compute any function of the model parameters and any chosen input, including intermediate computations of the feedforward pass, i.e. output of intermediate layers. For labelled input, the adversary can hence compute the corresponding loss and the gradients for each layer.

4.1.1.2 Active vs. Passive

PPML literature also distinguishes between passive and active behaviour for privacy attacks. A passive adversary can only observe the legitimate model updates and attempt to infer information by performing inference on the model, without changing anything in the local or global collaborative training procedure. In contrast, an active adversary influences the target model during training in order to coerce the data owners into unintentionally releasing more information through the model. The active adversary's actions may include choosing specific artificially crafted inputs (not originally included in the training dataset of the corrupted training client, or not drawn from the population distribution) for the training procedure, or performing gradient ascent on specific inputs.

4.1.1.3 Supervised vs. Unsupervised

When it comes to inferring information about the training dataset of a given model, it is also possible to distinguish between supervised and unsupervised attacks. In supervised attacks we assume the adversary already knows a portion of the training dataset. On the other hand, in unsupervised attacks, there is no such assumption, and the adversary is not assumed to possess any data point belonging to the training dataset.

4.1.1.4 Central vs. Distributed

Finally, privacy attacks can also be classified depending on whether they target model trained by a single party or by a federation composed of multiple data owners. We will describe the latter scenario in more detail in Section 4.2.

4.1.2 Attacker's Goal

Privacy attacks are typically categorized based on the specific type of information they aim to extract. Following the taxonomy provided by Rigaki et al. in their survey [3], we can classify privacy attacks into three categories: membership inference, model inversion, and property inference. For each category we describe the adversary's goal, the causes of the privacy leakage, some available mitigations, and an actual state-of-the-art attack from each given category.

4.1.2.1 *Membership Inference*

Membership inference attacks aim to determine whether or not a given data point was part of the training data set. These attacks exploit the intrinsic difference in the model's behaviour when performing predictions over known training data versus unseen data. Membership inference attacks reveal how much a model retains from its training data, helping to gauge the potential effectiveness of other privacy attacks such as data reconstruction, but they can also pose significant privacy risks on their own. For instance, consider being able to determine whether a specific patient's data was used to model the efficacy of a particular cancer treatment.

Since the introduction of the first membership inference attack by Shokri et al. [4], numerous studies have investigated the underlying causes of membership leakage in ML models. The primary contributing factor to membership leakage appears to be model overfitting or poor generalization [4, 5]. Several factors can exacerbate this issue, including a limited number of training samples [4, 6], high model complexity leading to overparameterisation [2], and high feature dimensionality [4].

A state-of-the-art attack in the membership inference category is the white-box attack by Nasr et al. [2]. Like other lines of work, Nasr et al.'s attack treats membership inference as a binary classification task, and it trains a machine learning model to accomplish this task. We describe the supervised version of their attack, in which the adversary is assumed to know a portion of the private training dataset and uses this knowledge to perform supervised training of the attack model. Given a target data point, the attacker performs a feedforward pass of the model over it, computing hidden layer outputs, loss, and subsequent backpropagation to calculate gradients for each layer.
These computed values, along with the true label, serve as input features for the attack model. The attack model suggested by the authors consists of a fully connected or convolutional component for each aforementioned value. Those components are then all connected to another fully connected component that produces a scalar output representing the membership probability of the input.

4.1.2.2 *Model Inversion*

Model inversion, also known as reconstruction attack, aims to recreate training samples and, in some cases, their associated labels. There are two main types of model inversion attacks: those that aim to reconstruct actual training samples [7, 8] and those that aim to craft a class representative [9, 10]. The latter type is particularly useful when all samples associated with a given label are similar, such as faces of the same person, or when the attacker has no prior knowledge about what a specific label encodes.

The effectiveness of model inversion attacks has been shown to increase with the target model's level of overfitting [5] and its predictive power, as measured by loss minimization [11]. To mitigate these attacks, one suggested approach is to partially prune the gradients before updating the model [7].

One of the first model inversion attacks on neural networks was developed by Fredrikson et al. [9]. The attacker crafts a dummy input for the target model and then uses gradient descent to optimize the dummy input. The high-level idea is that, instead of fitting the model parameters to the input, the attacker computes the gradient of the loss function with respect to the input and fits the latter to the model parameters. In contrast, other model inversion attacks use generative models to construct class representative. For instance, Hitaj et al. [10] proposed a method based on Generative Adversarial Networks (GANs). In this approach, the attacker designs a generator G with the purpose of producing examples for a specific class y, using the target model itself as the discriminator. The generator takes noise x_ϵ as input and generates $x_y = G(x_\epsilon)$, intended to represent class y. The parameters of G, denoted as θ_G, are optimized to minimize $L\left(f\left(x_y\right), y\right)$, which indicates how confidently the model classifies x_y as y. Another model inversion attack by Zhu et al. [7] assumes that the attacker has access to the gradients $\nabla L(f(x), y)$ computed by a training party for a data point (x, y), e.g., in a collaborative learning setting when the attacker corrupts the supporting server (with no secure aggregation ongoing) or if the attacker corrupts all parties but the target one. The attacker initialises a dummy data point (x', y'), computes the corresponding gradients $\nabla L(f(x'), y')$, and minimises the distance between these dummy gradients and the original gradients, which in turn brings the dummy input (x', y') closer to the original (x, y). To solve the minimization problem, the attacker differentiates $|\nabla L(f(x'), y') - \nabla L(f(x), y)|$ with respect to (x', y') and uses GD to find a local minimum (x', y').

4.1.2.3 Property Inference

Property inference attacks aim to extract properties about the training samples that are uncorrelated to the learning task at hand. For example, in a face recognition task, where the goal is gender classification, the attacker might try to infer whether people in the training dataset are wearing sunglasses. Similarly, for a model designed for handwriting recognition, the attacker may attempt to determine the font used to write the messages (e.g., cursive or block letters).

The underlying conditions and factors that enable property inference attacks are not yet fully understood [3]. It remains unclear what specific characteristics or vulnerabilities in a model make it susceptible to such attacks. Surprisingly, these attacks have shown effectiveness even on well-generalized models, and the relationship between their efficacy and overfitting is still unclear [12, 13]. It has been suggested that sharing only a small portion of the gradients, as in the collaborative approach by Shokri and Shmatikov [14], may contribute to mitigate these attacks [13].

To carry out a property inference attack, the adversary needs access to samples both with and without the property they want to infer. They calculate the gradients of the target model for both types of samples and trains a binary classifier to distinguish between the gradients of samples with the property of inference and samples without it. In collaborative learning, the adversary can obtain the gradients of honest parties by computing the difference between two subsequent model updates. However, if the adversary only has access to aggregated data from other parties, the attack may become less effective as the number of honest parties increases.

4.2 Federated Learning (FL) as a Privacy Enhancing Technology

FL is a collaborative learning approach where multiple data-owning clients jointly train a common model while keeping their data decentralized. Although developed as a privacy-preserving solution, recent lines of work show that this approach indirectly leaks information about the model's training data.

4.2.1 Recap of FL

The demand for more complex and accurate machine learning models in fields like image recognition and natural language processing has highlighted the need for extensive training data [15]. As a result, collaboration among data-owning entities has become crucial to leverage larger and more diverse datasets and enhance model performance. However, this collaborative approach raises privacy concerns, as sharing raw data can expose sensitive information, potentially violating privacy regulations and raising confidentiality concerns. To address this issue, collaborative learning solutions, like FL [16] and Split Learning (SL) [17], have been proposed. Those methods aim to protect privacy by enabling the training of a joint model without directly outsourcing the raw data.

Originally, Shokri and Shmatikov [14] introduced an FL scheme for neural networks where clients independently train local models on their datasets, while sharing portions of their model parameters with a global model hosted by a supporting server. The training process is asynchronous, with each client repeatedly downloading portions of the global model parameters, updating its local model, and then uploading portions of the gradients to the global model. However, a more widely adopted approach for FL is Federated Averaging (FedAvg), as proposed by McMahan et al. [16]. In FedAvg, the training process occurs in synchronous rounds. In each round, a subset of clients is selected to participate. These clients perform a local training step on their own data, updating their model parameters. These locally trained models are then sent to the central server, where they are averaged together to obtain a new global model. The aggregated global model is then distributed back to all participating clients.

4.2.2 A Privacy Illusion

FL was designed as a privacy-preserving approach to distributed learning, supposed to protect the privacy of the training data. Unfortunately, the recent developments in the PPML literature have shown that, even if the raw training data is not exposed, some information about them can be extracted from the model updates (i.e., gradients). The publication of white-box privacy attacks that specifically target the federated setting has made evident that solely withholding the training data is an insufficient strategy to ensure privacy protection in these scenarios [2, 13, 9, 10, 7]. These attacks can be carried out during the training stage (in addition to the threats during the inference phase) by an adversary who may corrupt one or more training clients and potentially the supporting server, making it possible to run these privacy attacks on the intermediate model updates.

4.2.2.1 Threat Actors: Aggregator vs. Clients

Besides the usual threat actor of the central setting (i.e., the querier), in the federated setting privacy attacks can be performed also during training in a white-box fashion against the joint model at each round. The actors that have access to these intermediate models are:

- the aggregator, and
- the training clients.

Note that an adversary that is able to corrupt the aggregator will have a much better view compared to the individual training clients. In fact, the aggregator has access to the individual model updates, and they can extract information about a specific local training dataset. On the other hand, a training client can only see the joint update (as the difference of two subsequent global model instances), and even by removing its own contribution, they would not be able to distinguish between the different contributions of the remaining clients.

4.2.3 Overview of Available Solutions

To mitigate such attacks, solutions based on DP [14, 18, 19], FHE [20], and MPC [21, 22, 23] have been introduced. However, these solutions come with trade-offs: DP introduces noise to protect privacy but may lead to accuracy loss; FHE provides strong privacy guarantees but has a high computation cost for deep arithmetic circuits that require bootstrapping, making it an unfeasible approach in many practical training contexts; MPC may require a large number of interactions between multiple parties and high communication bandwidth.

4.2.3.1 Differential Privacy

In the context of PPML, our goal is to protect the privacy of individual data records used to train the model by incorporating differential privacy into the training mechanism. Specifically for deep learning, achieving differential privacy involves injecting controlled noise in one or more parts of the training process [24], which usually comes at the cost of model's accuracy. The elements that can be subject to noisy perturbation include:

- *input*: adding noise directly to the input data, namely sanitising the dataset;
- *loss function*: incorporating noise into the loss function used during training;
- *output*: applying noise to the output of the training mechanism, that is the trained model parameters;
- *label*: introducing noise to the ground-truth labels during training;
- *gradient* [18]: perturbing the gradients, which is the most commonly used approach to achieve private deep learning, as it achieves a good trade-off between privacy and accuracy.

In collaborative learning settings, differential privacy can be applied not only at the data level [14] but also at the client level [19], to conceal the presence of the individual training parties. However, the latter requires a substantial number of clients to participate in the protocol to achieve meaningful privacy guarantees. In our work, following the approach of [14], we employ data level differential privacy by injecting noise into the gradients during training.

4.2.3.2 Secure Aggregation

Since the supporting server could also be compromised, some techniques have been designed to conceal individual model updates to the aggregator by performing secure aggregation of such values. Instead of sending the model updates to the orchestrator at the end of each round, the training parties run a protocol that outputs the aggregated update, which is sent to the orchestrator. The orchestrator's only job is then to apply the aggregated update to the global model and send back the result to the training parties. To have an immediate grasp of the effect of this mitigation, think about membership inference attacks: targeting a specific data point in a joint update is much more difficult than in an individual update, due to the "hiding in the crowd" effect. Secure aggregation can be based on different technologies:

- secret sharing [25],
- multiparty homomorphic encryption [26, 27], or
- additive masking [28, 29].

However, this security measure does still expose the global model to all the actors involved during training, making the leakage from intermediate (aggregated) models still a concern.

4.2.3.3 Training with Secret Sharing based Multiparty Computation

To prevent information leakage from the intermediate model updates, MPC can be employed, allowing the data owners to jointly execute the entire training mechanism in a secure manner, usually by exploiting secret sharing schemes. An FL protocol can be seen as an MPC, where:

- the players are the training parties;
- the function to be jointly evaluated is the training of a model, and the output of this function is the trained model itself;
- the distributed private inputs are the local training datasets.

However, a major challenge arises when scaling to a large number of parties, as it leads to impractical communication complexity. To work around such overhead, the data-owners can delegate the computations to a small cluster of non-colluding servers, usually composed of

- 2 parties: SecureML [30];
- 3 parties: ABY^3 [22], Falcon [21], SecureNN [23], or
- 4 parties: FLASH [31], Trident [32].

However, this delegation-based approach imposes strong assumptions on the non-collusion of the computing servers, strongly constraining the threat model. In the specific case of RE-SAMPLE, a direct application of MPC-based approaches would prevent a scalability of a n-out-of-n security model for more than 3 hospitals.

4.2.3.4 Training under Encryption

To overcome the limitations of small cluster MPC solutions to the threat model, a promising research direction has emerged, leveraging FHE schemes to encrypt the model. By employing FHE, the federated learning process can be conducted entirely under encryption, enabling secure collaboration among a large number of parties. Related literature is limited but promising:

- SPINDLE [33] for generalized linear models;
- POSEIDON [20] for neural networks.

While scaling well in terms of communication complexity, these solutions suffer from high computational overhead due to the heavy costs associated with performing operations under encryption, making their use unfeasible in real-world applications. We will see more about this kind of approach and its limitations in Section 5.2.

4.3 The importance of PPML for RE-SAMPLE

In the context of RE-SAMPLE, three hospitals (and potentially more in the future) collaborate to train a joint ML model on privacy-sensitive medical data. The sensitivity of medical data precludes its outsourcing to third-party entities for model training, making the adoption of FL-like solution necessary.

In this scenario, a potential adversary could corrupt one or more hospitals, and/or the supporting server hosting the aggregator. Our goal is then to prevent the adversary from stealing the information from the uncorrupted hospitals by adopting PPML measures both during the training and prediction phase.

5. Federated Training under Fully Homomorphic Encryption (FHE)

As mentioned in Section 4.2.3.4, an effective defence against white-box privacy attacks in the federated setting involves using FHE to encrypt the model and perform the entire training process under encryption. By adopting this approach, potential adversaries are prevented from accessing any meaningful information about the model's parameters, as all computations are performed on encrypted data. In this section, we provide some additional details on FHE schemes, in particular about the CKKS scheme [34], and its usage in a multiparty setting [35]. Additionally, we show how to use the multiparty variant of CKKS to perform federated learning with an MLP under encryption [20].

5.1 CKKS and Multiparty FHE

FHE enables the evaluation of unlimited-depth arithmetic circuits on encrypted data, utilising a technique called bootstrapping for refreshing ciphertexts after homomorphic operations. The CKKS scheme [34] is well-suited for floating-point-like arithmetic and performs computations on vectors of real/complex numbers, allowing for Single Instruction, Multiple Data (SIMD) operations. These properties make the scheme particularly suitable for the computations needed by neural network models.

The CKKS scheme works with residual polynomial rings of the form $R_q \coloneqq \mathbb{Z}_q[x]/(x^n+1)$, for some positive integers q and n, with n being a power of two. A Residue Number System (RNS) instantiation of CKKS [36] is usually adopted, which achieves the highest efficiency for CKKS among known variants of the scheme. Here, we briefly describe it at a high level. Given unique primes $q_0, q_1, \dots, q_M$, an RNS chain of moduli is built as $Q_i = \prod_{j=0}^{i} q_j$ for $i \in \{0, \dots, M\}$. A plaintext is an element $m \in R \coloneqq \mathbb{Z}[x]/(x^n+1)$, which can embed a vector of up to $n/2$ slots, as the encoding is a map $\mathbb{C}^{n/2} \to R$. A freshly encrypted ciphertext is a pair $c \in R_{Q_M} \times R_{Q_M}$. Then, after each multiplication, the ciphertext is rescaled to scale down the message and truncate the least significant bits, dropping the highest RNS limb, going from R_{Q_i} to $R_{Q_{i-1}}$ for $i > 0$. The maximum number of multiplications is given by $M-1$. However, not all of these levels can be used for the main computation as the bootstrapping procedure consumes levels, too.

5.1.1 Homomorphic Properties

Given two ciphertexts c_0, c_1 encrypting the plaintexts $m_0 = (m_{0,0}, \dots, m_{0,n/2}), m_1 = (m_{1,0}, \dots, m_{1,n/2})$, respectively, CKKS natively allows for the following homomorphic operations:

- Addition: $c_0 + c_1$, which corresponds to the component-wise addition of the underlying plaintexts $m_0 + m_1 = (m_{0,0} + m_{1,0}, \dots, m_{0,n/2} + m_{1,n/2})$.
- Multiplication: $c_0 c_1$, which corresponds to the component-wise multiplication of the underlying plaintexts $m_0 m_1 = (m_{0,0} m_{1,0}, \dots, m_{0,n/2} m_{1,n/2})$.
- Vector rotation: $c_0 \ll k$ for a given $k \in \{-n/2, \dots, 0, \dots, n/2\}$, which corresponds to the rotation of the underlying plaintext by k positions $m_0 \ll k = (m_{0,k}, \dots, m_{0,n/2}, m_{0,0}, \dots, m_{0,k-1})$ (the indices are modulo $n/2$). For convenience, we denote by $c_0 \gg k$ the rotation by $-k$.

Note that additions and multiplications can also be performed between a ciphertext and a plaintext.

To perform homomorphic computations efficiently, we can pack an entire vector in a single ciphertext and exploit the SIMD capabilities of CKKS to perform vector addition and component-wise multiplication in constant time. Note that if the vector is too long to fit the number of slots dictated by given FHE parameters, one can split the vector among multiple ciphertexts. However, in a neural network, we also need to multiply by weight matrices and evaluate non-linear activation functions. Now, we describe the approaches we can adopt to perform efficient vector-matrix multiplication under encryption and homomorphically evaluate non-polynomial functions.

5.1.1.1 Matrix Multiplication

To efficiently perform matrix multiplication under encryption, the idea is to encode the matrix as a vector in such a way that only one homomorphic multiplication is required. There exist multiple encoding schemes in the literature for matrices. For instance, in the column-based approach [37, 38], one encodes a matrix by concatenating its columns one after the other. The vector-matrix multiplication is then performed by first replicating the vector to match the number of columns of the matrix, then performing a SIMD multiplication between the two, and finally by performing cumulative addition of the result. All those operations can be realized by combining homomorphic additions, rotations, and multiplications by a masking vector. Moreover, the vector replication and the cumulative addition can be made more efficient by recursion, though requiring padding the inputs to a suitable power of two. Similarly to this column-based approach, a row-based encoding can be employed as well [39].

To perform subsequent matrix multiplications, it is convenient to use the alternating packing approach proposed in [20]. It is based on the observation that the result of a vector-matrix multiplication in column-based encoding requires extra rotations to prepare it for a multiplication with another column-based encoded matrix, while it is perfectly ready for a multiplication with another row-based encoded matrix. The idea is then to encode the matrices that are in consecutive secret layers by alternating between column- and row-based encodings. Note that multiplying by the transpose of a matrix is equivalent to multiplying by the matrix in the opposite encoding. We can exploit this property to efficiently compute gradients in the back-propagation step.

5.1.1.2 Evaluating Non-Polynomial Functions
By combining additions and multiplications it is possible to evaluate any polynomial function. However, when it comes to non-polynomial functions like the exponential function, we need to find alternative solutions. To evaluate non-polynomial functions with an FHE scheme, a possible strategy consists of using a high-degree polynomial to approximate the given function. This is usually done by exploiting the Chebyshev interpolation algorithm, which assures uniform convergence within a given interval $[a, b]$. Note the importance of correctly estimating the input range of the function to assure the correctness of the function evaluation.

5.1.2 Multiparty FHE
In Multiparty Homomorphic Encryption (MHE), the secret key is shared among multiple parties, who can use their shares to collectively generate joint public/evaluation keys and perform distributed decryption and bootstrapping protocols. MHE comes in two fashions, depending on the access structure of the private key:

- *threshold* FHE schemes [40], where the secret key and its access-structure are fixed at the beginning, and they usually allow for decryption if at least a given number of parties are online;
- *multikey* FHE schemes [41], where the secret key and its access-structure are dynamic, each party encrypts its data under its own secret key, and the output of homomorphic operations is a ciphertext encrypted under the joint keys of the parties involved in that specific computation.

On the one hand, the multikey FHE approach is more flexible, but on the other hand it also leads to an increase in ciphertext size (typically linear) and computation runtime (often quadratic) with the number of parties [42], which does not occur with the threshold FHE approach.

For ML applications, we usually consider the threshold FHE version of the CKKS scheme, which follows the design of Asharov et al. [43] and Mouchet et al. [35]. Threshold CKKS supports the same operations as the regular single-key CKKS. The only differences in threshold CKKS are:

- the *public key generation* is now distributed among all parties, each of which generates a public key share and broadcasts it to the other parties; these shares are then aggregated into the collective public key;
- the *rotation key generation* is done in a similar way to the public key generation;
- the *multiplication key generation* requires some extra rounds of communication with respect to the single-key variant;
- the *decryption* is also distributed among all parties, each of which generates a partial decryption share and broadcasts it to the other parties; these shares are then aggregated into the decryption result;
- the *bootstrapping* is done in a similar way to the decryption procedure, with the parties performing a masked partial decryption of the ciphertext and then adding an encryption of the negated mask to generate a refreshed encryption of the message. The masking requires ~3 additional multiplicative levels to achieve the desired statistical security [20].

The main drawbacks of MHE that often make it impractical in real-world contexts are the heavy computation costs and the significant communication overhead, especially due to the need for distributed bootstrapping, which can become a bottleneck in practical implementations. This is particularly relevant in ML scenarios when evaluating deep multiplicative circuits like neural networks. Additionally, the inherent noise in the encryption scheme and the approximation of non-linear functions can lead to a decrease in the model's accuracy.

5.2 Available Solutions for FL under FHE

Currently, the only available works that propose designs for FL under FHE are

- SPINDLE [33] for generalized linear models, and
- POSEIDON [20] for dense and convolutional neural networks.

Since deliverable D3.1 lists dense neural networks among the models that are deemed suitable for RE-SAMPLE, we will provide a privacy-preserving solution for a MLP model. Here, we focus on the description of POSEIDON. In this solution, the training parties instantiate the multiparty version of CKKS by performing a distributed generation of the keys. The orchestrator receives the public key, along with multiplication and rotation keys. Then, the orchestrator initialises the model's parameters, encrypts them, and the federated training process starts. All the computations are performed under encryption for a given number of iterations, after which the encrypted model can be used for prediction.

The inference phase can also be performed obliviously. The querier encrypts its query using the collective public key and sends the corresponding ciphertext to the training parties. Then, one of those performs a feedforward step of the model for the given encrypted input. The encrypted result undergoes a distributed decryption procedure, where each party computes its decryption share and sends it to the querier. Finally, the querier aggregates the shares to get the decryption of the model's output. Note that in this oblivious inference process none of the training parties can see the query's input nor output in plaintext, hence fully preserving the query's privacy.

5.2.1 Performance Limitations

Unfortunately, this solution is quite impractical for deep models or big datasets, since a training that would last a few hours in plaintext takes a few months under encryption [20]. The main cause for such a performance overhead is twofold:

- the high computational cost for the homomorphic operations, in particular for evaluating non-linear activation functions such as sigmoid and ReLU;
- the communication cost given by the bootstrapping protocol, which is distributed and must be invoked after every 1-2 layers.

6. Proposed Solution: Partial Encryption of the Model

In this section we describe the proposed PPML solution for RE-SAMPLE, built upon the fully encrypted solutions presented in Section 5. The underlying idea is that encrypting the full model might be unnecessary, as different portions of the model might leak different amounts of information. This leads to a trade-off between efficiency and privacy, orthogonal to the one between privacy and accuracy already provided by differential privacy. Here, we describe the general solution and assess it against state-of-the-art privacy attacks on publicly available datasets.

6.1 Design and Workflow

A flexible collaborative learning protocol that allows users to trade off privacy for efficiency has been designed. Our approach involves using an FHE scheme to encrypt the most vulnerable parts of the model and performing federated training on this partially encrypted model. The level of privacy protection is determined by the selection of layers to be encrypted (*secret layers* $\mathcal{L}_S$), while the remaining layers are left in plaintext (*exposed layers* $\mathcal{L}_E$). The more layers we encrypt, the less information potential adversaries can access, thus enhancing privacy, but it also leads to more computations performed under encryption, thus reducing efficiency. This flexibility allows our approach to achieve greater privacy than employing standard FL [16], while achieving a more practical level of efficiency than fully encrypted solutions [33, 20].

When performing feedforward and backpropagation on the model, we need to be careful about how to switch from secret to exposed layers and vice versa. Computations are conducted under encryption whenever an encrypted layer is encountered, possibly invoking bootstrapping to refresh intermediate computations. When an exposed layer is encountered, a decryption is called to allow continuing the training pass in plaintext (see ***Figure 1***).

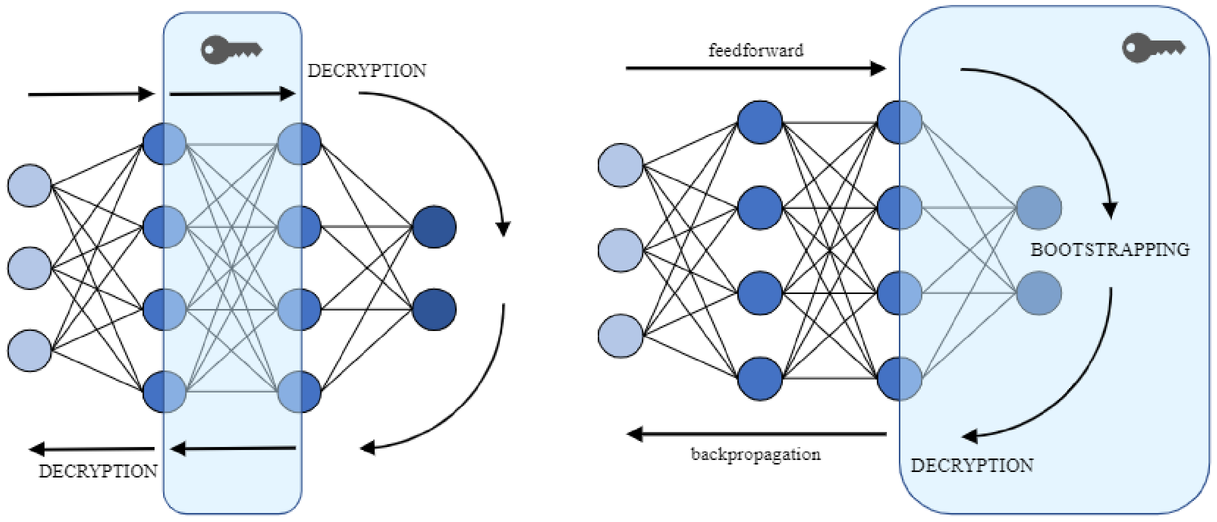

Figure 1: Diagram representation of the approach: encryption of a hidden layer on the left-hand side, encryption of the output layer on the right-hand side

6.1.1 Protocol Description

We describe the protocol for MLP models, trained with Stochastic Gradient Descent (SGD) optimizer, and Mean Squared Error (MSE) loss, but it can be easily generalized to any feed-forward model. Including simple momentum-based optimizers such as Nesterov Accelerated Gradient is straightforward and only requires an additional weight update. While adaptive optimizers such as AdaGrad [44], RMSProp [45], and Adam [46] may require additional care due to the division by the rescaling coefficient.

6.1.1.1 Global Training

The protocol involves N training parties $P_1, \dots, P_N$ and a central server S, whose role can potentially be taken by any P_i. The parties want to jointly train an MLP model, thus they agree on the model depth L, architecture, activation functions, training hyper-parameters, and on the set of layers to encrypt $\mathcal{L}_S$. The central server initiates the protocol by coordinating the FHE setup and key-generation phase, at the end of which $P_1, \ldots, P_N$ have their own private shares, and all actors possess the corresponding public key, relinearization, and rotation keys, which allow all parties to perform the necessary homomorphic operations. The central server initialises the model f in plaintext, by generating random weight matrices $w_1, \ldots, w_L$ and bias vectors $b_1, \ldots, b_L$ of appropriate sizes, according to the distribution given by the chosen initialization technique. Then, it uses the public key to encrypt the parameters w_i, b_i corresponding to the secret layers $L_i \in \mathcal{L}_S$, after proper encoding (see Section 5.1.1.1).

At this point, the training starts, and proceeds as in standard FL. The central server broadcast the partially encrypted model to $P_1, \ldots, P_N$. Each party P_i performs a certain number of local training iterations, and sends back the updated local model to the central server. The central server finally aggregates the local models, by averaging the parameters, using homomorphic addition and scalar multiplication by $1/N$ for the ones in $\mathcal{L}_S$. These steps are repeated for a fixed number of iterations E_g or until some convergence condition is satisfied (see Protocol 1).

```
Protocol 1 Global Training
ACTORS: central server $S$, training parties $P_1, \ldots, P_N$
 1: Parties agree on model depth $L$, architecture $a$, activation
    functions $\phi_j$, learning coefficient $\eta$, batch size $B$, param-
    eter initialization technique ModelInit, number of global
    iterations $E_g$, and the set of layers to encrypt $\mathcal{L}_S$
    FHE SCHEME SETUP:
 2: All actors collaborate to run the FHE setup and key
    generation, resulting in collective public key $pk$
    MODEL INITIALIZATION:
 3: $S$ initializes the model in plaintext $(w_1, b_1), \ldots, (w_L,$
    $b_L) \leftarrow \mathrm{ModelInit}(L, a)$
 4: $S$ encrypts parameters in secret layers:
 5: for $j = 1 \to L$ do
 6:     if $L_j \in \mathcal{L}_S$ then
 7:         $w_j \leftarrow \mathrm{Encrypt}(pk, w_j)$
 8:         $b_j \leftarrow \mathrm{Encrypt}(pk, b_j)$
 9:     end if
10: end for
    FEDERATED TRAINING
11: for all $g = 1 \to E_g$ do
12:     $S$ sends $(w_1, b_1), \ldots, (w_L, b_L)$ to the parties
        LOCAL TRAINING:
13:     Each $P_i$ runs $E_l$ local model updates, getting the local
        model $(w_1^i, b_1^i), \ldots, (w_L^i, b_L^i)$ and sends it to $S$ (see
        Protocol 2)
        AGGREGATION:
14:     $S$ aggregates the local models:
15:     for $j = 1 \to L$ do
16:         $w_j \leftarrow \frac{1}{N} \sum_{i=1}^{N} w_j^i$        (HE eval. if $L_j \in \mathcal{L}_S$)
17:         $b_j \leftarrow \frac{1}{N} \sum_{i=1}^{N} b_j^i$        (HE eval. if $L_j \in \mathcal{L}_S$)
18:     end for
19: end for
```

Protocol 1. Global Training.

6.1.1.2 Local Training

The local training subroutine, presented in Protocol 2, involves each party P_i performing E_l model updates locally before updating the central model. During each local update, a batch of size B is sampled from the local training set D_i. For each example in the batch, one training step is performed, which includes feedforward and backpropagation to compute gradients. These gradients are then averaged across the batch sample and used to perform a local model update. After E_l updates, the local model is sent to the central server, which proceeds to the aggregation step. We can add L2 regularization at the cost of an additional plaintext-scalar multiplication, by multiplying the weight matrices by $1 - \eta\lambda/B$ just before line 8, where λ is the weight decay coefficient.

```
Protocol 2 Local Training
 1: for $e = 1 \to E_l$ do
 2:     Sample $(x_1, y_1), \ldots, (x_B, y_B)$ from local dataset
 3:     for $b = 1 \to B$ do
 4:         Compute gradients $(\nabla w_1^b, \nabla b_1^b), \ldots, (\nabla w_L^b, \nabla b_L^b)$ by
            performing one training pass on $(x_b, y_b)$ (Protocol 3)
 5:     end for
 6:     Update local model:
 7:     for $j = 1 \to L$ do
 8:         $w_j \leftarrow w_j - \frac{\eta}{B}\sum_{b=1}^{B} \nabla w_j^b$      (HE eval. if $L_j \in \mathcal{L}_S$)
 9:         $b_j \leftarrow b_j - \frac{\eta}{B}\sum_{b=1}^{B} \nabla b_j^b$       (HE eval. if $L_j \in \mathcal{L}_S$)
10:     end for
11: end for
```

Protocol 2. Local Training.

Protocol 3 outlines one training pass of our approach. To feedforward an input through a partially encrypted model, we begin by feeding the vector in plaintext starting from the input layer. When an encrypted layer is encountered, the computation proceeds under encryption, with bootstrapping being called when necessary (we omit bootstrapping calls from the protocol description since their call frequency depends on the FHE parameters). As soon as an exposed layer is reached, a distributed decryption is invoked. The same process is followed during the backpropagation step. Note that if the last layer is encrypted, the loss is also computed under encryption.

```
Protocol 3 One Training Pass
 1: $l_0 \leftarrow x$
    FEEDFORWARD:
 2: for $j = 1 \to L$ do
 3:     $u_j \leftarrow l_{j-1} w_j + b_j$                  (HE eval. if $L_j \in \mathcal{L}_S$)
 4:     if $j < L \wedge L_j \in \mathcal{L}_S \wedge L_{j+1} \in \mathcal{L}_E$ then
 5:         $u_j \leftarrow \mathrm{Decrypt}(u_j)$
 6:     end if
 7:     $l_j \leftarrow \phi_j(u_j)$             (HE eval. if $(j = L \wedge L_L \in \mathcal{L}_S)$
                                      or $(j < L \wedge L_j, L_{j+1} \in \mathcal{L}_S)$)
 8: end for
    BACKPROPAGATION:
 9: $e_L \leftarrow y - l_L$                         (HE eval. if $L_L \in \mathcal{L}_S$)
10: $\nabla b_L \leftarrow e_L \phi'_L(u_L)$                 (HE eval. if $L_L \in \mathcal{L}_S$)
11: $\nabla w_L \leftarrow \nabla b_L l_{L-1}^T$                  (HE eval. if $L_L \in \mathcal{L}_S$)
12: for $j = L - 1 \to 1$ do
13:     $e_j \leftarrow \nabla b_{j+1} w_{j+1}^T$              (HE eval. if $L_{j+1} \in \mathcal{L}_S$)
14:     if $L_j \in \mathcal{L}_E \wedge L_{j+1} \in \mathcal{L}_S$ then
15:         $e_j \leftarrow \mathrm{Decrypt}(e_j)$
16:     end if
17:     $\nabla b_j \leftarrow e_j \phi'_j(u_j)$           (HE eval. if $L_j, L_{j+1} \in \mathcal{L}_S$)
18:     $\nabla w_j \leftarrow \nabla b_j l_{j-1}^T$            (HE eval. if $L_j, L_{j-1} \in \mathcal{L}_S$
                                         or $L_j, L_{j+1} \in \mathcal{L}_S$)
19: end for
```

Protocol 3. One Training Pass.

Unless we are at the last layer of the model, decrypting just after the linear transformation at line 5 is optimal. To show this, let us consider the case we are at the end of a group of encrypted layers, that is we are at layer L_j for some $j < L$, with $L_j \in \mathcal{L}_S$ and $L_{j+1} \in \mathcal{L}_E$. If instead of decrypting u_j, we perform an additional step under encryption, and decrypt after the evaluation of $\phi_j(u_j)$, then an adversary could just invert the activation function if bijective (e.g., sigmoid) or still get information about u_j for most of the activation functions commonly used. If we keep going under encryption for a step further, and decrypt for instance after the next linear transformation u_{j+1} in order to keep l_j private, then, since $L_{j+1} \in \mathcal{L}_E$, we would need to invoke a decryption for ∇w_{j+1}, which depends on l_j. However, an adversary could easily retrieve l_j given ∇w_{j+1} and ∇b_{j+1}, which is also in plaintext since $L_{j+1} \in \mathcal{L}_E$. Similar remarks hold for the decryption of the error in the backpropagation phase at line 15.

Consequently, when encrypting a single non-output layer, the corresponding layer output and gradient cannot be protected. Thus, to protect the initial or central portions of the model, at least two consecutive layers need to be encrypted, and even in that case, the gradient of the bias of the last layer of that group will be exposed. One approach to address this limitation is to omit the bias parameters on that specific layer.

An additional argument against encrypting only one non-output layer is the potential for an adversary to reconstruct the encrypted parameters. If the adversary can gather enough input and output pairs (x, y), where $y = wx + b$, they could retrieve the values of w and b by solving a system of linear equations with the layer parameters as the unknowns. Encrypting two consecutive layers (or the last one), already makes the system of equation significantly harder to solve. The system will involve many more variables and the activation function of the first layer as well, which is typically non-linear. As an additional measure, lowering the precision of the FHE scheme and introducing additional noise in the ciphertext can further complicate the reconstruction of the encrypted layers.

6.1.1.3 Prediction

After completing the training phase, the partially encrypted model can be directly used for predictions. However, cooperation among the training parties remains necessary for distributed bootstrapping and decryption calls. Prediction queries can be initiated by any of the training parties or an external entity. If the querier is one of the training parties, they can locally conduct the feedforward step and seek assistance from the others only for bootstrapping and decryption, including the potential output decryption if the last layer is also encrypted. If the querier is an external entity, additional precautions are needed due to the presence of exposed layers. To ensure the privacy of the query, the querier encrypts their input with the collective public key of the FHE scheme, and sends the encrypted input to one of the training parties. The selected party will then perform the feedforward pass on behalf of the querier. In this case, operations on the exposed layers must be adapted to work under encryption. While matrix multiplication and bias addition remain straightforward, the activation functions need to be approximated to be homomorphically evaluated, leading to a potential loss of accuracy. Moreover, unlike during training, the output of a group of adjacent layers should not be decrypted. (Note: It is possible to perform plaintext operations in exposed central layers, when they are sufficiently distant from both the input and output layers to lower the possibility of reconstructing the query input or output.) The final output of the model then remains encrypted, regardless of whether the last layer is private or not. At this point, the training parties can send the decryption shares of the output to the querier, who can then reconstruct the output in clear.

6.2 Assessment on Public Datasets

In this section, we evaluate our partially encrypted model approach for different choices of $\mathcal{L}_S$. Note that we conduct our experiments in this document in order to demonstrate the general approach and assess the trade-off of our approach for different data sets. Further experiments in the RE-SAMPLE specific context will be provided in D4.5: *Proof-of-concept privacy-preserving ML and data aggregation* and D4.6: *Report on the final parameter selection* at a later stage of this project.

6.2.1 Experimental Setup

We implement the prototype in C++, building on top of the OpenFHE library [47] for the multiparty CKKS functionalities (available at https://github.com/openfheorg/openfhe-development). Our implementation uses CKKS with a 5-bit integral precision, 55-bit decimal precision (scaling factor), a moduli tower with 8 levels, and a cyclotomic ring degree of 2^{15}. We use an N-out-of-N threshold scheme with additive sharing of the secret key, where all parties need to be present to perform decryption and bootstrapping. But the scheme can be easily modified to allow for arbitrary thresholds, i.e., t-out-of-N threshold FHE using Shamir secret sharing. To assess the performance of our prototype in a realistic scenario, we run the experiments within Mininet (available at https://github.com/mininet/mininet), a network emulator that allows us to configure different network topologies and impose constraints on bandwidth and network delay. Different virtual hosts are spawned within a server with an Intel Xeon Platinum 8358 running at 2.60 GHz, with 64 threads on 32 cores, and 512 GB RAM. For the evaluation in particular, we consider a setup with 3 training parties and a central server, communicating over TCP in a star topology network. The communication is constrained by 1Gbps bandwidth and 10ms network delay between the nodes.

Each party is provided with 30 examples from the MNIST dataset, and they jointly train an MLP model with two hidden layers of size 30, 20. Each layer uses sigmoid activation functions, which is approximated in $[-10, 10]$ by a polynomial of degree 13 for HE evaluation. To simplify the analysis of the trade-off given by the choice of $\mathcal{L}_S$, we decided to focus on the specific case of encrypting only one group of contiguous layers containing the output layer, which, from the investigation in [2], seems to be one of the most meaningful settings for our approach when considering inference attacks. That is, given a model f of depth L, we then have $\mathcal{L}_E = \{L_1, \dots, L_T\}$ and $\mathcal{L}_E = \{L_{T+1}, \dots, L_L\}$ for some $T \in \{0, \dots, L\}$. This way, the trade-off is controlled by the one-dimensional parameter T: when $T = 0$ we are in the extreme case of FL with full encryption of the model [20], while when $T = L$ we are in the extreme of FL in plaintext [16]. We report the results for the non-optimized version of our framework.

6.2.2 Datasets and Models Description

We selected well-known public datasets widely used in the PPML literature: with Texas-100, Purchase-100, Locations our datasets include tabular data, and with AT&T, MNIST, EMNIST Letters, LFW our datasets include images. This mix ensures a comprehensive evaluation of the privacy attacks in our prototype. We highlight that this work's main goal is not to achieve the highest possible accuracy on the given datasets, but to investigate the efficacy of privacy attacks across different layers of our target model. To accomplish this, we deliberately subsampled some of the datasets, using a reduced dataset for training the models. By doing so, we aim to create a vulnerable model that is more susceptible to privacy attacks. There are various ways to make a model vulnerable, constraining the training set to a subset is comparable with real-world scenarios, where training parties might struggle with limited data availability. This approach also helps in expediting the experimental assessment of our encrypted training solution, as the experiments can be run within a reasonable timeframe given the reduced training data size.

As for the models, we primarily focus on MLP architectures, for compatibility with our FHE prototype, using the plain SGD optimizer and minimising the MSE loss. Specifically, we train an MLP with two hidden layers of size 30 and 20 on the MNIST datasets, and 256, 128, and 64 on the Location dataset. For Purchase100 and Texas100, we adopt the MLP architecture proposed in [2], which consists of hidden layers with sizes 1024, 512, 256, and 128. Additionally, we train an MLP with two hidden layers of size 64 on the EMNIST Letters dataset, and a Convolutional Neural Network (CNN) on the LFW dataset. For the CNN, we adopt the architecture proposed in [13], which consists of three convolutional layers with 32, 64, and 128 filters, each with a 3x3 kernel and a max pooling layer, followed by two fully connected layers of size 256 and 2.

We will now give a more detailed description of the datasets we used for our experiments.

AT&T Database of Faces. This face dataset (available at https://cam-orl.co.uk/facedatabase.html) was created at the AT&T Laboratories Cambridge. It consists of 400 grey-scale images of size 112x92, depicting the faces of 40 individuals in various lighting conditions and facial expressions.

EMNIST Letters. This letter dataset (available at https://www.nist.gov/itl/products-and-services/emnist-dataset) is part of the extended version of the MNIST dataset by NIST [48]. It consists of 145,600 grey-scale images, representing both upper- and lower-case handwritten letters, which has been centred and resized to 28x28. The dataset contains 26 classes, one for each letter from 'a' to 'z'.

Labelled Faces in the Wild (LFW). This face dataset (available at http://vis-www.cs.umass.edu/lfw/) was developed by researchers at the University of Massachusetts, Amherst [49]. It consists of 13,233 RGB images, depicting the faces of 5,749 individuals. The dataset has been further labelled with attributes such as gender, race, age group, hair style, and eyewear.

Locations. This location dataset (available at https://github.com/privacytrustlab/datasets) was created by the authors of [4] from Foursquare check-in data for the city of Bangkok. The processed dataset contains 5010 examples, each corresponding to a unique user. Each record comprises 446 binary features, indicating whether a user visited a specific region or location type. The data is clustered into 30 classes, representing different geosocial types. Following [4], we use 1200 examples for training, and the remaining data for validation.

MNIST. Standard handwritten digits dataset by NIST (available at https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz). It consists of 70,000 grey-scale images, centred, and resized to 28x28. The dataset contains 10 classes, one for each digit from '0' to '9'. Due to the small number of classes and the low feature variability within the same class, this dataset has been observed to be particularly resilient against membership inference attacks [4]. For evaluation purposes, we want to start from a situation in which the target model is vulnerable. Thus, we drastically reduce the training set to a mere 100 examples. For compatibility with our current implementation of the prototype, we resize the images to 8x8.

Purchase-100. This purchase dataset (available at https://github.com/privacytrustlab/datasets) was created by the authors of [4] starting from Kaggle's "acquire valued shoppers" challenge dataset, containing the shopping history data of several users. The processed dataset contains 197,324 examples, each corresponding to a unique user. Each record comprises 600 binary features, indicating whether a user purchased a given product. The data is clustered into 100 classes, representing different purchase styles. For our experiments, we use 1000 examples for training and the remaining data for validation.

Texas-100. This hospital dataset (available at https://github.com/privacytrustlab/datasets) was created by the authors of [4] starting from the Hospital Discharge Data records released by the Texas Department of State Health Services. The processed dataset contains 67,330 examples, each corresponding to a unique patient. Each record comprises 6,169 binary features, containing information about the patient, the causes of injury, the diagnosis, and the procedures the patient underwent. The data is clustered into 100 classes, representing the 100 most frequent medical procedures present. For our experiments, we use 1000 examples for training and the remaining data for validation.

6.2.3 Runtime and Communication Performance

In this section, we discuss the efficiency of the proposed approach in terms of computation time, communication time, and communication size, by performing both a theoretical analysis and an experimental assessment.

6.2.3.1 Theoretical Analysis

Compared to a fully encrypted approach, partially encrypted models offer significant efficiency advantages, including reduced number of computations under encryption, lighter model updates, and fewer communication rounds. Providing a precise efficiency analysis is challenging due to the variable presence of bootstrapping and the exceptions to which computation is performed homomorphically depending on which layers are encrypted. Nonetheless, we can offer general considerations that cover any choice of $\mathcal{L}_S$. To simplify the analysis, we assume all layers to have same order of magnitude sizes, and plaintext size and operation costs to be negligible with respect to their encrypted counterpart.

In general, since computations are carried out in plaintext in the exposed layers, we expect a lower-bound for the gain in computational complexity to be at least linear in the number of exposed layers relative to the total number of layers in the model, i.e. $|\mathcal{L}_E|/L$. Additionally, we can avoid the homomorphic evaluation of the last activation function in each group of encrypted layers (except the ones containing the last layer). Moreover, as the training parties decrypt their computations at the end of each encrypted block, the need for distributed bootstrapping decreases or even disappears. This results in gains in computation efficiency, communication size, and communication rounds, as each bootstrapping process typically requires one round-trip of communication. This advantage is amplified by the fact that bootstrapping in the CKKS scheme needs the ciphertext to have some levels left, further increasing the computational overhead in fully encrypted models. Depending on the number of contiguous encrypted layers, even faster somewhat HE schemes can be adopted.

Regarding the communication size during model update and broadcast, we again observe a linear gain in $|\mathcal{L}_E|/L$, as the model parameters corresponding to exposed layers are sent in plaintext. On the other hand, during the aggregation phase, fewer parameters need to be averaged under encryption, leading to an additional gain in terms of computational complexity. Finally, we note that in the optimized version of our solution, the performance gain factor $|\mathcal{L}_E|/L$ changes according to the number of exposed layers across the training epochs.

6.2.3.2 Experimental Evaluation

For the experimental evaluation, we only report the results related to the MNIST dataset, since no significant difference appeared among different datasets. As shown in Table 1, the efficiency of our approach scales approximately linearly with T, both in terms of run time and communication size.

Table 1: Execution time and communication size of our approach on the MNIST dataset for a 3-layer MLP, with varying levels of layer encryption: none (T = 3), last layer (T = 2), last two layers (T = 1), and full model encryption (T = 0)

T	Training		Inference	
	Run time (h)	Comm. size (GB)	Run time (s)	Comm. size (MB)
0	2232.7	3083.4	76.8	36.3
1	1504.1	1988.8	50.0	23.3
2	726.7	980.8	23.5	10.3
3	1.8e-2	3.9e-2	5.0e-3	0.0

The reported run time and communication size have been averaged among the training parties for consistency. In particular, the communication size refers to the total volume of messages received and sent per party over 300 epochs of training. The model achieves the same test accuracy as its plaintext counterpart. The noise from the FHE scheme and the approximation error of the activation functions do not have a significant impact on the training procedure.

In ***Figure 2***, we observe that communication time is the dominant factor on the overall performance.

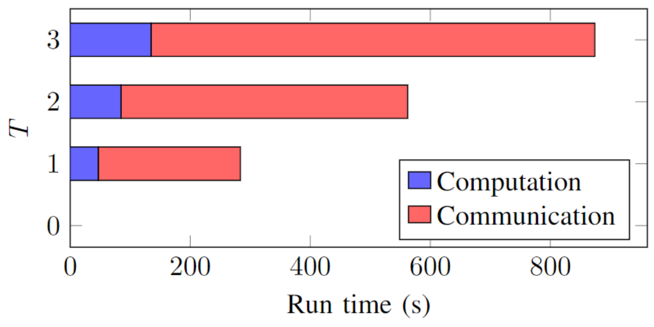

Figure 2: Computation vs. communication time for one training pass in our approach on MNIST for a 3-layer MLP, with varying levels of encryption: full model encryption (T = 0), last two layers (T = 1), last layer (T = 2), and none (T = 3)

Note that the high communication and overall run time are partly due to our current implementation being still a prototype, and the use of relatively high-degree approximation for the activation functions. We acknowledge that there are large margins for optimization, starting from compressing the model updates before transmitting them. Also, using a tile-packing of the weight matrices as suggested in [50] may increase the performance of our approach. However, our main focus is on comparing the relative efficiency measures between different choices of encrypted layers, rather than their absolute values.

6.2.3.3 *Micro-benchmark*

In this section, we provide measurements for various FHE functionalities, enabling estimation of our approach's scalability for different model architectures. The execution run time and communication size per training party, averaged over multiple runs, are presented in Table 2. For each functionality, we have divided the execution time into computation and communication time. Note that the missing time from the total execution time reflects the idle time when parties are waiting for other parties to complete their computations in order to proceed.

Table 2: Microbenchmarks of different FHE functionalities, for 3 parties, 5-bit integral precision, 55-bit decimal precision, 8-level moduli tower, and 215 cyclotomic ring degree. The "one layer" functionality refers to a fully connected layer, while the "one pass" functionality refers to an entire pass (forward or backward) of the model under encryption

Functionality	Execution time (s)			Comm.
	Comp.	Comm.	Total	size (MB)
Vector-Matrix mult. (64x32)	6.289	-	6.289	-
Vector-Matrix mult. (32x16)	3.291	-	3.291	-
Sigmoid evaluation (deg. 13)	11.046	-	11.046	-
FHE setup	0.043	0.020	0.143	0.001
Pub./Priv. key gen.	0.151	0.347	0.611	12.005
Relin. key gen.	0.900	0.788	2.336	81.020
Rot. key gen. (x23)	4.378	86.458	101.662	1296.182
Model generation	0.989	-	0.989	-
Decryption	0.376	34.090	34.999	15.013
Bootstrapping	1.265	49.287	51.160	22.767
One layer forward	20.294	147.737	170.125	68.302
One layer backward	24.617	98.699	124.262	45.535
One pass forward	60.944	443.211	510.438	204.907
One pass backward	73.913	296.096	372.849	136.605
One training pass	134.857	739.307	883.287	341.512
Send model	0.157	17.674	17.831	27.015
Update params.	2.039	-	2.039	-
Aggregation	1.697	56.454	60.265	37.532

6.2.4 *Privacy Analysis*

In this section, we sketch a security proof for the encrypted layers of the model, and we discuss the capabilities of different types of adversaries for each class of privacy attack against the exposed layers.

6.2.4.1 *Security Proof for the Encrypted Layers*

Our approach aims to preserve the privacy of the training data, during both the training and prediction phases, by encrypting the most vulnerable layers of the model. In the semi-honest setting, we prove that no party, including the central server, can learn more information about the training data of any other party or the model parameters corresponding to any layer in $\mathcal{L}_S$, other than what can be deduced from their own data (including the model output, in case of predictions), and from the parameters and intermediate computations of the layers in $\mathcal{L}_E$. In the case of predictions requested by an external entity, we can make this claim stronger, as the querier should not learn anything, other than what can be deduced from only their own input data and the query output.

For the security proof, we proceed as in [20], but assuming the simulator is also given access to the parameters of the exposed layers at each iteration, and to the output of each decryption call. The idea is to see the overall scheme as a composition of the underlying FHE protocols, which are all simulatable. For the basic protocols like key generation and decryption, we rely on the proofs by Mouchet et al. [35], while for the distributed bootstrapping, we rely on the proof by Sav et al. [20]. Note that the security of our approach does not hold for a malicious adversary, which can exploit the decryption call in the protocol decrypt the secret layers.

6.2.4.2 Privacy Assessment for the Exposed Layers

Threat model. We study privacy attacks in the *grey-box setting*, which represents a more flexible threat model generalising white-box scenarios for privacy-attacks, by considering intermediate adversary capabilities. In this grey-box setting, the adversary has only access to the subset of the model parameters corresponding to the exposed layers, for which the adversary has the same access as in the white-box setting. Thus, the adversary can compute the loss value for a given labelled example only if the last layer is exposed. Conversely, the adversary has no access to any parameters of the encrypted layers.

In the rest of the section, we discuss whether specific attacks can still run on the exposed layers. We consider three threat model configurations, based on the possible combinations of the ML and cryptographic adversary's capabilities:

1. ML-passive and crypto-passive, where the adversary follows the protocol and can only use inputs from the original dataset (no maliciously crafted inputs);
2. ML-active and crypto-passive, where the adversary follows the protocol but may craft malicious inputs for the training procedure;
3. ML-active and crypto-active, where the adversary can arbitrarily deviate from the protocol and may craft arbitrary malicious input.

Note that the distinction between the first two settings is important in real-world scenarios since it has implications in terms of detectability and liability. If a client becomes corrupted during the training process, an ML-active attack is potentially more detectable than an ML-passive attack. Detection can occur by analysing the intermediate updates or the final model, or by employing some form of commitment to the training dataset.

In general, if at least one layer is encrypted, no attack can be carried out in the ML-passive and crypto-passive settings, as they all require to at least perform an inference on an example not belonging to the original training dataset (the target of inference, or the dummy example for model inversion). In the ML-active and crypto-passive settings, the inference attacks can work limited to the exposed layers, while the model inversion attacks can work only if the first layer is exposed. In the ML-active and crypto-active settings, all attacks are feasible, as it would be enough to ask for the decryption of the secret parameters. The active versions of each attack are possible as well. In Table 3, we outline the capabilities of each attack discussed in Section Privacy Attacks4.1 for the threat model configurations described above. Note that, in contrast to FL in plaintext, the presence of encrypted layers restricts the adversary from freely conducting any inference on the model.

Table 3: Description of attack capabilities in different threat models, for different privacy attacks. For a description of the active variant of the attacks, we refer to the corresponding works

Attack	Class	ML-passive & crypto-passive	ML-active & crypto-passive	ML-active & crypto-active
Nasr et al. (2019)	Membership inference	The attack is not possible, since the attacker cannot pass its target data point through the model.	The attack is possible, but limited to its passive variant on the exposed layers.	The active variant of the attack is also possible, since the attacker can perform gradient ascent.
Fredrikson et al. (2015)	Model inversion	The attack is not possible, since the attacker cannot pass the dummy input through the model.	The attack is not possible if the first layer is encrypted, since the attacker cannot backpropagate over the input layer.	The attack is always possible, the malicious adversary can decrypt the first layer if necessary.
Hitaj et al. (2017)	Model inversion	The attack is not possible, since the	The attack is not possible if the first	The attack is always possible,

		attacker cannot pass the generator output through the model.	layer is encrypted, since the attacker cannot backpropagate to the generator.	the malicious adversary can decrypt the first layer if necessary.
Zhu et al. (2019)	Model inversion	The attack is not possible, since the attacker cannot pass the dummy input through the model.	The attack is not possible, since the attacker cannot compute the derivative of the gradients with respect to the input.	The attack is always possible, the malicious adversary can decrypt any layer if necessary.
Melis et al. (2019)	Property inference	The attack is not possible, since the attacker cannot pass the target batch through the model.	The attack is possible, but limited to the exposed layers. The active variant is not possible if the last layer is encrypted.	The attack is always possible, the malicious adversary can decrypt any layer if necessary.

More details about the individual attacks is as follows.

Membership Inference. In the grey-box setting, the efficacy of membership inference attacks heavily depends on access to the last layers of the model. While the initial layers of a neural network tend to extract simple features from the input, enabling them to generalize well, the later layers specialize in detecting higher-level abstract features in the input, making them prone to overfitting and memorising the specific training examples. For instance, in a CNN model trained for image classification, you can expect the first layers to learn more about edges and abstract shapes of the input image, while the last layers more about intricate texture and artifacts within those shapes [51]. Moreover, as the neural network progresses to the later layers, the parameter capacity increases, causing the target model to store information about the exact training samples [2]. This behaviour can be attributed to the vanishing gradient effect, where the impact of a training step diminishes for the earlier layers compared to the later layers. Therefore, if the last layers of the model are accessible, membership inference attacks tend to be stronger due to the higher degree of membership information leakage.

In line with the results by Nasr et al. attacking the last layers, we observe a similar effect for our generalized setting: the combination of multiple (intermediate) layers does not leak significantly more membership information than the just the last of those layers. For instance, attacking layers 1, 3, and 4 of a given model does not provide a significant advantage over attacking just layer 4. Consequently, we simplified our experimental setting and attack each layer individually and do not expect significantly different accuracy compared to attacks that include any combination of previous layers. In scenarios where the attacker knows the ground-truth label associated with the target example, they can perform backpropagation. We account for this case by conducting an additional cycle of attacks, where the attack model is provided with the ground-truth label, the layer gradient, and, when the last layer is accessible, the loss value.

The number of members and non-members used for both training and testing the attack model is the same, resulting in a baseline attack accuracy of 50%. In ***Figure 3***, we present the outcome of our experimental assessment on different datasets. We report the average and maximum attack accuracy over four runs. Our experimental results confirm the trend of the later layers of a model to leak more information compared to earlier layers. In particular, the very last layer leaks considerably more information than the others. This is especially evident in the cases of Purchase100 and MNIST, where the attack accuracy for layer output increases from 55.12% to 90.10% (~7.8 times increase offset to the baseline) and from 50.81% to 64.49%

(~17.8 times increase offset to the baseline), respectively, when passing from the second-to-last layer to the last layer.

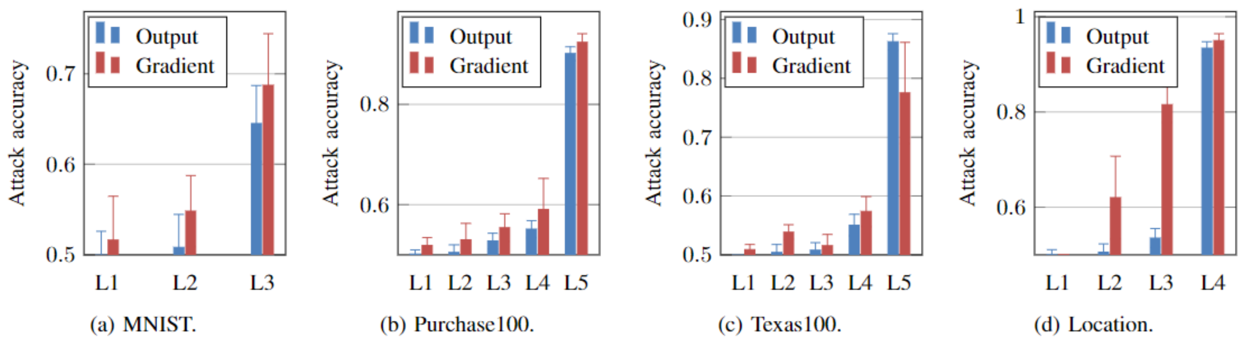

Figure 3: Layer-wise accuracy of the white-box membership inference attack by Nasr et al. (2019) against different datasets and models, exploiting both the layer's output and gradient

Additionally, in line with the findings of Nasr et al [2], our experiments confirm that the availability of gradients contributes to a higher attack accuracy.

Model inversion. All the model inversion attacks described in Section 4.1.2.2 share the common requirement of computing the derivative of the target model's loss with respect to the model input: $\partial L/\partial x$, which can be written as $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial l_1}\frac{\partial l_1}{\partial x}$, where l_1 is the output of the first layer. To compute this derivative, the attacker then needs access to the first layer's gradients and parameters. Blocking access to the first layer straightforwardly prevents the attack by Fredrikson et al. [9]. This limitation also prevents backpropagation from the target model to the generator for GAN-based approaches like [10], and hinders the ability to solve the gradient difference minimization problem in the case of the attack by Zhu et al. [7]. We conclude that denying the attacker access to the first layer of the model appears to be sufficient in preventing these specific types of model inversion attack, thus no experimental assessment is needed in this case. However, we refrain from making a general claim, as there might still be potential attacks that can circumvent this limitation and leave this as future research direction.

Property Inference. Due to the lack of a clear understanding of the underlying causes of property inference leakage, it is challenging to predict how such attacks will scale in the grey-box setting. The general idea is that the less gradients are exposed to the attacker, the less information is available for inference. However, it remains unclear whether specific types of layers (e.g., convolutional or fully-connected) or their positions in the model contribute to higher or lower information leakage.

To assess this category of attacks in the grey-box setting, we build upon the white-box property inference attack proposed by Melis et al. [13] and adapt it to target only a subset of the gradients. This attack works in batches, aiming to determine whether a batch of data points enjoys the target property or not. In our variant of the attack, we feed to the attack model only the gradients computed with respect to parameters in the exposed layers.

We conduct the assessment on two datasets: Labeled Faces in the Wild (LFW) and EMNIST letters. For the LFW dataset we train a CNN model following the architecture provided in [13], with 3 convolutional and 2 fully connected layers, using gender as main classification task, and race:black as inference task, which has been reported to yield the highest attack rate. While for the EMNIST letter dataset, we train a custom MLP model with 3 layers, using the standard 26 letters classification as the main task, and the letter case (upper or lower) as the inference task. The gradients of the exposed layers are fed to a Random Forest classifier with 50 trees, and the attack accuracy is averaged over multiple instances of the attack model. For both datasets, we use batches of size 32, which are balanced with respect to the inference property, resulting in an attack accuracy baseline of 50%.

For the CNN model trained on LFW, we did not find any specific patterns indicating whether some layers are more or less susceptible to inference than others. The attack exhibited significant variability when conducting multiple iterations of training the target model and conducting repeated testing. In Table 4, we report the attack accuracy layer-wise for multiple attempts of the experiment, revealing no consistent vulnerability or resistance of any layer across the runs. On the other hand, for the MLP model trained on the EMNIST letters dataset, the attack did not achieve accuracy significantly above the baseline, regardless of the choice of the layers to attack (including combinations of multiple layers).

Table 4: Property inference attack by Melis et al. (2019) against 5-layer CNN trained on the LFW dataset. The attack accuracy is reported per layer, demonstrating high variability across multiple training attempts of the same target model

Run	Conv. 1	Conv. 2	Conv. 3	Dense 1	Dense 2
1	66.88	81.44	88.13	95.06	81.69
2	60.25	74.31	72.00	53.19	56.13
3	63.31	69.63	79.06	66.25	55.88
4	86.69	93.44	91.75	99.88	96.00

Mitigation for the Exposed Layers. While encryption can prevent attackers from directly targeting the most vulnerable layers of a model, there remains a risk of information leakage from exposed layers, as shown above. Mitigating this potential leakage is out of the scope of this work, but we propose a few ideas as follows. Incorporating differential privacy into the exposed layers of the model would provide a theoretical-level privacy guarantee, at the cost of introducing an accuracy element in the trade-off. We describe a possible implementation of DP to our solution in Section 6.3, where we show how to adapt the approach by Shokri and Shmatikov [14] to partially encrypted models. Additionally, allowing for larger encryption error on the secret layers by reducing the precision of the FHE scheme. This error would propagate to the exposed layers during backpropagation, inducing a DP-like effect. Finally, secure aggregation [28] could be used to conceal the individual model updates corresponding to the exposed layers in case of a corrupted central server.

6.2.5 Trade-Off between Privacy and Efficiency

The more layers we encrypt, the higher the privacy, as less parameters are available to a potential adversary. However, encrypting more layers also leads to lower performance due to the overhead introduced by homomorphic evaluations and distributed bootstrapping. Thus, there is a trade-off between privacy and efficiency when deciding how many and which layers to encrypt in a model. The optimal choice depends on the specific use case, and in particular on the types of attacks one wants to protect the model from, and the desired balance between privacy and performance. In ***Figure 4***, we represent this trade-off, using membership inference accuracy as the metric for privacy leakage.

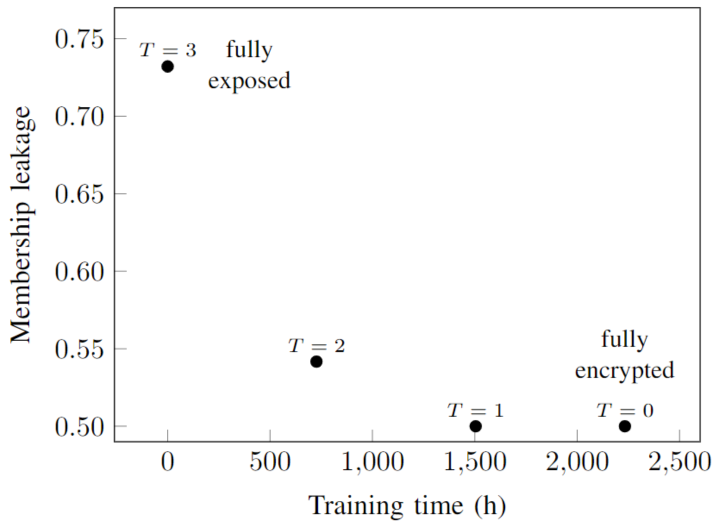

Figure 4: Trade-off between privacy and efficiency for the MNIST dense neural network, where the privacy is measured by the membership inference attack by Nasr et al. (2019) on gradients, assuming attacker corrupted 2 out of 3 parties

Each point on the plot represents a particular configuration of our approach, for different choices of $\mathcal{L}_S$. The optimum point of the trade-off occurs when both the leakage and the training time are minimized (i.e., at the origin point of the chart).

6.2.5.1 Comparison with Prior Work

The flexibility of our approach allows the user to sacrifice some privacy in order to gain training performance in terms of computation and communication time compared to fully encrypted solution like SPINDLE [33] or POSEIDON [20]. At the same time, it provides higher privacy levels than training entirely in plaintext, without compromising significant accuracy. Note that setting $T = 0$ corresponds to the original POSEIDON idea resulting in a fully encrypted model.

For the very specific case presented in ***Figure 4***, encrypting only the last layer ($T = 2$) provides a good trade-off. It offers a membership leakage very close to a random guess (54.17%, a 5.6 times reduction from the fully plaintext solution's 73.21%, relative to the random guess baseline), while reducing the training time with respect to the fully encrypted solution by a factor of 3.1. Assuming the adversary does not possess the target label, the leakage reduction factor increases to 17.8. The advantage provided by our solution may become even more evident for models with deeper architectures [52, 53, 54], particularly in settings with constrained communication networks.

6.3 Integration with Differential Privacy

To mitigate the leakage from the exposed layers additional privacy-enhancing techniques can be employed, such as differential privacy [1]. Applying DP to the parameters or gradients of exposed layers would provide a theoretical privacy guarantee to our solution, albeit introducing an additional trade-off between privacy and accuracy. By applying noise only on the exposed layers, rather than the entire model, our approach can achieve a higher level of privacy for the same privacy budget compared to standard FL solutions with differential privacy [14, 19, 55, 56].

We adapt the approach of Shokri and Shmatikov [14], which uses the sparse vector technique [57, 58] to privately upload a small, perturbed subset of the gradients to the global model. Given a privacy budget ϵ per epoch allocated to each training party P_i, we split this budget among the exposed parameters, and use Laplacian mechanism to add noise to the corresponding gradient value. The sensitivity of the training mechanism is estimated by clipping the gradient values within the range $[-\gamma, \gamma]$, resulting in a sensitivity of 2γ. The clipping range value should be independent of the specific training dataset, to avoid leaking sensitive information. We suggest setting it by calculating the median of the unclipped gradients over the course of training, as proposed in [18].

For each value g in ∇w_j for $L_j \in \mathcal{L}_E$, random Laplacian noise $r_g \sim \mathrm{Lap}(2c\gamma/\epsilon)$ is generated, where γ_g is the estimated clipping bound for g, and c is the total number of exposed gradients $c = \sum_{L_j \in \mathcal{L}_E}(|\nabla w_j| + |\nabla b_j|)$. The gradient g is then clipped within $[-\gamma_g, \gamma_g]$, and the noise r_g is added before uploading to the central server. A similar process can be followed for the exposed bias gradients. These operations are performed on the aggregated gradient obtained after several local iterations, each computed over a randomly sampled batch. Applying noise to each computed gradients could be done as well, and the overall effect over batches can be analysed using the privacy amplification theorem [59, 60]. More advanced techniques, such as privacy accountants [19, 18], can also be potentially adapted to work in our partially encrypted model solution.

6.4 Further Optimization: Delayed Encryption

In this section, we exploit the fact that some attacks start to be effective only after some number of training epochs to further optimize our solution.

Attacking Intermediate Models: Membership Inference. A model acquires more information about its training data the more training iterations it undergoes, thus leaking progressively more information as it approaches the end of the training process. To assess how the membership leakage changes across the epochs, we use the attack by Nasr et al. [2] against the intermediate models. Specifically, in ***Figure 5***, we present the attack accuracy against the model trained on the MNIST dataset. We carry out the attack at 10-epoch intervals, targeting each layer within the model independently. Our experiment reveals a consistent upward trend in attack accuracy with respect to the number of training epochs, especially for the later layers. Notably, a significant deviation from the attack baseline appears only from epoch 90.

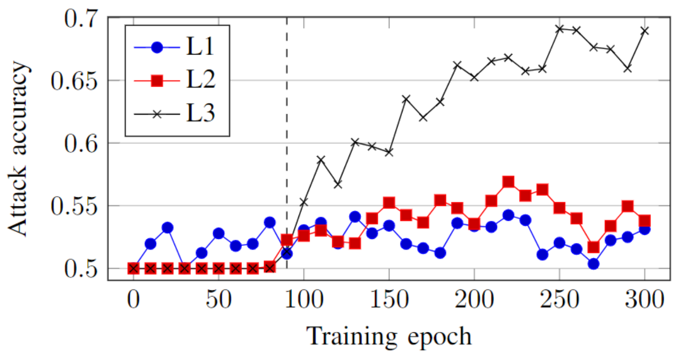

Figure 5: Layer-wise accuracy of the membership inference attack by Nasr et al. (2019) against intermediate models for the MNIST classification task. The model leaks more membership information as the number of training epochs grows. This behaviour is particularly evident for the output layer

Attacking Intermediate Models: Model Inversion. We also evaluated the effectiveness of model inversion attacks against intermediate training models. Similarly to membership inference, our experiments reveal that as the number of training epochs increases, the reconstructed class representative becomes more and more visually similar to the corresponding training examples. In ***Figure 6***, we display the reconstruction of a face from the AT&T dataset performed during different training epochs. We use the model inversion attack by Fredrikson et al. [9]. Following their work, we train an MLP with one hidden layer with 3000 nodes, sigmoid activation function, and a SoftMax output layer, using the SGD optimizer and cross-entropy loss function. The reconstruction becomes more and more clear as the training proceeds. However, we notice that the attack works already well even after just a few epochs. This happens since a model inversion attack works well as soon as the model is generalising well enough.

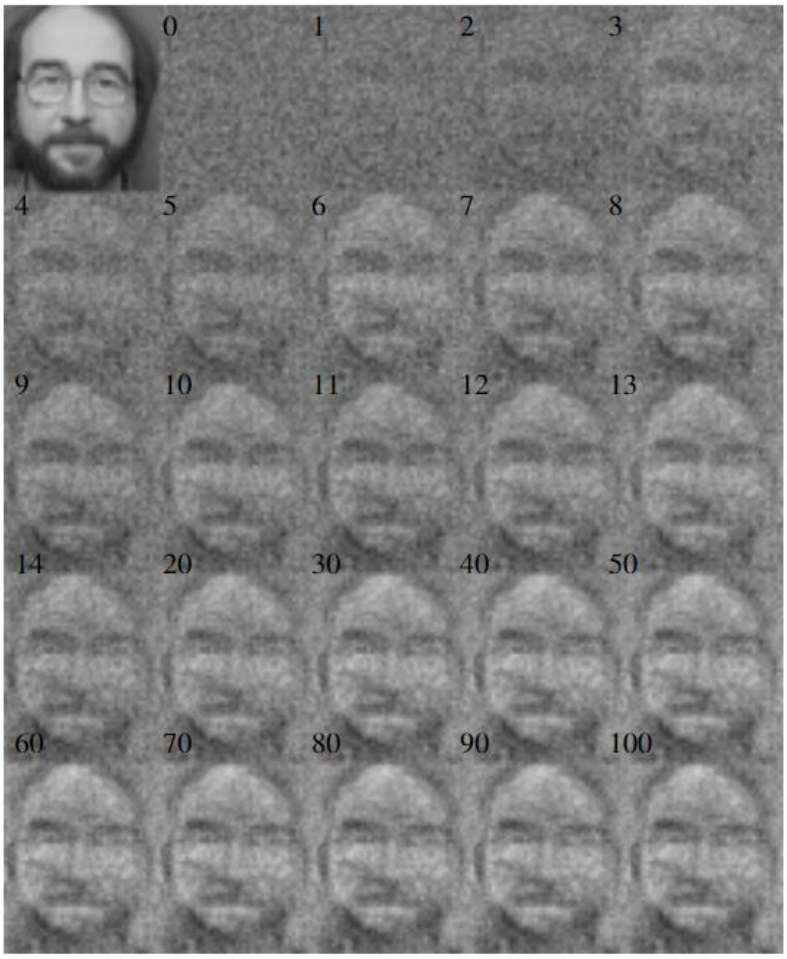

Figure 6: Reconstruction of a face in the AT&T dataset performed at different training epochs. The first picture is a class representative, while the number at the top-left of each picture denotes the corresponding training epoch of the model

Attacking Intermediate Models: Property Inference. We also assessed how property inference varies across training epochs. However, the experiments were inconclusive as no consistent trend emerged from the attack accuracy. In ***Figure 7***, we report the attack accuracy of the property inference attack by Melis et al. [13] against intermediate training models for the LFW classification task. The attack is performed every 10 training epochs, and it targets each layer of the model individually to get better insights. The model exhibits a general upward trend for information leakage as the number of training epochs grows. However, this trend is not much consistent, and the leakage is already substantial since the very beginning.

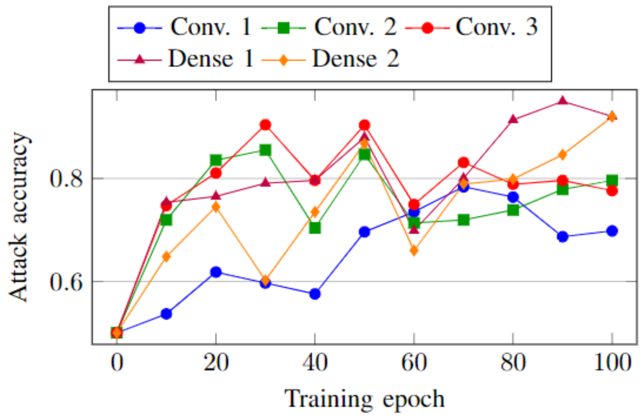

Figure 7: Layer-wise accuracy of a property inference attack against intermediate models for the LFW classification task

Optimization: Delayed Encryption. We can leverage the investigation of privacy attacks on intermediate training models to further optimize our solution. An optimization for our solution consists of starting the federated learning process fully in plaintext and encrypting (some of) the layers only once the model becomes vulnerable. The number of layers to encrypt can be dynamically adjusted during the training process. We will discuss this more in detail in Section 6.5.

6.5 Hyperparameters Choice

In this section we suggest a procedure to choose which layers to encrypt in a FL setting. Since the privacy leakage of a model strongly depends on the training dataset, there is no general-purpose guideline on how many and which layers to encrypt. A practical approach we propose consists of getting a lower bound estimate on the privacy leakage through an assessment on the local training data. To do so, the parties can train a dummy model on their own private datasets and perform a privacy assessment locally. Instead of exploring all possible combinations of private-exposed layers, the parties can rely on the insights discussed in Section 6.2.4 to determine which configurations are the most meaningful to assess. Since each local dataset is a subset of the joint dataset, the privacy leakage assessed locally provides an empirical worst-case for the privacy leakage of the joint model. From the efficiency point of view, the parties can use the insights from Section 6.2.3, by also taking into account their specific computation and communication constraints (e.g., bandwidth and network delay between the nodes). Finally, the parties can collectively agree on which layers to encrypt by leveraging MPC techniques, avoiding leaking potential information about their local dataset. Depending on the specific situation and requirements, the parties can perform a majority vote or compute the union of the local choices to reach a consensus on the layers to be encrypted.

We provide an example of how to agree on $\mathcal{L}_S$ for the optimized version of our solution in case membership inference attacks are considered. Each party P_i trains a model locally and assess it against the considered privacy attack across the training epochs, as in ***Figure 5***. Then, they select a privacy threshold $\tau_i \in [0.5, 1]$, which fixes an upper bound on the model leakage they are willing to allow. The party then proceeds to compute their preferred choice for the layers to encrypt $\mathcal{L}_S^{i,g}$ for each epoch $g = 1, \dots, E_g$ as the minimum set of layers that keeps the attack accuracy under τ_i. For consistency reasons, if a layer L_j is included in $\mathcal{L}_S^{i,g}$, then all subsequent layers L_k for $k > j$ should be included as well. Moreover, the layer should be included in all the future epochs as well, that is $L_j \in \mathcal{L}_S^{i,g'}$ for all $g' > g$. Then, the parties use MPC to compute $\mathcal{L}_S^g$ as the union of the $\mathcal{L}_S^{i,g}$ for each $g = 1, \dots, E_g$.

7. Conclusions and Next Steps

In this deliverable, we presented a flexible solution for privacy-preserving training of neural networks in a federated setting such as RE-SAMPLE. Our system allows users to trade-off little privacy for higher training performance, by selectively encrypting specific portions of the model using a multiparty FHE scheme. Through an investigation of various privacy attacks in the grey-box setting, where the adversary's access is limited to the unencrypted layers of the model, we determine the layers that tend to leak more information and, consequently, identify which layers are advisable to encrypt. Our findings indicate that encrypting the last layers is particularly effective to mitigate membership inference attacks, while encrypting the first layers helps preventing model inversion attacks.

The next steps consist of:

1. implementing the prototype within the RE-SAMPLE framework in D4.5, where we will detail the API calls and the interactions among the hospitals and the coordinating server, and
2. selecting the hyperparameters of the solution in D4.6 by assessing the solution on the actual RE-SAMPLE data, to choose the model architecture, the subset of layers to encrypt, and the encryption schedule for the delayed encryption.

Due to the high costs associated with FHE, a major challenge will be making sure that the runtime performance of the implementation meets practical deployment thresholds. The hyperparameters selection will play a key role in this, namely in finding a sweet spot in the efficiency-privacy trade-off that achieves a requisite operational efficiency level while safeguarding patient privacy.

References

[1] C. Dwork, F. McSherry, K. Nissim and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Theory of Cryptography,* 2006.

[2] M. Nasr, R. Shokri and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in *Symposium on security and privacy (SP),* 2019.

[3] M. Rigaki and S. Garcia, “A survey of privacy attacks in machine learning,” *arXiv preprint arXiv:2007.07646,* 2020.

[4] R. Shokri, M. Stronati, C. Song and V. Shmatikov, “Membership inference attacks against machine learning models,” in *Symposium on security and privacy (SP),* 2017.

[5] S. Yeom, I. Giacomelli, M. Fredrikson and S. Jha, “Privacy risk in machine learning: Analyzing the connection to overfitting,” in *Computer security foundations symposium (CSF),* 2018.

[6] B. Hilprecht, M. Härterich and D. Bernau, “Monte Carlo and Reconstruction Membership Inference Attacks against Generative Models,” in *Proceedings on Privacy Enhancing Technologies,* 2019.

[7] L. Zhu, Z. Liu and S. Han, “Deep leakage from gradients,” *Advances in neural information processing systems,* vol. 32, 2019.

[8] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang and H. Qi, “Beyond inferring class representatives: User-level privacy leakage from federated learning,” in *IEEE INFOCOM 2019-IEEE conference on computer communications*, 2019.

[9] M. Fredrikson, S. Jha and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015.

[10] B. Hitaj, G. Ateniese and F. Perez-Cruz, “Deep models under the GAN: information leakage from collaborative deep learning,” in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017.

[11] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li and D. Song, “The secret revealer: Generative model-inversion attacks against deep neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020.

[12] K. Ganju, Q. Wang, W. Yang, C. A. Gunter and N. Borisov, “Property inference attacks on fully connected neural networks using permutation invariant representations,” in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security,* 2018.

[13] L. Melis, C. Song, E. De Cristofaro and V. Shmatikov, “Exploiting unintended feature leakage in collaborative learning,” in *Symposium on security and privacy (SP),* 2019.

[14] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proceedings of the SIGSAC conference on computer and communications security*, 2015.

[15] X. Zhu, C. Vondrick, D. Ramanan and C. C. Fowlkes, “Do We Need More Training Data or Better Models for Object Detection?.,” in *BMVC,* 2012.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial intelligence and statistics,* 2017.

[17] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta and R. Raskar, “Split learning for collaborative deep learning in healthcare,” *arXiv preprint arXiv:1912.12115,* 2019.

[18] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the SIGSAC conference on computer and communications security,* 2016.

[19] H. B. McMahan, D. Ramage, K. Talwar and L. Zhang, “Learning differentially private recurrent language models,” *arXiv preprint arXiv:1710.06963,* 2017.

[20] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa and J.-P. Hubaux, “Poseidon: Privacy-preserving federated neural network learning,” in *Network And Distributed System Security Symposium,* 2021.

[21] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.

[22] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proceedings of the SIGSAC conference on computer and communications security*, 2018.

[23] S. Wagh, D. Gupta and N. Chandran, "SecureNN: 3-Party Secure Computation for Neural Network Training.," *Proceedings on Privacy Enhancing Technologies*, 2019.

[24] F. Mireshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar and H. Esmaeilzadeh, "Privacy in deep learning: A survey," *arXiv preprint arXiv:2004.12254*, 2020.

[25] M. Burkhart, M. Strasser, D. Many and X. Dimitropoulos, "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics," in *USENIX Security Symposium*, 2010.

[26] E. Shi, H. T. H. Chan, E. Rieffel, R. Chow and D. Song, "Privacy-preserving aggregation of time-series data," in *Annual Network & Distributed System Security Symposium (NDSS)*, 2011.

[27] T. H. H. Chan, E. Shi and D. Song, "Privacy-preserving stream aggregation with fault tolerance," in *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers 16*, 2012.

[28] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[29] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proceedings of the SIGSAC Conference on Computer and Communications Security*, 2020.

[30] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Symposium on security and privacy (SP)*, 2017.

[31] M. Byali, H. Chaudhari, A. Patra and A. Suresh, "FLASH: Fast and robust framework for privacy-preserving machine learning," *Cryptology ePrint Archive*, 2019.

[32] H. Chaudhari, R. Rachuri and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," *arXiv preprint arXiv:1912.02631*, 2019.

[33] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat and J.-P. Hubaux, "Scalable Privacy-Preserving Distributed Learning," *Proceedings on Privacy Enhancing Technologies*, 2021.

[34] J. H. Cheon, A. Kim, M. Kim and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT: International Conference on the Theory and Applications of Cryptology and Information Security*, 2017.

[35] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, 2021.

[36] A. Kim, A. Papadimitriou and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," in *Cryptographers' Track at the RSA Conference*, 2022.

[37] A. Kim, Y. Song, M. Kim, K. Lee and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC medical genomics*, vol. 11, p. 23–31, 2018.

[38] K. Han, S. Hong, J. H. Cheon and D. Park, *Efficient Logistic Regression on Large Encrypted Data*, 2018.

[39] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff and V. Vaikuntanathan, "Optimized homomorphic encryption solution for secure genome-wide association studies," *BMC Med Genomics*, vol. 13, p. 83, July 2020.

[40] Y. G. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, p. 449–458, 1994.

[41] A. López-Alt, E. Tromer and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012.

[42] H. Chen, W. Dai, M. Kim and Y. Song, “Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2019.

[43] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan and D. Wichs, “Multiparty computation with low communication, computation and interaction via threshold FHE,” in *Advances in Cryptology–EUROCRYPT: 3Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012.

[44] J. Duchi, E. Hazan and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.,” *Journal of machine learning research*, vol. 12, 2011.

[45] T. Tieleman, G. Hinton and others, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, p. 26–31, 2012.

[46] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[47] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, R. V. Saraswathy, K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan and V. Zucca, “OpenFHE: Open-Source Fully Homomorphic Encryption Library,” in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, New York, NY, USA, 2022.

[48] G. Cohen, S. Afshar, J. Tapson and A. Van Schaik, “EMNIST: Extending MNIST to handwritten letters,” in *2017 international joint conference on neural networks (IJCNN)*, 2017.

[49] G. B. Huang, M. Mattar, T. Berg and E. Learned-Miller, “Labeled faces in the wild: A database forstudying face recognition in unconstrained environments,” in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[50] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik and others, “HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data,” *Proceedings on Privacy Enhancing Technologies*, vol. 1, p. 325–342, 2023.

[51] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, 2014.

[52] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.

[53] A. F. Agarap, “Training Deep Neural Networks for Image Classification in a Homogenous Distributed System,” 2019.

[54] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly and others, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[55] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso and others, “Privacy-preserving federated brain tumour segmentation,” in *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*, 2019.

[56] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy and W. Wei, “LDP-Fed: Federated learning with local differential privacy,” in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020.

[57] C. Dwork, A. Roth and others, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, p. 211–407, 2014.

[58] M. Hardt and G. N. Rothblum, “A multiplicative weights mechanism for privacy-preserving data analysis,” in *2010 IEEE 51st annual symposium on foundations of computer science*, 2010.

[59] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova and A. Smith, “What can we learn privately?,” *SIAM Journal on Computing*, vol. 40, p. 793–826, 2011.

[60] A. Beimel, H. Brenner, S. P. Kasiviswanathan and K. Nissim, “Bounds on the sample complexity for private learning and private data release,” *Machine learning,* vol. 94, p. 401–437, 2014.

[61] F. Tramer and D. Boneh, “Slalom: Fast, verifiable and private execution of neural networks in trusted hardware,” *arXiv preprint arXiv:1806.03287,* 2018.

[62] S. Song, K. Chaudhuri and A. D. Sarwate, “Stochastic gradient descent with differentially private updates,” in *2013 IEEE global conference on signal and information processing*, 2013.

[63] V. Shejwalkar and A. Houmansadr, “Membership privacy for machine learning models through knowledge transfer,” in *Proceedings of the AAAI conference on artificial intelligence*, 2021.

[64] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider and F. Koushanfar, “Chameleon: A hybrid secure computation framework for machine learning applications,” in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018.

[65] V. Pihur, A. Korolova, F. Liu, S. Sankuratripati, M. Yung, D. Huang and R. Zeng, “Differentially-private ``draw and discard" machine learning,” *arXiv preprint arXiv:1807.04369,* 2018.

[66] D. Pasquini, G. Ateniese and M. Bernaschi, “Unleashing the Tiger: Inference Attacks on Split Learning,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security,* New York, NY, USA, 2021.

[67] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow and K. Talwar, “Semi-supervised knowledge transfer for deep learning from private training data,” *arXiv preprint arXiv:1610.05755,* 2016.

[68] J. Liu, M. Juuti, Y. Lu and N. Asokan, “Oblivious neural network predictions via minionn transformations,” in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017.

[69] Z. Li and Y. Zhang, “Membership leakage in label-only exposures,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[70] B. Li, D. Micciancio, M. Schultz and J. Sorrell, “Securing Approximate Homomorphic Encryption Using Differential Privacy,” in *Advances in Cryptology – CRYPTO 2022*, Cham, 2022.

[71] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi and R. Sharma, “Cryptflow: Secure tensorflow inference,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[72] K. Kluczniak and G. Santato, *On Circuit Private, Multikey and Threshold Approximate Homomorphic Encryption,* 2023.

[73] C. Juvekar, V. Vaikuntanathan and A. Chandrakasan, “{GAZELLE}: A low latency framework for secure neural network inference,” in *USENIX Security Symposium*, 2018.

[74] M. Juuti, S. Szyller, S. Marchal and N. Asokan, “PRADA: protecting against DNN model stealing attacks,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[75] J. Jia, A. Salem, M. Backes, Y. Zhang and N. Z. Gong, “Memguard: Defending against black-box membership inference attacks via adversarial examples,” in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019.

[76] M. Hardt, B. Recht and Y. Singer, “Train Faster, Generalize Better: Stability of Stochastic Gradient Descent,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48,* New York, NY, USA, 2016.

[77] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in *International conference on machine learning*, 2016.

[78] J. Fan and F. Vercauteren, “Somewhat Practical Fully Homomorphic Encryption.,” *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[79] C. A. Choquette-Choo, F. Tramer, N. Carlini and N. Papernot, “Label-only membership inference attacks,” in *International conference on machine learning*, 2021.

[80] H. Chen, I. Chillotti and Y. Song, “Improved Bootstrapping for Approximate Homomorphic Encryption,” in *Advances in Cryptology – EUROCRYPT 2019*, Cham, 2019.

[81] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical GapSVP,” in *Annual Cryptology Conference,* 2012.

[82] Z. Brakerski, C. Gentry and V. Vaikuntanathan, “(Leveled) Fully Homomorphic Encryption without Bootstrapping,” *ACM Trans. Comput. Theory*, vol. 6, July 2014.