Split Without a Leak: Reducing Privacy Leakage in Split Learning[⋆]

Khoa Nguyen[1][0000−0001−5089−5250], Tanveer Khan[1][0000−0001−7296−2178][⋆⋆], and Antonis Michalas[1,2][0000−0002−0189−3520]

[1] Tampere University, Finland
[2] University of Westminster
{khoa.nguyen,tanveer.khan, antonios.michalas}@tuni.fi

Abstract. The popularity of Deep Learning (DL) makes the privacy of sensitive data more imperative than ever. As a result, various privacy-preserving techniques have been implemented to preserve user data privacy in DL. Among various privacy-preserving techniques, collaborative learning techniques, such as Split Learning (SL) have been utilized to accelerate the learning and prediction process. Initially, SL was considered a promising approach to data privacy. However, subsequent research has demonstrated that SL is susceptible to many types of attacks and, therefore, it cannot serve as a privacy-preserving technique. Meanwhile, countermeasures using a combination of SL and encryption have also been introduced to achieve privacy-preserving deep learning. In this work, we propose a hybrid approach using SL and Homomorphic Encryption (HE). The idea behind it is that the client encrypts the activation map (the output of the split layer between the client and the server) before sending it to the server. Hence, during both forward and backward propagation, the server cannot reconstruct the client's input data from the intermediate activation map. This improvement is important as it reduces privacy leakage compared to other SL-based works, where the server can gain valuable information about the client's input. In addition, on the MIT-BIH dataset, our proposed hybrid approach using SL and HE yields faster training time (about 6 times) and significantly reduced communication overhead (almost 160 times) compared to other HE-based approaches, thereby offering improved privacy protection for sensitive data in DL.

Keywords: Homomorphic Encryption · Machine Learning · Privacy-preserving Techniques · Split Learning

1 Introduction

Machine learning (ML) can boost productivity in various fields by leveraging data to make predictions and decisions without explicit human instructions.

[⋆] This work was funded by the Technology Innovation Institute (TII) for the project ARROWSMITH and from Horizon Europe for HARPOCRATES (101069535).

[⋆⋆] The first two authors contributed equally to this work.

However, the data should be of excellent quality in order to achieve good results after training ML models. Any ML algorithm only performs well when it is provided with a large amount of high-quality training data [10]. In a nutshell, one can say that data drives ML. Organizations that create and gather data can build and train their ML models. This allows them to make use of their models and provide ML as a service (MLaaS) [38] to other organizations. This is beneficial for organizations that are unable to build their own models but wish to use this method for data predictions. However, a cloud-hosted model poses numerous privacy issues [20,37,44]. In order to use it, external organizations must either upload their input data or download the model. Regarding privacy, uploading input data can be problematic, while downloading the model might not be a suitable option due to intellectual property.

A possible solution is to use privacy-preserving techniques [6] – to encrypt the model and the data in a way that allows one organization to use a model held by another organization without either party disclosing their intellectual property. Homomorphic Encryption (HE) [2] and Secure Multi-Party Computation (SMPC) [30] are the best-known techniques enabling computations over encrypted data, hence preserve input privacy. Both of these techniques seem like a silver bullet solution because they enable computation on encrypted data without revealing the underlying information. However, some issues lie behind the surface that can make their implementation more difficult, such as communication cost in SMPC and computation complexity in HE [9].

In addition to HE and SMPC, collaborative learning techniques like Federated Learning (FL) [5] and Split Learning (SL) [18] can also be used to preserve user data privacy. Both FL and SL are similar in that they both allow for the training of ML models on large distributed datasets, *without* the need to centralize the data. However, they differ in how the models are split and trained, and in the types of applications they are used for. FL is used to train a ML model on a large number of devices, each using local data. In FL, each device trains a local copy of the model, and the models are then combined on a central server. Through this approach the model is trained on a large amount of non-centralized data. This can be useful in the case of sensitive data or devices not connected to the internet. SL is a distributed deep learning technique that splits a neural network into two parts, first half directed to the client(s) and second to the server. Both the client(s) and the server collaboratively train the split model, though they *cannot* access each other's parts, SL provides multiple benefits such as: *(i)* joint neural network training by multiple parties, such as a client and a server, in which each party is protected from disclosing their parts of the model, *(ii)* ML model training by users. In this case, users do not share their raw data with a server running part of a Deep Neural Network Model (DNN), hence preserving their privacy, *(iii)* computational burden reduction for the client, as SL does not run the entire model (i.e. it utilizes a smaller number of layers), and *(iv)* model classification accuracy that is comparable to non-split model. Also SL offers several significant advantages over FL, such as [41,36,45] *(i)* *not* full disclosure of the model's architecture and weights *(ii)* peak performance through minimal resource consumption *(iii)* required bandwidth reduction *(iv)* reduced computational burden on the data owners' devices.

By definition, SL provides an additional layer of privacy protection, however, Abudhabba *et al.* [1] showed that adding SL to one-dimensional (1D) Convolutional Neural Network (CNN) models for time-series data like ECG signals could result in significant privacy leakage. To reduce privacy leakage, the authors used two more mitigation techniques, namely differential privacy [3] and additional layers. However, based on the findings, none of these techniques can significantly reduce privacy leakage from the channels of the SL activation. Additionally, the accuracy of the joint model is greatly decreased by both methods.

While there is a plethora of Privacy-preserving ML (PPML) works utilizing HE to protect users' input, to the best of our knowledge, until now there is only one work [23] that combines HE with SL. In this work, we concentrate on collaborative client-server training of an ML model in a privacy-preserving manner by constructing a model that uses HE to reduce privacy leakage in SL.

1.1 Contributions

C1. In [23], the authors proposed an SL protocol with HE. However, by analyzing the proposed protocol, we found that the server can gain some valuable information on the client's input data by exploiting the gradients sent from the client in the backward pass of the protocol. This, results in important privacy leakage. In this paper, we propose an improved protocol to face this privacy leakage.

C2. We took the U-shaped split 1D CNN proposed in [23], tested this model with plaintext activation maps, and an encrypted version using HE to allow the server to perform computations on encrypted activation maps. Even though the 1D CNN model is identical to that of [23], our proposed protocol optimizes the amount of information being transferred between the client and the server, through that not only solve the privacy leakage in [23], but also improve the communication cost of the whole protocol.

C3. We have designed a new training protocol permitting the extensive use of CKKS HE scheme's packing feature [13]. Combining packing and optimization techniques in C2, our training becomes much faster (almost 6 times) compared to [23] and also requires much less overhead communication (about 160 times) on the MIT-BIH dataset.

1.2 Organization

The rest of the paper is organized as follows: In section 2, we present important published works in the area of SL. In section 3, we provide the necessary background information about 1D CNN, HE and SL. The architectures of the local and split models on plaintext data are presented in section 4. In section 5, we report the privacy leakage we noted in the paper [23] while section 6 presents the design and implementation of the split 1D CNN training protocol on encrypted data that solves this privacy leakage. Then, section 7 refers extensively to experimental results and finally the paper is concluded in section 8.

2 Related Works

SL has been proposed as a privacy-preserving implementation of collaborative learning [47,46,36] and has gained particular interest due to its efficiency and simplicity. For user data privacy, SL relies on the fact that only intermediate activation maps are shared between the parties. It can enable more privacy-preservation and efficient use of deep learning models in a variety of settings [18]. This is why SL-based solutions have been implemented and adopted in commercial as well as open-source applications. For example, SL could enable medical image analysis on a patient's device, while securing their privacy and sensitive data [47]. In addition, SL can efficiently process data generated by IoT devices, by keeping the computationally intensive part of a model on a remote server [26,49,29,40].

Although the previous studies [18,47,36] assumed that SL is designed to protect the intellectual property of the shared model and to reduce the risk of inference attacks[3] perpetrated by a malicious server, recent studies have shown that these assumptions are false and privacy leakage risks do exist in SL. In [46], the authors analyzed the privacy leakage of SL and found a considerable leakage from the split layer in the 2D CNN model. Also, the authors in paper [1] performed experiments on the 1D CNN model and found that sharing the activation from the split layer results in severe privacy leakage. Also, the paper described various structural vulnerabilities of SL and showed how to exploit them and violate the protocol's privacy-preserving property. This makes SL a prime target for various types of attacks such as model inversion attack [19,1,43] and property inference attack [34].

Various SL privacy threats and attacks have been presented. In addition, countermeasures have also been introduced to achieve privacy-preserving deep learning. For example, Vepakomma et al. [46] proposed a method for limiting data reconstruction in split neural networks (SplitNN) by minimizing the distance correlation between the input data and the intermediate tensors during model training. Also, to protect the SL from property inference attack, one approach is to use secure aggregation protocol that can protect the privacy of the clients' data while still allowing for collaborative learning [34]. Another approach is to use differential privacy techniques to add noise, which can help prevent attackers from inferring sensitive information from the model updates [11]. Abuadbba *et al.* [1] applies noise to the intermediate tensors in a SplitNN as a defence mechanism against model inversion attacks on one-dimensional ECG data. A similar method, Shredder [31], was introduced by Mireshghallah *et al.*. It adaptively generates a noise mask to minimize mutual information between input and intermediate data. Also, the authors in [43], proposed a simple additive noise method to defend against model inversion. This revealed that the method can significantly reduce attack efficacy. In addition to the above-mentioned techniques, hybrid approaches have also been combined with FL to yield a more scal-

[3] A type of attack where an attacker can infer only specific properties of the private training instances, rather than reconstructing the entire input.

able privacy-preserving training protocol, such as [42]. Pereteanu *et al.,* propose splitting the server network into private sections separated by a public section that can be assessed in plaintext by client to speed up classification while using HE [35]. While their approach are somewhat similar to our work, it is limited to client input classification and does not permit a client to customize a network to their private dataset. Recently, the authors of [23] proposed an approach that combines SL and HE that can be considered as the closest work to ours. In their model, prior to sending the intermediate input to the server, the client first encrypts it and sends the encrypted activation map to the server. However, one of the shortcomings of this hybrid approach, as we show in section 5, is that during the backward propagation, by exploiting the gradients sent from the client, the server can gain valuable information on the client's input data, thus causing privacy leakage. We propose an improved protocol to address this privacy leakage [23]. Our protocol is faster and incurs less communication overhead.

3 Preliminaries

3.1 Convolutional Neural Networks

CNNs are a class of neural network that uses the convolution kernels to slide along the input signal and produce output feature maps. CNNs are used to process grid-like topology data, for example, they were used to recognize handwritten zip codes in 2D images [27]. CNNs can also process 1D [25] or 3D signals [21]. In this work, we employ a 1D CNN on ECG data to classify heartbeats.

If we have a kernel $\mathbf{w}$ of size m, and at every step we stride the kernel n steps, then the 1D convolution layer performs the operation: $\mathbf{z}_i = \sum_{j=0}^{m-1} \mathbf{w}_j \cdot \mathbf{x}_{i\times n+j}$ Apart from the Conv1D layers, our 1D CNN also consists of other layers:

– Leaky ReLU (LReLU): Leaky Rectify Linear Unit is a non-linear function that can be described as: $f(x) = \begin{cases} x, & \text{if } \ x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases}$
where α is called the "negative slope" – a small number such as 0.01.
– Max Pooling: An operation that calculates maximum value in each patch of a feature map. The result is a down-sampled feature map that contains the maximum values that highlight important features of those patches.
– Fully Connected (FC) or Linear Layer: An FC layer performs a weighted affine transformation to its input: $\mathbf{y} = \mathbf{x}W^T + \mathbf{b}$.
– Softmax: The Softmax function is used to transform a vector $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ into a vector of probabilities: $\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$

By stacking these layers on top of each other, we construct our 1D CNN as a feature detector and classifier for the ECG datasets. The 1D CNN model can be written as a function f_Θ, where Θ is a set of adjustable (or learnable) parameters. Θ is first initialized to small random values in the range $[-1, 1]$. The training process consists of two phases named forward propagation and backward propagation. More specifically, during forward propagation, an input heartbeat x is propagated forward through the Conv1D layer, pooling layer and fully connected layer to obtain the final output value $\hat{y}$. As for each x, there is a corresponding encoded label vector y that represents its ground truth class. We aim to find the best set of parameters Θ that maps x to a predicted output vector $\hat{y}$, where $\hat{y}$ is as closed as possible to y which is measured by a loss (or distance) function $\mathcal{L}(\hat{y}, y)$. $\mathcal{L}$ is chosen to be cross entropy loss in our work. To reach the minimized loss function, we update Θ using backward propagation. Backward propagation moves from the network's output layer back to the input layer to calculate the gradients of the loss function $\mathcal{L}$ w.r.t the weights Θ. Then, these weights are updated according to the gradients. The full process of calculating the predicted output, the loss function and the gradients, followed by updating the weights on one batch of data is called an "iteration". In this paper, we train the neural network with thousands of samples of x and associated y. We train the neural network using mini-batch gradient descent with the batch size specified by n. The total number of training batches is $N = \frac{|D|}{n}$, where $|D|$ is the size of the dataset. N is equal to the number of training iterations. Once the neural network goes through all the training batches, it has completed one training epoch. This process is repeated for a total of E epochs.

3.2 Split Learning

A distributed deep learning method that allows collaboration between different parties to train a model without sharing raw data [47]. There are three main configurations of SL, namely vanilla SL, U-shaped SL, and SL for vertically partitioned data [47]. These SL configurations are visually demonstrated in Figure 1.

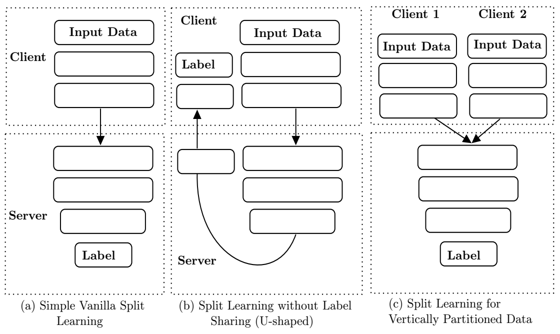
Client
Input Data
Server
Label
(a) Simple Vanilla Split Learning
Client
Input Data
Label
Server
(b) Split Learning without Label Sharing (U-shaped)
Client 1
Client 2
Input Data
Input Data
Label
(c) Split Learning for Vertically Partitioned Data

Fig. 1: Different Split Learning Configurations

We employ the U-shaped SL configuration, where we train the split 1D CNN between the client and the server with the client sharing neither his input data nor the labels. We can see from Figure 1(b) that in the forward pass, the client first trains his part of the model until the split layer, then sends the activation maps to the server that continues training his part of the model. After finishing its part, the server sends its outputs to the client. The client computes the outputs of the network and compares them with the labels to calculate the difference between the predicted outputs and ground-truth labels. In the backward pass, the information flow moves in the opposite direction to the forward pass.

3.3 Homomorphic Encryption

With the help of HE, computations on encrypted data can be performed without prior decryption. Homophormism is based on the idea that when elements are transformed from one set to another, the relationships between them are preserved. In encryption terms, this means whether operating on encrypted data or plaintext data will yield equivalent results. Let's take a homomorphic addition operation as an example. Given two plaintext messages x_1, x_2 and their corresponding encrypted messages c_1, c_2. An operation $\oplus$ is an HE addition, if $c_3 = c_1 \oplus c_2$ is the encryption of $x_3 = x_1 + x_2$.

The HE journey started in 1978 with the paper [39], in which the authors envisioned homomorphism as a construction for secure computation and a resort for user data privacy. From then on, we have witnessed the creation of several homomorphic cryptosystems. Cryptosystems that only enable homomorphic addition or multiplication on ciphertexts (but not both) are called *partial HE*. Examples of partial HE systems are the ElGamal (1985) [16] or Paillier cryptosystem (1999) [33]. *Fully HE* (FHE) cryptosystems that allow both addition and multiplication on encrypted data, were thought of as impractical until Gentry's first FHE scheme in 2009 [17]. Gentry introduced the concept of bootstrapping, an operation performed to reset the noise level in ciphertexts after homomorphic operations. Following Gentry's work, new HE schemes were developed. There are two main approaches to new HE schemes, namely fast bootstrapping and leveled. Schemes that belong to the *fast bootstrapping FHE* branch aim to maximally reduce the computing overhead induced by the bootstrapping operation. Examples of fast bootstrapping schemes are FHEW [15], TFHE [14]. *Leveled HE* schemes assign levels to different noise sizes and only operate within these noise levels. In this way, they do not have to resort to the computationally expensive bootstrapping operation. However, when the noise in a ciphertext exceeds the maximal level, they will provide incorrect decryptions. Examples of leveled FHE schemes are BGV [8], BFV [7] and CKKS [13]. Leveled HE schemes can also be extended to FHE, for example, in [12], the authors proposed a bootstrapping procedure to make CKKS an FHE scheme.

In this work, we employ the CKKS scheme. Using CKKS, we can perform computations on vectors of complex values and real values. Suppose we have a plaintext message $\mathbf{z} \in \mathbb{C}^{N/2}$ that we want to encrypt, where N is called polynomial degree modulus and is a power of 2. Before encrypting, $\mathbf{z}$ is encoded into a polynomial $\mathbf{m} \in \mathbb{Z}[X]/\left(X^N + 1\right)$. Compared to standard computations on vectors, polynomials provide a better trade-off between security and efficiency. During encoding, $\mathbf{z}$ is also multiplied by a scaling factor Δ to keep a level of precision. The encoded message $\mathbf{m}$ is then encrypted into $\mathbf{c} \in \left(\mathbb{Z}_q[X]/\left(X^N + 1\right)\right)^2$. We can apply addition or multiplication on the ciphertext $\mathbf{c}$ to produce $\mathbf{c}' = f(\mathbf{c})$. In order to get the decrypted message, we need to follow the inverse procedure. First, the ciphertext $\mathbf{c}'$ is decrypted into an encoded message $\mathbf{m}' \in \mathbb{Z}[X]/\left(X^N + 1\right)$. Finally, $\mathbf{m}'$ is decoded to produce the transformed plaintext $\mathbf{z}' = f(\mathbf{z})$. In summary, the most important parameters of the CKKS schemes are:

- **Polynomial Modulus (Degree of the polynomial)** N**:** This parameter directly affects the number of coefficients in plaintext polynomial and the size of ciphertext elements. The value N must be a power of 2, such as 1024, 2048, etc[4]. The bigger N is, the more secure the scheme. However, the computational overhead incurred is also higher.
- **Coefficient Modulus** $\mathcal{C}$**:** This is a list of prime numbers that define scheme noise levels. After each multiplication, a different prime in $\mathcal{C}$ is used as the coefficient modulus. After all primes in the list are used, we can no longer perform multiplications, since each homomorphic multiplication on ciphertexts increases the noise level by one.
- **Scaling Factor** Δ: During the encoding process, the plaintext message is multiplied by Δ to maintain a certain level of precision. Δ is a positive integer and is often chosen to be a power of 2.

4 Network Architecture

In this section, we describe the architecture of our 1D CNN model. The model has two forms: a local model and a split model. A local model is actually a non-split model in which all the layers of the network are executed by one party. In the split model, the network is split between two parties: the client and the server. The first few layers of the network are executed on the client side, while the remaining layers are executed on the server side. We discuss both models in more detail in subsection 4.1 and subsection 4.2. Additionally, we discuss the threat model considered as well as the parties involved in the split model training process, concentrating on their roles and the parameters allocated to them throughout the training process.

4.1 Local Model

In this section, we describe the non-split version of the 1D CNN model to classify the ECG signals. The model was selected from paper [28]. As can be seen in Figure 2, the non-split version of 1D CNN model contained an input layer, two Conv1D layers, two pooling layers, a fully connected layer and an output layer. The input of each layer is connected to the output of the previous layer.

As shown in Figure 3, the 1D CNN model has two parts. The first one is feature extraction, which consists of two types of layers: Conv1D and pooling layer. This latter automatically learns the features from the raw input data. In the Conv1D layer, the 1D CNN model carries out the convolution operation on

[4] `https://bit.ly/3ABOEfQ`

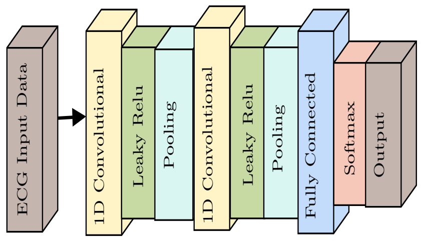

Fig. 2: Local Model

the input to generate the corresponding one-dimensional feature maps. Different convolution kernels are used to extract varying features from the input signal as convolution kernels of size 7 are used for the first Conv1D layer, while kernels of size 5 are used for the second Conv1D layer. Consequently, the Conv1D serves as a filter that learns the necessary characteristics of the original signal. The outputs of the Conv1D layers are called feature maps, or activation maps. Following, each feature map is now processed using the max pooling operation. The pooling layer reduces the dimension of the feature maps and prevents the network from over-fitting. The second part of the neural network is classification. Through it, signals are classified accurately by utilizing the extracted features. It consists of a fully connected layer and a softmax layer. The fully connected layer performs a weighted sum of the inputs and adds bias. It integrates and normalizes highly abstracted features used as the input of a softmax classifier to classify the segments into different types. More detailed descriptions of the hyper-parameters for each layer are given in Figure 3.

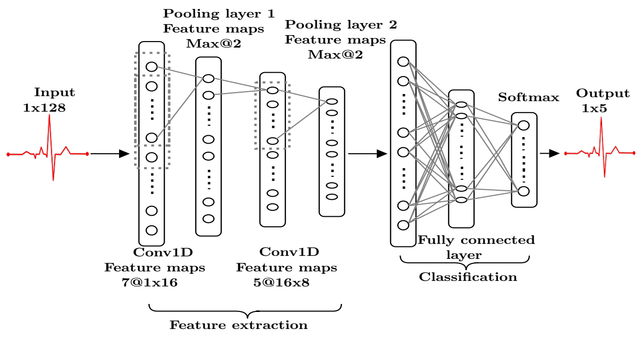

Fig. 3: Architecture and parameters of the CNN model used for ECG classification

4.2 Split Model

In this section, we discuss the split version of the above 1D CNN model. We split the 1D CNN model into multiple parts, each of which is processed and computed by a separate party. For our setting, we consider two parties, a client and a server, with one part of the model executed on the client and another part of the model executed on the server. Both parties participate in training the 1D CNN model. As can be seen in Figure 4, activation maps and gradients are passed between client and server to collaboratively train the joint model, but neither can access the other's data. In addition, we used the U-shaped SL, where the first few layers $(1 \dots l)$ and the last layer L are executed on the client side, while the remaining layers $(L-1)$ are executed on the server side. As a result, only the client has access to the model's output, while the server does not know the 1D CNN model's output. We will not go into the details of protocol training with the plaintext activation map as the authors in [23] have shown that SL can be applied into 1D CNN without the model classification accuracy degradation. Note that with the current split model architecture, we only have one fully connected layer on the server side, which goes against the advantage of SL where the client can outsource a big part of the neural network to the server to reduce computation cost on the client side. The reason for this is that in our encrypted training protocol, we will encrypt the activation maps with HE before sending them to the server (more details in section 6). With the current limitations of HE, e.g. big computation complexity and difficulties in non-linear calculations..., in this work, we limit to have only one linear layer on the server side. Adding more layers on the server side will be done in future works.

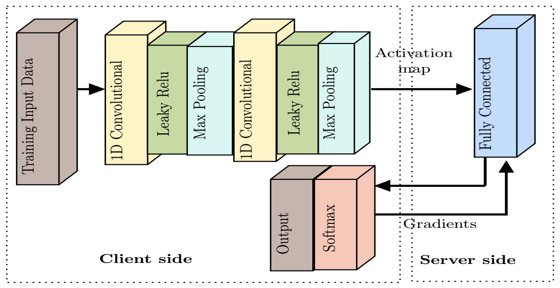

Fig. 4: U-shaped Split Model

4.3 Threat Model

For this setting, we consider a semi-honest threat model in which both parties follow the protocol while trying to gather as much information as possible from the received messages. As mentioned earlier, in the SL context, we have two parties: a client and a server, each of which plays a specific role and has access to specified parameters. Both parties do not access each other's devices and cannot target attacks on each other. More specifically, the server performs all its operations as defined, does not collude with the client but is curious about the raw data stored with the client. The server's goal is to reconstruct the raw data from the activation maps of the split layer delivered from the client during the forward propagation and also the gradients delivered by the client during backward propagation. Furthermore, we assume that the clients are trustworthy and that they will participate in the learning process as long as the raw data remains in their possession, however, the client also tries to learn the server's neural network weights and biases. In short, both client and server follow the protocol but may try to extract maximum information from the exchanged messages.

5 Privacy Leakage Analysis

In this section, we report the privacy leakage we noted in the backward propagation phase, while training the U-shaped split learning model on HE encrypted data proposed in [23]. Here, we also present possible solutions to mitigate this privacy leakage and in section 6 we discuss in detail how we used homomorphic encryption to address this privacy leakage. In addition, we provide an evaluation of the privacy leakage of the Algorithm 3 in [23].

In SL, the training is performed through a vertically distributed back-propagation that requires clients to exclusively share the intermediate network's output; rather than the raw, private training instances [34]. In [23], before the training phase, the client decides on the model's architecture and hyper-parameters and sends the required information to the server. Hence, the authors assumed the server has no information on the client side architecture and its weights. This is one of the important security features of SL: it conceals information about the model's architecture and hyper-parameters. Furthermore, because the client encrypts the activation map before transmitting it to the server during forward propagation, the next assumption is that the server is unaware of the activation maps as they are encrypted, as long as the HE algorithm is secure. However, during backward propagation in the Algorithm 3 of [23], the client sends both the gradients of error J w.r.t a^L ($\frac{\partial J}{\partial \mathbf{a}^{(L)}}$) and w^L ($\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$) to the server, allowing the server to determine the activation map. Following, we present detailed information on the privacy breach we detected during backpropagation.

In the backward pass in Algorithm 3 from [23], the client calculates $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ and sends them in plaintext to server to continue backward propagation, where $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ is calculated by the client using the chain rule as follows:

$$\frac{\partial J}{\partial \boldsymbol{w}^{(L)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \boldsymbol{w}^{(L)}}. \tag{1}$$

Furthermore, since on the server side there is only one linear layer, we have:

$$\frac{\partial \mathbf{a}^{(L)}}{\partial \boldsymbol{w}^{(L)}} = \mathbf{a}^{(l)}. \tag{2}$$

Equation (1) now becomes:

$$\frac{\partial J}{\partial \boldsymbol{w}^{(L)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \cdot \mathbf{a}^{(l)}. \tag{3}$$

Therefore, if the client sends both $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ and $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ to the server in plaintext, the server can then find out the plaintext activation maps by multiplying both sides of equation (3) with $\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)^{-1}$ to figure out $\mathbf{a}^{(l)}$ as:

$$\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)^{-1} \cdot \frac{\partial J}{\partial \boldsymbol{w}^{(L)}} = \left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)^{-1} \cdot \left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right) \cdot \mathbf{a}^{(l)} = \mathbf{a}^{(l)}. \tag{4}$$

This privacy leakage will happen if the server is able to find the inverse matrix of $\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)^{-1}$, which is highly possible as the server can manipulate the training batch size to make $\mathbf{a}^{(L)}$ and $\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)$ square matrices. In the next section, we provide a more detailed evaluation of this privacy leakage analysis. To resolve this privacy leakage, we propose a new training protocol in section 6.

5.1 Privacy Leakage Evaluation

Given that the server instructs the client to train with the proper batch size to make $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ a square matrix, after the server receives the gradients that include $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ from the client, the server tries to reconstruct $\mathbf{a}^{(l)}$ like in Equation 4. We evaluate this privacy leakage by building the attack on top of the code provided by the authors of [23].

First, the server instructs the client to train with the batch size of 5 (since we have 5 output classes), making $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ a square matrix. Then, using Equation 4, the server is able to reconstruct an activation map plotted in Figure 5 (left), which looks very similar compared to the client's plaintext activation map Figure 5 (right). Furthermore, when the server plots the reconstructed activation

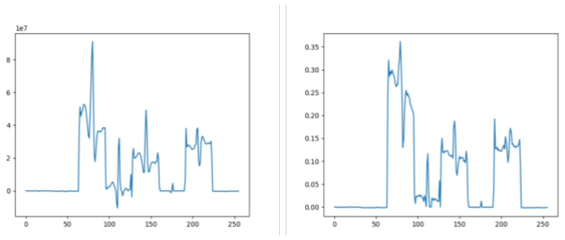

Fig. 5: Server's reconstructed (Left) and Client's plaintext (Right) activation maps

map in chunks of 32 data points as in Figure 6, it can reveal some very similar patterns compared to the client's original data plotted in Figure 7. Observing two figures, we can see that the 4th, 5th and 7th chunk look very similar to the plaintext data, which shows the effectiveness of the privacy leakage we presented in section 5. Furthermore, note that in [23], the neural network can only produce about 88% accuracy on the test dataset. Once the neural network gets to higher accuracy (e.g. 99%), the privacy leakage can become even more evident.

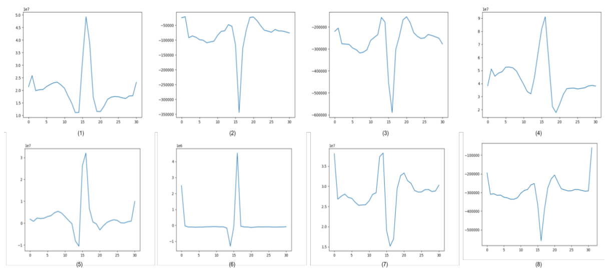
Fig. 6: The reconstructed activation map in chunks.

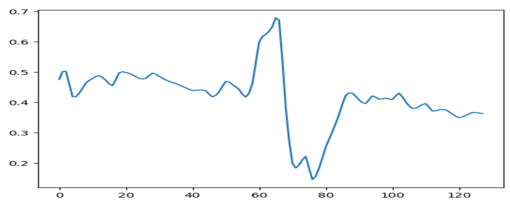
Fig. 7: The client's plaintext data.

6 Split Model Training Protocol using Encrypted Activation Map

In this section, we discuss in detail the U-shaped split model with the encrypted activation map. We refer to this protocol as HESplit network. Before delving into the details, consider a model with L layers in total, with the L^{th} layer serving as the model's output layer. Assume the model is divided into layers l and $l+1$. As discussed earlier, most of the layers of the 1D CNN model are executed on the client side, while only one layer is executed on the server side. Hence, the client owns the first l layers from layer 1 to layer $l+1$, as well as the final output layer, whilst the server owns only one linear layer. The main reason for having only one linear layer on the server side is due to computational constraints when training on HE encrypted data. Furthermore, more layers on the server's side require us to insert non-linear operations, which have shown difficulties for encrypted computation. In particular, for the CKKS scheme, we need to resort to approximation [22] which may lead to further degradation in accuracy. Our future works will focus on additional layers on the server side for encrypted split training. Below, we describe the U-shaped SL training protocol with the encrypted activation map.

Socket initialization and context generation In the initialization phase, the client first connects to the server through the socket and receives a number of training parameters to synchronize (see Figure 8). The parameters that are synchronized between the client and server are E, η, n, N. E is the number of training epochs, n is the batch size, N is the number of batches to be trained and, η represents the learning rate. These parameters should be synchronized on both sides in order to train the 1D CNN model in the same way (see algorithm 1, algorithm 2). In addition, both the client and server use the random weight initializer ϕ to initialize the weights w_i and biases b_i of their respective layers. The variables $\mathbf{a}^{(i)}$ (activation map), $\mathbf{z}^{(i)}$ (output of a tensor vector), and the gradients are also set to zero at the start of the phase. During this phase, the context is also generated, which holds the public key pk and secret key sk of the HE scheme as well as other parameters like polynomial modulus $\mathcal{P}$, coefficient modulus $\mathcal{C}$ and scaling factor Δ.

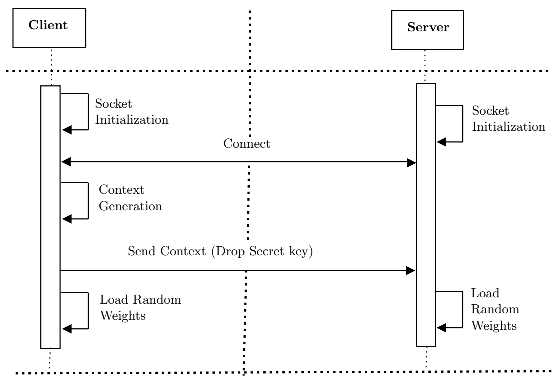

Fig. 8: Socket Initialization

Forward propagation The forward propagation starts from the client side. As can be seen in Figure 9, the client and server initialize their part of the model. The client first generates the batch (x, y), which has n data examples extracted from the train set D. x represents the input data, and y shows their labels. During the forward propagation phase, the client forward propagates the input x (see Equation 5) until layer l and generates an activation map (a^i).

$$\begin{aligned} z^i &\leftarrow f^{(i)}(\mathbf{a}^{(i-1)}) \\ a^i &\leftarrow g^{(i)}(\mathbf{z}^{(i)}) \end{aligned} \tag{5}$$

where f^i and g^i are the convolution operation of layer i and activation function of layer i respectively.

The client encrypts the activation map and sends the encrypted activation map $(\overline{\mathbf{a}^{(l)}})$ to the server (see algorithm 1). The server performs computation on the encrypted activation map and sends the output to the client $(\overline{\mathbf{a}^{(L)}})$. Finally, the client executes the final softmax layer of the 1D CNN model and calculates the loss function (J) between the original y and the predicted output $\hat{y}$. The loss function produces error J by computing the cross entropy loss.

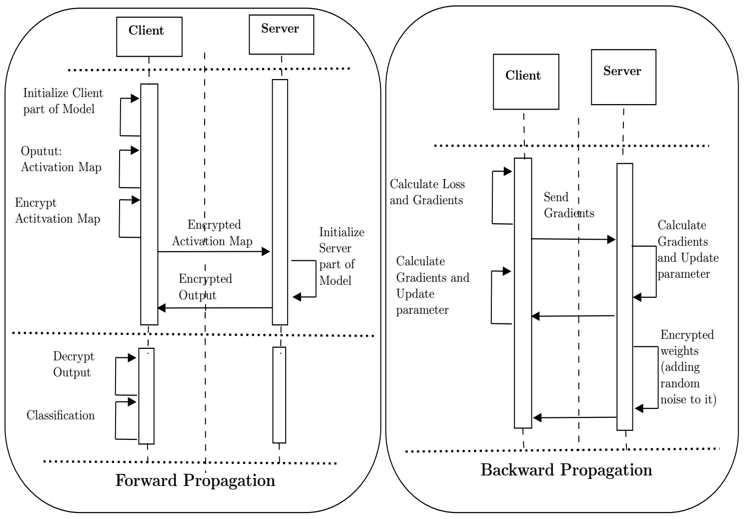
Client
Server
Initialize Client part of Model
Oputut: Activation Map
Encrypt Actitvation Map
Encrypted Activation Map
Initialize Server part of Model
Encrypted Output
Decrypt Output
Classification
Forward Propagation
Client
Server
Calculate Loss and Gradients
Send Gradients
Calculate Gradients and Update parameter
Calculate Gradients and Update parameter
Encrypted weights (adding random noise to it)
Backward Propagation

Fig. 9: Forward and Backward Propagation

Backward propagation As can be seen in Figure 9, the backpropagation starts from the client side. The client calculates the loss function, the gradients of the loss function w.r.t the activation map and send these gradients to the server. On receiving the gradients from the client, the server calculates the gradients of the loss function w.r.t its linear layer and update the parameter of this layer. More specifically, as shown in algorithm 1 and algorithm 2, the client calculates and sends the gradients of error w.r.t $a^{(L)}$ to the server. The server continues the backpropagation, calculates $\frac{dJ}{db}$, $\overline{\frac{dJ}{dw'}}$ and $\frac{dJ}{da}$. The server then sends $\frac{dJ}{da}$ to the client. After receiving the gradients, the backpropagation continues to the first hidden layer on the client side. Note that as $\frac{dJ}{dw'}$ becomes encrypted, the server's weights will become encrypted after updating the parameters as in Equation 6. After updating this parameters for several forward and backward pass, the noise level in w can cause overflow, rendering the computation in correct. To solve this issue, the server also sends the encrypted weights to the client after adding some fixed noise to it to preserve the weight's privacy. Upon reception, the client decrypts the weights to reset the HE noise level and sends the weights back to the server. After receiving the decrypted weights, the server subtract the fixed noise added earlier and continues to the next forward propagation pass.

$$\begin{aligned}\overline{(\boldsymbol{w}'^{(L)})} &= HE.Enc(\boldsymbol{w}'^{(L)})\\ \overline{(\boldsymbol{w}'^{(L)})} &= \overline{(\boldsymbol{w}'^{(L)})} - \overline{\frac{\partial J}{\partial \boldsymbol{w}'^{(L)}}}\end{aligned} \tag{6}$$

The difference between our new proposed protocol and the protocol in algorithms 3 and 4 of [23] is that instead of sending both $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ to the server, in our protocol's backward pass, the client only sends $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ to the server, hence resolving the privacy leakage reported in section 5.

7 Performance Analysis

In this section, we describe the experimental setups, datasets, and results from training our framework.

7.1 Evaluation

Experimental Setup: To process the data and train the neural networks, we use a machine with Intel Core i7-8700 CPU processor, 32 Gb of RAM, GeForce GTX 1070 Ti GPU with 8 Gb of GPU memory and running Ubuntu 20.04 LTS. The algorithms are implemented using Python 3.9.7, the PyTorch framework version 1.8.1+cu102 for constructing the neural network, the TenSeal framework [4] version 0.3.10 for HE and are evaluated on the local network.
Datasets We evaluate the performance of our framework on two Electrocardiography (ECG) datasets: the MIT-BIH dataset [32] and the PTB-XL dataset [48].

MIT-BIH This dataset contains 48 half-hour excerpts of two-channel ECG recordings, obtained from 47 subjects from 1975 to 1979. In our work, we use the processed version of the dataset from [1], which contains 26,490 heartbeat samples. Each sample belongs to one out of five classes: Normal (N), Left bundle branch block (L), Right bundle branch block (R), Atrial premature contraction (A), Ventricular premature contraction (V). Figure 10 shows some example signals from the processed MIT-BIH datset. To train the NNs, the dataset is split into a 50-50 ratio. A train or test split contains 13,245 examples, each example is a time series signal of length 128.

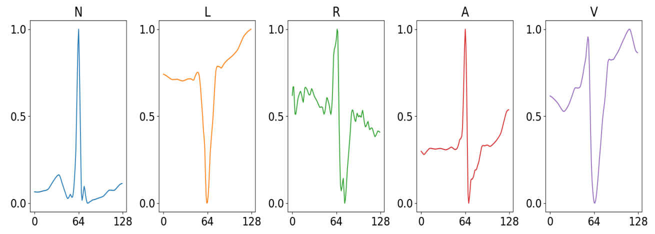

Fig. 10: Example heartbeats from the MIT-BIH dataset.

PTB-XL It is a large ECG dataset that contains 21837 clinical ECGs from 18885 patients, collected between 1989 and 1996. Each ECG waveform has 12 channels (leads) and is 10 seconds long. The raw waveforms are sampled at two different sampling rates (100 Hz and 500 Hz) and annotated by up to two cardiologists, who may assign multiple statements to one ECG signal. An ECG signal can be diagnosed to belong to one out of five superclasses: Normal ECG (NORM), Myocardial Infarction (MI), ST/T Change (STTC), Conduction Disturbance (CD), and Hypertrophy (HYP). Figure 11 shows an example of an ECG signal that belongs to the CD class; we can see that there are a total of 12 time series signals in the figure, as we are dealing with 12-lead ECG signals. The dataset is processed according to [23], where a sampling rate of 100 Hz is used. Following the example of [23], if an ECG signal belongs to more than one classes, we only choose one (the first class). The whole dataset is then split into train-test splits at a ratio of 90%-10%. More specifically, the training split contains 19,267 data samples and the test split contains 2,163 data samples.

Algorithm 1: Client Side

```
Initialization:
$s \leftarrow$ socket initialization;
$s.connect$
$\eta, n, N, E \leftarrow s.synchronize()$
$\{\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow \Phi$
$\{\mathbf{z}^{(i)}\}_{\forall i \in \{0..l\}}, \{\mathbf{a}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow \emptyset$
$\left\{\frac{\partial J}{\partial \mathbf{z}^i}\right\}_{i \in \{0..l\}}, \left\{\frac{\partial J}{\partial \mathbf{a}^i}\right\}_{i \in \{0..l\}} \leftarrow \emptyset$
Context Initialization:
$\mathsf{ctx}_{\mathsf{pri}}, \leftarrow \mathcal{P}, \mathcal{C}, \Delta, \mathsf{pk}, \mathsf{sk}$
$\mathsf{ctx}_{\mathsf{pub}}, \leftarrow \mathcal{P}, \mathcal{C}, \Delta, \mathsf{pk}$
$s.send(\mathsf{ctx}_{\mathsf{pub}})$
for $e$ in $E$ do
    for each batch $(\mathbf{x}, \mathbf{y})$ from $\mathbf{D}$ do
        Forward propagation :
        $O.zero\_grad()$
        $\mathbf{a}^0 \leftarrow \mathbf{x}$
        for $i \leftarrow 1$ to $l$ do
            $\mathbf{z}^{(i)} \leftarrow f^{(i)}\left(\mathbf{a}^{(i-1)}\right)$
            $\mathbf{a}^i \leftarrow g^{(i)}\left(\mathbf{z}^{(i)}\right)$
        end
        $\overline{\mathbf{a}^{(l)}} \leftarrow \mathsf{HE.Enc}\left(\mathsf{pk}, \mathbf{a}^{(l)}\right)$
        $s.send\ \overline{(\mathbf{a}^{(l)})}$
        $s.receive\ \overline{(\mathbf{a}^{(L)})}$
        $\mathbf{a}^{(L)} \leftarrow \mathsf{HE.Dec}\left(\mathsf{sk}, \overline{\mathbf{a}^{(L)}}\right)$
        $\hat{\mathbf{y}} \leftarrow Softmax\left(\mathbf{a}^{(L)}\right)$
        $\mathbf{J} \leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
        Backward propagation :
        Compute $\left\{\frac{\partial J}{\partial \hat{\mathbf{y}}} \& \frac{\partial J}{\partial \mathbf{a}^{(L)}}\right\}$
        $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)$
        $s.receive\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
        for $i \leftarrow l$ down to $1$ do
            Compute $\left\{\frac{\partial J}{\partial \boldsymbol{w}^{(i)}}, \frac{\partial J}{\partial \boldsymbol{b}^{(i)}}\right\}$
            Update $\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}$
        end
        $\boldsymbol{w}'^{(L)} = HE.Dec\overline{(\boldsymbol{w}'^{(L)})}$
        $s.send\left(\boldsymbol{w}'^{(L)}\right)$
    end
end
```

Algorithm 2: Server Side

```
Initialization:
$s \leftarrow$ socket initialization;
$s.connect$
$\eta, n, N, E \leftarrow s.synchronize()$
$\{\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow \Phi$
$\{\mathbf{z}^{(i)}\}_{\forall i \in \{l+1..L\}} \leftarrow \emptyset$
$\left\{\frac{\partial J}{\partial \mathbf{z}^{(i)}}\right\}_{\forall i \in \{l+1..L\}} \leftarrow \emptyset$
for $e \in E$ do
    for $i \leftarrow 1$ to $N$ do
        Forward propagation :
        $O.zero\_grad()$
        $s.receive\ (\overline{(\mathbf{a}^{(l)})})$
        $\overline{\mathbf{a}^{(L)}} \leftarrow \left(\overline{(\mathbf{a}^{(l)})}.\ w' + b\right)$
        $s.send\left(\overline{\mathbf{a}^{(L)}}\right)$
        Backward propagation :
        $s.receive\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)$
        Compute $\left\{\overline{\frac{\partial J}{\partial \boldsymbol{w}'^{(L)}}}, \frac{\partial J}{\partial \boldsymbol{b}^{(L)}}, \frac{\partial J}{\partial \boldsymbol{a}^{(L)}}\right\}$
        Encrypt and add noise to $\boldsymbol{w}'^{(L)}$
        Update $\boldsymbol{w}'^{(L)},, \boldsymbol{b}^{(L)}$.
        Compute $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$
        $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
        $s.send\overline{(\boldsymbol{w}'^{(L)})}$
        $s.receive\left(\boldsymbol{w}'^{(L)}\right)$
        Subtract noise from $\boldsymbol{w}'^{(L)}$
    end
end
```

Hyper- and HE- parameters settings: First, we reproduced the training results for the local version and the split version on HESplit network plaintext data from [23] and concluded in the same results, namely 88.06% accuracy on

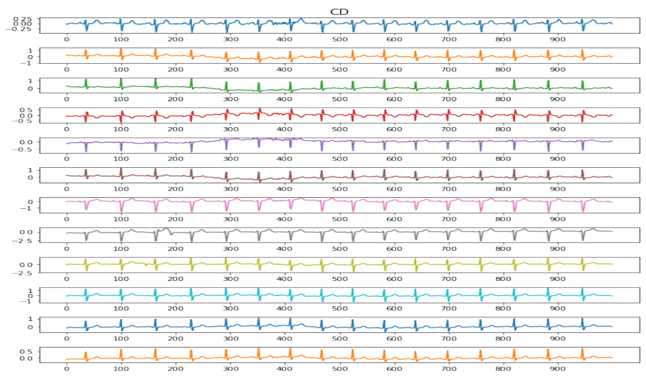

Fig. 11: An example CD ECG signal from the PTB-XL dataset.

the MIT-BIH dataset. We also trained a plaintext split network on the PTB-XL dataset and got 67.68% accuracy. Note that these accuracies are low, however, as we only focus on comparing the accuracies when training on plaintext versus encrypted data, getting high predictive accuracies on these datasets is not the focus of this work. To train the network on these datasets, we use 10 epochs with a learning rate of 0.001 and batch size 4. For plaintext training, we used the Adam optimizer [24] on both the client and server side.

We use the same NN hyperparameters when training the HESplit network on encrypted activation maps, however, the optimizer on the server side is a simple stochastic gradient descent. To encrypt the activation maps before sending them to the server, we employ two sets of HE parameters as following:

1. S1: $N = 2^{13} = 8192,\ \mathcal{C} = [40, 21, 21, 21, 40],\ \varDelta = 2^{21}$
2. S2: $N = 2^{14} = 16384,\ \mathcal{C} = [40, 21, 21, 21, 40],\ \varDelta = 2^{21}$

The only difference between the two sets of HE parameters is the polynomial modulus degree N, which directly affects the computational performance of the scheme (bigger is worse) and its security level (bigger is better). Apart from that, the coefficient modulus $\mathcal{C}$ is a list of prime numbers that define scheme noise levels and HE multiplication depth. Each HE multiplication consumes a prime in $\mathcal{C}$, and after all primes in the list are consumed, we will no longer be able to perform multiplication in the encrypted domain. $\varDelta$ is called the scaling factor, which is the factor where the plaintext message is multiplied by to maintain a certain level of precision during the encoding process. We vary only N to assess the trade-off between security and performance of our protocol. The results of training the HESplit network with S1 and S2 on the MIT-BIH dataset can be found in Table 1. Due to computational limitations, we can only train HESplit with $\mathcal{S}_1$ on the PTB-XL dataset.

For the MIT-BIH dataset, the best test accuracy, after training on encrypted activation maps, is 83.49% when using S1 with the training batch size of 4. Overalls, we can see that for MIT-BIH, S1 achieve better accuracy than S2, and the training time as well as communication cost using S1 are also about

Table 1: Training results of HESplit network on the MIT-BIH abd PTB-XL dataset using different sets of HE parameters and batch sizes. The training duration (in sec) and communication (in Gb) overhead are reported as the average value per epoch

Dataset	HE param set	Batch Size	Accuracy (%)	Training Time (s)	Communication (Gb)
MIT-BIH	S1	4	83.49	8100.60	239.53
		8	78.01	4780.73	121.57
		16	74.77	2855.29	62.58
		32	73.79	1658.73	33.09
		64	62.08	987.42	18.35
	S2	4	83.09	19746.85	471.57
		8	67.94	10378.92	239.33
		16	70.33	5358.59	123.20
		32	72.47	3152.82	65.14
		64	64.42	1913.11	36.11
PTB-XL	S1	4	58.71	100060.60	2624.85
		8	58.95	49334.95	1315.38
		16	57.10	22501.79	660.65
		32	59.36	12370.38	333.23
		64	56.45	5702.29	169.60

half when using S2. This indicates the trade off between security and utility. For both S1 and S2, the batch size of 4 achieve the best accuracies overall, since once the batch size get bigger, more compression will be done in HE batch processing and more noise will be introduced in HE batch computations. Nonetheless, the bigger the batch size, the less training time and communication needed, and this relationship is linear as observed from our experiments. When the batch size is small ($n = 4$), S1 and S2 produce very comparable accuracies (83.49% vs 83.09%), and S2 incur double training time as well as communication cost compared to S1. However, when n gets bigger, the accuracies of S2 degrade very quickly compared to S1, as can observed when $n = 16$ and $n = 32$. When $n = 64$, both S1 and S2 produce very low predictive power with 62.08% and 64.42%. We reason that when n gets big, the accumulated noise also become very big that the neural network does equally bad on both cases. For the PTB-XL dataset, accuracies across all batch sizes vary much less compared to those from MIT-BIH. The reason for this is that because an ECG example from PTB-XL is much bigger than that from MIT-BIH (length 1000 vs length 128), compressing along the batch size when doing HE encryption and noise incurred during HE computation produce less effects on the results.

In Table 2, we provide a comparison between HESplit network, the work from [23] (Split Ways) and the split plaintext training results when the training batch size is 4. In terms of accuracy, for MIT-BIH, HESplit network produces 4.57% and 1.82% less than the plaintext version and the Split Ways network. For PTB-XL, the difference is bigger: 6.71% less than Split Ways and 8.97% less than plaintext. W.r.t training time and communication, both HESplit and Split Ways require thousands of time more to train compared to the plaintext version. However, compared to Split Ways, HESplit is 6× faster and about 160× less communication overhead on MIT-BIH. On PTB-XL, HESplit requires 1.3× more time to train but 44× less communication overhead compared to Split Ways. The reason for HESplit having longer training time compared to Split Ways on PTB-XL is that the batch size is only small ($n = 4$). We predict when n get bigger, we will achieve shorter training time compared to Split Ways. On the other hand, we can see that overalls, HESplit produces much less communication cost compared to Split Ways.

Table 2: . The training time (in seconds) and communication (in Gb) overhead are reported as the average value per epoch.

Dataset	Framework	Batch Size	Accuracy (%)	Training Time (s)	Communication (Gb)
MIT-BIH	HESplit	4	83.49	8100.60	239.53
	Split Ways [23]	4	85.31	50318	37840
	Plaintext	4	88.06	8.56	0.033
PTB-XL	HESplit	4	58.71	100060.60	2624.85
	Split Ways [23]	4	65.42	72534	115640
	Plaintext	4	67.68	15.55	0.317

Open Science and Reproducible Research: To support open science and reproducible research, and provide researchers with the opportunity to use, test, and hopefully extend our work, our source code has been made available online[5].

8 Conclusion

Split learning is a collaborative learning technique that has been proposed as a privacy-preserving technique. Split learning-based solutions have been used in both open-source and commercial applications. However, recent studies have indicated that this approach is not infallible. Hence, it is important to consider other privacy-preserving techniques in order to ensure the security of the data. In this research, we demonstrate how the privacy leakage in split learning can be reduced using a different privacy-preserving technique – homomorphic encryption[6]. In homomorphic encryption, the computation is performed on encrypted data (which is computationally expensive). While this work features a single fully connected layer on the server side due to the constraints of encrypted training with HE, we acknowledge the potential for further expansion in future work. In upcoming research, we plan to explore advanced privacy enhancing technologies to incorporate more complex layers on the server side, balancing efficiency and model complexity. Furthermore, this work only considers a one-client setting, leaving the multi-client setting as a future work.

[5] https://github.com/khoaguin/HESplitNet

[6] There is a plethora of other PPML techniques utilizing HE but to the best of our knowledge, until now there is only one work that combines SL with encrypted data.

References

1. Abuadbba, S., Kim, K., Kim, M., Thapa, C., Camtepe, S.A., Gao, Y., Kim, H., Nepal, S.: Can we use split learning on 1d cnn models for privacy preserving training? In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security. pp. 305–318 (2020) 3, 4, 16
2. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) **51**(4), 1–35 (2018) 2
3. Bakas, A., Michalas, A., Dimitriou, T.: Private lives matter: A differential private functional encryption scheme. In: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy. p. 300–311. CODASPY '22, Association for Computing Machinery, New York, NY, USA (2022) 3
4. Benaissa, A., Retiat, B., Cebere, B., Belfedhal, A.E.: Tenseal: A library for encrypted tensor operations using homomorphic encryption (2021) 16
5. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al.: Towards federated learning at scale: System design. Proceedings of Machine Learning and Systems **1**, 374–388 (2019) 2
6. Boulemtafes, A., Derhab, A., Challal, Y.: A review of privacy-preserving techniques for deep learning. Neurocomputing **384**, 21–45 (2020) 2
7. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. Proceedings of Advances in Cryptology-Crypto **7417** (08 2012) 7
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014) 7
9. Cabrero-Holgueras, J., Pastrana, S.: Sok: Privacy-preserving computation techniques for deep learning. Proceedings on Privacy Enhancing Technologies **2021**(4), 139–162 (2021) 2
10. Chai, C., Wang, J., Luo, Y., Niu, Z., Li, G.: Data management for machine learning: A survey. IEEE Transactions on Knowledge and Data Engineering (2022) 2
11. Chen, J., Wang, W.H., Shi, X.: Differential privacy protection against membership inference attack on machine learning for genomic data. In: BIOCOMPUTING 2021: Proceedings of the Pacific Symposium. pp. 26–37. World Scientific (2020) 4
12. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018) 7
13. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017) 3, 7
14. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016) 7
15. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015) 7
16. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985) 7

17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009) 7
18. Gupta, O., Raskar, R.: Distributed learning of deep neural network over multiple agents. Journal of Network and Computer Applications **116**, 1–8 (2018) 2, 4
19. He, Z., Zhang, T., Lee, R.B.: Model inversion attacks against collaborative inference. In: Proceedings of the 35th Annual Computer Security Applications Conference. pp. 148–162 (2019) 4
20. Hesamifard, E., Takabi, H., Ghasemi, M., Wright, R.N.: Privacy-preserving machine learning as a service. Proc. Priv. Enhancing Technol. **2018**(3) (2018) 2
21. Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(1), 221–231 (2013) 5
22. Khan, T., Bakas, A., Michalas, A.: Blind faith: Privacy-preserving machine learning using function approximation. In: 2021 IEEE Symposium on Computers and Communications (ISCC). pp. 1–7. IEEE (2021) 13
23. Khan, T., Nguyen, K., Michalas, A.: Split ways: Privacy-preserving training of encrypted data using split learning. arXiv preprint arXiv:2301.08778 (2023) 3, 5, 10, 11, 12, 15, 17, 19, 20
24. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014) 18
25. Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., Inman, D.J.: 1d convolutional neural networks and applications: A survey. Mechanical systems and signal processing **151**, 107398 (2021) 5
26. Koda, Y., Park, J., Bennis, M., Yamamoto, K., Nishio, T., Morikura, M.: One pixel image and rf signal based split learning for mmwave received power prediction. In: Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies. pp. 54–56 (2019) 4
27. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Computation **1**(4), 541–551 (1989) 5
28. Li, D., Zhang, J., Zhang, Q., Wei, X.: Classification of ecg signals based on 1d convolution neural network. In: 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom). IEEE (2017) 8
29. Lim, W.Y.B., Ng, J.S., Xiong, Z., Niyato, D., Leung, C., Miao, C., Yang, Q.: Incentive mechanism design for resource sharing in collaborative edge learning. arXiv preprint arXiv:2006.00511 (2020) 4
30. Lindell, Y.: Secure multiparty computation. Communications of the ACM **64**(1), 86–96 (2020) 2
31. Mireshghallah, F., Taram, M., Ramrakhyani, P., Jalali, A., Tullsen, D., Esmaeilzadeh, H.: Shredder: Learning noise distributions to protect inference privacy. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 3–18 (2020) 4
32. Moody, G.B., Mark, R.G.: The impact of the mit-bih arrhythmia database. IEEE Engineering in Medicine and Biology Magazine **20**(3), 45–50 (2001) 16
33. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999) 7
34. Pasquini, D., Ateniese, G., Bernaschi, M.: Unleashing the tiger: Inference attacks on split learning. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2113–2129 (2021) 4, 11

35. Pereteanu, G.L., Alansary, A., Passerat-Palmbach, J.: Split he: Fast secure inference combining split learning and homomorphic encryption. arXiv preprint arXiv:2202.13351 (2022) 5
36. Poirot, M.G., Vepakomma, P., Chang, K., Kalpathy-Cramer, J., Gupta, R., Raskar, R.: Split learning for collaborative deep learning in healthcare. arXiv preprint arXiv:1912.12115 (2019) 2, 4
37. Riazi, M.S., Rouani, B.D., Koushanfar, F.: Deep learning on private data. IEEE Security & Privacy **17**(6), 54–63 (2019) 2
38. Ribeiro, M., Grolinger, K., Capretz, M.A.: Mlaas: Machine learning as a service. In: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). pp. 896–902. IEEE (2015) 2
39. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978) 7
40. Samikwa, E., Di Maio, A., Braun, T.: Ares: Adaptive resource-aware split learning for internet of things. Computer Networks **218**, 109380 (2022) 4
41. Singh, A., Vepakomma, P., Gupta, O., Raskar, R.: Detailed comparison of communication efficiency of split learning and federated learning. arXiv preprint arXiv:1909.09145 (2019) 2
42. Thapa, C., Arachchige, P.C.M., Camtepe, S., Sun, L.: Splitfed: When federated learning meets split learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 8485–8493 (2022) 5
43. Titcombe, T., Hall, A.J., Papadopoulos, P., Romanini, D.: Practical defences against model inversion attacks for split neural networks. arXiv preprint arXiv:2104.05743 (2021) 4
44. Topol, E.J.: High-performance medicine: the convergence of human and artificial intelligence. Nature medicine **25**(1), 44–56 (2019) 2
45. Turina, V., Zhang, Z., Esposito, F., Matta, I.: Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). pp. 250–260. IEEE (2021) 2
46. Vepakomma, P., Gupta, O., Dubey, A., Raskar, R.: Reducing leakage in distributed deep learning for sensitive health data. arXiv preprint arXiv:1812.00564 (2019) 4
47. Vepakomma, P., Gupta, O., Swedish, T., Raskar, R.: Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564 (2018) 4, 6
48. Wagner, P., Strodthoff, N., Bousseljot, R.D., Kreiseler, D., Lunze, F.I., Samek, W., Schaeffter, T.: Ptb-xl, a large publicly available electrocardiography dataset. Scientific data **7**(1), 1–15 (2020) 16
49. Yansong, G., KIM, M., ABUADBBA, S., et al.: End-to-end evaluation of federated learning and split learning for internet of things. In: Proceedings of 2020 International Symposium on Reliable Distributed Systems (SRDS), Shanghai, China 4