Love or Hate? Share or Split? Privacy-Preserving Training Using Split Learning and Homomorphic Encryption

1st Tanveer Khan
Tampere University
Tampere, Finland
tanveer.khan@tuni.fi

2nd Khoa Nguyen
Tampere University
Tampere, Finland
khoa.nguyen@tuni.fi

3rd Antonis Michalas
Tampere University, Finland,
University of Westminster
antonios.michalas@tuni.fi

4th Alexandros Bakas
Tampere University and Nokia Bell Labs
Tampere, Finland
alexandros.bakas@tuni.fi

***Abstract*—Split learning (SL) is a new collaborative learning technique that allows participants, e.g. a client and a server, to train machine learning models without the client sharing raw data. In this setting, the client initially applies its part of the machine learning model on the raw data to generate activation maps and then sends them to the server to continue the training process. Previous works in the field demonstrated that reconstructing activation maps could result in privacy leakage of client data. In addition to that, existing mitigation techniques that overcome the privacy leakage of SL prove to be significantly worse in terms of accuracy. In this paper, we improve upon previous works by constructing a protocol based on U-shaped SL that can operate on homomorphically encrypted data. More precisely, in our approach, the client applies homomorphic encryption on the activation maps before sending them to the server, thus protecting user privacy. This is an important improvement that reduces privacy leakage in comparison to other SL-based works. Finally, our results show that, with the optimum set of parameters, training with HE data in the U-shaped SL setting only reduces accuracy by 2.65% compared to training on plaintext. In addition, raw training data privacy is preserved.**

***Index Terms*—Activation Maps, Homomorphic Encryption, Machine Learning, Privacy, Split Learning**

I. Introduction

SL [1] is a collaborative learning approach that divides a Machine Learning (ML) into two parts: the client-side and the server-side. It facilitates training the Deep Neural Networks (DNN) among multiple data sources, while mitigating the need to directly share raw labeled data with collaboration parties. The advantages of SL are multifold: *(i)* it allows multiple parties to collaboratively train a neural network, *(ii)* it allows users to train ML models without sharing their raw data with a server running part of a DNN model, thus preserving user privacy, *(iii)* it protects both the client and the server from revealing their parts of the model, and *(iv)* it reduces the client's computational overhead by not running the entire model (i.e. utilizing a smaller number of layers) [2].

Though SL offers an extra layer of privacy protection by definition, there are no works exploring how it is combined with popular techniques that promise to preserve user privacy (e.g. encryption). In [3], the authors studied whether SL can handle sensitive time-series data and demonstrated that SL alone is *insufficient* when performing privacy-preserving training for 1-dimensional (1D) CNN models. More precisely, the authors showed raw data can be reconstructed from the activation maps of the intermediate split layer. The authors also employed two mitigation techniques, adding hidden layers and applying differential privacy to reduce privacy leakage. However, based on the results, none of these techniques can effectively reduce privacy leakage from all channels of the SL activation. Furthermore, both these techniques result in significantly reducing the joint model's accuracy.

In this paper, we focus on training an ML model in a privacy-preserving manner, where a client and a server collaborate to train the model. More specifically, we construct a model that uses Homomorphic Encryption (HE) [4] to mitigate privacy leakage in SL. In our model, the client first encrypts the activation maps and then sends the encrypted activation maps to the server. The encrypted activation maps do *not* reveal anything about the raw data (i.e. it is *not* possible to reconstruct the original raw data from the encrypted activation maps).

Contributions: The main contributions of this paper are:

C1. We construct a U-shaped split 1D CNN model and experiment using plaintext activation maps sent from the client to the server. Through the U-shaped 1D CNN model, clients do *not* need to share the ground truth labels with the server – this is an important improvement that reduces privacy leakage compared to [3].

C2. We constructed the HE version of the U-shaped SL technique. In the encrypted U-shaped SL model, the client encrypts the activation map using HE and sends it to the server. The advantage of HE encrypted U-shaped SL over the plaintext U-shaped SL is that server performs computation over encrypted activation maps.

C3. To assess the applicability of our framework, we performed experiments on the PTB-XL heartbeat datasets [5], currently being the largest open-source electrocardiography dataset to our knowledge. We experimented with activation maps of lengths 256 for both plaintext and homomorphically encrypted activation maps and we have measured the model's performance by considering training duration, test accuracy, and communication cost.

Organization: The rest of the paper is organized as follows: In section II, we provide necessary background for 1D CNN, HE and SL. In section III, we present works in the area of SL, followed by design and implementation of the split 1D CNN training protocols in section IV, extensive experimental results in section V and conclude the paper in section VI.

II. Preliminaries

A. Convolutional Neural Network

CNN is a special type of neural network using convolution layers to extract features from data [6]. In this work, we employ a 1D CNN [3], [7] as a feature detector and classifier for a heartbeat datasets, namely PTB-XL [5]. The employed 1D CNN consists of the following layers stacked on top of each other, namely: Conv1D, Leaky ReLU, max pooling and a single fully connected and a softmax layer (see Figure 1).

B. Homomorphic Encryption

HE is an emerging cryptographic technique for computations on encrypted data. HE schemes are divided into three main categories according to their functionality: Partial HE [8], leveled (or somewhat) HE [4], and fully HE [9]. Each scheme has its own benefits and disadvantages. In this work, we use the CKKS Leveled HE scheme [4]. CKKS allows users to do additions and a limited number of multiplications on vectors of complex values (and hence, real values too). The most important parameters of the CKKS scheme are Polynomial Modulus $\mathcal{P}$, Coefficient Modulus $\mathcal{C}$ and Scaling Factor Δ.

C. Split Learning

Although the configuration of SL could potentially take many forms, widely known configurations are simple vanilla split and U-shaped SL. In a simple two-party vanilla split, a client and a server collaborate to train the split model with no access to each other's parts. More specifically, the layers of a DNN model are split into two parts (A and B) and distributed to the client and the server. The client owning the data, uses forward propagation to train part A (comprising the first few layers) and sends the activated output from the split layer (part A's final layer) to the server. The server performs the forward training on the outputs activated from the client using part B. Furthermore, the server performs the backward propagation on part B, only returning to the client the gradients of the split layer's active outputs (part B's first layer) to complete the backward propagation on part A. This process is repeated until the model converges. Although the client and server do not share any raw input data, this configuration requires label sharing. The sharing of labels can be eliminated by using a U-shaped configuration. The U-shaped SL configuration is different from the simple vanilla setup, as it does not require of clients to share labels [2]. On the server-side, the network is wrapped at the end layer, and the outputs are sent back to client entities (see Figure 1). Clients generate gradients from the end layers and utilize them for backward propagation without revealing the corresponding labels.

III. Related Work

Initially, it was believed that SL is a promising approach in terms of client raw data protection, and for doing PPML [10] as it does *not* share raw data and also has the benefit of *not* disclosing the model's architecture and weights.For example, Gupta and Raskar [1] predicted that reconstructing raw data on the client-side, while using SL would be difficult. In addition, the authors of [2] employed the SL model to the healthcare applications to protect the users' personal data. Vepakomma *et al.* found that SL outperforms other collaborative learning techniques like federated learning [11] in terms of accuracy [2].

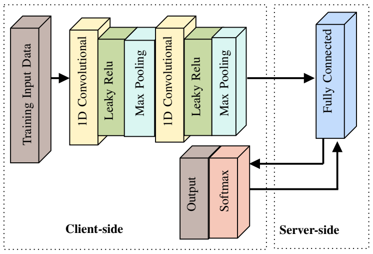

Fig. 1: U-shaped Split-Learning

However, SL provides data privacy on the grounds that only activation maps are shared between the parties. Different studies showed the possibility of privacy leakage in SL. In [12], the authors analyzed the privacy leakage of SL and found a considerable leakage from the split layer in the 2D CNN model. Furthermore, the authors mentioned that it is possible to reduce the distance correlation (a measure of dependence) between the split layer and raw data by slightly scaling the weights of all layers before the split. This works well in models with a large number of hidden layers before the split.

The work of Abuadbba *et al.* [3] is the first study exploring whether SL can deal with time-series data. According to the results, SL can be applied to a model without the model accuracy degradation. The authors also showed that when SL is directly adopted into 1D CNN models for time series data could result in significant privacy leakage. Two mitigation techniques were employed to limit the potential privacy leakage in SL: *(i)* increasing the number of layers before the split on the client-side and *(ii)* applying differential privacy to the split layer activation before sending the activation map to the server. However, both techniques suffer from a loss of model accuracy, particularly when differential privacy is used.

In [3], during the forward propagation, the client sends the activation map in plaintext to the server, while the server can easily reconstruct the original raw data from the activated vector of the split layer leading to clear privacy leakage. In our work, we constructed a training protocol, where, instead of sending plaintext activation maps, the client first conducts an encryption using HE and then sends said maps to the server. In this way, the server is unable to reconstruct the original raw data, but can still perform a computation on the encrypted activation maps, enabling the the training process.

IV. Split Model Training Protocols

In this section, we first present the protocol for training the U-shaped split 1D CNN on plaintext activation maps, followed by the protocol for training the U-shaped split 1D CNN on encrypted activation maps.

A. Actors in the U-Split Learning Model

We have two parties: client and a server. Each party plays a specific role and has access to certain parameters. We construct the U-shaped split 1D CNN in such a way that the first few as well as the last layer are on the client-side, while the remaining layers are on the server-side, as demonstrated in Figure 1. The client and server collaborate to train the split model by sharing the activation maps and gradients. To elaborate further, we describe their respective roles and access as follows:

- Client: In the plaintext version, the client holds two Conv1D layers and can access their weights and biases in plaintext. However, other layers such as Max Pooling, Leaky ReLU, Softmax do not have associated weights and biases. Apart from these, in the HE encrypted version, the client is also responsible for generating the context for HE and has access to all context parameters including Polynomial modulus $\mathcal{P}$, Coefficient modulus $\mathcal{C}$, Scaling factor Δ, Public key pk and Secret key sk. Note that for both training on plaintext and encrypted activation maps, the raw data examples $\mathbf{x}$'s and their corresponding labels $\mathbf{y}$'s reside on the client side and are never sent to the server during the training process.
- Server: The computation performed on the server-side is limited to only one linear layer. The reason for having only one linear layer on the server-side is due to computational constraints when training on HE encrypted data. Hence, the server can exclusively access the weights and biases of this linear layer. Regarding the HE context parameters, the server has access to $\mathcal{P}$, $\mathcal{C}$, Δ, and pk shared by the client, with the exception of the sk. Not holding the secret key, the server cannot decrypt the HE encrypted activation maps sent from the client. The hyperparameters shared between the client and the server include the learning rate (η), batch size (n), number of batches to be trained (N), and the number of training epochs (E).

B. Plaintext Activation Maps

We have used algorithm 1 and algorithm 2 to train the U-shaped split 1D CNN model. First, the client and server start the socket initialization process and synchronize the hyperparameters η, n, N, E. They also initialize the weights of their layers using the randomly initialized weights Φ.

During the forward propagation phase, the client forward-propagates the input $\mathbf{x}$ until the l^{th} layer and sends the activation $\mathbf{a}^{(l)}$ to the server. The server continues to forward propagate and sends the output $\mathbf{a}^{(L)}$ to the client. Next, the client applies the Softmax function on $\mathbf{a}^{(L)}$ to get $\hat{\mathbf{y}}$ and calculates the error J using the loss function $J = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$.

The client starts the backward propagation by calculating and sending the gradient of the error w.r.t $\mathbf{a}^{(L)}$, i.e. $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$, to the server. The server continues the backward propagation, calculates $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ and sends $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ to the client. After receiving the gradients $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ from the server, the backward propagation continues to the first hidden layer on the client-side. Note that the exchange of information between client and server in these algorithms takes place in plaintext. As can be seen in algorithm 1, the client sends the activation maps $\mathbf{a}^{(l)}$ to the server in plaintext and receives the output of the linear layer $\mathbf{a}^{(L)}$ from the server in plaintext. The same applies on the server side: receiving $\mathbf{a}^{(l)}$ and sending $\mathbf{a}^{(L)}$ in the plaintext as can be seen in algorithm 2. Sharif *et al.* [3] showed that the exchange of plaintext activation maps between client and server using SL reveals important information regarding the client's raw sequential data. Later, in subsection V-E we show in detail how passing the forward activation maps from the client to the server in the plaintext will result in information leakage. To mitigate this privacy leakage, we propose the protocol, where the client encrypts the activation maps before sending them to the server, as described in subsection IV-C.

Algorithm 1: Client Side

```
Initialization:
$s \leftarrow$ socket initialized with port and address;
$s.connect$
$\eta, n, N, E \leftarrow s.synchronize()$
$\{\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow initialize\ using\ \Phi$
$\{\mathbf{z}^{(i)}\}_{\forall i \in \{0..l\}}, \{\mathbf{a}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow \emptyset$
$\left\{\frac{\partial J}{\partial \mathbf{z}^{(i)}}\right\}_{\forall i \in \{0..l\}}, \left\{\frac{\partial J}{\partial \mathbf{a}^{(i)}}\right\}_{\forall i \in \{0..l\}} \leftarrow \emptyset$
for $e \in E$ do
    for each batch $(\mathbf{x}, \mathbf{y})$ generated from $D$ 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
            $s.send\ (\mathbf{a}^{(l)})$
            $s.receive\ (\mathbf{a}^{(L)})$
            $\hat{y} \leftarrow Softmax\left(\mathbf{a}^{(L)}\right)$
            $J \leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
        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 $i \leftarrow 1$ to $l$ 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
    end
end
```

C. Encrypted Activation Maps

The protocol for training the U-shaped 1D CNN with a homomorphically encrypted activation map consists of four phases: initialization, forward propagation, classification, and backward propagation. The initialization phase only takes place once at the beginning of the procedure, whereas the other phases continue until the model iterates through all epochs. Each phase is described in detail in the following subsections.

Initialization: The initialization phase consists of socket initialization, context generation, and random weight loading. The client first establishes a socket connection to the server and synchronizes the four hyperparameters $\eta,\ n,\ N, E$ with the server, shown in algorithm 3 and algorithm 4. These parameters

Algorithm 2: Server Side

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

Algorithm 3: Client Side

```
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 $(\mathbf{x}, \mathbf{y})$ generated 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 $\left\{\frac{\partial J}{\partial \hat{\mathbf{y}}} \& \frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right\}$
            $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right)$
            $s.receive\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
        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
    end
end
```

must be synchronized on both sides to be trained in the same way. Also, the weights on the client and server are initialized with the same set of corresponding weights in the local model to accurately assess and compare the influence of SL on performance. On both the client and the server sides, $\boldsymbol{w}^{(i)}$ are initialized using corresponding parts of Φ. The activation map at layer i ($\mathbf{a}^{(i)}$), output tensor of Conv1D layer ($\mathbf{z}^{(i)}$), and the gradients are initially set to zero. In this phase, the context generated is a specific object that holds pk and sk of the HE scheme as well as certain other parameters like $\mathcal{P}$, $\mathcal{C}$ and Δ.

Further information on the HE parameters and how to choose the best-suited parameters can be found in the TenSEAL's benchmarks tutorial. As shown in algorithm 3 and algorithm 4, the context is either public ($\mathsf{ctx}_{\mathsf{pub}}$) or private ($\mathsf{ctx}_{\mathsf{pri}}$) depending on whether it holds the secret key sk. Both the $\mathsf{ctx}_{\mathsf{pub}}$ and $\mathsf{ctx}_{\mathsf{pri}}$ have the same parameters, though $\mathsf{ctx}_{\mathsf{pri}}$ holds a sk and $\mathsf{ctx}_{\mathsf{pub}}$ does not. The server does not have access to the sk as the client only shares the $\mathsf{ctx}_{\mathsf{pub}}$ with the server. After completing the initialization phase, both the client and server proceed to the forward and backward propagation phases.

Forward propagation: The forward propagation starts on the client side. The client first zeroes out the gradients for the batch of data $(\mathbf{x}, \mathbf{y})$. He then begins calculating $\mathbf{a}^{(l)}$ from $\mathbf{x}$, algorithm 3 where each $f^{(i)}$ is a Conv1D layer.

The Conv1D layer can be described as following: given a 1D input signal that contains C channels, where each channel $\mathbf{x}_{(i)}$ is a 1D array ($i \in \{1, \ldots, C\}$), a Conv1D layer produces an output that contains C' channels. The j^{th} output channel $\mathbf{y}_{(j)}$, where $j \in \{1, \ldots, C'\}$, can be described as

$$\mathbf{y}_{(j)} = \boldsymbol{b}_{(j)} + \sum_{i=1}^{C} \boldsymbol{w}_{(i)} \star \mathbf{x}_{(i)}, \tag{1}$$

where $\boldsymbol{w}_{(i)}, i \in \{1, \ldots, C\}$ are the weights, $\boldsymbol{b}_{(j)}$ are the biases of the Conv1D layer, and $\star$ is the 1D cross-correlation operation. The $\star$ operation can be described as

$$\mathbf{z}(i) = (\boldsymbol{w} \star \mathbf{x})(i) = \sum_{j=0}^{m-1} \boldsymbol{w}(j) \cdot \mathbf{x}(i+j), \tag{2}$$

where $\mathbf{z}(i)$ denotes the i^{th} element of the output vector $\mathbf{z}$, and i starts at 0. Here, the size of the 1D weighted kernel is m.

In algorithm 3, $g^{(i)}$ can be seen as the combination of Max Pooling and Leaky ReLU functions. The final output activation maps of the l^{th} layer from the client is $\mathbf{a}^{(l)}$. The client then homomorphically encrypts $\mathbf{a}^{(l)}$ and sends the encrypted activation maps $\overline{\mathbf{a}^{(l)}}$ to the server. In algorithm 4, the server receives $\overline{\mathbf{a}^{(l)}}$ and then performs forward propagation, which is a linear layer evaluated on HE encrypted data $\overline{\mathbf{a}^{(l)}}$ as

$$\overline{\mathbf{a}^{(L)}} = \overline{\mathbf{a}^{(l)}}\boldsymbol{w}^{(L)} + \boldsymbol{b}^{(L)}. \tag{3}$$

After that, the server sends $\overline{\mathbf{a}^{(L)}}$ to the client (algorithm 4). Upon reception, the client decrypts $\overline{\mathbf{a}^{(L)}}$ to get $\mathbf{a}^{(L)}$, performs Softmax on $\mathbf{a}^{(L)}$ to produce the predicted output $\hat{\mathbf{y}}$ and calculate the loss J, as can be seen in algorithm 3. Having finished the forward propagation we may move on to the backward propagation part of the protocol.

Backward propagation: After calculating J, the client starts the backward propagation by initially computing $\frac{\partial J}{\partial \hat{\mathbf{y}}}$ and then $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ using the chain rule (algorithm 3).

$$\frac{\partial J}{\partial \mathbf{a}^{(L)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a}^{(L)}}, \text{and} \quad \frac{\partial J}{\partial \boldsymbol{w}^{(L)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \boldsymbol{w}^{(L)}} \quad (4)$$

Following, the client sends $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ to the server. Upon reception, the server computes $\frac{\partial J}{\partial \boldsymbol{b}}$ by simply doing $\frac{\partial J}{\partial \boldsymbol{b}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}}$, based on equation (3). The server then updates the weights and biases of his linear layer according to equation (5).

$$\boldsymbol{w}^{(L)} = \boldsymbol{w}^{(L)} - \eta \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}, \quad b^{(L)} = \boldsymbol{b}^{(L)} - \eta \frac{\partial J}{\partial \boldsymbol{b}^{(L)}} \quad (5)$$

Next, the server calculates

$$\frac{\partial J}{\partial \mathbf{a}^{(l)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{a}^{(l)}}, \quad (6)$$

and sends $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ to the client. After receiving $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$, the client calculates gradients of J w.r.t the weights and biases of the Conv1D layer using the chain-rule: as:

$$\frac{\partial J}{\partial \boldsymbol{w}^{(i-1)}} = \frac{\partial J}{\partial \boldsymbol{w}^{(i)}} \frac{\partial \boldsymbol{w}^{(i)}}{\partial \boldsymbol{w}^{(i-1)}} \quad (7)$$

$$\frac{\partial J}{\partial \boldsymbol{b}^{(i-1)}} = \frac{\partial J}{\partial \boldsymbol{b}^{(i)}} \frac{\partial \boldsymbol{b}^{(i)}}{\partial \boldsymbol{b}^{(i-1)}} \quad (8)$$

After calculating the gradients $\frac{\partial J}{\partial \boldsymbol{w}^{(i)}}$, $\frac{\partial J}{\partial \boldsymbol{b}^{(i)}}$, the client updates $\boldsymbol{w}^{(i)}$ and $\boldsymbol{b}^{(i)}$ using the Adam optimization algorithm [13].

Algorithm 4: Server Side

```
Context Initialization:
    $s.receive(\mathsf{ctx_{pub}})$
for e in E do
    for $i \leftarrow 1$ to $N$ do
        Forward propagation :
            $O.zero\_grad()$
            $s.receive\ (\overline{\mathbf{a}^{(l)}})$
            $\overline{\mathbf{a}^{(L)}} \leftarrow \mathsf{HE.Eval}\left(f^{(i)}\left(\overline{\mathbf{a}^{(l)}}\right)\right)$
            $s.send\left(\overline{\mathbf{a}^{(L)}}\right)$
        Backward propagation :
            $s.receive\left\{\frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right\}$
            Compute $\frac{\partial J}{\partial \boldsymbol{b}^{(L)}}$
            Update $\boldsymbol{w}^{(L)}$, $\boldsymbol{b}^{(L)}$
            Compute $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$
            $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
    end
end
```

V. Performance Analysis

A. ECG Datasets

We evaluate our method on the PTB-XL dataset [5].

*a) **PTB-XL:**:* According to [5], PTB-XL is the largest open-source ECG dataset since 2020. The dataset contains 12-lead ECG-waveforms from 21837 records of 18885 patients. Each waveform from PTB-XL has a duration of 10 seconds. Two sampling rates are used to collect the data: 100 Hz and 500 Hz. In our experiment, we employ the 100 Hz waveforms. Each 12-lead ECG waveform is associated with one or several classes out of five classes: normal (NORM), conduction disturbance (CD), myocardial infarction (MI), hypertrophy (HYP), and ST/T change (STTC). For waveforms that belong to multiple classes, we choose only the first one and remove the others for simplicity. Figure 2 shows an example of a 12-lead normal heartbeat from the PTB-XL dataset. The dataset is then split into the train-test splits with a ratio of 90%-10%. In summary, after processing, we have a training split of size $[19267, 12, 1000]$, with 19,267 ECG waveform samples, of 12 channels (or leads) and 1,000 timesteps each. The test split's size is $[2163, 12, 1000]$.

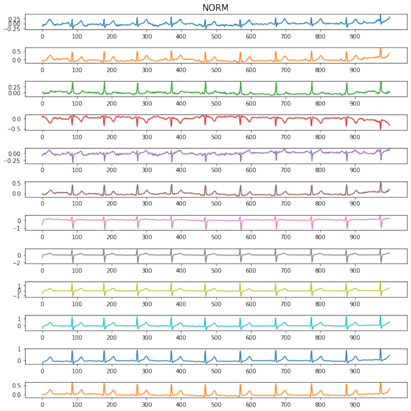

Fig. 2: A 12-lead normal heartbeat from the PTB-XL dataset.

B. Experimental Setup

All neural networks are trained on a machine with Ubuntu 20.04 LTS, processor Intel Core i7-8700 CPU at 3.20GHz, 32Gb RAM, GPU GeForce GTX 1070 Ti with 8Gb of memory. We write our program in the Python programming language version 3.9.7. The neural nets are constructed using the PyTorch library version 1.8.1+cu102. For HE algorithms, we employ the TenSeal library version 0.3.10.

In terms of hyperparameters, we train all networks with 10 epochs, a $\eta = 0.001$ learning rate, and a $n = 4$ training batch size. For the split neural network with HE activation maps, we use the Adam optimizer for the client model and mini-batch Gradient Descent for the server model. We use GPU for networks trained on the plaintext. For the U-shaped SL model on HE activation maps, we train the client model on GPU, and the server model on CPU.

C. Evaluation

In this section, we report the experimental results in terms of accuracy, training duration and communication throughput.

The 1D CNN models used for the PTB-XL datasets have two Conv1D layers and one linear layer. The activation maps are the output of the last Conv1D layer.

For the PTB-XL dataset [1], the number of the input channels for the first Conv1D layer are 12, since the input data are 12-lead ECG signals. Besides, we only experiment with 8 output channels for the second Conv1D layer. We denote this

[1] For the MIT-BIH dataset, please have a look at [14]

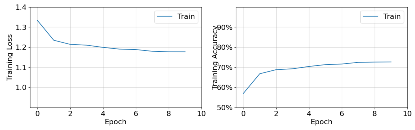

Fig. 3: Training locally on the plaintext PTB-XL dataset.

network by M. Using M, the output activation map size is [batch size, 2000].

Training the model M locally (without any split) on the plaintext PTB-XL dataset results in Figure 3. As we notice the best training accuracy after 10 epochs is only 72.65%. The best test accuracy is 67.68%. The reason for this low accuracy is that our neural network is small (it contains only 12013 trainable parameters), and we only train it for a particularly small epoch. Each training epoch takes 10.56 seconds on average.

D. U-shaped Split Learning using Plaintext Activation Maps

Based on our experiments, training U-shaped split model on plaintext yields same results in terms of accuracy compared to local training [3]. Although the authors of [3] only used the vanilla version of the split model, they too found that, compared to training locally, accuracy was not reduced.

We will now discuss the training time and communication overhead of the U-shaped split models and compare them to their local versions. For M, the communication overhead is on average 316.9 Mb per epoch, which is much bigger due to the bigger activation maps sent from the client during training.

E. Visual Invertibility

We visualize the activation maps produced by the model M to access their visual similarity compared to the original signals. In SL, the activation maps are sent from client to server to continue the training process. A visual representation of the activation maps reveals a high similarity between certain activation maps and the input data from the client, as shown in Figure 4, Figure 5, Figure 6, Figure 7, Figure 8 for five different classes of heartbeat in the PTB-XL dataset.

The figure indicates that, compared to the raw input data from the client, some activation maps have exceedingly similar patterns. This phenomenon clearly compromises the privacy of the client's raw data. The authors of [3] quantify the privacy leakage by measuring the correlations between the activation maps and the raw input signal by using two metrics: distance correlation and Dynamic Time Warping. This approach allows them to measure whether their solutions mitigate privacy leakage work. Since our work uses HE , said metrics are unnecessary as the activation maps are encrypted.

F. U-shaped Split 1D CNN with Encrypted Activation Maps

The results of training different settings of M on the PTB-XL dataset are in Table I. The HE set of parameters $\mathcal{P} = 8192$, $\mathcal{C} = [40, 21, 21, 40]$, $\Delta = 2^{21}$ achieves the best test accuracy at 65.42%. This result is only 2.26% lower than the result obtained by the plaintext version. However, this set of parameters incurs the most communication overhead (115.64 Tb per epoch) and takes the longest to train (72 534 seconds per epoch). Overall, test accuracies achieved by different HE parameters are quite close to each other, with 65.42% being the lowest. Interestingly, the smallest set of HE parameters $\mathcal{P} = 2048, \mathcal{C} = [18, 18, 18], \Delta = 2^{16}$ achieves the second-best accuracy at 65.33% while only requiring about 1/10 of the training duration and 1/100 of the communication overhead compared to $\mathcal{P} = 8192$. The two sets of parameters with $\mathcal{P} = 4096$ produce quite similar results, while roughly taking same amount of time and communication overhead to train.

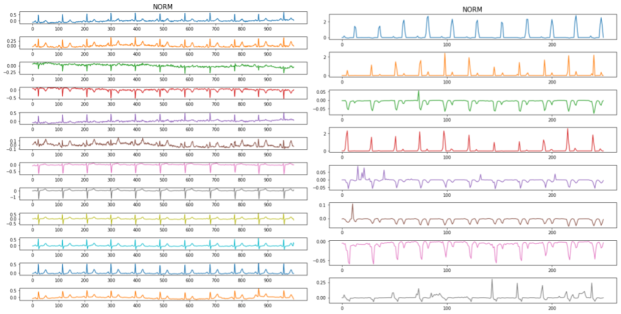

Fig. 4: Visual invertibility of the model M on the PTB-XL dataset. Left: input data (NORM class). Right: corresponding activation maps.

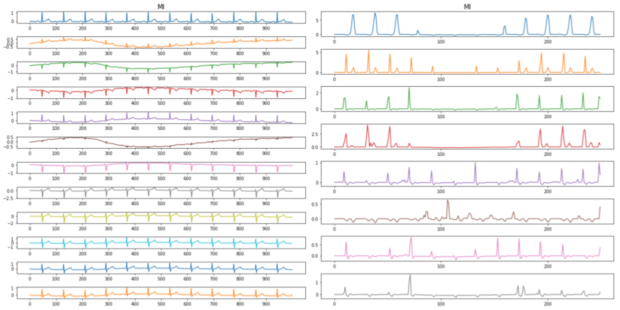

Fig. 5: Visual invertibility of the model M on the PTB-XL dataset. Left: input data (MI class). Right: corresponding activation maps.

Through our experiments, we see that training on encrypted activation maps can produce very optimistic results, with accuracy dropping by 2-3% for the best sets of HE parameters. Furthermore, training using BE can significantly reduce the amount of training time and communication overhead needed, while producing comparable results when it come to training without BE. The set of parameters with $\mathcal{P} = 8192$ always achieve the highest test accuracy, though incurring the highest communication overhead and the longest training time. The set of parameters with $\mathcal{P} = 4096$ can offer a good trade-off as they can produce on-par accuracy with $\mathcal{P} = 8192$, while requiring significantly less communication and training time. Experimental results show that with the smallest set of HE parameters $\mathcal{P} = 2048$, $\mathcal{C} = [18, 18, 18]$, $\Delta = 2^{16}$, the least amount of communication and training time is required. In addition, this only works well when used together with BE. When training the network M on the PTB-XL dataset, this set of parameters produces even better test accuracy compared to $\mathcal{P} = 4096$. However, this result may be because the network M is small. The test accuracy on the plaintext version is 67.68%, hence the noises produced by the HE algorithm do not yet

TABLE I: Training and testing results on the PTB-XL dataset. Training duration and communication are reported per epoch.

Network	Type of Network	HE Parameters				Training duration (s)	Test accuracy (%)	Communication (Tb)
		BE	$\mathcal{P}$	$\mathcal{C}$	Δ			
M	Local					10.56	67.68	0
	Split (plaintext)					15.55	67.68	316.9e-6
	Split (HE)	True	8192	[40,21,21,40]	2^{21}	72 534	65.42	115.64
			4096	[40,20,20]	2^{21}	24 061	64.22	18.20
			4096	[40,20,40]	2^{20}	22 570	65.23	18.77
			2048	[18,18,18]	2^{16}	7 605	65.33	1.93

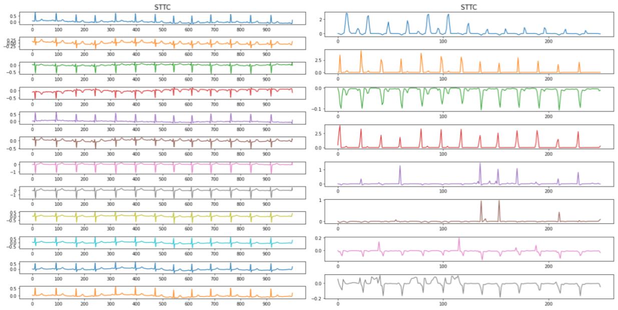

Fig. 8: Visual invertibility of the model M on the PTB-XL dataset. Left: input data (STTC class). Right: corresponding activation maps.

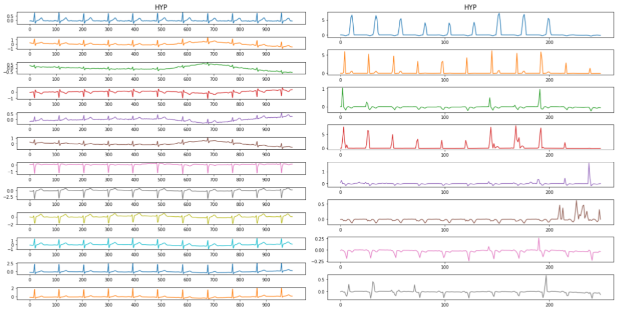

Fig. 6: Visual invertibility of the model M on the PTB-XL dataset. Left: input data (HYP class). Right: corresponding activation maps.

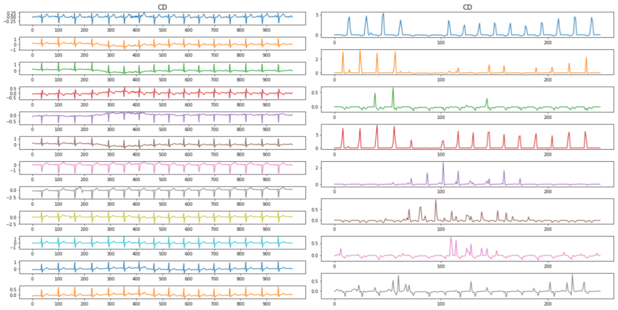

Fig. 7: Visual invertibility of the model M on the PTB-XL dataset. Left: input data (CD class). Right: corresponding activation maps.

have a significant role in reducing the model's accuracy.

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, the source code used for the evaluations is publicly available online[2].

Acknowledgment

This work was funded by the HARPOCRATES EU research project (No. 101069535) and the Technology Innovation Institute (TII), UAE, for the project ARROWSMITH.

VI. Conclusion

In this paper, we focused on training ML models in a privacy-preserving way. We used the concept of SL in combination with HE and constructed protocols allowing a client to train a model in collaboration with a server without sharing valuable information about the raw data. To the best of our knowledge, this is the first work that uses SL on encrypted data. Our experiments show that our approach has achieved high accuracy, especially when compared with less secure approaches that combine SL with differential privacy.

References

[1] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[2] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," in *AI for social good workshop*, ICLR, 2019.

[3] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "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.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "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.

[5] P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F. I. Lunze, W. Samek, and T. Schaeffter, "Ptb-xl, a large publicly available electrocardiography dataset," *Scientific data*, vol. 7, no. 1, pp. 1–15, 2020.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] D. Li, J. Zhang, Q. Zhang, and X. Wei, "Classification of ecg signals based on 1d convolution neural network," in *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pp. 1–6, IEEE, 2017.

[8] P. Paillier, "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.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[10] T. Khan, A. Bakas, and A. Michalas, "Blind faith: Privacy-preserving machine learning using function approximation," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, IEEE, 2021.

[11] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[12] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," in *AI for Social Good Workshop*, ICLR, 2019.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, ICLR, 2015.

[14] T. Khan, K. Nguyen, and A. Michalas, "Split ways: Privacy-preserving training of encrypted data using split learning," in *Proceedings of the Workshops of the EDBT/ICDT 2023 Joint Conference, Ioannina, Greece, March, 28, 2023* (G. Fletcher and V. Kantere, eds.), vol. 3379 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

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