SplitFed: When Federated Learning Meets Split Learning

Chandra Thapa[1], M.A.P. Chamikara[1,2], and Seyit Camtepe[1]

[1] CSIRO Data61, Australia
[2] RMIT University, Australia
{chandra.thapa, chamikara.arachchige, seyit.camtepe}@data61.csiro.au

Abstract. Federated learning (FL) and split learning (SL) are two recent distributed machine learning (ML) approaches that have gained attention due to their inherent privacy-preserving capabilities. Both approaches follow a model-to-data scenario, in that an ML model is sent to clients for network training and testing. However, FL and SL show contrasting strengths and weaknesses. For example, while FL performs faster than SL due to its parallel client-side model generation strategy, SL provides better privacy than FL due to the ML model architecture split between clients and the server. In contrast to FL, SL enables ML training with clients having low computing resources as the client trains only the first few layers of the split ML network model. In this paper, we present a novel approach, named splitfed (SFL), that amalgamates the two approaches eliminating their inherent drawbacks. SFL splits the network architecture between the clients and server as in SL to provide a higher level of privacy than FL. Moreover, it offers better efficiency than SL by incorporating the parallel ML model update paradigm of FL. Our empirical results, on uniformly distributed horizontally partitioned HAM10000 and MNIST datasets with multiple clients, show that SFL provides similar communication efficiency and test accuracy as SL, while significantly decreasing - by four to six times - its computation time per global epoch than in SL for both datasets. Furthermore, as in SL, its communication efficiency over FL improves with the number of clients. To further enhance privacy, we integrate a differentially private local model training mechanism to SFL and test its performance on AlexNet with the MNIST dataset under various privacy levels.

1 Introduction

Machine learning (ML) usually relies on large datasets and high computational resources. For example, deep learning (DL) is a type of ML that requires extensive computing resources (e.g., GPUs) as well as large datasets to produce high accuracy [1,2]. Due to the excellent learning capabilities, DL is heavily used in several fields, including healthcare and finance, which often operate on privacy-sensitive data. Moreover, data are usually distributed and isolated in these fields due to privacy concerns. For example, in a healthcare system, data reside in hospitals, imaging centers, and research centers that are located in different locations. Finding the privacy-friendly methods that enable DL to access these distributed data is essential in ML applications.

Broadly, there are two approaches to perform ML on the mode of data access. A straightforward approach is to conduct a central pooling of the raw data, and then, analysts access the central repository of the pooled data to conduct ML over the overall dataset. This approach is called Data-to-Modeler (DTM) [3]. However, it has challenges due to ethical requirements, regulation (e.g., GDPR in Europe [4]), privacy concern, single-point failure, competitiveness between the data custodians, and scalability issues.

The second approach is to leverage distributed collaborative learning (DCL). It enables computation on multiple systems or servers and end devices such as mobile phones, while data reside in the source devices. As the distributed data are not pooled to one central repository, DCL provides a level of privacy to the datasets. However, there can still be privacy risks as the server, or distributed systems may not be trusted platforms. In this regard, various techniques, including homomorphic encryption (HE) [5], differential privacy (DP) [6], multi-party computation (MPC) [7], and distributed collaborative machine learning (DCML) have been proposed for privacy-preserving computations. These techniques enable privacy-by-design in the system. HE and MPC are cryptographic approaches and presents challenges, including high computational requirements, communication cost, and the necessity of participants being online all time. Low efficiency in ML model training and high communication latency are two main drawbacks of the existing DP approaches. However, compared to encryption scenarios, DP approaches are computationally efficient; hence, they provide feasible solutions for privacy preservation. In DCML, the analyst has no access to the raw data; instead, the ML model is submitted to the data curator for processing. This approach is called Model-to-data (MTD) [3] and depicted in Fig. 1. It enables data processing without accessing the raw data. This way, MTD provides privacy to sensitive raw data. On top of this, DCML can integrate DP or HE [8] and MPC [9] to provide robust privacy.

Federated learning (FL) [10,11], split learning (SL) [12], and distributed synchronous SGD [13] are the most popular DCML approaches. Although DCML optimization has been a frequently addressed topic, in this work, our

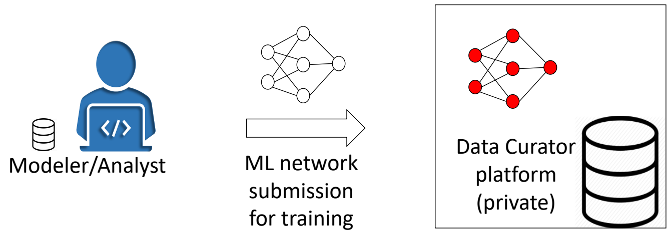


Fig. 1: Model-to-data approach

primary focus is on privacy preservation and efficiency (e.g., training time for an ML model) in DCML. In this regard, based on whether the machine learning model is split and trained separately or not, we broadly divide these approaches into two types; the first type is without network splitting, e.g., FL and distributed synchronous SGD, and the second type with network splitting, e.g., SL. Distributed synchronous SGD (e.g., downpour SGD [14]) works in a similar essence to FL. The difference is that the (local and global) updates are based on one batch of training data in the distributed synchronous SGD, whereas, in FL, a client trains the network over its all local data for some time, i.e., local epochs[3], before updating the model to the server. Considering the similarity and high latency in distributed synchronous SGD, we choose only FL from the first type in this work.

The main advantage of FL is that it allows parallel (hence efficient) ML model training across multiple clients, whereas the main advantage of SL is that it provides better privacy due to the split of the ML model between the clients and the server. These features offer two benefits. Firstly, the model privacy as the client has no access to the server-side model and vice-versa. Secondly, assigning only a part of the network to train at the client-side reduces processing load (compared to that of running the complete network), which is significant in ML computation on resource-constrained devices. However, due to the sequential nature of ML model training across the clients, SL is significantly slower than FL. Now by considering the pros and cons of FL and SL, we have a natural question: *Can we combine the SL and FL strategically to exploit their main advantages?*

This paper answers this question and associated research questions related to performance and privacy. The contributions are stated in the following.

1.1 Our contributions

- Propose a novel architecture that blends FL and SL to produce *splitfed learning (SFL)*: The architecture and algorithm of SFL are described in Section 3.1.
- Comparative performance measurements of FL, SL, and SFL: SFL provides an excellent solution that offers better privacy than FL (Section 5.1), and it is faster than SL with a similar performance in model accuracy and communication efficiency (Section 7). Moreover, SFL enables distributed processing across clients with low computing resources as SL, but with an improvement in its training speed by four to six times based on our empirical studies (Section 7.4).
- Implement and analyze splitfed learning with differential privacy: To reduce the potential privacy leakage among clients, client-fed server, and client-main server, differential privacy based measures are implemented (Section 5.2). The performance evaluation for various levels of privacy under AlexNet on MNIST is presented in Section 7.5.

SFL is beneficial for resource-constrained environments where full model training and deployment is not feasible, and fast model training time is required to periodically update the model based on a continually updating dataset with time. These environments characterize various domains, including health, e.g., real-time anomaly detection in a network with multiple medical internet of things[4] connected via gateways, and finance, e.g., privacy-preserving credit card fraud detection.

[3] In one local epoch of a client, there is a completion of one forward and its respective back-propagation for all available local data in that client.

[4] Examples of medical internet of things (MIoTs) include glucose monitoring devices, open artificial pancreas systems, wearable electrocardiogram (ECG) monitoring devices, and smart lenses.

Table 1: Notations

t	Round (time instance)	$\nabla\ell$	Gradient of the loss ℓ
k, K	K clients, each indexed by $k \in \{1, 2, \ldots, K\}$	η	Learning rate
$\mathbf{W}$	Joint global machine learning model	E	No. of local epochs at client side
$\mathbf{W}_t$	$\mathbf{W}$ at t	$\mathcal{B}$	A set of batches of local data
$\mathbf{W}^{\mathrm{C}}_{k,t}$	Client-side model of client k at t	S_t	A set of n_t clients at t, $n_t \leq K$
$\mathbf{W}^{\mathrm{S}}_t$	Server-side Model at t	$\mathbf{A}_t$	All activations at t
$f(x; y)$	Any function f on input x given y	$\mathbf{A}_{k,t}$	Activations of the cut layer (i.e., smashed data) of client k at t
$\mathbf{Y}_k$	True labels from the client k	$\hat{\mathbf{Y}}_k$	Predicted labels for the client k

2 Background

2.1 Federated learning

Federated learning [10,11,15] is a collaborative machine learning technique developed by google to train machine learning models on distributed devices (e.g., mobile phones). It pushes the computations to the edge devices and removes the necessity to pool the raw data from the data curator to train an ML algorithm. This way, it enables the privacy of the raw data. In FL, a complete ML network/algorithm is trained by each client on its local data in parallel for some local epochs, and then they send the local updates to the server. Afterward, the server aggregates the local updates from all clients to form a global model, and this completes one global epoch[5]. The learned parameters of the global model are then sent back to all clients to train for the next round. This process continues until the algorithm converges to a certain level (i.e., runs for multiple global epochs). Besides, in FL, the local epochs contribute to stabilizing the locally trained model, which can contribute to the fast convergence of the global model in terms of the number of communications with the server. In this paper, we consider the FederatedAveraging algorithm of FL. For more detail, refer to Algorithm 1.

Algorithm 1: Federated learning [11]. Model $\mathbf{W}$ is a global model, and the fraction of the participant C is kept 1 (i.e., $n_t = K$).

```
/* Runs on Server */
Ensure the Server executes at round t ≥ 0:
    Distribute W_t to a random subset S_t of K clients
    for each client k ∈ S_t in parallel do
        W_{k,t} ← ClientUpdate(W_{k,t})                ▷ Receives updates from n_t clients
    end
    Perform W_{t+1} ← Σ_{k=1}^{K} (n_k/n) W_{k,t}   ▷ n = Σ_k n_k, and server performs weighted average of the gradients and update
     the model
/* Runs on Client k */
EnsureClientUpdate(W_{k,t}):
    Set W_{k,t} = W_t                                  ▷ W_t is downloaded from server
    for each local epoch e from 1 to E do
        for batch b ∈ B do
            Forward (calculate loss), back-propagation (calculate ∇ℓ(W_{k,b,t}))
            W_{k,t} ← W_{k,t} − η∇ℓ(W_{k,b,t})
        end
    end
    Send W_{k,t} to server                 ▷ n_k is the entire size of the dataset at each client
```

[5] When forward propagation and back-propagation are completed for all available datasets across all participating clients for one cycle, then it is called one global epoch.

Algorithm 2: Split learning with label sharing [16]. The learning rate η remains same in both server and client side.

```
/* Runs on Server */
EnsureMainServer executes at round t ≥ 0:
    for a request from client k ∈ S_t with new data do
        (A_{k,t}, Y_k) ← ClientUpdate(W^C_{k,t})                ▷ A_{k,t} and Y_k are from client k
        Forward propagation with A_{k,t} on W^S_t      ▷ W^S_t is the server-side part of the model W_t
        Loss calculation with Y_k and Ŷ_k
        Back-propagation and model updates with learning rate η: W^S_{t+1} ← W^S_t − η ∇ℓ(W^S_t; A^S_t)
        Send dA_{k,t} := ∇ℓ(A^S_t; W^S_t) (i.e., gradient of the A_{k,t}) to client k for its ClientBackprop(dA_{k,t})
    end

/* Runs on Client k */
EnsureClientUpdate(W^C_{k,t}):
    Set A_{k,t} = φ
    if Client k is the first client to start the training then
        W^C_{k,t} ← Randomly initialize (using Xavier or Gaussian initializer)
    else
        W^C_{k,t} ← ClientBackprop(dA_{k−1,t−1})        ▷ k − 1 is the last trained client with the main server
    end
    for each local epoch e from 1 to E do
        for batch b ∈ B do
            Forward propagation on W^C_{k,b,t}    ▷ W^C_{k,t} of batch b, where W^C_{k,t} is the client-side part of the model W_t
            Concatenate the activations of its final layer to A_{k,t}
            Concatenate respective true labels to Y_k
        end
    end
    Send A_{k,t} and Y_k to server

/* Runs on Client k */
EnsureClientBackprop(dA_{k,t}):
    for batch b ∈ B do
        Back-propagation with dA_{k,b,t}                                ▷ dA_{k,t} of the batch b
        Model updates W^C_{t+1} ← W^C_t − η dA_{k,b,t}
    end
    Send W^C_{t+1} to the next client ready to train with the main server.
```

2.2 Split learning

Split learning [16,12,17] is a collaborative deep learning technique, where a deep learning network $\mathbf{W}$ is split into two portions $\mathbf{W}^{\mathrm{C}}$ and $\mathbf{W}^{\mathrm{S}}$, called client-side network and server-side network, respectively. $\mathbf{W}$ includes weights, bias, and hyperparameters. The clients, where the data reside, commit only to the client-side portion of the network, and the server commits only to the server-side portion of the network. The client-side and server-side portions collectively form the full network $\mathbf{W}$. The training of the network is done by a sequence of distributed training processes. In a simple setup, the forward propagation and the back-propagation take place in the following way: With the raw data, a client trains the network up to a certain layer of the network, called the *cut layer*, and sends the activations of the cut layer, also called *smashed data*, to the server. Then, the server carries out the training of the remaining layers with the smashed data that it received from the client. This completes a single forward propagation. Next, the server carries out the back-propagation up to the cut layer and sends the gradients of the smashed data to the client. With the gradients, the client carries out its back-propagation on the remaining network (i.e., up to the first layer of the network). This completes a single pass of the back-propagation between a client and the server. This process of forward propagation and back-propagation continues until the network gets trained with all the available clients and reaches its convergence. In SL, the architectural configurations are assumed to be conducted by a trusted party that has direct access to the main server. This authorized party selects the ML model (based on the application) and network splitting (finding the cut layer) at the beginning of the learning. The synchronization of the learning process with multiple clients is done either in centralized mode or peer-to-peer mode. In the centralized mode, before starting training with the (main) server, a client updates its client-side model by downloading the model parameters from a trusted third-party server, which retains the updated client-side model uploaded by the last trained client. On the other hand, in peer-to-peer mode, the client updates its client-side model by directly downloading it from the last trained client. Overall, the training takes place in a relay-based approach,

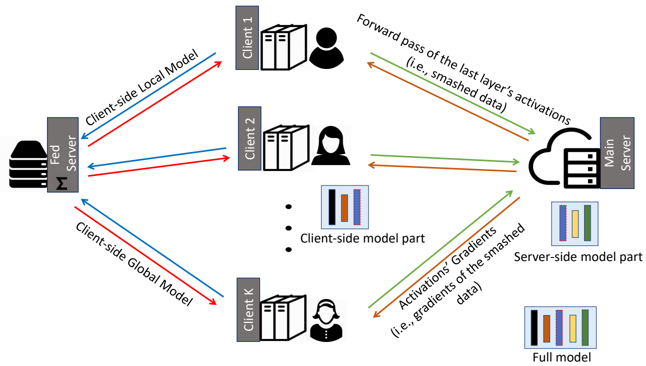


Fig. 2: Splitfed system model

where the server trains with one client and then move to another client sequentially. The SL algorithm is provided in Algorithm 2.

SL limits the client-side network portion down to a few layers. Thus, it enables the reduction in client-side computation compared to other collaborative learning such as FL, where the complete network is trained at the client-side. Besides, in the SL setup, the client-side updates, and the updated model $\mathbf{W}^{\mathrm{C}}$ are not accessible to the server once the training starts. The server knows only the portion of the network it trains, i.e., $\mathbf{W}^{\mathrm{S}}$. Consequently, SL provides a certain level of privacy to both the trained model and the inputs because the server needs to predict all parameters of the client-side network to infer the information. In contrast, the server has full access (i.e., white box access) to the whole trained network in FL. However, there is a primary issue with SL. The relay-based training in SL makes the clients' resources idle because only one client engages with the server at one instance. This significantly increases the training overhead if there are many clients.

Now we present a privacy measure that applies to the DCML approaches.

2.3 Differential Privacy

Differential privacy (DP) is a privacy model that allows the definition of privacy using stochastic frameworks [6,18]. DP comes with two important parameters, namely ϵ and δ, where ϵ and δ refer to *privacy level*, and *probability of failure* (on the size of dataset), respectively. These parameters indicate the level of difficulty in predicting individual data points with high confidence. Let us consider two datasets x and y such that y is formed from x by removing or adding only one data point of x. Hence, x and y are called adjacent datasets. The requirement of differential privacy for any mechanism $\mathcal{M}$ on these two datasets is that $\mathcal{M}(x)$ and $\mathcal{M}(y)$ should be almost indistinguishable for a given range of results, R. We can provide a formal definition to differential privacy as given below:

Definition 1. *A mechanism $\mathcal{M}$ is considered to be (ϵ, δ)-differential private if, for all adjacent datasets x and y, and for all possible subsets of results R of the mechanism, the following holds:*

$$\mathbb{P}[\mathcal{M}(x) \in R] \leq e^{\epsilon} * \mathbb{P}[\mathcal{M}(y) \in R] + \delta.$$

Practically, the values of ϵ and δ should be kept as small as possible to maintain a high level of privacy. However, the smaller the values of ϵ and δ, the higher the noise applied to the data by the DP algorithm. A higher level of noise will result in a lower level of accuracy. Hence, there is always a trade-off between utility and privacy.

3 Proposed architecture

3.1 Splitfed learning

In this section, we present splitfed learning that utilizes the strengths of SL and FL. SFL combines the strength of FL, which is parallel processing among distributed clients, and the strength of SL, which is network splitting (client-side and server-side) during training. Refer to Fig. 2 for the SFL system model. The right-hand side of the figure shows how the clients and the (main) server perform the network training. Unlike SL, all clients (e.g., Hospitals, IoMTs with low computing resources) carry out the forward propagations on their client-side model in parallel, then pass their smashed data to the (main) server. Then the server, which is assumed to have sufficient computing resources (e.g., cloud server and researchers with high-performance computing resources), process the

Table 2: Comparative description of split, federated and splitfed learning

Learning approach	Model aggregation	Privacy advantage by splitted model (client & server sides)	Client-side training	Distributed computing	Access to raw data
Split	No	Yes	Sequential	Yes	No
Federated	Yes	No	Parallel	Yes	No
Splitfed (proposed)	Yes	Yes	Parallel	Yes	No

Algorithm 3: Splitfed learning (our approach) with label sharing. The splitfed network ($\mathbf{W}$) configuration follows the same setup used in SL. Learning rate η remains the same for the server-side model and the client-side model.

/* Runs on Main Server */
EnsureMainServer executes at round $t \geq 0$**:**
  for each client $k \in S_t$ in parallel **do**
    $(\mathbf{A}_{k,t}, \mathbf{Y}_k) \leftarrow$ ClientUpdate($\mathbf{W}^{\text{C}}_{k,t}$)
    Forward propagation with $\mathbf{A}_{k,t}$ on $\mathbf{W}^{\text{S}}_t$, compute $\hat{\mathbf{Y}}_k$
    Loss calculation with $\mathbf{Y}_k$ and $\hat{\mathbf{Y}}_k$
    Back-propagation calculate $\nabla \ell_k(\mathbf{W}^{\text{S}}_t; \mathbf{A}^{\text{S}}_t)$
    Send $d\mathbf{A}_{k,t} := \nabla \ell_k(\mathbf{A}^{\text{S}}_t; \mathbf{W}^{\text{S}}_t)$ (i.e., gradient of the $\mathbf{A}_{k,t}$) to client k for ClientBackprop($d\mathbf{A}_{k,t}$)
  end
  Server-side model update: $\mathbf{W}^{\text{S}}_{t+1} \leftarrow \mathbf{W}^{\text{S}}_t - \eta \frac{n_k}{n} \sum_{i=1}^{K} \nabla \ell_i(\mathbf{W}^{\text{S}}_t; \mathbf{A}^{\text{S}}_t)$

/* Runs on Client k */
EnsureClientUpdate($\mathbf{W}^{\text{C}}_{k,t}$)**:**
  Model updates $\mathbf{W}^{\text{C}}_{k,t} \leftarrow$ FedServer($\mathbf{W}^{\text{C}}_{t-1}$)
  Set $\mathbf{A}_{k,t} = \phi$
  for each local epoch e from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      Forward propagation on $\mathbf{W}^{\text{C}}_{k,b,t}$
      Concatenate the activations of its final layer to $\mathbf{A}_{k,t}$
      Concatenate respective true labels to $\mathbf{Y}_k$
    end
  end
  Send $\mathbf{A}_{k,t}$ and $\mathbf{Y}_k$ to the main server

/* Runs on Client k */
EnsureClientBackprop($d\mathbf{A}_{k,t}$)**:**
  for batch $b \in \mathcal{B}$ **do**
    Back-propagation, calculate gradients $\nabla \ell_k(\mathbf{W}^{\text{C}}_{k,b,t})$
    $\mathbf{W}^{\text{C}}_{k,t} \leftarrow \mathbf{W}^{\text{C}}_{k,t} - \eta \nabla \ell_k(\mathbf{W}^{\text{C}}_{k,b,t})$
  end
  Send $\mathbf{W}^{\text{C}}_{k,t}$ to the Fed server

/* Runs on Fed Server */
EnsureFedServer executes:
  for each client $k \in S_t$ in parallel **do**
    $\mathbf{W}^{\text{C}}_{k,t} \leftarrow$ ClientBackprop($d\mathbf{A}_{k,t}$)
  end
  Client-side global model updates: $\mathbf{W}^{\text{C}}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{W}^{\text{C}}_{k,t}$
  Send $\mathbf{W}^{\text{C}}_{t+1}$ to all K clients for ClientUpdate($\mathbf{W}^{\text{C}}_{k,t}$)

forward propagation and back-propagation on its server-side model with each client's smashed data in (somewhat) parallel. It then sends the gradients of the smashed data (i.e., activations' gradients) to the respective clients for their back-propagation. Afterward, the server updates its model by a weighted average of the gradients that it computes during the back-propagation on each client's smashed data. At the client's side, after receiving the gradients of the smashed data, each client performs the back-propagation on their client-side local model and computes its gradients. Then, the clients send the gradients to the fed server, which conducts the federated averaging of the client-side local updates and send that back to all participating clients. This way, the fed server synchronizes the client-side global model in each round of network training. The fed server's computations are not costly, and it is hosted within the

Table 3: Total cost analysis of the four DCML approaches

Method	Comms. per client	Total comms.	Total model training time	Total cost
FL	$2\|W\|$	$2K\|W\|$	$t + K\|W\|$	$t + 3K\|W\|$
SL	$(\frac{2p}{K})q + 2\beta\|W\|$	$2pq + 2\beta K\|W\|$	$tK + T(K-1)$	$2pq + 2\beta K\|W\| + tK + T(K-1)$
SFLV1	$(\frac{2p}{K})q + 2\beta\|W\|$	$2pq + 2\beta K\|W\|$	$t + K\|W\|$	$2pq + t + (1+2\beta)K\|W\|$
SFLV2	$(\frac{2p}{K})q + 2\beta\|W\|$	$2pq + 2\beta K\|W\|$	$t + \beta K\|W\|$	$2pq + t + 3\beta K\|W\|$

local edge boundaries. For more detail, refer to the SFL algorithm in Algorithm 3. Although the algorithm is for the label sharing configuration, all possible configurations of SL, including U-shaped without label sharing, vertically partitioned data, extended vanilla, and multi-task SL [16], can be carried out similarly in SFL.

A brief comparative description of SL, FL, and SFL is provided in Table 2.

Variants of splitfed learning In this paper, we propose two variants of SFL. The first one is called *splitfedv1 (SFLV1)*, which is depicted in Algorithm 3, and explained in the previous section. The next algorithm is called *splitfedv2 (SFLV2)*, and it is motivated from the intuition of the possibility to increase the model accuracy by removing the model aggregation part in the server-side computation module in Algorithm 3. In this algorithm, the server-side models of all clients are executed in parallel and then aggregated to obtain the global server-side model at each global epoch. In contrast, SFLV2 processes the forward-backward propagations of the server-side model sequentially with respect to the client's smashed data. The client order is chosen randomly in the server-side operations, and the model gets updated in every single forward-backward propagation. Besides, the server receives the smashed data from all participating clients synchronously. The client-side operation remains the same as in the SFLV1; the fed server conducts the federated averaging of the client-side local updates and sends averaged weights back to all participating clients. This operation is not affected by the client order as the local client-side models are aggregated by the weighted averaging method.

4 Total cost analysis

In this section, we analyze the total communication cost and model training time for federated, split, and splitfed learning. Assume that K be the number of clients, p be the total data size, q be the size of the smashed layer, $|W|$ be the total number of model parameters, β be the fraction of model parameters (weights) available in a client in SL, t be the model training time (for any architecture) for one global epoch, and T be the wait time (delay) for one client to receive updates from SL during one global epoch. Table 3 provides the communication costs and the total costs of each of the DCML approaches for one global epoch. The term $K|W|$ in FL training time is due to the latency caused by federated averaging, and the term $2\beta|W|$ in communication per client is due to the download and upload of the client-side model updates before and after training, respectively, by the client. As shown in the table, due to T, SL can become inefficient when there is a large number of clients (refer to Fig. 7 for empirical results). Besides, we see that when K increases, the total cost (communication and computation) increases in the order of SFLV2<SFLV1<SL, which can also be observed in our empirical results in Fig. 6 and 7.

5 Threat model and measures

In this work, for FL, SL, and SFL, we consider all participants, including clients, the fed server, and the main server of the system model, are honest-but-curious adversaries. They perform the tasks as specified but can be curious about the local private data of the clients. The attack scenario considered here is passive, where they only observe the updates and possibly do some calculations to get the information, but they do not maliciously modify their own inputs or parameters for the attack purpose. Besides, the servers and clients are non-colluding with each other. However, in the latter part of this section (refer to Section 5.2), we consider a stricter variant of the proposed threat model in which we assume both servers to be potential adversaries, and there can be data leakage.

We assume that SFLV1 and SFLV2 (our approaches) adheres to the standard client-server security model where the clients and servers establish a certain level of trust before starting the network model training. For example, in the health domain, the hospitals (clients) only allow researchers (server) which have a level of trust. Hospitals opt-out if the corresponding platform has malicious clients or servers. Besides, we assume that all communications between the participating entities (e.g., exchange of smashed data and gradients between clients and the main server) are performed in an encrypted form. Privacy in SFL (both versions) is enabled by the MTD approach, and network split, as discussed in Section 5.1. In comparison with the threat model of FL and SL, SFL has an extra honest-but-curious and non-colluding fed server at the client-side. The fed server and clients in SFL have access only to the gradients/weights of the client-side model portions only. In FL, the main server has access to the entire model. In contrast, the main server has access only to the server-side portion of the model and the smashed data (i.e., activation vectors of the cut layer) from the client in SL and SFL.

5.1 Privacy preservation due to splitfed learning

The privacy preservation aspect of SFL is due to two reasons: firstly, it adopts the model-to-data approach, and secondly, SFL conducts ML over a split network. A network split in ML learning enables the clients/fed server and the main server to maintain privacy by not allowing the server to get the client-side model updates and vice-versa. The main server has access only to the smashed data (i.e., activation vectors of the cut layer). The curious main server needs to invert all the client-side model parameters, i.e., weight vectors. The possibility of inferring the client-side model parameters and raw data is highly unlikely if we configure the client-side ML networks' fully connected layers with sufficiently large numbers of nodes [12]. However, for a smaller client-side network, this possibility can be high. This issue can be reduced by modifying the loss function at the client-side [19]. Due to the same reason as above, the clients or the fed server, which has access only to the gradients of the smashed data from the server, cannot infer the server-side model parameters.

As mentioned above, in FL, there is no network split and separate training on client-side and server-side as in SFL. Thus, SFL provides more privacy for ML model training than FL.

5.2 Splitfed learning with differential privacy at client side

Although splitfed learning comes with a privacy infused architecture, we want to investigate its performance under strict privacy configurations with differential privacy. In our architecture, the clients communicate with two servers: (1) the main server (split server) and (2) the fed server. The clients share their smashed data (activations) from their cut layer to the main server, whereas the fed server aggregates the client-side model portion (refer to Section 3.1). During both of these communications, the clients do not share their raw data with these two servers or other clients. We provide a comprehensive discussion of the inherent privacy of the proposed model in Section 5.1. However, there can be an advanced adversary exploiting the underlying information representations of the shared smashed data or parameters (weights) to violate data owners' privacy. This can happen if the data communications between the clients and the servers get breached, or any server becomes vulnerable or malicious. To avoid this possibility (i.e., potential privacy leakage), we apply differential privacy to the client-side model training algorithm based on the differentially private deep learning approach developed by Abadi et al [20]. As discussed in Section 3.1, the clients and the main server (split server) collaboratively train the client-side model portion and the server-side model portion separately while training one whole model that is split between the clients and the main server. Thus, the application of the differential privacy on the client-side model guarantees a differentially private client-side model training that is independent of the main server-side model training.

Considering Algorithm 3, we present the process for implementing differential privacy at a client k. We assume the following: σ represents the noise scale, and C' represents the gradient norm bound. Now, firstly, after t time, the client k, receives the gradients $d\mathbf{A}_{k,t}$ from the server, and with this, it calculates client-side gradients $\nabla \ell_k(\mathbf{W}^{\mathrm{C}}_{k,i,t})$ for each of its local sample x_i, and

$$\mathbf{g}_{k,t}\left(x_i\right) \leftarrow \nabla \ell_k(\mathbf{W}^{\mathrm{C}}_{k,i,t}). \tag{1}$$

Secondly, the ℓ_2-norm of each gradient is clipped according to the following equation:

$$\overline{\mathbf{g}}_{k,t}\left(x_i\right) \leftarrow \mathbf{g}_{k,t}\left(x_i\right) / \max\left(1, \frac{\left\|\mathbf{g}_{k,t}\left(x_i\right)\right\|_2}{C'}\right). \tag{2}$$

Thirdly, calibrated noise is added to the average gradient:

$$\tilde{\mathbf{g}}_{k,t} \leftarrow \frac{1}{n_k} \sum_i \left(\overline{\mathbf{g}}_{k,t}\left(x_i\right) + \mathcal{N}\left(0, \sigma^2 C'^2 \mathbf{I}\right)\right). \tag{3}$$

Finally, the client-side model parameters of client k are updated as follows:

$$\mathbf{W}^{\mathrm{C}}_{k,t+1} \leftarrow \mathbf{W}^{\mathrm{C}}_{k,t} - \eta_t \tilde{\mathbf{g}}_{k,t}. \tag{4}$$

The application of noise will continue iteratively until the model converges or reaches a specified number of iterations. As the iterations progress, the final convergence will exhibit a privacy level of (ε, δ)- differential privacy, where (ε, δ) is the overall privacy cost of the model.

However, for the first forward propagation based on Algorithm 3, the activation, including the smashed data, are not subjected to noise application. Thus, to avoid the potential privacy leakage at the first pass of the smashed data from the client to the main server and further strengthen the model/data privacy, we implement an additional privacy measure in the cut layer of the client-side model. This measure utilizes the noise application mechanism involved in local differential privacy to add a calibrated noise over the smashed data while maintaining utility. In this process, firstly, we calculate the bounds (max $\mathbf{A}_{k,i}$, min $\mathbf{A}_{k,i}$) of smashed data, and generate a vector of intervals $\Delta I_i = \max \mathbf{A}_{k,i} - \min \mathbf{A}_{k,i}$. Secondly, Laplacian noise with scale $\frac{\Delta I_i}{\varepsilon'}$ is applied to randomize the smashed data as follows:

$$\mathbf{A}^{\mathrm{P}}_{k,i} = \mathbf{A}_{k,i} + \mathrm{Lap}\left(\frac{\Delta I_i}{\varepsilon'}\right), \tag{5}$$

where, $\mathbf{A}^{\mathrm{P}}_{k,i}$ represents a private version of the smashed data, and ϵ' is the privacy budget used for the Laplacian noise. With this approach, the noisy smashed data is sent to the main server. To follow a strict privacy configuration, we consider the final client-side model to be $((\varepsilon+\varepsilon'), \delta)$-differentially private with the assumption that all the clients work on IID data. Empirical results are presented in Section 7.5.

6 Experiment setup

Experiments are carried out on uniformly distributed and horizontally partitioned image datasets among clients. All programs are written in python 3.7.2 using the PyTorch library (PyTorch 1.2.0). For quicker experiments and developments, we use the High-Performance Computing (HPC) platform that is built on Dell EMC's PowerEdge platform with partner GPUs for computation and InfiniBand networking. We run clients and servers on different computing nodes of the cluster provided by HPC. We request the following resources for one slurm job on HPC: 10GB of RAM, one GPU (Tesla P100-SXM2-16GB), one computing node with at least one task per node. The architecture of the nodes is x86_64. During the experiments, we investigate the performance by observing the training time (also called latency) with respect to global epochs, and the amount of data communication by each client. In our setup, we consider that all participants update the model in each global epoch (i.e., $C = 1$ during training). We choose ML network architectures and datasets based on their performance and their need to include in our studies for proportionate participation. The learning rate for LeNet is maintained at 0.004, and 0.0001 for the remainder network architectures (AlexNet, ResNet and VGG16). We choose the learning rate based on their performance during our initial observations. For example, for LeNet on FMNIST, we observed train and test accuracy of 94.8% and 92.1% with a learning rate of 0.004, whereas 87.8% and 87.3% with a learning rate of 0.0001 in 200 global epochs. We set up a similar computing environment for comparative analysis. We do not conduct hyperparameter tunning or any other model optimization scenarios, as our work is on the privacy-preserving approaches in DCML rather than the optimization of the individual ML network.

6.1 Datasets

We use four public image datasets in our experiments, and these are summarized in Table 4. HAM10000 dataset is a medical dataset, i.e., the Human Against Machine with 10000 training images [21]. It consists of colored images of pigmented skin lesion, and has dermatoscopic images from different populations, acquired and stored by different modalities. Each sample has 600×450 pixel images. HAM10000 has a total of $10,015$ samples with seven cases of important diagnostic categories: Akiec, bcc, bkl, df, mel, nv, and vasc. MNIST dataset consists of handwritten digit images of 0 to 9 (i.e., 10 classes). Each image has 28×28 pixels with a pixel value ranges from 0 to 255. It has a total of $60,000$ training and $10,000$ test samples. Fashion MNIST consists of images of ten clothing, including T-shirts, trousers, pullover, dress, and coat. Each sample has 28×28 pixel grayscale images the number of training and test samples equal to the MNIST dataset. CIFAR10 consists of color images of ten objects (classes), including airplane, cat, dog, bird, automobile, horse, and ship. It has $50,000$ training and $10,000$ test samples. Each sample in CIFAR10 has 32×32 pixels. For our experiments, we consider the training and testing sample sizes described in Table 4.

6.2 Model Architecture

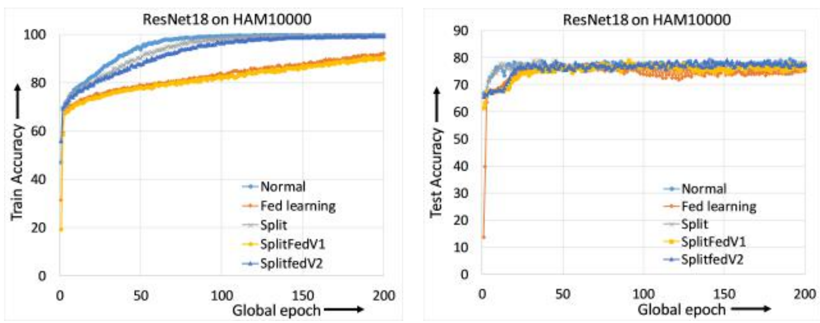
We consider four ML model architectures in our experiments. These four architectures fall under Convolutional Neural Network (CNN) architectures and are summarized in Table 5. We restrict our experiments to CNN architectures, and further experiments on other architectures such as recurrent neural networks will be investigated in

Table 4: Datasets

Dataset	Training samples	Testing samples	Image size
HAM10000 [21]	9013	1002	600×450
MNIST	$60,000$	$10,000$	28×28
FMNIST	$60,000$	$10,000$	28×28
CIFAR10	$50,000$	$10,000$	32×32

Table 5: Model Architecture

Architecture	No. ofparameters	Layers	Kernel size
LeNet [22]	60 thousands	5	(5×5), (2×2)
AlexNet [2]	60 million	8	(11×11), (5×5), (3×3)
VGG16 [23]	138 million	16	(3×3)
ResNet18 [24]	11.7 million	18	(7×7), (3×3)



(a) (b)

Fig. 3: (a) Training, and (b) testing convergence of ResNet18 on HAM10000 under various learning with five clients.

future work. LeNet [22] is a five-layer CNN consisting of convolution, average pooling, Sigmoid or Tanh, and fully connected layers. It uses the 5×5 and 2×2 sized kernels in its layers. The input image dimension is $32 \times 32 \times 1$. AlexNet [2] consists of eight network layers, including convolution (five layers), pooling (three layers), and fully connected (two) layers. It uses 11×11, 5×5, and 3×3 sized kernels in its layers. The input image dimension is $227 \times 227 \times 3$. VGG16 [23] consists of sixteen network layers, including convolution (sixteen layers), max pooling, and fully connected layers. It uses 3×3 sized kernels in its layers. The input image dimension is $224 \times 224 \times 3$. ResNet18 [24] consists of eighteen network layers, including convolution, max pooling, ReLU, and fully connected layers. It uses 7×7 and 3×3 sized kernels in its layers.

For all experiments in SL, SFLV1, and SFLV2, the network layers are split at the following layer: second layer of LeNet (after 2D MaxPool layer), second layer of AlexNet (after 2d MaxPool layer), fourth layer of VGG16 (after 2D MaxPool layer), and third layer (after 2D BatchNormalization layer) of ResNet18.

7 Empirical Results and Discussion

7.1 How do the distributed collaborative machine learning, including splitfedv1 and splitfedv2 perform?

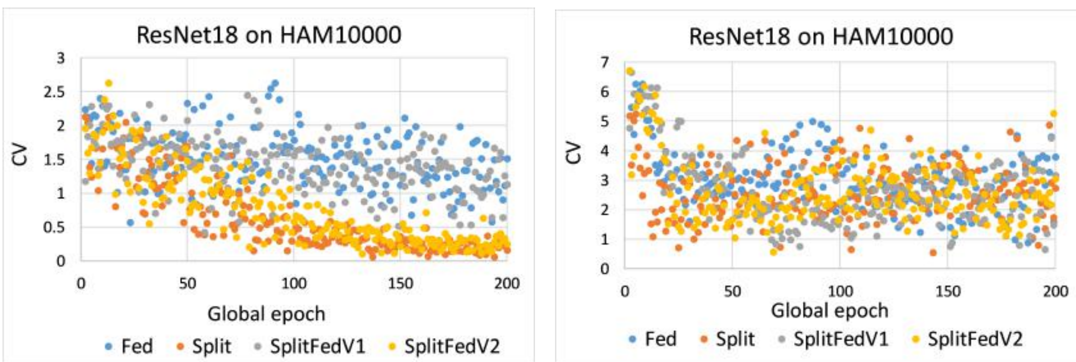
We consider the results under standard learning (centralized learning) as our benchmark. In standard learning, all data are available centrally to a server that performs ML training and testing. Table 6 and 7 summarize our first result, where the observation window is 200 global epochs with one local epoch, batch size of 1024, and five clients for DCML. The tables show the highest accuracy observed at 200 global epochs; thus, it is not necessarily the accuracy at the final global epoch (i.e., 200). Moreover, the train/test accuracy is averaged over all clients in the DCML setup at each global epoch. The process of calculating the accuracy is the following: the model is trained with each client, then based on the model's predictions and true labels, the training accuracy is calculated. Afterward, the test accuracy is calculated on the test data. The performance observations (e.g., calculating the average accuracy) are carried out in each global epoch, both in centralized and DCML setups.

Table 6: Training Results

Dataset	Architecture	Normal	Federated	Split	Splitfedv1	Splitfedv2
HAM10000	ResNet18	99.8%	92.2%	99.5%	90.2%	99.3%
HAM10000	AlexNet	98.8%	74.2 %	70.3%	69%	72.1%
FMNIST	LeNet	95.1%	92.5 %	90.6%	89%	89.6%
FMNIST	AlexNet	97.2%	90.6 %	83.9%	84.9%	79%
CIFAR10	LeNet	78.7%	70.5 %	62.6%	61.3%	62%
MNIST	AlexNet	99.7%	98.8%	93.7%	95.7%	89.8%
MNIST	ResNet18	99.9%	99.9%	99.9%	99.8%	99.9%

Table 7: Test Results

Dataset	Architecture	Normal	Federated	Split	Splitfedv1	Splitfedv2
HAM10000	ResNet18	79.3%	77.5%	79.1%	79%	79.2%
HAM10000	AlexNet	80.1%	75 %	73.8%	70.5%	74.9%
FMNIST	LeNet	92.7%	91.9 %	90.4%	89.6%	90.4%
FMNIST	AlexNet	90.5%	89.7%	84.7%	86%	81%
CIFAR10	LeNet	72.1%	69.4 %	62.7%	62.6%	63.8%
MNIST	AlexNet	98.8%	98.7 %	95.1%	96.9%	92%
MNIST	ResNet18	99.3%	99.2 %	99.2%	99%	99.2%



(a) (b)

Fig. 4: (a) Training, and (b) testing coefficient of variation (CV) of ResNet18 on HAM10000 under various learning configurations with five clients.

Analyzing Table 6 and 7: As presented in the tables, SL and SFL (both versions) performed well under the proposed experimental setup. However, we also observed that among DCML, FL shows better learning performance in most cases as the original model is not split while trained among the multiple clients. In all cases except ResNet18 on MNIST, the four approaches failed to achieve the benchmark results. Based on the results in Table 6 and 7, we can observe that SFLV1 and SFLV2 have inherited the characteristics of SL. We noticed that VGG16 on CIFAR10 did not converge in SL, which was the same for both versions of splitfed learning, although there were around 66% and 67% of training and testing accuracies, respectively, for FL. We assume that this was because of the unavailability of certain other factors such as hyper-parameters tuning or change in data distribution or adding additional regularization terms in the loss function, which are out of the scope of this paper. Besides, in some cases, the training accuracy was less than the test accuracy. This may be due to the use of dropouts, the less diversity in test data, and the consideration of the highest accuracy within the observation window (i.e., 200 global epochs).

In the following, we present and discuss the performance of ResNet18 on the HAM10000 dataset for standard configuration, FL, SL, SFLV1, and SFLV2 under similar settings. The training and testing performances of the remaining architectures and datasets are provided in Appendix A.3.

Analyzing Fig. 3 : For ResNet18 on HAM10000, the convergence patterns during SFLV1 and FL training were close to each other, but slower than SFLV2 and SL. The training accuracy reached around 90% for them, whereas others got 99% in the observation window of 200 global epochs. However, the test accuracy convergence was almost the same for all learning approaches, and they reached to around 76% in the observation window of 200 global epochs. Based on the observations, the performances of SFLV1 and SFLV2 were similar for ResNet18 on

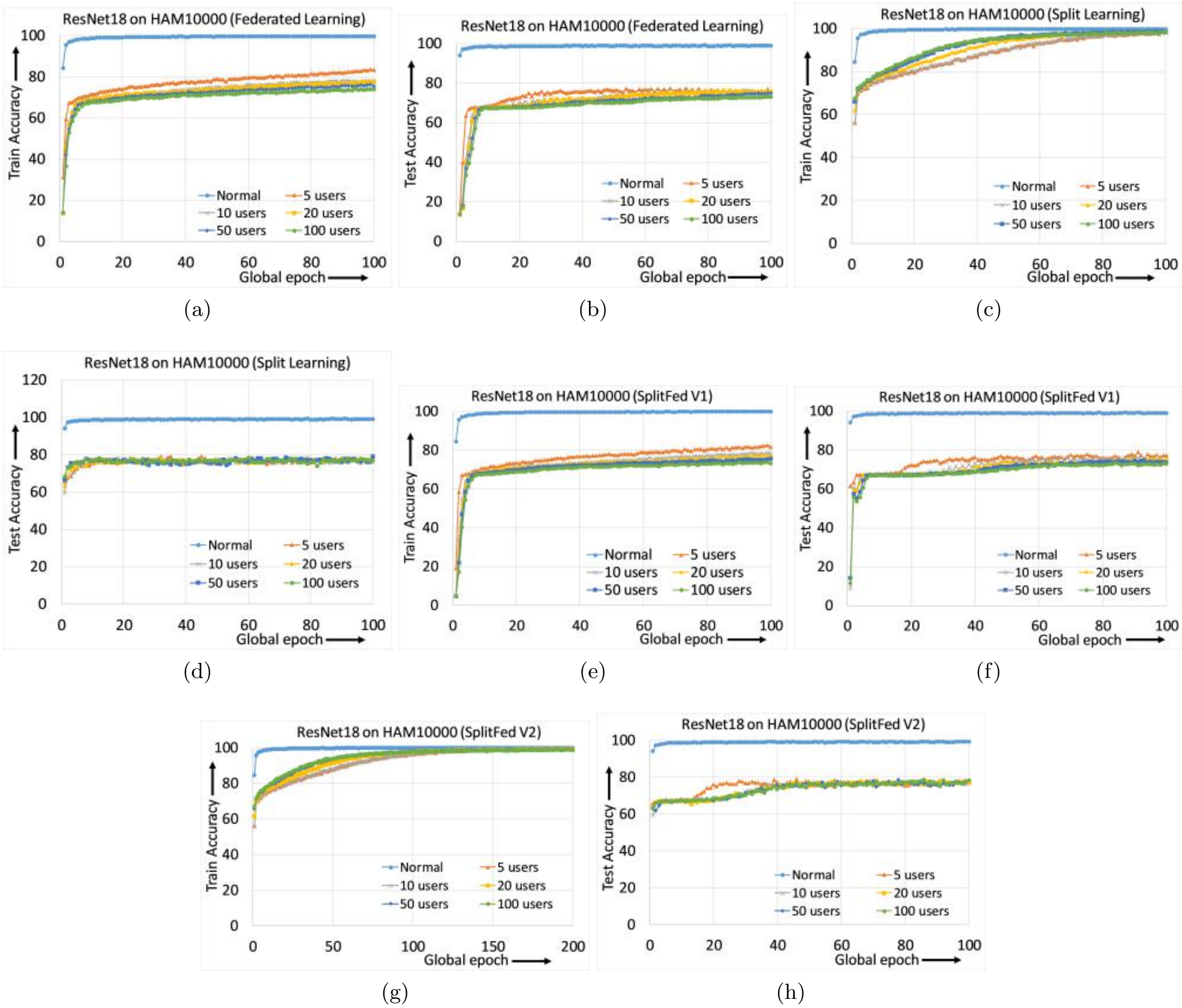


Fig. 5: Training and testing convergence of ResNet18 on HAM10000 with various number of clients.

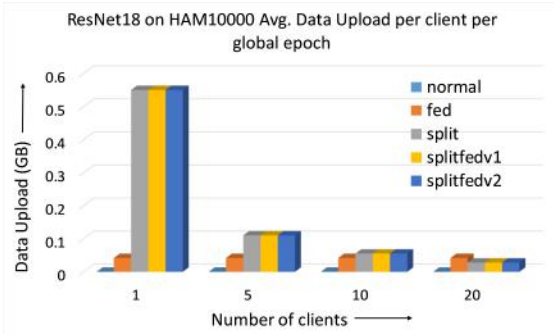
HAM10000. However, SFLV1 and SFLV2 struggled to converge if SL failed to converge, as shown in Fig. 12 for the case of VGG16 on CIFAR10.

Analyzing Fig. 4: So far, we present the training and testing mean accuracy over the five clients with respect to the global epochs. Fig. 4 illustrates the variations of the performance (i.e., accuracy) over five clients at each global epoch. In this regard, we compute the coefficient of variation (CV), which is a ratio of the standard deviation to the mean, and it measures the dispersion. Moreover, we calculate the CV over the five accuracies provided by five clients at each global epoch. Based on our results for ResNet18 on HAM10000, the CVs for SL, FL, SFLV1, and SFLV2 are bounded between 0.06 and 2.63 while training, and 0.54 and 6.72 while testing after epoch 2; at epoch 1, the CV is slightly higher. The results indicate uniform individual client-level performance across the clients.

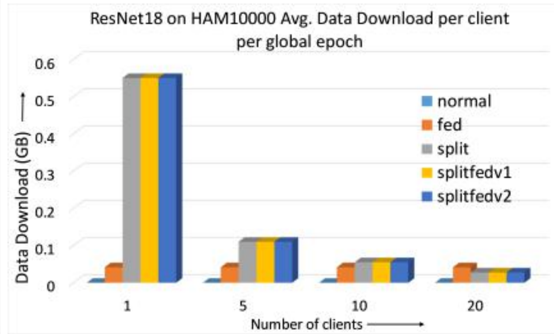
In some datasets and architecture, the training accuracy of the model was still improving and showing better performance at higher global epochs than 200. For example, going from 200 epochs to 400 epochs, we noticed an accuracy increment from 84.7% to 86.9% for FL with LeNet on FMNIST with 100 users. However, we limited our observation window to 200 global epochs as some network architecture such as AlexNet on HAM10000 in FL was taking an extensive amount of training time on the HPC (a shared resource).

7.2 Effect of number of users on the performance

In this section, we present the analysis of the effect of the number of users for ResNet18 on HAM10000. Refer to Appendix A.4 for the results of LeNet on FMNIST and AlexNet on HAM10000. For ResNet18 on HAM10000, we



(a)



(b)

Fig. 6: (a) Data upload, and (b) data download for ResNet18 on HAM10000 with various number of clients.

observed that up to 100 clients (clients ranging from 5 to 100), the training and testing curves for all numbers of clients followed a similar pattern in each plot. Moreover, they achieved a similar level of accuracy within each of our DCMLs. We got comparative test accuracies of 74% (federated), 77% (split), 75% (SFLV1), and 77% (SFLV2) at 100 global epochs. While training, only SL and SFLV2 achieved the standard training accuracy at around 100 global epochs. In contrast, FL and SFLV1 could not achieve this result even at 200 global epochs. The experimental results for clients ranging from five to hundred showed a negligible effect on the performance due to the increase in the number of clients in FL, SL, SFLV1, and SFLV2. However, there was a significant gap between standard (centralized) learning and DCML (refer to Fig. 5). This was not the case in general. For LeNet on FMNIST with a fewer number of clients, the performances were close to the standard case in FL and SL (refer to Fig. 13). Moreover, for SL with AlexNet on HAM10000, the performance degraded and even failed to converge with the increase in the number of clients, and we saw a similar effect on the SFLV2 (refer to Fig. 14). Overall, the convergence of the learning and performance slowed down (sometimes failed to progress) with the increase in the number of clients in our DCML techniques due to the resource limitations and other constraints, such as the change in data distribution among the clients with the increase in its number, and a regular global model aggregation to synchronize the model across the multiple clients. However, the model convergence will not slow down in a non-resource-constrained environment.

7.3 Communication measurement

The amount of data uploaded and downloaded by a client indicates the operability of a DCML approach in a resource-constrained environment. High data communication slows down the ML training and testing process, and the clients need to have sufficient resources to handle the high communication cost. In this regard, we measure the amount of data communication in our experiments and present the relative performance of the four DCML techniques. To make the experimental setup normalized for different numbers of clients, we run our program under the following configurations: The main server's and fed server's programs run in two different HPC nodes, and clients' programs run in five separate nodes (except for experiment with one client). Each client HPC node runs one, two, and four clients (client-side programs) for five, ten, and twenty client cases, respectively. We record the total communication for the observation window of 11 global epochs with one local epoch and a batch size of 256. Then, the results were averaged over all global epochs and clients to obtain the communication cost per client per global epoch.

During the experiments, we observed the same amount of data communication for SL, SFLV1, and SFLV2. For example, refer to Fig. 6 and 9. This shows that SFLV1 and SFLV2 provided the same communication efficiency as in SL. As the number of clients increased, the amount of data communication (which was averaged over clients) per global epoch decreased for SL, SFLV1, and SFLV2. This was because of the following two reasons: (1) SL and both versions of SFL were communicating smashed data rather than (model) weights. Thus, the amount of communication was the function of the data samples. (2) The overall dataset was uniformly distributed among clients, so the dataset size per client decreased with the increase in the number of clients. On the other hand, this effect was not applicable to FL, which had the same level of data communication in all cases. This was because, in FL, for given network architecture, each client sends the locally trained network to the server and receives the global network independent of the sample size.

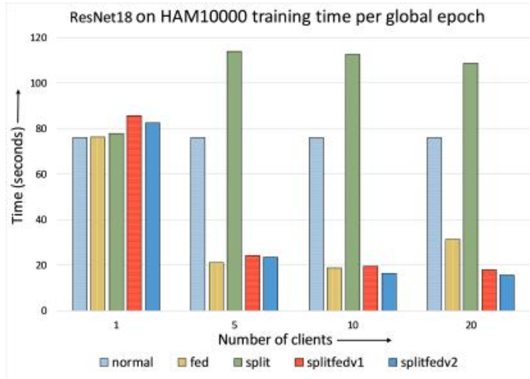
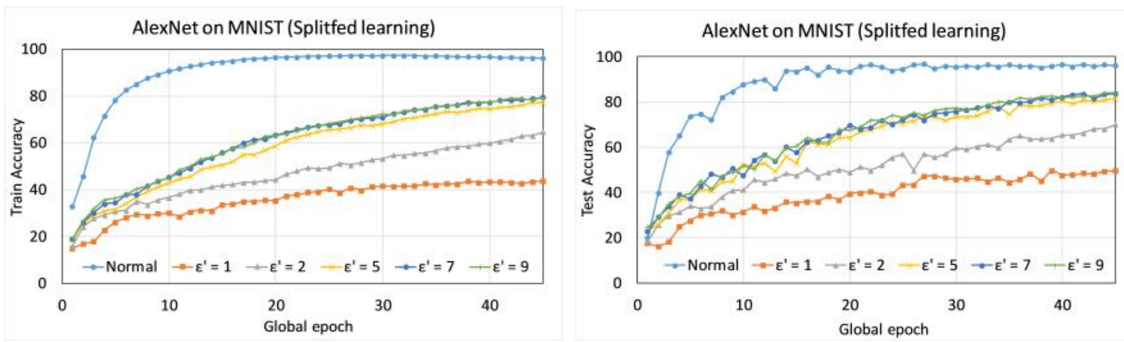


Fig. 7: Time measurement for ResNet18 on HAM10000 under various number of clients.



(a) (b)

Fig. 8: (a) Training, and (b) testing convergence of AlexNet on MNIST with five clients, sensitivity $\delta = 1e^{-5}$, $\varepsilon = 0.5$ ($\sigma = 1.3$), and under various ε'.

In recent work, it was shown that SL is more communication efficient than FL with an increase in the number of clients or model size [25]. In contrast, FL is preferred if the amount of data samples is increased by keeping the number of clients or model size low [25,26]. Our results in Fig. 6 and 9 complement this result for SL and FL.

7.4 Time measurement

To show the time efficiency of splitfed learning (both versions) compared to SL, we analyze the training time taken for one global epoch. Considering Section 4, Algorithms 2 and 3, it is not difficult to see the following: (1) For SL, the total training time for one global epoch depends on the product of *the number of clients*, and *the time to process one global epoch by the clients and the server*, and (2) in SFLV1, there are no such product terms while calculating the time because the server can be a supercomputing resource and processes all clients in parallel. Consequently, SFLV1 is faster than SL for multiple clients. Besides, this also shows that the training time measurement depends not only on the implementation but also on the algorithm. For our experiments in SFLV1, we implemented a multithreaded python program for the main server. By using the same experimental setup (which was used to measure the communication cost), we ran each experiment for eleven global epochs and recorded the time for each global epoch. Unlike in the communication measurement setup, the training time was averaged by considering the time from the second global epoch onward for all clients after running each experiment for ten times. The time for the first global epoch was excluded because it included the time taken by clients to connect to the server, i.e., the initial connection overhead (in our setup, all clients got connected to the server at first and kept the connection during the experiment). We ran each experiment ten times - different HPC slurm jobs in each instance - to exclude the effects of the change in the computing environment in each run.

Based on our observations on ResNet18 on HAM10000 and AlexNet on MNIST, the time statistics for the cases with multiple clients indicated that SFLV1 and SFLV2 were significantly faster - by four to six times - than SL. It had a similar or even better speed than FL (refer to Fig. 7 and Fig. 10). For the single client case, all other DCML approaches spent similar time; SFLV1 and SFLV2 spent slightly more time than the other.

7.5 Splitfed learning with differential privacy at client side

We implement the differential privacy measures as described in Section 5.2. For illustration, experiments are performed for SFLV1 with AlexNet on MNIST data distributed over five clients. With the application of differential privacy measures, the training time increases for our programs, which is evident due to the introduction of additional privacy-related computations. Hence, we limit our observation window to 45 global epochs. For 45 global epochs with five local epochs at each client per global epoch, the training and testing accuracy curves converge as shown in Fig. 8. Besides, as for illustration, we change the values of ε' from one to nine to see the effect on the overall performance. Moreover, we maintain ε at 0.5 during all experiments to examine the behavior of SFL under strict client model privacy settings. As expected, the convergence of accuracy curves with differential privacy is gradual and slow compared to non-differentially private training. Besides, testing accuracy of around 50%, 70%, 82%, 84%, and 84% are observed at global epoch 45 for ε' equal to $1, 2, 5, 7$, and 9, respectively. Clearly, the accuracy increases with the increase in the privacy budget, i.e., $\varepsilon + \varepsilon'$. Overall, the utility is decreased with a decrease in the privacy budget (which means a higher level of privacy). As the client-side architecture in SFLV2 is the same as in SFLV1, the application of differential privacy in SFLV2 is done in the same way as in SFLV1.

8 Limitations and future works

In this study, we have analyzed the performance of FL, SL, SFLV1, and SFLV2 based on their privacy-preserving features, model accuracy, communication, and computation costs. In addition, we have implemented and analyzed the SFLV1 with differential privacy. The comparative performance analysis of these privacy-preserving distributed ML approaches with the integration of DP [6,27] and HE [5] is remained as future work. Moreover, DP provides provable privacy, whereas fully HE provides guaranteed privacy. However, they have a negative effect on model performance (also seen in our experimental results) and computational overhead, respectively. In our experiments on SFLV1 with differential privacy, a detailed trade-off analysis of privacy and utility is not done. Thus, the studies related to privacy-utility trade-off analysis are left as future work.

9 Conclusion

By bringing federated learning (FL) and split learning (SL) together, we proposed a novel distributed machine learning approach, named splitfed, that offered a higher level of privacy than FL due to network splitting, faster than SL due to parallel processing across clients. Results showed that splitfed provides similar performance in terms of model accuracy compared to SL. Thus, as being a hybrid approach, it is suitable for ML with low-computing resources (enabled by network split), fast training (enabled by handling clients in parallel), and analytic over private and sensitive data. Our empirical results for ResNet18 on HAM10000 and AlexNet on MNIST showed that the two versions of splitfed learning (presented in this paper) were significantly faster than SL for multiple clients. Moreover, their speed was similar or even better than FL in some instances, and they had the same communication efficiency of SL and improved its efficiency than FL with the increase in the number of clients. To further strengthen the privacy in splitfed learning, differential privacy measures were implemented both on the gradients' update process and the smashed data transmission from the clients to the main server. This enabled us to eliminate the potential privacy leakage among the client-client, client-main server, and client-fed server during collaborative learning.

References

1. Guosheng Hu, Xiaojiang Peng, Yongxin Yang, Timothy M. Hospedales, and Jakob Verbeek. Frankenstein: Learning deep face representations using small data. *IEEE Trans. Image Processing*, 27(1):293–303, 2018.
2. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS'12 - Vol. 1*, pages 1097–1105, USA, 2012.
3. J. Guinney and J. Saez-Rodriguez. Alternative models for sharing confidential biomedical data. *Nature BioTech.*, 36(5):391–392, May 2018.
4. EU. Regulation (eu) 2016/679 general data protection regulation. *Official Journal of the European Union*, May 2016.
5. Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, Stanford, California, Sept. 2009.
6. Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
7. Andrew C. Yao. Protocols for secure computations. In *Proc. SFCS '82*, pages 160–164. IEEE Computer Society, 1982.

8. Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Trans. Info. Forensics and Security*, 13(5):1333–1345, 2018.

9. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proc. CCS*, pages 1175–1191. ACM, 2017.

10. Jakub Konecný, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arxiv*, 2015. https://arxiv.org/pdf/1511.03575.pdf.

11. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, pages 1273–1282, 2017.

12. Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *J. Network and Computer Applications*, 116:1–8, 2018.

13. Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jzefowicz. Revisiting distributed synchronous sgd. In *2016 Proc. of ICLR workshop track*, 2016.

14. Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, pages 1232–1240, 2012.

15. Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *arxiv*, 2019. http://arxiv.org/abs/1902.01046.

16. Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arxiv*, 2018. http://arxiv.org/abs/1812.00564.

17. Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we use split learning on 1d cnn models for privacy preserving training? In *Proc. ACM AsiaCCS*, 2020. https://arxiv.org/pdf/2003.12365.pdf.

18. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7(3):17–51, 2016.

19. Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and Ramesh Raskar. Reducing leakage in distributed deep learning for sensitive health data. In *Proc. ICLR*, 2019.

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

21. Philipp Tschandl. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, 2018. doi:10.7910/DVN/DBW86T.

22. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, Nov 1998.

23. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. 3rd ICLR*, 2015.

24. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE CVPR*, pages 770–778, June 2016.

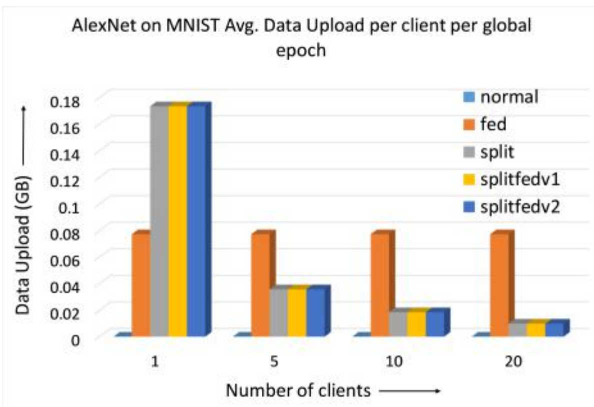
25. Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *arxiv*, 2019. https://arxiv.org/abs/1909.09145.

26. Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A. Camtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. In *Proc. SRDS*, 2020. https://arxiv.org/pdf/2003.13376.pdf.

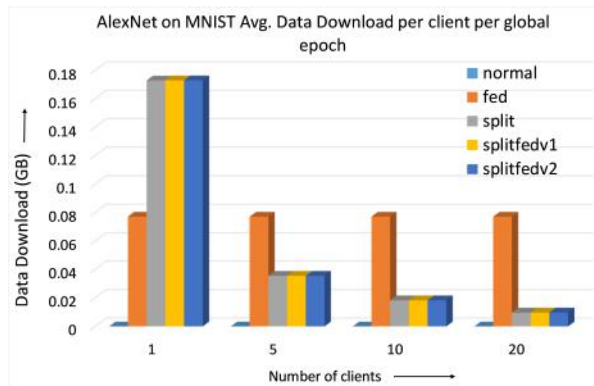
27. Pathum Chamikara Mahawaga Arachchige, Peter Bertok, Ibrahim Khalil, Dongxi Liu, Seyit Camtepe, and Mohammed Atiquzzaman. Local differential privacy for deep learning. *IEEE Internet of Things Journal*, 2019.

A Supplemental Figures

A.1 Communication measurement



(a)



(b)

Fig. 9: (a) Data upload, and (b) data download for AlexNet on MNIST with various number of clients.

A.2 Training time measurement

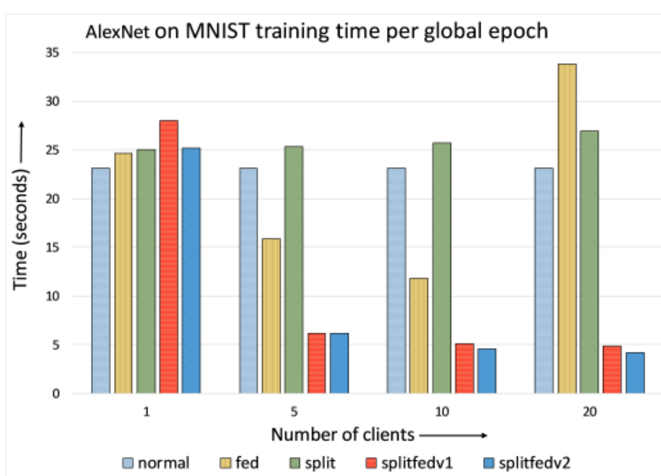
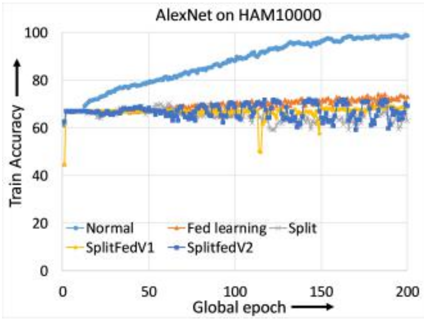
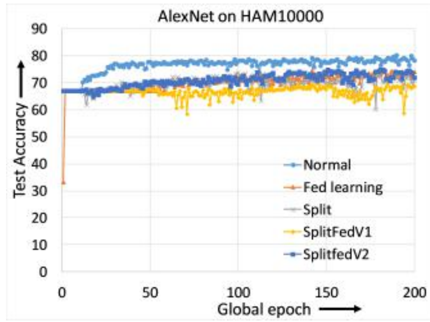


Fig. 10: Time measurement for AlexNet on MNIST with various number of clients.

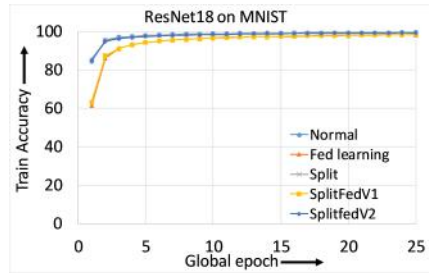
A.3 Learning and performance curves



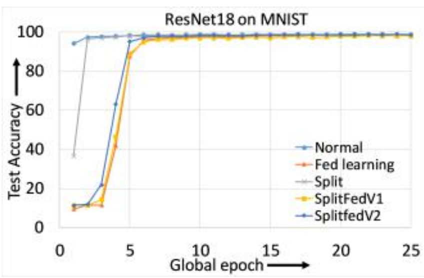
(a)



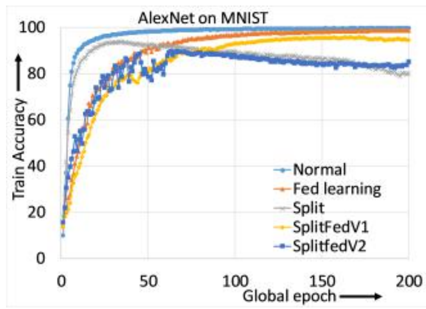
(b)



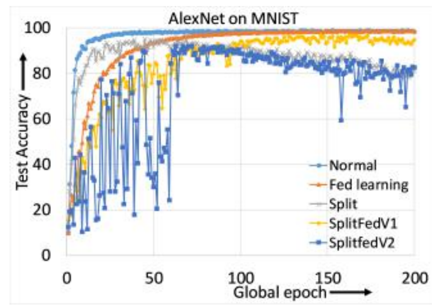
(c)



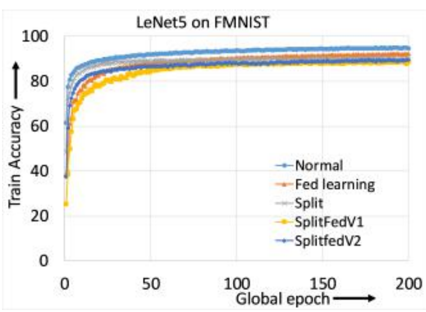
(d)



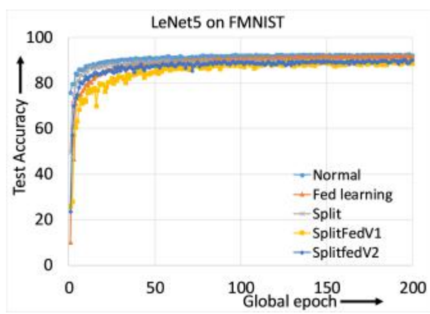
(e)



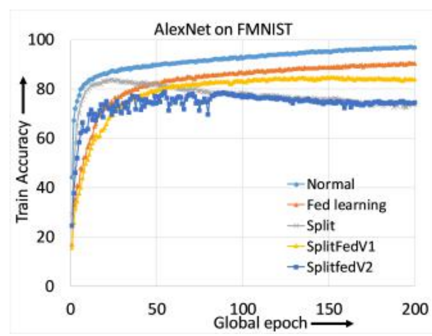
(f)



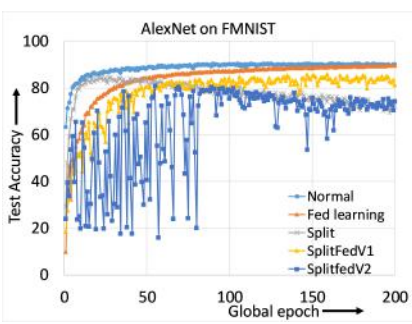
(g)



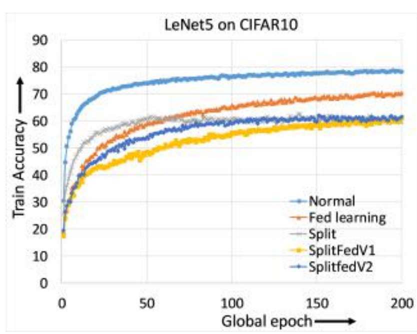
(h)



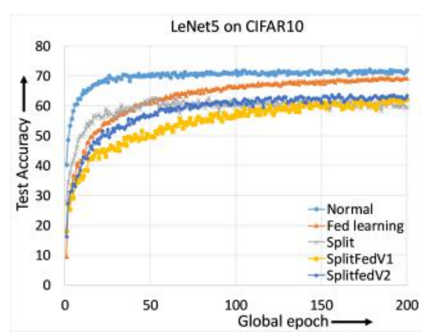
(i)



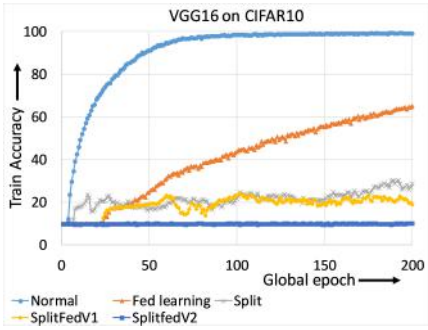
(j)



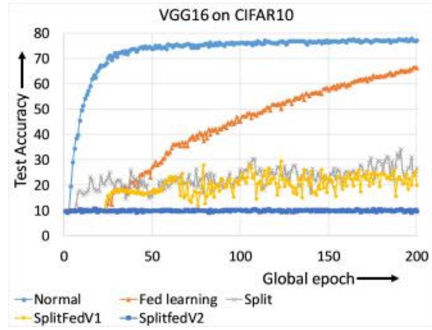
(k)



(l)



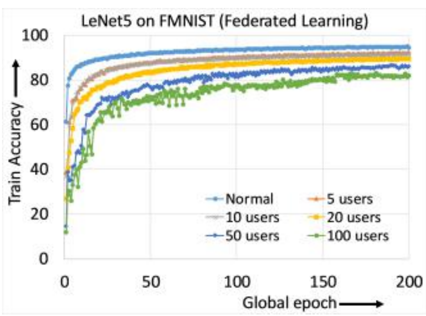
(m)



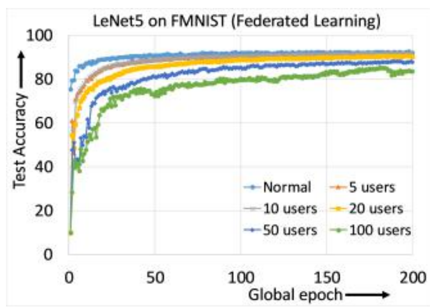
(n)

Fig. 12: Training and test convergence with five clients under various DCML approaches, architectures and datasets.

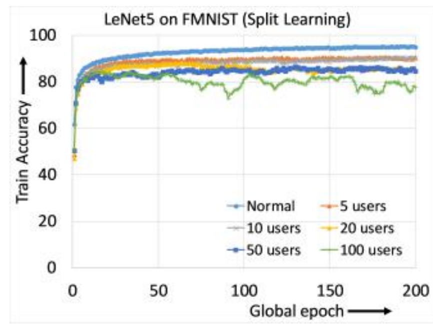
A.4 Effects of number of users on the performance



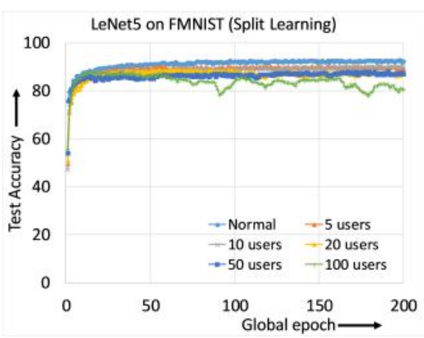
(a)



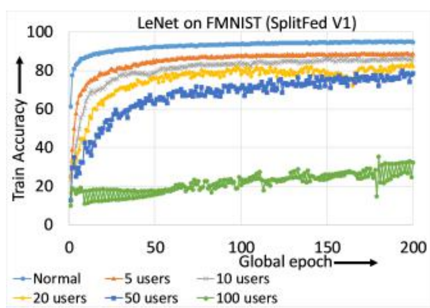
(b)



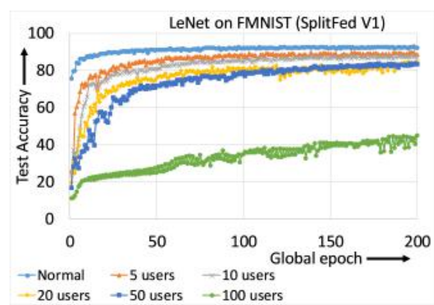
(c)



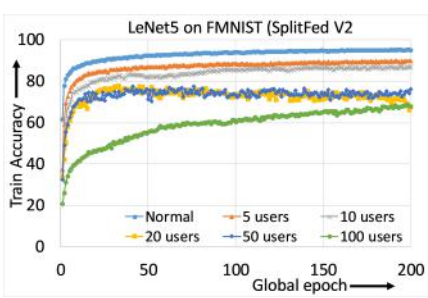
(d)



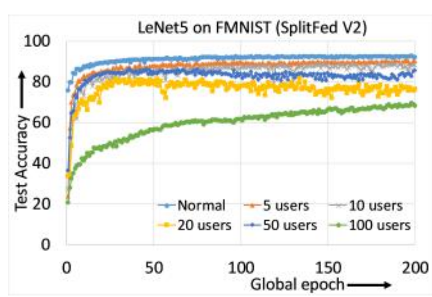
(e)



(f)



(g)



(h)

Fig. 13: Effects of number of users for LeNet on FMNIST: Training and test convergence with various number of clients and DCML approaches.

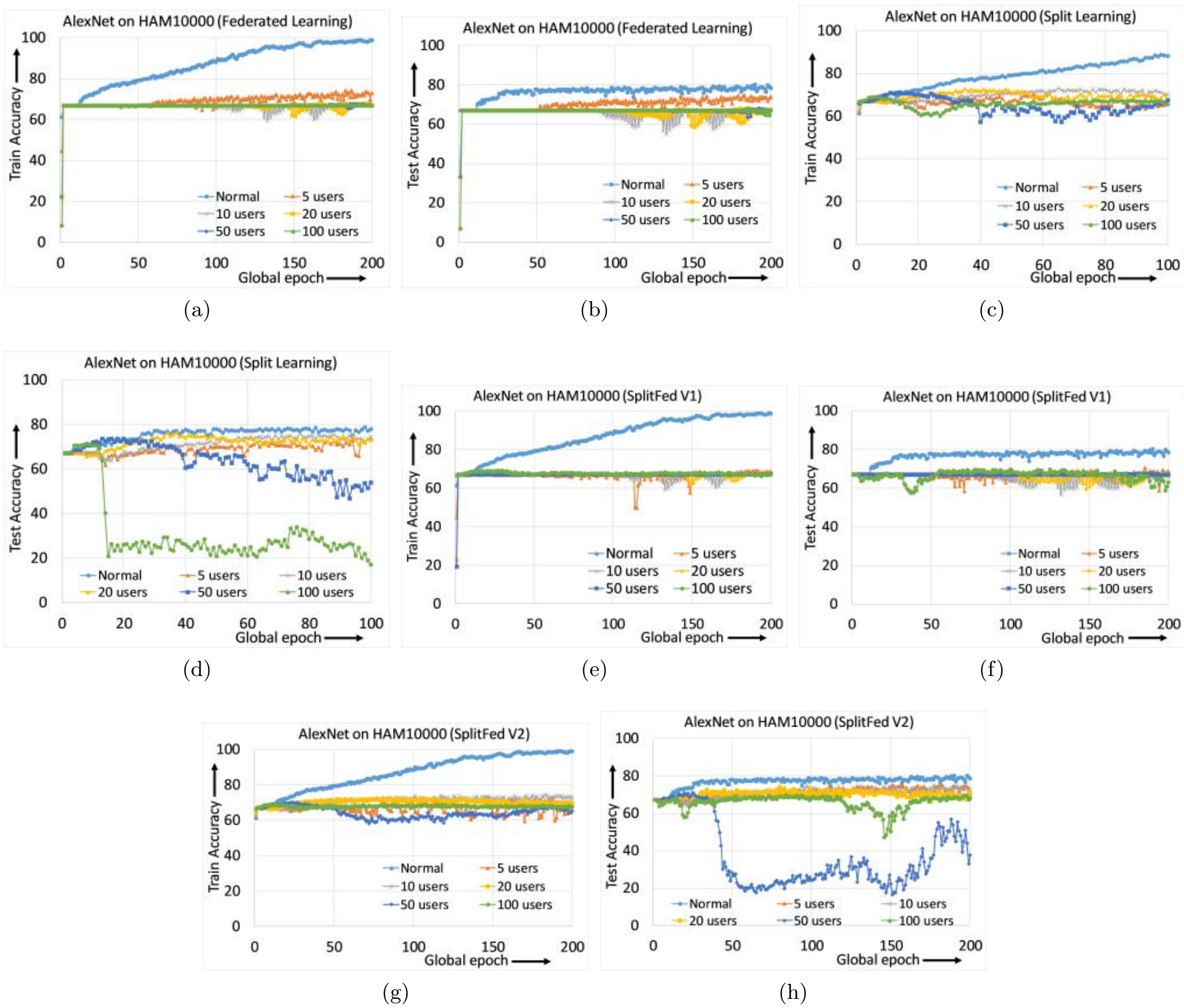


Fig. 14: Effects of number of users for AlexNet on HAM10000: Training and test convergence with various number of clients and DCML approaches.