Can We Use Split Learning on 1D CNN Models for Privacy Preserving Training?

Sharif Abuadbba
sharif.abuadbba@data61.csiro.au
Data61, CSIRO, Australia
Cyber Security CRC

Kyuyeon Kim
kyuyeon.kim@data61.csiro.au
Data61, CSIRO, Australia
Sungkyunkwan University, South Korea

Minki Kim
minki.kim@data61.csiro.au
Data61, CSIRO, Australia
Sungkyunkwan University, South Korea

Chandra Thapa
chandra.thapa@data61.csiro.au
Data61, CSIRO, Australia

Seyit A. Camtepe
seyit.camtepe@data61.csiro.au
Data61, CSIRO, Australia

Yansong Gao
garrison.gao@data61.csiro.au
Data61, CSIRO, Australia
Cyber Security CRC

Hyoungshick Kim
hyoung.kim@data61.csiro.au
Data61, CSIRO, Australia
Sungkyunkwan University, South Korea

Surya Nepal
surya.nepal@data61.csiro.au
Data61, CSIRO, Australia
Cyber Security CRC

ABSTRACT

A new collaborative learning, called *split learning*, was recently introduced, aiming to protect user data privacy without revealing raw input data to a server. It collaboratively runs a deep neural network model where the model is split into two parts, one for the client and the other for the server. Therefore, the server has no direct access to raw data processed at the client. Until now, the split learning is believed to be a promising approach to protect the client's raw data; for example, the client's data was protected in healthcare image applications using 2D convolutional neural network (CNN) models. However, it is still unclear whether the split learning can be applied to other deep learning models, in particular, 1D CNN.

In this paper, we examine whether split learning can be used to perform privacy-preserving training for 1D CNN models. To answer this, we first design and implement an 1D CNN model under split learning and validate its efficacy in detecting heart abnormalities using medical ECG data. We observed that the 1D CNN model under split learning can achieve the same accuracy of 98.9% like the original (non-split) model. However, our evaluation demonstrates that split learning may fail to protect the raw data privacy on 1D CNN models. To address the observed privacy leakage in split learning, we adopt two privacy leakage mitigation techniques: 1) adding more hidden layers to the client side and 2) applying differential privacy. Although those mitigation techniques are helpful in reducing privacy leakage, they have a significant impact on model accuracy. Hence, based on those results, we conclude that split learning alone would not be sufficient to maintain the confidentiality of raw sequential data in 1D CNN models.

KEYWORDS

split learning, neural networks, privacy leakage, 1D CNN

ACM Reference Format:
Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. 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 (ASIA CCS '20), June 1–5, 2020, Taipei, Taiwan.* ACM ASIACCS, Taipei, Taiwan, 13 pages. https://doi.org/10.1145/3320269.3384740

1 INTRODUCTION

Deep learning has been successfully applied to many applications, including genomics [22] and healthcare systems [27]. In such health applications, those models monitor patients' status effectively and detect their disease earlier. To achieve high accuracy of deep learning models, they need to be trained with sufficient data collected from a wide range of institutions [10]. However, sharing raw data, especially in health applications, may raise privacy concerns that violate certain rules such as reusing the data indiscriminately and risk-agnostic data processing [26] required by General Data Protection Regulation (GDPR) [5] and HIPAA [2].

In 2018, Otkrist et al. [10] introduced a new collaborative learning technique, called *split learning*, to protect user privacy by allowing training without sharing users' raw data to the server that runs a deep neural network (DNN) model [30, 31]. Generally, split learning divides the DNN layers into two parts (A and B) between client and server. The client, who owns the raw data, trains part A that consists of the first few layers using forward propagation and only sends their activation outputs from the split layer (the last layer of the part A) to the server. After receiving the activation outputs from the

ASIA CCS '20, June 1–5, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-6750-9/20/06…$15.00
https://doi.org/10.1145/3320269.3384740

client, the server performs the forward training with those outputs on part B. Next, the server runs the backward propagation on part B and only sends back the gradients of the activation outputs of the split layer (first layer of part B) to the client to complete the backward propagation on part A. This process continues until the model is converged.

Goals of split learning are: 1) the raw data is no longer required to be shared with the server, 2) the model classification accuracy is comparable to the non-split model [10], and 3) reducing the computational overhead of the client who only needs to run a few layers rather than the whole model. To date, the effectiveness of split learning has been validated in vision domains such as the medical image classification problem via a 2D convolution neural network (CNN) [31]. However, health data includes not only images but also sequential/time-series data such as ECG signals.

As a first study towards exploring the feasibility of split learning to deal with sequential data, we adopt an 1D CNN model for detecting heart abnormalities using ECG signals that are collected from electrodes attached to human skin [8] as a case study. Recently, several 1D CNN models were introduced to classify sequential data, including biomedical ECG signals [13, 14, 17]. Considering the fact that the exposure of raw ECG data would raise privacy concerns because ECG signals can reveal people's disease status and also be used to identify people uniquely [24], it is crucial to protect the privacy of raw data for 1D CNN models. Split learning would be a promising candidate to fulfill this privacy requirement.

This work is dedicated to investigating the answers to the following two **research questions (RQs):**

RQ 1: Can split learning be applied to deal with sequential or time-series data in particularly using 1D CNN to achieve comparable model accuracy as that of trained on centralized data?

To answer RQ 1, we first investigate the applicability of split learning to 1D CNN models to deal with sequential data. To the best of our knowledge, this is the first elaborated study on split learning using 1D CNN models, where we confirm that split learning is applicable to sequential data.

RQ 2: Can split learning be used to protect privacy in sequential data trained using 1D CNN?

Then, we focus on understanding the privacy leakage of split learning to answer RQ 2. We find that the impact of split learning was rather limited to reduce privacy leakage.

Correspondingly, we have made the following contributions:

- We implement * split learning on 1D CNN model and apply it for time-series sequential data exemplified by using medical ECG signals to detect heart abnormalities, where sharing medical data with other party is inherently avoided but can still achieve the same accuracy of the non-split model.
- We propose a privacy assessment framework for CNN models employing split learning, with three metrics: visual invertibility, distance correlation [33], and Dynamic Time Warping (DTW) [25]. This is to answer RQ 2. We observed that direct application of split learning into 1D CNN has a high privacy leakage in the applications with sensitive data such as ECG signals.
- To address the shortcoming of direct application of split learning into 1D CNN, we apply two countermeasures: i) increasing the number of layers of a CNN model split at the client and ii) exploiting differential privacy. The results suggest that although these techniques seem helpful to reduce privacy leakage, they have a significant impact on the accuracy of the model.

The rest of the paper is organized as follows: Section 2 provides background information about CNN, split learning, and privacy issues in using deep neural network models on cloud services. The design and implementation of split learning on 1D CNN are detailed in Section 3. Section 4 analyzes the privacy leakage question on 1D CNN models under split learning based on our identified threat model. Section 5 discusses the possibility of two mitigation techniques and evaluates them. Section 6 discusses our findings and future work. Section 7 presents the related work, followed by the conclusion in Section 8.

2 BACKGROUND

This section provides the necessary information to understand our work. It includes an 1D convolution neural network, split learning technique, and privacy issues in the field of machine learning.

2.1 1D Convolution Neural Network (CNN)

A CNN is a part of broader machine learning methods based on artificial neural networks where input feature extraction is performed automatically [23]. A CNN can be depicted as a mapping function $f_\Theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps an input $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$ based on the calculated parameters Θ. For example, let us assume x is an ECG sample taken from a patient that has to be classified by f_Θ into y as a vector of probabilities corresponding to $C \in [1\,..\,5]$ classes, where $[1\,..\,5]$ represents five different heart diseases. The output with the highest probability $\arg\max_{i\in\{1,\ldots,5\}} y_i$ is pointing to the ECG prediction label C_i, e.g., 'A' (atrial premature contraction).

A CNN is constructed with L hidden layers. Each layer l, $l \in \{1, \ldots, L\}$, has n_l neurons, and activation vector $a^{(l)}$. The vector consists of values of each neuron of that layer, and it is computed in a feed-forward propagation manner as follows:

$$a^{(l)} = \varphi(w^{(l)}a^{l-1} + b^{(l)}) \quad \forall\, l \in \{1, \ldots, L\}, \tag{1}$$

where $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a non-linear function that ensures only crucial neurons to be fired (i.e., > 0) and forwards its output as an input to the next layer. $w^{(l)} \in \mathbb{R}^{n_{l-1}\times n_l}$ is the *weights* and $b^{(l)} \in \mathbb{R}^n_l$ is the *biases*; both of them are learned during training. The CNN output of the L layer, i.e., the last hidden layer, is a function which can be calculated as $y = \vartheta(w^{(L+1)}a^{(L)} + b^{(L+1)})$, where $\vartheta : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

After performing forward propagation reaching the output y_i, the difference between the ground truth label C_i and predicted y_i is calculated as the loss $E_i = C_i - y_i$. Then, the contribution of every w towards this loss is calculated in a backpropagation way. For that, the partial derivatives of the loss with respect to each individual weight is calculated. As an example, we present the calculation

* https://github.com/SharifAbuadbba/split-learning-1D

with respect to a single weight $w_{12}^{(L)}$ that connects node 2 in layer $L-1$ to node 1 in Layer L as follows:

$$\frac{\partial E}{\partial w_{12}^{(L)}} = (\frac{\partial E}{\partial a_1^{(L)}})(\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}})(\frac{\partial z_1^{(L)}}{\partial w_{12}^{(L)}}), \tag{2}$$

$$= 2(a_1^{(L)} - y_l)(g^{(L)}(z_1^{(L)}))(a_2^{(L-1)}), \tag{3}$$

where g is the activation output of a neuron in layer L and z is the input for that neuron.

The widely used CNN models have 1 or 2 dimensions. The 2D CNN is popular in image recognition to extract features from 2D images. The 1D CNN also recently produced several promising results in extracting features from sequential time-series data [17]. Both types are using similar steps as in Equation 1, 2 and 3, but the major difference is the structure of the input data and how the filter, also known as a convolution kernel or feature detector, moves across the data for feature extraction. The kernel size is a *vector scalar* in 1D CNN and *2D matrix* in 2D CNN, as shown in Fig. 1.

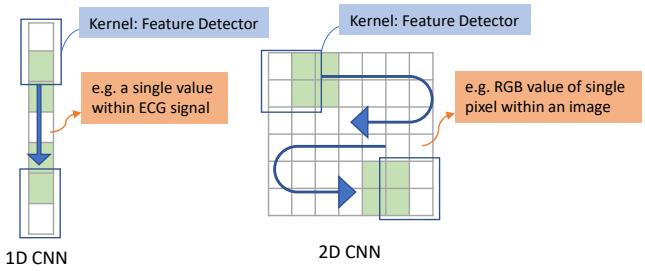

Figure 1: 1D CNN vs. 2D CNN in feature detection. The kernel (feature filter) is a *vector scalar* in 1D CNN while a *2D matrix* in 2D CNN.

2.2 Split Learning

Split learning is a distributed deep learning technique that splits a CNN into two parts; the first part is provided to the client and the second part to the server. Both client and server collaboratively train the split model without *accessing each other's part*. To perform the split learning for 2D CNN models, several networks such as LeNet, VGG, and AlexNet were considered and validated [10].

2.3 Privacy Preserving Machine Learning

Cloud servers or major model providers have been popularly used for collecting and processing data. However, users often have privacy concerns when their sensitive data is processed and stored at cloud servers [21]. In practice, user data on the cloud server can be misused to identify individuals even though their explicit identifier information is not provided. For example, a previous study [24] showed ECG signals can be used to uniquely identify individuals. Perhaps, identification of individuals may violate important privacy rules such as reusing the data indiscriminately, and risk-agnostic data processing [26] required by regulations such as GDPR in Europe [5].

In this regard, the privacy-preserving machine learning technique through distributed learning, such as federated learning [16] and split learning [30], is promising. Split learning as the focus of this work aims to reduce the privacy leakage of sensitive localized data by splitting the network during training—allowing raw data being remained in the data owner (i.e., client). However, there is a possibility of privacy leakage from the information sent from the client during the machine learning process. Precisely, the privacy leakage is that given the activation vector of the cut layer l, i.e., $a^{(l)}$, how much one (the server) infer about the training data X.

3 DESIGN AND IMPLEMENTATION OF THE SPLIT 1D CNN

In this section, we design and implement the split 1D CNN to answer the RQ 1: ***Can split learning be applied to deal with sequential/time-series data in particularly using 1D CNN to achieve comparable model accuracy as that of trained on centralized data?***

We first introduce the 1D CNN ECG classification models [14, 17] that we reproduced. We then detail our implementation of splitting the 1D CNN model. Consequently, we validate that the split 1D CNN is able to achieve the same model accuracy of the non-split 1D CNN, where our RQ 1 is answered.

3.1 1D CNN for ECG Signal Classification

In this section, we describe the non-split version of 1D CNN ECG classification models, which were recently introduced to classify ECG signals [14, 15, 17, 34, 35]. We chose two 1D CNN model architectures given in [14] and [17] as they are most recent and showed the best-achieved accuracy. Both works [14, 17] aim to classify ECG signals into five classes with less than or equal to 5 layers of 1D CNN. For the model architecture in [14], it has three 1D CNN layers and two fully connected layers, exhibiting about 96.6% accuracy. In [17], only two 1D CNN layers are used with two fully connected layers, demonstrating about 97.5% accuracy. We first implement these original non-split model from those two studies and then implement them using split learning to validate consistency in the model accuracy.

3.1.1 ***ECG Dataset and Preprocessing.*** We use MIT-BIH arrhythmia [20] which is a popular dataset for ECG signal classification or arrhythmia diagnosis detection models. Arrhythmia is short for Abnormal Heart Rythm, which is an indication of various heart diseases. Following the models [14, 17], we collect 26,490 samples in total which represents 5 heartbeat types as classification targets: N (normal beat), L (left bundle branch block), R (right bundle branch block), A (atrial premature contraction), V (ventricular premature contraction). We preprocess these samples to remove the noise before feeding them to the 1D CNN as shown in Fig. 2. We explain the detailed preprocessing steps in Appendix A.

3.1.2 ***1D CNN Model Architecture.*** The first 1D CNN [17] model architecture we adopted is illustrated in Fig. 3 (a). Specifically, each convolutional layer has 16 filters: the size of the filter used for the first convolutional layer is 7, and the rest is 5. Zero padding is applied. Leaky ReLU rather than ReLU as an activation function is chosen to prevent the dying ReLU problem. Softmax is used for the activation function of the last fully connected layer. We call this model 'Two-layer Model'. The second 1D CNN model adopted

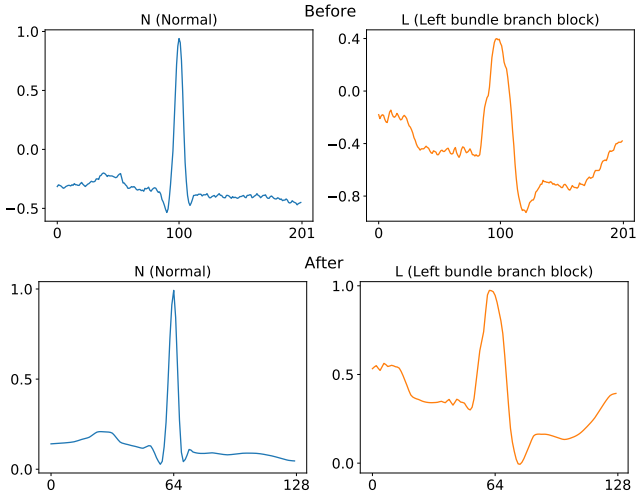

Figure 2: ECG signals before and after preprocessing.

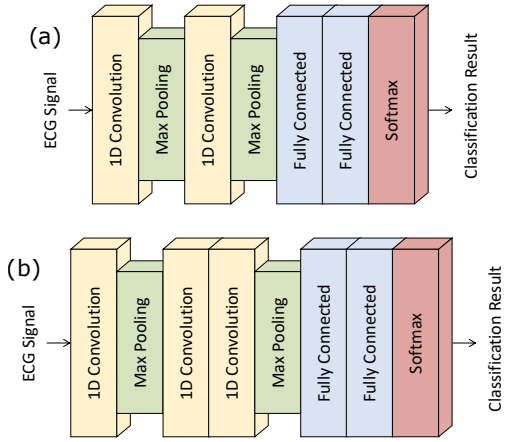

Figure 3: ECG signal classification with (a) two and (b) three 1D CNN layers.

by us [14] is with three 1D CNN layers (see Fig. 3 (b)), termed as 'Three-layer Model'. Parameter settings are similar to the 'Two-layer Model'.

Table 1: ECG dataset specifications.

Type	N	L	R	A	V	Total
Train	3,000	3,000	3,000	1,245	3,000	13,245
Test	3,000	3,000	3,000	1,245	3,000	13,245
Total	6,000	6,000	6,000	2,490	6,000	26,490

3.1.3 ***Training Result.*** Table 1 shows our training and testing dataset distribution which follows a similar proportion setting for each of 5 classes as in the previous work [17]. Both 'Two-layer Model' and 'Three-layer Model' are trained with 400 epochs, respectively. The learning rate is set to 0.001. Adam optimizer with a batch size of 32 is set.

We measure the accuracy of the models on the test set, after each epoch. As shown in Fig. 4 (non-split), both models' accuracy converged to 98% around or less than 200 epochs. The accuracy is not necessarily improved after reaching the optimal value. The model test accuracy of two 1D CNN layers is 98.9%, which is also similar to the model test accuracy of three 1D CNN layers. Note that the original work by Dan et al. [17] that we follow shows a 97.5% accuracy in a similar setting. In other words, we have successfully reproduced their work and slightly improved the accuracy, where the improvement attributes to the denoising step of the beats we added before training, as detailed in Appendix A.

3.2 Splitting 1D CNN

We now split the above 1D CNN models. We focus on two parties in the split learning setting: the client and the server. As we show later, the leakage is resulted from passing the forward activation to the server, which is irrelevant to the number of participated clients.

The 1D CNN model is split into two parts, A and B as shown in Fig. 5. Activation and Gradients are passed between client and server to collaboratively train the joint model.

We follow the vertical split method [10, 30] to split 1D CNN. Those previous studies provide the conceptual process of split learning for 2D CNN models and show that model accuracy of split training is the same as that in the non-split training [10, 30]. Other than focusing on 2D CNN models as [10, 30], we, herein, elaborate on 1D CNN split implementation strategies for client and server sides, respectively (see Algorithm 1 and 2). Those detailed algorithms can be useful as a guideline for implementing split learning models. We also clearly specify what information is exchanged between client and server with the socket instructions in the pseudocodes in Algorithm 1 and 2, which will be especially helpful for practitioners along with our source code.

3.2.1 ***Client.*** Assume a model that has L layers—input layer is excluded—in total. The L-th layer is the output layer, and the remaining layers are hidden layers. Suppose that the model is split between layer l and layer $l+1$. The client holds first l layers from the layer 1 to l—part A, whereas the server holds remaining layers from the layer $l+1$ to L—part B. Weights in the layer i are denoted as $w^{(i)}$. In addition, let $f^{(i)}$ denotes the forward propagation over the i-th layer, and $z^{(i)}$ denotes the output tensor just after the forward propagation in i-th layer. $a^{(i)}$ is denoted as the output after the activation function in layer i, which can be given by $a^{(i)} = g(z^{(i)})$, where g is the activation function. In backpropagation, $f_{\mathrm{T}}^{(i)}$ denotes the function which returns the gradient of activation in $(i-1)$-th layer, using weights and gradient of i-th layer. Finally, when the client has the ECG raw dataset $\mathbb{D}$ to train with, the split learning process on client follows Algorithm 1.

The client first connects to the server via socket and synchronizes some train configurations. With a single batch given from $\mathbb{D}$, the client forward propagates it until the l-th layer and sends the activation $a^{(l)}$ from the l-th layer to the server. When the client

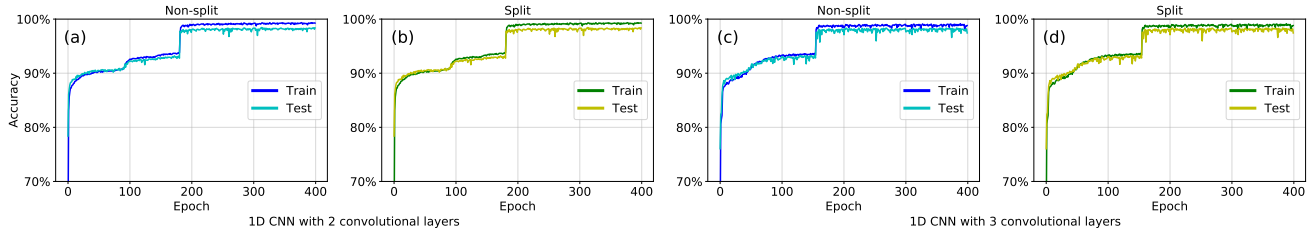

Figure 4: Accuracy over the training of split and non-split 1D CNN models. Notably, we have used the exact same initial parameter randomization, through the same seed, for both split and non-split tests in this example.

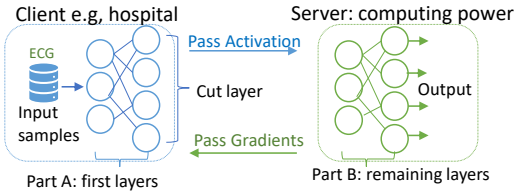

Figure 5: Split learning overview.

Algorithm 1: Split learning on the client side

Initialization:
- $s \leftarrow$ socket initialized with port and address
- $s.connect(Bob)$
- $\phi, \eta, o, n, N \leftarrow s.synchronize()$
- $\{w^{(i)}\}_{\forall i \in \{0,\dots,l\}} \leftarrow$ initialize using ϕ
- $\{z^{(i)}\}_{\forall i \in \{0,\dots,l\}}, \{a^{(i)}\}_{\forall i \in \{0,\dots,l\}} \leftarrow \emptyset$
- $\{\frac{\partial E}{\partial z^{(i)}}\}_{\forall i \in \{0,\dots,l\}}, \{\frac{\partial E}{\partial a^{(i)}}\}_{\forall i \in \{0,\dots,l\}} \leftarrow \emptyset$

for each batch (x, y) generated from $\mathbb{D}$ **do**
- **Forward propagation:**
  - $a^{(0)} \leftarrow x$
  - **for** $i \leftarrow 1$ **to** l **do**
    - $z^{(i)} \leftarrow f^{(i)}(a^{(i-1)})$
    - $a^{(i)} \leftarrow g(z^{(i)})$
  - $s.send((a^{(l)}, y))$
- **Backward propagation:**
  - $\frac{\partial E}{\partial a^{(l)}} \leftarrow s.receive()$
  - **for** $i \leftarrow l$ **downto** 1 **do**
    - $\frac{\partial E}{\partial z^{(i)}} \leftarrow \frac{\partial E}{\partial a^{(i)}} \times g'(z^{(i)})$
    - Compute $\frac{\partial E}{\partial w^{(i)}}$ using $\frac{\partial E}{\partial z^{(i)}}$ and $a^{(i-1)}$
    - Update $w^{(i)}$ using η, $\frac{\partial E}{\partial w^{(i)}}$, and o
    - **if** $i \neq 1$ **then**
      - $\frac{\partial E}{\partial a^{(i-1)}} \leftarrow f_{\mathrm{T}}^{(i)}(\frac{\partial E}{\partial z^{(i)}})$

$s.close()$

receives the gradient of $\frac{\partial E}{\partial a^{(l)}}$ from the server, the backpropagation continues to the first hidden layer.

3.2.2 ***Server.*** The server continues forward propagation after receiving activation from l-th layer. The server then calculates the loss between the activated output from the last layer and the label passed from the client. Let E denotes the loss value calculated from the cost function J. With E, the server starts backpropagation until $(l+1)$-th layer. The server finally sends the gradient of the output to the client, which is $\frac{\partial E}{\partial a^{(l)}}$, to make the client continues the backpropagation. The rest of the denotations except E and J, are the same as used in Algorithm 1.

The training flow between the client and the server is illustrated in Fig. 6 and further detailed in Appendix B.

Algorithm 2: Split learning on the server side

Initialization:
- $s \leftarrow$ server socket initialized with port and address
- $s_A \leftarrow s.accept(Alice)$
- $\phi, \eta, o, n, N \leftarrow s_A.synchronize()$
- $\{w^{(i)}\}_{\forall i \in \{l+1..L\}} \leftarrow$ initialize using ϕ
- $\{z^{(i)}\}_{\forall i \in \{l+1..L\}}, \{a^{(i)}\}_{\forall i \in \{l..L\}} \leftarrow \emptyset$
- $\{\frac{\partial E}{\partial z^{(i)}}\}_{\forall i \in \{l+1..L\}}, \{\frac{\partial E}{\partial a^{(i)}}\}_{\forall i \in \{l..L\}} \leftarrow \emptyset$

for $i \leftarrow 1$ **to** N **do**
- **Forward propagation:**
  - $(a^{(l)}, y) \leftarrow s_A.receive()$
  - **for** $i \leftarrow l+1$ **to** L **do**
    - $z^{(i)} \leftarrow f^{(i)}(a^{(i-1)})$
    - $a^{(i)} \leftarrow g(z^{(i)})$
  - $E \leftarrow J(a^{(L)}, y)$
- **Backward propagation:**
  - Compute $\frac{\partial E}{\partial a^{(L)}}$
  - **for** $i \leftarrow L$ **downto** $l+1$ **do**
    - $\frac{\partial E}{\partial z^{(i)}} \leftarrow \frac{\partial E}{\partial a^{(i)}} \times g'(z^{(i)})$
    - Compute $\frac{\partial E}{\partial w^{(i)}}$, using $\frac{\partial E}{\partial z^{(i)}}$ and $a^{(i-1)}$
    - Update $w^{(i)}$, using η, $\frac{\partial E}{\partial w^{(i)}}$, and o
    - $\frac{\partial E}{\partial a^{(i-1)}} \leftarrow f_{\mathrm{T}}^{(i)}(\frac{\partial E}{\partial z^{(i)}})$
  - $s_A.send(\frac{\partial E}{\partial a^{(i)}})$

$s_A.close()$

3.2.3 ***Influence on Performance.*** Based on the above split implementation, we test the split learning on both 1D CNN model architectures: Two-layer Model and Three-layer Model. We split the former after two layers and latter after three layers. To measure and

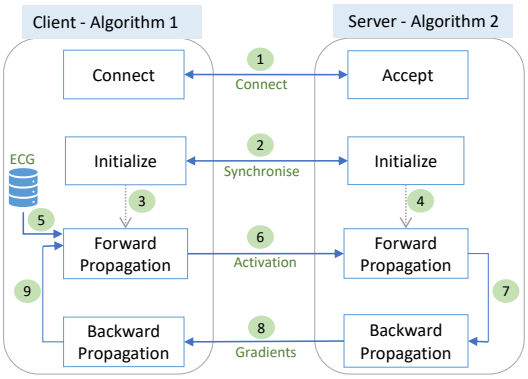

Figure 6: Data flow between the client and the server in Algorithm 1 and 2.

compare the influence of split learning on performance accurately, we initialize both models before and after the split with the same set of weights. We have repeated the training processes for up to ten(10) times. Fig. 4 depicts the exact accuracy conversion of both models before/after applying split. It is clear that the accuracy is the same. In other words, our split learning implementation has no noticeable impact on the accuracy of the models.

Summary: Split learning can be applied into 1D CNN without the model classification accuracy degradation as demonstrated in Fig. 4. Therefore, RQ 1 can be is answered affirmatively.

4 PRIVACY LEAKAGE ANALYSIS

In this section, we propose a privacy assessment framework under our identified threat model to answer the RQ 2: ***Can split learning be used to protect privacy in sequential/time-series data trained using 1D CNN?***

This framework has three metrics: visual invertibility, distance correlation and Dynamic Time Warping. Based on these metrics, we present our empirical results validated from systematic experiments to demonstrate that it is possible to reconstruct raw data from the activation of the intermediate split layer, which indicates that our RQ 2 is answered unfavourably.

We firstly elaborate on the considered threat model.

4.1 Threat Model

We consider the server is an honest-but-curious single entity adversary. It performs all its operations as specified, but curious about the raw data localized at the client. We assume that the server has no access to the client's device, and it does not target attacks on those devices. Furthermore, the server does not collude with any client. The server's goal is to reconstruct the raw data (e.g., the client's medical data) from the activation vector of the cut layer, which is delivered from the client during the forward propagation. We assume that all participating clients are trusted, and they participate in the learning process provided that the raw data always remain within their custody. In the health domain, examples of trusted clients are patients, hospitals, and (health) research organizations.

4.2 Visual Invertibility

We first perform visual invertibility on the intermediate split layer activation as an initial assessment to observe the possibility of reconstructing the original ECG signals. The model batch size is 32 (i.e., the number of ECG samples fed to the model) and uses 16 filters to extract features. These filters produce the layer activation, which is passed to the server from any split layer. Fig. 7 shows five original ECG samples on the top row vs the reconstructed version from 5 corresponding filters activation after two layers on the bottom row. We can observe that there is high similarity, which means significant leakage.

Our framework goes further to generalise this observation by measuring the correlations between the split layer activation and original samples. To quantify our results, we employ two other metrics: distance correlation and Dynamic Time Warping as explained below.

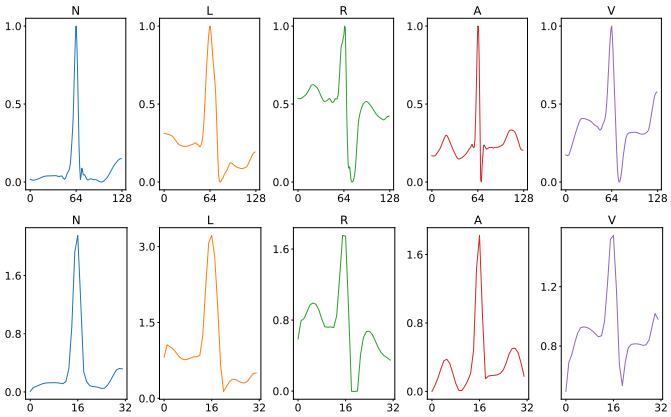

Figure 7: Visual invertibility (top: raw data, bottom: the output of two convolutional layers) where the length of the raw input is 128 while the length of the activated outputs after two convolutional layers is 32 because features are max pooled twice.

4.3 Distance Correlation

In statistics, distance correlation is a measure of dependence between two paired vectors of arbitrary, not necessarily equal, dimensions. It is on the scale of 0-to-1. Correlation of 0 refers to independent vectors, whereas value 1 means highly dependent and fully similar. Distance correlation has already been used in the context of deep learning to measure the autoencoder correlation [33] and reconstruction of raw data from intermediate layers of 2D CNN [29]. Therefore, we apply distance correlation as a measure to monitor the dependency between each split layer filter activation and corresponding raw 1D ECG signal.

The distance correlation of two random variables is assessed by dividing their distance covariance by the product of their distance standard deviations, given below 4 [9].

$$\mathrm{dCor}(X, Y) = \frac{\mathrm{dCov}(X, Y)}{\sqrt{\mathrm{dVar}(X)\ \mathrm{dVar}(Y)}} \tag{4}$$

Fig. 8 (a) shows the mean of distance correlation intermediate split layer between all 16 filters activation and corresponding raw data. We repeat these experiments after two and three convolutional layers, respectively. Splitting after two layers, the results suggest that the correlation of the top filters activation is very high (0.89), which indicates high leakage and can be exploited to reconstruct the raw data, as shown previously in visual invertibility Fig 7. Similarly, the filters activation after three layers exhibits high dependency, as well. However, the correlation of the top filters reduced by (0.03); from (0.89) to (0.86) between splitting after 2 and 3 layers. This gives us intuition to increase the number of layers as a mitigation strategy investigated in the next section.

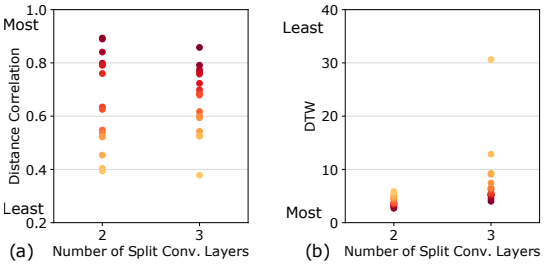

Figure 8: Distance correlation and DTW between the raw input and the activated outputs after two and three 1D convolutional split layers, respectively.

4.4 Dynamic Time Warping (DTW)

To generalise our observations, we utilise another well-known similarity measurement in time series analysis called Dynamic Time Warping (DTW) [25]. DTW is an algorithm that can accurately measure the similarity between two temporal sequences, which may vary in speed. It is widely used in speech recognition and signature recognition.

Given two time series $X = (x_1, x_2, ...x_N),\ N \in \mathbb{N}$ and $Y = (y_1, y_2, ...y_M),\ M \in \mathbb{N}$ represented by a sequence of values. DTW algorithm starts by constructing the distance matrix $C \in \mathbb{R}^{N \times M}$ representing all pairwise distances between X and Y. This distance matrix called local cost matrix for the alignment of two sequences X and Y and calculated as follows

$$C_l \in \mathbb{R}^{N \times M} : c_{i,j} = \left\| x_i - y_j \right\|,\ i \in \{1 \,..\, N\},\ j \in \{1 \,..\, M\} \quad (5)$$

Once the local cost matrix constructed, the algorithm explores the alignment path (or warping path), which runs through the low-cost areas. The warping path, which has the lowest cost associated with alignment called the optimal warping path. Zero(0) warping path refers to high similarity, whereas increasing warping path toward, e.g., 1000 means very dissimilar. In this paper, we apply DTW as a measure to monitor the similarity between each split layer filter activation and corresponding raw 1D ECG signal.

Fig. 8 (b) shows the mean of DTW between the intermediate split layer of all 16 filters activation and corresponding raw data. We repeat these experiments after two and three convolutional layers, respectively. Splitting after two layers, DTW indicates that the similarity of the top filters activation is very high and close to zero (2.7), which exhibits high leakage and can be exploited to reconstruct the raw data as visually shown previously in Fig 7. Similarly, the filters activation after three layers shows high similarity as well; Again, the dissimilarity of the top filters increases by (0.21); from (2.70) to (2.98) between splitting after 2 and 3 layers. This suggests again that increasing the number of layers may reduce the leakage.

Summary: Our framework leakage analysis to test our RQ 2 via three empirical (visual invertibility) and numerical metrics (distance correlation and DTW) indicate that the filters activation after splitting the 1D CNN models at two and three convolutional layers can be used to reconstruct the raw data. In other words, sharing the filters activation from these layers may result in severe privacy leakage. Therefore, RQ 2 is answered unfavourably.

5 MITIGATE THE SHORTCOMING?

To further answer our RQ 2, we investigate a number of strategies which can be deployed in 1D CNN model to mitigate the privacy leakage. Specifically, we apply and evaluate two mitigation techniques to reduce potential privacy leakage by i) adding more hidden layers before splitting and ii) applying differential privacy on split layer activation before transmitting them to the server. We measure the efficacy of mitigation techniques via both i) privacy leakage reduction using distance correlation and DTW as well as ii) model accuracy after applying mitigation techniques.

5.1 Adding More Hidden Layers

Specifically, we add more layers—ranging from two to eight—to the client before the cut layer. In other words, the model architecture becomes more complex, given the number of layers held by the server being constant. To be consistent, we follow the original model architecture in [14] in terms of hidden layers and filter size. We utilize 16 filters for each convolutional layer. Zero paddings are applied to keep the size of output as a power of 2. Moreover, the size of the filter used for each hidden convolutional layer is 5. We select activation function as Leaky ReLU, rather than ReLU, to prevent the dying ReLU problem.

Correlation Mean: Fig. 9 (a) shows the mean of distance correlation between the intermediate split layer of all 16 filters activation and corresponding raw data after second to eighth convolutional layer. The correlation of each filter is measured against the corresponding raw data ECG and represented as a *dot* in the figure. We then sort the mean correlation in descending order on the Y-axis, where one(1) is a high risk of leakage, and zero(0) is low risk. It is clear from the top filters that there is a slight reduction in the correlation from (0.89) to (0.69) as the number of hidden convolutional layers increases; however, there is still a high correlation (above 0.5).

DTW Mean: Fig. 9 (b) further shows the mean of DTW between the intermediate split layer of all 16 filters activation and corresponding raw data after second to the eighth layer. The similarity of each filter is calculated against the corresponding raw data ECG and represented as a *dot* in the figure. We again grade the mean similarity in ascending order from zero(0) as a high risk of leakage to 600 as low risk. It is clear from the filters that there is a significant dissimilarity improvement from zero(0) to (600+) of some filters;

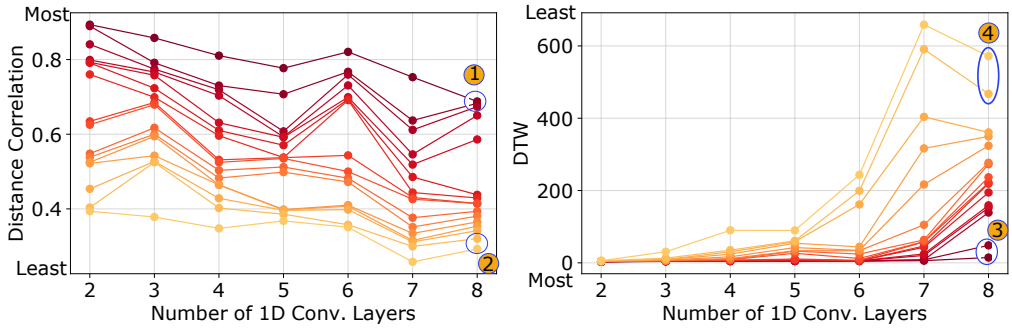

Figure 9: Mean of distance correlation and DTW calculated for each filter in split output (10,000 ECG samples), thicker color means higher correlation. Downsampled original sample from 128 (original size) to 32 (split layer output size) by average pooling to adjust the length of each sample.

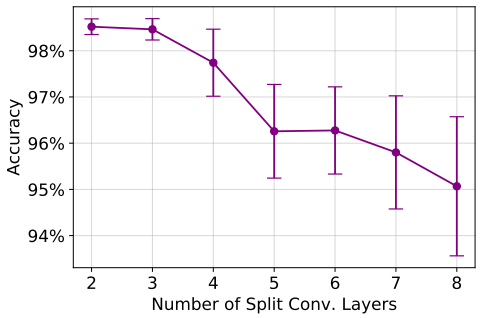

Figure 10: Mean and standard deviation of test accuracy with varying the number of convolutional layers from 2 to 8 for split learning. The experiments were performed under the same configuration 10 times to avoid bias.

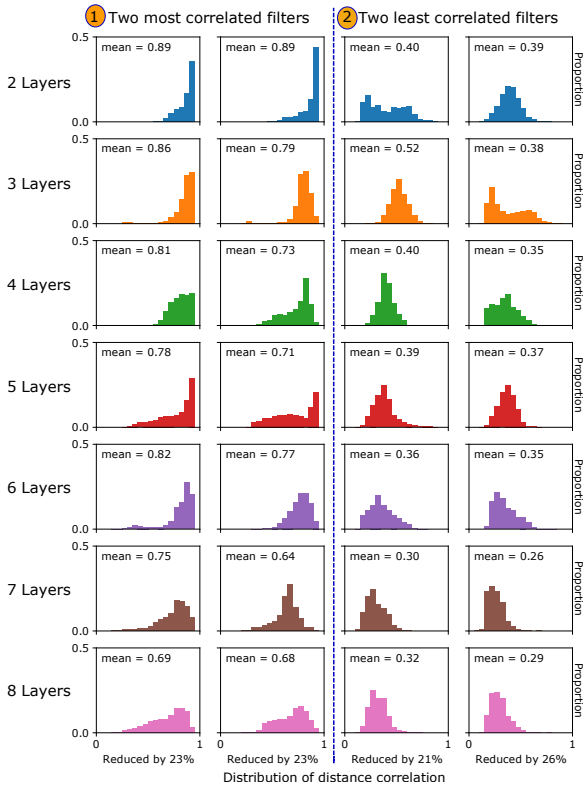

Figure 11: Effects of adding more layers on distance correlation distribution of the two most correlated filters, ① in Fig. 9, and two least correlated filters, ② in Fig. 9. Privacy mitigation by increasing the number of more layers seems effective for least correlated filters, but it is ineffective for those most correlated filters since the correlation is still high, more than 0.68, when up to 8 layers run on the client. In other words, mitigation efficiency appears to be dependent on filters.

However, many other filters are still close to zero, which indicates high leakage and can be potentially exploited to reconstruct the raw ECG data.

Distribution: We further investigate the distribution of distance correlation, as detailed in Fig. 11, which illustrates the most two correlated filters vs the least two correlated filters. The correlation distribution is continuously reduced but seems ineffective to protect the most correlated filters. Similarly, Fig. 12 presents DTW distribution of the most two correlated filters vs the least two correlated filters. The least correlated shows clear improvements by e.g., 84 times; however, DTW also emphasizes that increasing the number of layers seems ineffective with the most correlated filters i.e., improved only by 5 times.

5.2 Applying Differential Privacy on Split Layer

As the second mitigation technique, we apply differential privacy on split layer filters activation before transmitting them to the server. Differential privacy has already been widely used in deep learning to protect privacy [1]. Given input space X, output space Y, privacy parameter ϵ, and a randomisation mechanism μ. $\mu : X \longrightarrow Y$ is ϵ-differentially private if for all neighbouring inputs $X \simeq X^{'}$ and all sets of outputs $E \subseteq Y$ satisfy the following:

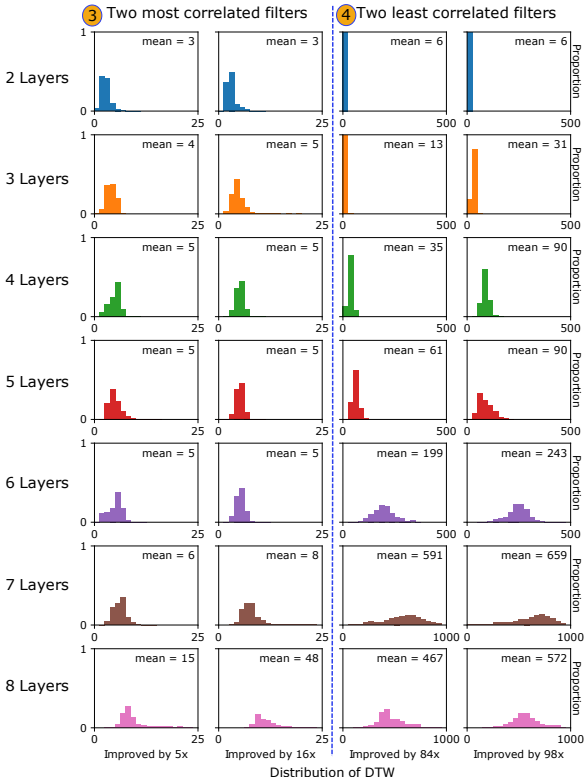

Figure 12: Effects of adding more layers on DTW distribution of the two most correlated filters, ③ in Fig. 9, and two least correlated filters, ④ in Fig. 9. Privacy mitigation by increasing the number of more layers seems effective for least correlated filters—DTW improved by e.g., 84 times, but it is ineffective for those most correlated filters since the DTW improvement is less by, e.g., 5 times, when up to 8 layers run on the client. In other words, mitigation efficiency appears to be dependent on filters.

$$\mathbb{P}\left[\mu(X) \in E\right] \leq e^{\epsilon} \times \mathbb{P}\left[\mu(X') \in E\right] \tag{6}$$

Therefore, the value of ϵ determines the strength of privacy. Specifically, we employ the Laplace differential privacy mechanism on the split layer activation, which is widely used for numerical data [12]. It adds Laplace noise from the Laplace distribution, which can be expressed by a probability density function. The level of noise relies on pre-determined ϵ on a scale of 10 (weakest privacy)-to-0 (strongest privacy where the data can not be used).

Correlation Mean: Fig. 13 (a) shows the mean of distance correlation between the intermediate split layer of all 16 filters activation after applying different ϵ levels of differential privacy and corresponding raw data. It is under the expectation that the strongest differential privacy level of, e.g., $\epsilon = 1$ can fully protect all the filters; However, it comes with the cost of degrading the classification accuracy significantly from 98.9% to only 50% (which is close to random guessing) as shown in Fig.14.

DTW Mean: Fig. 13 (b) shows the mean of DTW between the intermediate split layer of all 16 filters activation after applying various ϵ levels of differential privacy and corresponding raw data. The results also suggest that strongest differential privacy level of $\epsilon = 1$ can increase the dissimilarity between the filter activation and corresponding raw data which protects them against potential reconstruction attempts; However, strongest privacy level of $\epsilon = 1$ damages the accuracy significantly as explained earlier and makes it close to the random prediction of 50%.

Distribution: To look closely at the impact on filters after various ϵ levels, we also deep dive into the distribution of distance correlation and DTW. Fig. 15 presents the distance correlation of the most two correlated filters vs the least two correlated filters after two layers split without differential privacy. The correlation distribution is continuously improved after applying stronger ϵ (10,7,5,3,1) on activation before transmitting. Also, Fig. 16 shows DTW of the most two correlated filters vs the least two correlated filters. The DTW also improved after applying stronger ϵ noise; However, it seems less effective for most correlated filters.

Summary: None of the two applied mitigation techniques can efficiently mitigate the privacy leakage from all filters—it appears that the mitigation is dependent on the applied mitigation techniques as well as filters. On top of that, both of them come with a cost of accuracy reduction of the joint model, which appears not acceptable especially in the differential privacy case. Therefore, the RQ 2 is still hard to be answered favourably even when our mitigation attempts are applied.

6 DISCUSSION AND FUTURE WORK

Why split learning is ineffective to preserve the privacy for 1D CNN? To reduce the privacy leakage from cut layers revealed by us, we have consequently applied two specific techniques: i) increase the number of split layers at the client-side, and ii) applying differential privacy to the filters activation before transmission. However, both techniques suffer the reduction of model accuracy, especially differential privacy. For the first technique of increasing the number of split layers to the client, it eventually renders the other trade-off that is the computational overhead of the client. As a matter of fact, saving the computational overhead of the client is the other main motivation of split learning since the client only needs to train a small fraction of parameters of the whole model in comparison with the federated learning required to train the whole model [10]. Therefore, increasing the number of layers before the split layers to the client further diminishes the benefits of the split learning from the client computational overhead perspective, especially, when the split learning is mounted on resource-restrict devices/agents.

Framework to evaluate split learning models: We use three metrics to measure the privacy leakage and explore the possibility of reconstructing the ECG samples' raw data from split layer filters. These measurements are visual invertibility, distance correlation, and Dynamic Time Warping. The first measurement is to show an empirical way to reconstruct the raw data from activation. The second and third are used for quantification. We believe these three

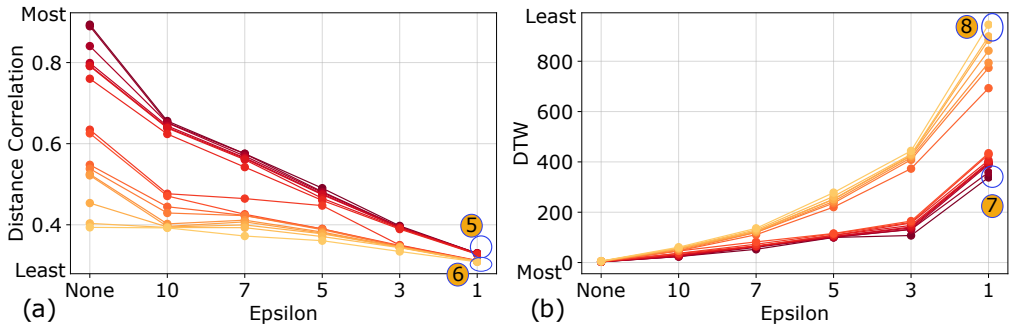

Figure 13: Mean of distance correlation and DTW after applying differential privacy with various values of epsilon.

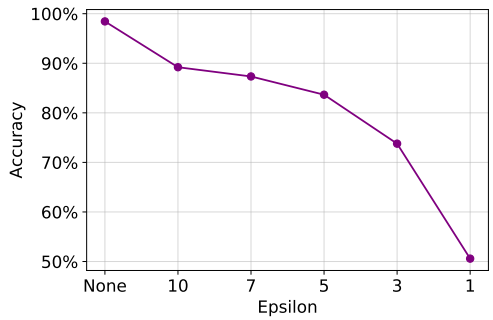

Figure 14: Accuracy changes after applying differential privacy with stronger epsilon values between 10 (Weakest) and 1 (Strongest).

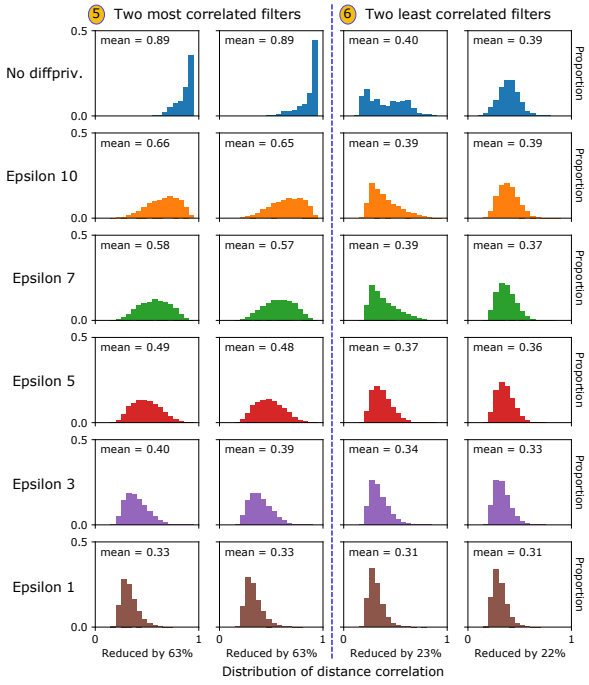

Figure 15: Effects of applying differential privacy on distance correlation distribution of the two most correlated filters, ⑤ in Fig. 13, and two least correlated filters, ⑥ in Fig. 13. In contrast to applying more hidden layers, privacy mitigation by applying differential privacy seems effective for most correlated filters—reduced by 63%, but it is ineffective for those least correlated filters, e.g., about 23%. In other words, mitigation efficiency appears to be dependent on filters as well as mitigation techniques applied.

metrics can be used as a general framework to evaluate the potential privacy leakage from split learning models.

How to develop more effective split learning for 1D CNN? This is an open question that is remaining to be answered as we do not have an affirmative answer. However, we suggest two, but not limited to, future strategies. (i) Our first mitigation attempt to increase the number of hidden layers shows that it is possible to protect many of the filters activation, but not all of them without significant accuracy degradation. One would explore leakage mitigation techniques for only a few revealing filters. (ii) Existing split learning follows a vertical split mechanism, which means all the cut layer filters activation should be shared. To reduce the exposure of all filters activation, one would explore a multiple-horizontal split and share only the protected filters activation with the server.

Can split learning be applied to LSTM and/or RNN? As a matter of fact, employing LSTM [11] and/or RNN [19] served as the first trial when we intended to investigate the practicality of dealing with sequential/time-series data. However, we realized that LSTM and RNN are sequential networks, while the current split technique is always vertically split. Therefore, we found that there is no efficient means of splitting LSTM and RNN. This eventually motivated us to adopt 1D CNN that can be vertically split as well to deal with pervasive sequential/time-series data. Whether split learning can be properly applied to LSTM and/or RNN remains to be further investigated.

Limitations: Our work has two main limitations. Firstly, we explore the leakage in 1D CNN using one sensitive health application, which is ECG biomedical signals. We experiment with the ECG heartbeat samples from the widely-used MIT-BIH dataset. Other 1D

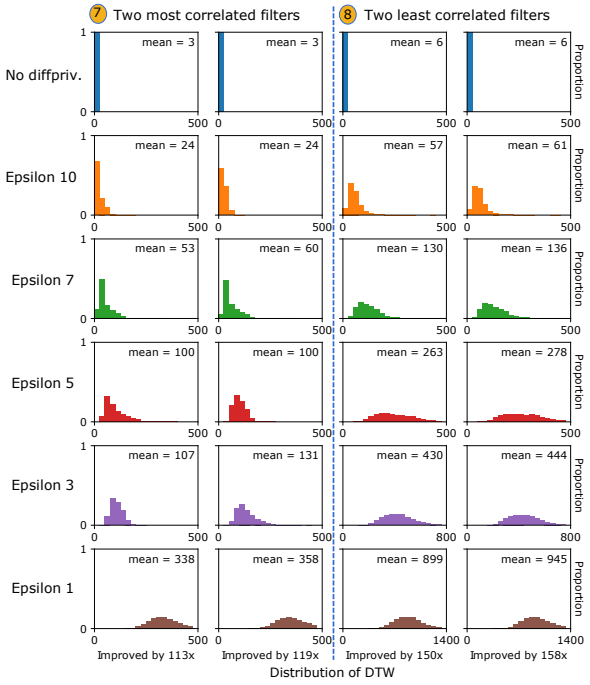

Figure 16: Effects of applying differential privacy on DTW distribution of the two most correlated filters, ⑦ in Fig. 13, and two least correlated filters, ⑧ in Fig. 13. In contrast to applying more hidden layers, privacy mitigation by applying differential privacy seems effective for least correlated filters—improved 158 times, but it is less effective for those most correlated filters, e.g., about 113 times. In other words, mitigation efficiency appears to be dependent on filters as well as mitigation techniques applied.

CNN applications and datasets remain to be investigated. Secondly, privacy leakage from the split layer of 2D CNN various applications remains to be addressed.

7 RELATED WORK

In [10], the authors envision that it should be challenging to reconstruct the data localized on the client-side when split learning is employed. The main argument, generally, is that the server can not access the weight of split layers held by the client—this is the case for federated learning. Inverting the weight is computationally infeasible in practice. While their analysis focuses on 2D CNN, this paper has shown that reconstructing the raw data in 1D CNN is more than possible.

Vepakomma et al. [29] investigated the potential privacy leakage of split learning in 2D CNN using the MNIST dataset. They also found high potential leakage from the cut layer. They showed that by slightly scaling the weights of all layers before the split, it is possible to reduce the distance correlation between the split layer activation and raw data. This scaling mechanism is effective in 2D CNN because of the large number of hidden layers before the split layer. However, existing 1D CNN models such as in [14, 17] have only 2-3 hidden layers, which renders scaling their weights ineffective and degrades the accuracy.

The other privacy leakage that has been revealed is the membership inference. For example, Melis *et al.* [18] demonstrated the membership inference attack on federated learning by observing the gradients aggregated from the model trained on clients. It is not surprising to envision such a membership inference attack could be applicable to split learning. To which extent the split learning can be resistant to inference attack leaves interesting future work.

Other than privacy concerns of distributed learning, there are also emerging security concerns inherent to the distributed learning, in particular, the backdoor attacks [6, 7, 32]. Generally, a backdoor attack occurs when the training data is tampered by a malicious party under the model training outsource scenario. The backdoor model behaves normally for clean inputs while misclassifying any input to the target class when the input is stamped with a trigger. It has been shown that federated learning is inherently vulnerable to such a backdoor attack because the server has no control over the local data by design [3, 28]. Therefore, participants can manipulate their data as they want to insert a backdoor to the joint model. We believe that split learning, for the same reason, is also inevitably vulnerable to backdoor attacks from the security perspective besides the privacy leakage revealed in this work. Thus, it is important to consider this security concern when deploying split learning in realistic security-critical applications.

8 CONCLUSION

In this paper, we explored the feasibility of split learning to deal with sensitive time-series data in particular personal ECG signals to detect heart abnormalities. We introduced the first implementation of split learning into the 1D CNN model. We proposed a privacy assessment framework for CNN models when using split learning, with three metrics: visual invertibility, distance correlation, and DTW. Based on this framework, we extensively evaluated the privacy leakage exemplified by the ECG dataset. Our results demonstrate that adopting split learning directly into 1D CNN models for time-series/sequential data would exhibit a possibility of high privacy leakage from feature values. Initial mitigation attempts via i) increasing the number of layers in a CNN model and ii) using differential privacy explicitly indicate there is a trade-off between the degree of privacy leakage reduction and joint model accuracy deterioration—substantial when applying the differential privacy. Perhaps, it would be more challenging to preserve privacy via 1D CNN models compared with 2D CNN models. This is because 1D CNN usually has much less hidden layers where more 1D CNN layers usually tend to affect the model accuracy adversely. As future work, privacy leakage through the split layer should be thoroughly evaluated for (1D and 2D) CNN models, and corresponding effective mitigation techniques need to be developed.

ACKNOWLEDGMENT

The work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme. This work was also supported in part by the ITRC support program

(IITP-2019-2015-0-00403). The authors would like to thank all the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[2] HIPAA Compliance Assistance. 2003. Summary of the HIPAA privacy rule. *Office for Civil Rights* (2003).

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459* (2018).

[4] David L Donoho and Jain M Johnstone. 1994. Ideal spatial adaptation by wavelet shrinkage. *biometrika* 81, 3 (1994), 425–455.

[5] EU GDPR. [n.d.]. General Data Protection Regulation (GDPR). https://www.eugdpr.org/. [Online; accessed 10-Nov-2019].

[6] Yansong Gao, Yeonjae Kim, Bao Gia Doan, Zhi Zhang, Gongxuan Zhang, Surya Nepal, Damith C Ranasinghe, and Hyoungshick Kim. 2019. Design and Evaluation of a Multi-Domain Trojan Detection Method on Deep Neural Networks. *arXiv preprint arXiv:1911.10312* (2019).

[7] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. 2019. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *35th Annual Computer Security Applications Conference (ACSAC)*.

[8] Tomas B Garcia and Daniel J Garcia. 2019. *Arrhythmia Recognition: The Art of Interpretations*. Jones & Bartlett Publishers.

[9] Arthur Gretton, Kenji Fukumizu, and Bharath K Sriperumbudur. 2009. Discussion of: Brownian distance covariance. *The annals of applied statistics* 3, 4 (2009), 1285–1294.

[10] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Naoise Holohan, Stefano Braghin, Pól Aonghusa, and Killian Levacher. 2019. Diffprivlib: The IBM Differential Privacy Library. *arXiv preprint arXiv:1907.02444* (2019).

[13] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 2019. 1D Convolutional Neural Networks and Applications: A Survey. *arXiv preprint arXiv:1905.03554* (2019).

[14] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. 2015. Real-time patient-specific ECG classification by 1-D convolutional neural networks. *IEEE Transactions on Biomedical Engineering* 63, 3 (2015), 664–675.

[15] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. 2017. Personalized monitoring and advance warning system for cardiac arrhythmias. *Scientific reports* 7, 1 (2017), 9270.

[16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[17] Dan Li, Jianxin Zhang, Qiang Zhang, and Xiaopeng Wei. 2017. Classification of ECG signals based on 1D convolution neural network. In *2017 IEEE 19th International Conference on e-Health Networking, Apps and Services*. IEEE, 1–6.

[18] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 691–706.

[19] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[20] George B Moody and Roger G Mark. 2001. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine* 20, 3 (2001), 45–50.

[21] European Network and Information Security Agency. 2009. An SME perspective on Cloud Computing—Survey. *Survey Report* (2009), 1–16.

[22] D. Ravì, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G. Yang. 2017. Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics* 21, 1 (Jan 2017), 4–21. https://doi.org/10.1109/JBHI.2016.2636665

[23] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[24] Abdelkader Sellami, Amine Zouaghi, and Abdelhamid Daamouche. 2017. ECG as a biometric for individual's identification. In *2017 5th International Conference on Electrical Engineering-Boumerdes (ICEE-B)*. IEEE, 1–6.

[25] Pavel Senin. 2008. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855, 1-23 (2008), 40.

[26] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. The Seven Sins of Personal-data Processing Systems Under GDPR. In *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'19)*. USENIX Association, Berkeley, CA, USA, 1–1. http://dl.acm.org/citation.cfm?id=3357034.3357036

[27] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi. 2018. Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis. *IEEE Journal of Biomedical and Health Informatics* 22, 5 (Sep. 2018), 1589–1604. https://doi.org/10.1109/JBHI.2017.2767063

[28] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can You Really Backdoor Federated Learning? *arXiv preprint arXiv:1911.07963* (2019).

[29] Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. 2019. Reducing leakage in distributed deep learning for sensitive health data. *arXiv preprint arXiv:1812.00564* (2019).

[30] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018).

[31] Praneeth Vepakomma, Tristan Swedish, Ramesh Raskar, Otkrist Gupta, and Abhimanyu Dubey. 2018. No Peek: A Survey of private distributed deep learning. *arXiv preprint arXiv:1812.03288* (2018).

[32] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks* (2019), 0.

[33] Rick Wang, Amir Karimi, and Ali Ghodsi. 2018. Distance correlation autoencoder. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[34] Yunan Wu, Feng Yang, Ying Liu, Xuefan Zha, and Shaofeng Yuan. 2018. A comparison of 1-D and 2-D deep convolutional neural networks in ECG classification. *arXiv preprint arXiv:1810.07088* (2018).

[35] Özal Yıldırım, Paweł Pławiak, Ru-San Tan, and U Rajendra Acharya. 2018. Arrhythmia detection using deep convolutional neural network with long duration ECG signals. *Computers in biology and medicine* 102 (2018), 411–420.

A APPENDIX: ECG DATASET PREPROCESSING IN DETAIL

The MIT-BIH database contains 48 records retrieved from 47 different patients. About 110K ECG signals are distributed in 48 records. Each record includes a 30-minute excerpt of two-channel ECG signals. Also, each record has an annotation pair that contains time positions and beat types of all ECG signals in the paired record. Similar to [14, 17], we first eliminate four records (record 102, 104, 107, and 217) containing paced heartbeats. Among two channels in ECG data, we extract only the upper channel from each record as it highlights abnormal ECG patterns [17]. In most cases, the upper channel signals come from modified limb II (ML II); however, in the case of record 114, the ML II signal is located at its bottom channel. Therefore, we exclude record 114 too. A total of 43 out of 48 records are selected for further preprocessing.

Each ECG record contains n number of cardiac cycles of heartbeat collected using electrodes placed on the skin. Each heartbeat has three main components: the P wave that reflects the depolarization of the atria; the QRS complex that reflects the depolarization of the ventricles; and the T wave that reflects the repolarization of the ventricles. Together, they can tell if the heart is normal or has a problem.

The next step is to extract every single beat from selected 43 records. We took an equal number of samples from the left and right side of R-peaks which is done by iterating through the time-series annotations. One hundred values were taken from both sides; However, the segment was discarded if another R-peak existed in the sampled interval. The current beat after this step has 201 sampling values containing only single R-peak. Each signal was then rescaled by the min-max normalization shown in Equation (7), where x is original, and x' is normalized value in the signal. The normalized signal was downsampled to 128 by adopting the Fourier method.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{7}$$

Denoising was the last step of refining ECG signals. We chose biorthogonal wavelet as decomposition and reconstruction wavelet function. The level of signal decomposition was 3. As shown in Equation (8), we applied a soft thresholding technique to decomposed coefficients w, based on calculated universal threshold [4].

$$w' = \begin{cases} \text{sgn}(w)(|w| - \lambda) & (|w| \geq \lambda) \\ 0 & (|w| < \lambda) \end{cases} \tag{8}$$

$$\lambda = \sigma\sqrt{2\log N} \tag{9}$$

Equation (9) shows the universal threshold value, where σ is the median absolute deviation of the wave coefficients at the last level divided by 0.6745, and N is the length of the denoising input signal, in this case, 128.

ECG heartbeat samples in the MIT-BIH database are classified into 17 categories according to morphological patterns in a signal, labeled by independent cardiologists. Following [17], we select 5 heartbeat types as classification targets: N (normal beat), L (left bundle branch block), R (right bundle branch block), A (atrial premature contraction), V (ventricular premature contraction). After excluding beat samples whose labels are not included among five types, 96K beat samples remain. We randomly pick 6,000 samples for N, L, R, V beat types/classes, and 2,490 samples for A beat type, as a number of A labeled beats are not as sufficient as other categories. We divide this data pool equally into the train and test set as shown in Table 1.

B APPENDIX: TRAINING FLOW IN DETAIL.

We further detail the training flow in Algorithm 1 and Algorithm 2 in terms of 1D CNN, as shown in Fig. 6. This training flow can be directly applied to the 1D CNN ECG classification model we made in Section 3.1.2.

In initialization, the client first connects to the server through the socket and gets some training parameters to synchronize. The initialization in Algorithm 1 and 2 require five parameters to be synchronized between the client and the server. The server sends ϕ, η, o and n to the client that the client should adjust to, just after accepting the connection. ϕ is a random weight initializer, η is learning rate, o is a type of optimizer, and n is the batch size. These four hyperparameters should be synchronized on both sides to let them trained in which is the same way. When the client receives n, it sends N to the server, which is the number of total batches to be trained. n is implicitly used not only for the batch generation but also for determining the shape of the matrix in forward and backward propagation. N helps the server to know how many times the forward backward propagation should be done. If needed, the number of epochs should also be synchronized between the client and the server. In Algorithm 1 and 2, they give only a single epoch training process. Hence, the training repetition process is omitted.

Forward propagation starts from the client-side. The client first generates the batch, (x, y), which has n data extracted from the train set $\mathbb{D}$. x represents the features of data, and y shows their labels. Starting with x, the client does the forward propagation until layer l. If layer i is 1D convolutional layer, $f^{(i)}$ can be expressed as follows [14]:

$$z_k^{(i)} = f^{(i)}(a^{(i-1)}) = b_k^{(i)} + \sum_{j=1}^{C_{i-1}} \text{Conv1D}(a_j^{(i-1)}, w_{jk}^{(i)}) \tag{10}$$

where $z_k^{(i)}$ is the k-th filter of the output from the i-th layer, and $a_j^{(i-1)}$ is the j-th filter of activation from layer $i-1$. Also, $w_{jk}^{(i)}$ is a convolution filter which connects between $a_j^{(i-1)}$ and $z_k^{(i)}$. C_{i-1} means the number of filters in the output from the $(i-1)$-th layer. Conv1D means the regular 1D convolution operation without zero padding. $b_k^{(i)}$ is the bias value added after the convolution operation. Let us assume that the server has at least one convolutional layer on its previous side. Equation (10) tells that forward propagation on layer i only depends on the activated output of $(i-1)$-th layer. In other words, the server can continue forward propagation on layer $l+1$ by only receiving $a^{(l)}$ from the client. After receiving $a^{(l)}$ and label y, the server is able to do forward propagation to the L-th layer.

Before starting the backpropagation, the server calculates loss between $a^{(L)}$ and y. Here, the cost function J produces E by computing either mean square error or cross-entropy loss in the usual case. The backpropagation function $f_{\text{T}}^{(i)}$ can be written as follows if i-th layer is 1D convolutional layer [14].

$$\frac{\partial E}{\partial a_k^{(i)}} = f_{\text{T}}^{(i)}(\frac{\partial E}{\partial z_k^{(i+1)}}) = \sum_{j=1}^{C_{j+1}} \text{Conv1Dz}(\frac{\partial E}{\partial z_k^{(i+1)}}, \text{rev}(w_{kj}^{(i+1)})) \tag{11}$$

rev means reversing 1D convolution filter array, and Conv1Dz is full 1D convolution operation with (filter length $-$ 1) zero padding on both side. Again, Equation (11) indicates that computing $\frac{\partial E}{\partial a^{(i)}}$ only depends on $\frac{\partial E}{\partial z^{(i+1)}}$. This means the client can continue backpropagation by receiving just $\frac{\partial E}{\partial a^{(l)}}$ from the server. When the client receives $\frac{\partial E}{\partial a^{(l)}}$, the client can generate $\frac{\partial E}{\partial z^{(l)}}$ by multiplying $\frac{\partial a^{(l)}}{\partial z^{(l)}}$, which is $g'(z^{(l)})$. Then the client is able to do backpropagation to the first hidden layer with (11). In backpropagation, there is no need for calculating gradients of input because it is not trainable. For example, the client does not have to calculate $\frac{\partial E}{\partial a^{(0)}}$, because the input is a variable which is directly given from the user. However, from the server perspective, the server has to calculate the gradient of input, $\frac{\partial E}{\partial a^{(l)}}$, because the client requires it. Therefore, the model on the server part should be forced to calculate the gradient to the input level. To update weights, computing the gradient of weights in layer i can be given as follows [14]:

$$\frac{\partial E}{\partial w_{kj}^{(i)}} = \text{Conv1D}(a_k^{(i-1)}, \frac{\partial E}{\partial z_j^{(i)}}) \tag{12}$$