FedDCT: Federated Learning of Large Convolutional Neural Networks on Resource Constrained Devices using Divide and Co-Training

Quan Nguyen, Hieu H. Pham*, Member, IEEE, Kok-Seng Wong, Member, IEEE, Phi Le Nguyen, Truong Thao Nguyen and Minh N. Do, Fellow, IEEE

Abstract—In Federated Learning (FL), the size of local models matters. On the one hand, it is logical to use large-capacity neural networks in pursuit of high performance. On the other hand, deep convolutional neural networks (CNNs) are exceedingly parameter-hungry, which makes memory a significant bottleneck when training large-scale CNNs on hardware-constrained devices such as smartphones or wearables sensors. Current state-of-the-art (SOTA) FL approaches either only test their convergence properties on tiny CNNs with inferior accuracy or assume clients have the adequate processing power to train large models, which remains a formidable obstacle in actual practice. To overcome these issues, we introduce FedDCT, a novel distributed learning paradigm that enables the usage of large, high performance CNNs on resource-limited edge devices. As opposed to traditional FL approaches, which require each client to train the full-size neural network independently during each training round, the proposed FedDCT allows a cluster of several clients to collaboratively train a large deep learning model by dividing it into an ensemble of several small sub-models and train them on multiple devices in parallel while maintaining privacy. In this co-training process, clients from the same cluster can also learn from each other, further improving their ensemble performance. In the aggregation stage, the server takes a weighted average of all the ensemble models trained by all the clusters. FedDCT reduces the memory requirements and allows low-end devices to participate in FL. We empirically conduct extensive experiments on standardized datasets, including CIFAR-10, CIFAR-100, and two real-world medical datasets HAM10000 and VAIPE. Experimental results show that FedDCT outperforms a set of current SOTA FL methods with interesting convergence behaviors. Furthermore, compared to other existing approaches, FedDCT achieves higher accuracy and substantially reduces the number of communication rounds (with 4–8 times fewer memory requirements) to achieve the desired accuracy on the testing dataset without incurring any extra training cost on the server-side.

I. INTRODUCTION

Federated learning (FL) [8] has emerged as a state-of-the-art machine learning (ML) paradigm that allows clients to cooperatively learn a shared model in a decentralized fashion where no private dataset is sent to a central repository. Specifically, the data for the ML tasks are acquired and processed locally at the edge nodes, and only the updated ML parameters are transmitted to the server for aggregation purposes [9], [10]. FL has been successfully deployed in many industries, including healthcare, telecommunications, IoT, manufacturing, and surveillance systems [11], [12]. Given the broad applications of FL, guaranteeing that such a cooperative learning process is reliable is becoming essential.

Despite significant recent milestones in FL [13], several fundamental challenges are yet to be addressed [14]. For instance, edge nodes often have substantial constraints on their resources, such as memory, computation power, communication, and energy, severely limiting their capability of locally training large models. The FL algorithm designs often assume all clients can train the entire ML model independently. However, the dedicated memory of edge devices can be insufficient to store all weights and/or intermediate states of large deep learning models during training [15]. Fig. 1 compares the memory consumption during the training of deep neural networks of

Fig. 1. Memory consumption of training SetNet-154 [1], ResNet-200 [2] on ImageNet dataset [3], DensetNet [4] on CIFAR-100 dataset and ResNet-110 [2] on CIFAR-10 dataset [5] with different batch sizes. All implementations are in PyTorch (version 1.10.2) [6]. The red lines indicate the common memory capacities of mobile devices with their relative amount of devices in percentage [7].

Index Terms—Federated learning, deep convolutional neural networks, split learning, edge devices, edge learning, co-training.
different model sizes with the memory capacity of current mobile devices. Consequently, the availability of the on-device memory constraints the size of neural networks model in training [16], [17]. Typically, the model size is reduced, and the input is similarly reduced in dimension, potentially destroys information during training. [18]. As data collection systems continue to advance, the dimensionality of real-world datasets continues to grow, leading to an ever more pressing need for the ability to train larger and more complex models in FL [19], [20]. This necessitates the development of more memory-efficient training algorithms. However, the majority of state-of-the-art techniques designed to reduce inference memory [21], [22] are applicable in training memory reduction, as they often require first training a full-size model that is later compressed. The techniques to reduce training memory used in the centralized training at a data center/supercomputer are also not applicable in FL. For example, the out-of-core [23] method moves data out/into extra memory of the CPU from the GPU’s memory. Gradient checkpointing [15], [24]–[26] proposes to redundantly recompute the intermediate states of the forward propagation in the backward propagation instead of storing it. Such extra memory and increased compute requirement cannot be fulfilled by the edge devices. As a result, FL algorithms are still restricted to using light-weight models or low-bit training. [27], [28]. Other Distributed Collaborative Machine Learning (DCML) methods either require significant training time overhead (e.g., Split Learning [29]) or are highly dependent on the computing resources on the server side (e.g., SplitFed [27]).

To this end, we propose FedDCT (Federated Divide and Co-training), a novel FL architecture that allows for DCML with considerably lower client memory requirements over traditional FL while introducing no additional training cost on the central server. FedDCT enables a cluster of several clients to cooperatively train a large deep learning model, in contrast to standard FL, which demands that each client train the full-size neural network independently during each training round. Specifically, FedDCT (channel-wise) divides the single large global model into several small sub-models regarding its parameters and regularization components. The ensemble model of these sub-models is the new global model. Each client in a cluster will train the deep ensemble network’s sub-models up to a certain “cut layer” during the training phase using their private dataset. The outputs at the cut layers of the sub-models are sent to other clients in the cluster which completes forward propagation without looking at raw data from data-holding client. Similarly, these clients perform back propagation until the cut layer and the gradients at the cut layers (and only these gradients) are sent back to the data-owning client to finish the rest of back propagation. Together clients in a cluster train the full network. To boost network variety and improve ensemble performance, we also train the sub-networks using various viewpoints of the same data [30], [31]. This is implemented by applying different data augmentation in practice. We also add Jensen Shannon divergence among all predictions of sub-networks. From the predicted posterior probability distributions of its peers, one network can gain insightful knowledge about intrinsic object structure information [33], [34]. In the model aggregation stage, the server combines corresponding sub-models to create an ensemble model for each cluster and aggregates the ensemble models from all clusters to obtain an updated global ensemble model. This process allows for co-training of a large global model with multiple low-computing clients in FL. Overall, FedDCT is beneficial for resource-constrained environments where training the large model are not feasible.

In summary, the main contributions of this study are threefold:

- We propose FedDCT, a novel approach that reduces the memory requirements of training in the FL setting. FedDCT allows lightweight edge devices to overcome resource constraints and participate in the FL task to train large CNN models. To the best of our knowledge, this is the first work that enables the training of large deep learning networks in edge computing devices in FL settings.

- The proposed FedDCT architecture can divide large CNN networks into sub-networks and co-training them with local clients in parallel. This co-training process allows clients from the same cluster to learn representations from each other. The server takes a weighted average of all the ensemble models trained by all the clusters to improve the learning performance of the global model.

- We conduct extensive experiments on natural and real-world medical image datasets. FedDCT significantly outperforms a set of the latest state-of-the-art FL methods and sets new state-of-the-art results on these datasets. Our codes are made publicly available at https://github.com/vinuni-vishc/fedDCT.

- To the best of our knowledge, we are the first to consolidate several advantages of DCML algorithms into a single framework: data privacy, reduced memory demand and asynchronous training, all while achieving superior accuracy compared to other DCML methods.

The remainder of our paper is organized as follows. Related works are reviewed in Section II. In Section III, we
provide brief preliminaries of an FL setting and an overview formulation for training large CNNs. Section IV describes the proposed FL framework and the key techniques behind it. The experimental results and key properties of FedDCT are presented in Section V. Finally, we conclude the paper and discuss the key research challenges, limitations of our method, and potential directions of future works in Section VI.

II. RELATED WORKS

Improving efficiency and effectiveness is extensively researched, but few works address the memory barrier in FL. Existing FL methods such as FedAvg [35] and its variant [14], [36]–[38] face significant hurdles in training large CNNs on resource-constrained devices. Recent works like FedNAS [39] work on large CNNs, but they rely on GPU training to complete the evaluations. Others [40]–[43] optimize the communication cost without considering edge computational limitations. In Table I, we summarize the techniques for dealing with the memory limitation issue during training. In which each component of memory can be categorized as one of the following [15]: (i) model memory which consists of the ML model parameters; (ii) optimizer memory which consists of the gradients of the ML model and the hyper-parameters of the optimization methods such as the momentum vectors; and (iii) the activation memory which consists of the intermediate activation, e.g., the output of each layer in the network stored to reuse during the backpropagation phase and its gradients. The activation memory consumes most of the required memory and could far outstrip the model memory, up to 10s–100s times. The optimizer memory is always $2 - 3 \times$ more than the model memory.

A. Training memory reduction techniques

One of the simplest approaches to reduce the required training memory is to reduce the size of the input samples (downsampling), which may potentially lead to accuracy degradation [18]. On the other hand, authors in [44] proposed reducing the minibatch size by using the micro-batching technique. The mini-batch is divided into smaller micro-batches, which are run in a sequential manner that leads to a reduction in activation size. However, this approach may not work well with other machine learning techniques such as the batch normalization used by Wide-ResNet model [45]. Alternatively, checkpointing [15], [24]–[26] reduces the activation memory by storing the activations of a subset of layers only and redundantly recomputing the un-stored activations during the backpropagation phases. In the centralized training environment, checkpointing becomes more efficient when combined with the out-of-core method [23] to avoid too much recomputing by temporarily storing some of the activations in extra memory, e.g., CPU memory. However, such kind of approach is not applicable to the edge devices environment due to the hardware and power consumption restriction.

B. Compression techniques

In literature, compression techniques have been well-design for inference on edge-mobile devices [46]–[48]. Because most of those techniques often require training with a full model before compressing, it could not help for low-memory training. Alternatively, one can reduce the required training memory by removing the less important connections of the ML networks (model pruning), so that storing fewer parameters with a little or no loss of training accuracy [49], [50]. Another direction aims to train very low-precision, e.g., use a lower number of bits to present a weight value (quantization or low-precision). For example, [51] demonstrated that training with most storage in half-precision (16 bits) could achieve the same accuracy with 32 and 64-bits numbers.

C. Memory reduction in DCML

Split learning (SL) [29] attempts to break the computation constraint by splitting the ML model into two portions $W^c$ and $W^s$ in a layer-wise manner and offloading the larger portion into the server-side. The communication involves sending activations, called smashed data, at the cut layer of the client-side network to the server, and receiving the gradients of the smashed data from the server-side. When doing ML computation on devices with limited resources, assigning only a portion of the network to train at the client-side minimizes processing burden (compared to running a complete network as in FL). In addition, neither a client nor a server can access the other’s model. However, the relay-based training in SL idles the clients’ resources because only one client engages with the server at one instance; causing a significant increase in the training time overhead. SplitFed [27] (SFL) attempts to solve this by processing the forward propagation and back-propagation on its server-side model with each client’s smashed data separately in parallel, assuming that the server is equipped to handle the training of all clients simultaneously, which is impractical in real-world FL scenarios where millions of device can participate in training [52]. As we have reached a state of maturity sufficient to deploy the system in production and solve applied learning problems with millions of real-world devices [9]; and we anticipate uses where the number of devices reaches billions, there is a need for a more efficient way to resolve the computation problem in FL without being highly dependent on a central server.

D. Ensemble learning and collaborative learning

Ensemble techniques are a ML strategy that integrates many base models to generate a single best prediction model. Some layer splitting algorithms [53], [54] implicitly employ a “divide and ensemble” strategy, i.e., they divide a single layer of the model and then fuse the resulting outputs to improve performance. Dropout [55] can likewise be regarded as an implicit aggregation of many sub- networks inside a single unified network. Slimmable network [56] creates many networks with varying widths from a full network and trains them by sharing parameters to obtain adaptive accuracy-efficiency trade-offs at run time. MutualNet [57] mutually trains these sub-networks to improve the performance of the entire network. Recent works [58] show that ensembles can outperform single models with both higher accuracy and requiring similar or fewer total FLOPs to compute, even when those
individual models’ weights and hyperparameters are highly optimized. Collaborative learning methods [30], [34], [59] can enhance the performance of individual neural networks by training them with some peers or teachers. Zhao et al. [60] shows that increasing the number of networks (ensemble) and collaborative learning can achieve better accuracy-efficiency trade-offs than traditional model scaling methods. Based on this observation, we bring model ensemble and collaborative learning to the FL task.

III. PRELIMINARIES

We leverage the FL paradigm to collaboratively train large convolutional neural networks on resource-constrained devices (that may not be equipped with GPU accelerators) without transmitting the clients’ data to the server. In the following, we first introduce the traditional FL framework in Section III-A. Note that this approach is unsuitable for resource-constrained devices as it requires all clients to train a whole network whose size is usually significant. To this end, we propose a novel theoretical formulation of FL in Section III-B, where each device is only responsible for training a portion of the model, and the ensemble learning paradigm is used to build the whole model.

A. CONVENTIONAL FL FRAMEWORK

Let us consider the baseline version of FL algorithm, Federated Averaging (FedAvg) [35] as described in Algorithm 1. We denote the set of \( K \) mobile devices as \( \mathcal{K} \). Each device \( k \in \mathcal{K} \) participates in training a shared global model \( W \) by using its own dataset \( D_{k \in \mathcal{K}} := \{(X^{(k)}_i, y^{(k)}_i)\}_{i=1}^{N^{(k)}} \), where \( X^{(k)}_i \) is the \( i \)-th training sample of device \( k \), \( y^{(k)}_i \) is the corresponding label of \( X^{(k)}_i \), \( y^{(k)}_i \in \{1, 2, \ldots\} \) (a multi-classification learning task), and \( N^{(k)} \) is the cardinality of dataset \( D_{k \in \mathcal{K}}; N = \sum_{k \in \mathcal{K}} N^{(k)} \).

At the start of each training round, each device \( k \) trains a local model \( W^k \) to minimize a loss function \( \mathcal{F}^{(k)}(W) \) as follows

\[
W^{(k)} = \arg \min_{W} \mathcal{F}^{(k)}(W), k \in \mathcal{K}. \tag{1}
\]

The loss function might vary according to the FL algorithm. For example, a typical loss function for a multi-classification task can be defined as

\[
\mathcal{F}^{(k)}(W) = \frac{1}{N^{(k)}} \sum_{i=1}^{N^{(k)}} \ell \left( W; X^{(k)}_i, y^{(k)}_i \right),
\]

where \( \ell \left( W; X^{(k)}_i, y^{(k)}_i \right) \) quantifies the difference between the expected outcome \( y^{(k)}_i \) and the result predicted by the model. After finishing the local training, each device \( k \) uploads its computed update \( W^k \) to the central server, which then uses the weighted averaging method to aggregate and calculate a new version of the global model as follows

\[
W = \frac{1}{N} \sum_{k=1}^{K} N^{(k)} W^{(k)}. \tag{2}
\]

The global model \( W \) is then sent back to all devices for the next communication round until the global learning is completed. Consequently, an FL process of a CNN-based global model \( W \) can be formulated by the following distributed optimization problem

\[
\min_W \mathcal{F}(W) \overset{\text{def}}{=} \min_W \frac{1}{N} \sum_{k=1}^{K} N^{(k)} \mathcal{F}^{(k)}(W),
\]

subject to \( \mathcal{F}^{(k)}(W) = \frac{1}{N^{(k)}} \sum_{i=1}^{N^{(k)}} \ell \left( W; X^{(k)}_i, y^{(k)}_i \right) \).

Most federated optimization approaches such as FedAvg [35], FedProx [14], FedNova [38], and FedMA [37] offer to solve objective function Eq. (3) with local SGD (Stochastic Gradient Descent) variants. These methods focus on communication-efficient training and demonstrate their characteristics with experiments on shallow neural networks or linear models. However, the main drawback of these methods is the incapability to train large CNN at the lightweight edge devices due to resource constraints such as the lack of GPU accelerators and insufficient memory.

B. A NOVEL FL FORMULATION FOR Training LARGE CNNs

To overcome the resource constraints on lightweight edge devices, we propose a novel methodology for the FL optimiza-
Consequently, our proposed FL framework can be theoretically instead of sending the global model by the server to produce the updated global model ensemble model. These ensemble models are then aggregated in each cluster then perform a co-training process to obtain an ensemble of $S$ models. Each cluster $C$ consists of devices $\{W_1, W_2, \ldots, W_S\}$. For each communication round, instead of sending the global model $W$ to each device $k \in K$, the server transfers only a portion of the global ensemble model to corresponding devices in each cluster. The devices in each cluster then perform a co-training process to obtain an ensemble model. These ensemble models are then aggregated by the server to produce the updated global model $W_{en}$. Consequently, our proposed FL framework can be theoretically formulated as follows

$$\begin{align}
\min_{W_{en}} F(W_{en}) & \overset{\text{def}}{=} \min_{W_{en}} \frac{1}{|C|} \sum_{C \in C} |d_C| \cdot F^{(C)}(W_{en}), \\
\text{subject to } F^{(C)}(W_{en}) &= F^{(C)}(E\{W_1, \ldots, W_S\}),
\end{align}$$

(4)

where $F^{(C)}(E\{W_1, \ldots, W_S\})$ is the objective function of cluster $C$.

**Key Insight** The core advantage of the above reformulation is that since the size of a sub-model $W_i$ ($i = 1, \ldots, S$) is multiple orders of magnitude smaller than that of the original model $W$, the edge training is affordable. Moreover, as discussed in [58, 60], instead of using the common practice of tuning a single large model, one can increase the number of networks (ensemble) as a more flexible method for model scaling without sacrificing accuracy. Our experiments prove that employing an ensemble of small models as the global model does not degrade but rather substantially improves FL’s performance.

### IV. The Proposed Framework

Based on the above reformulation, we propose FedDCT, a novel method for distributed training of large-scale neural networks on light-weight edge devices under the FL settings. First, we discuss the model division and co-training techniques in Sections IV-A and IV-B, respectively. We outline the FedDCT workflow based on these methods in Section IV-C, and we then dig into depth about the federated co-training and cluster aggregation procedure in Sections IV-D and IV-E, respectively.

#### A. Model division

Given one large network, we first (channel-wise) divide it according to its width, or more specifically, the parameters or FLOPs of the network. For instance, if we want to divide a network into four small networks, the number of parameters of one small network will become one-fourth of the original. Let $W$ be the original model. As demonstrated in Fig. 3, we want to divide $W$ into an ensemble of $S$ small sub-models $\{W_1, \ldots, W_S\}$. Our goal is to keep the metrics: the total number of parameters and FLOPs roughly unaltered before and after division.

**Parameters.** We view $W$ as a stack of convolutional layers. Following Pytorch’s definition [61], consider a convolutional layer with $M \times M$ as the kernel size, $C_{in}$ and $C_{out}$ as the number of channels of input and output feature maps, respectively, and $d$ as the number of groups (i.e., every $\frac{C_{in}}{d}$ input channels are convolved with its own set of filters, of size $\frac{C_{in}}{d}$). In the case of depthwise convolution [18], $d = C_{in}$. Then, the number of parameters and FLOPs of that convolutional layer can be derived as follows

$$\text{Params: } M^2 \times \frac{C_{in}}{d} \times \frac{C_{out}}{d} \times d,$$

(5)

$$\text{FLOPs: } \left(2 \times M^2 \times \frac{C_{in}}{d} - 1\right) \times H \times W \times C_{out},$$

(6)

where $H \times W$ is the size of the output feature map. Bias is omitted here for brevity. Generally, $C_{out} = t_1 \times C_{in}$, where $t_1$ is a constant. Therefore, the number of parameters is equivalent to

$$\text{Params: } M^2 \times t_1 \times (C_{in})^2 \times \frac{1}{d}.$$  

(7)

Therefore, in order to divide a convolutional layer by a factor of $S$, we just need to divide $C_{in}$ by $\sqrt{S}$ as follows

$$\frac{M^2 \times t_1 \times (C_{in})^2}{S} \times \frac{1}{d} = M^2 \times t_1 \times \left(\frac{C_{in}}{\sqrt{S}}\right)^2 \times \frac{1}{d}.$$  

(8)
one of the factors affecting the effectiveness of an ensemble models. Training loss to mutually exploit knowledge between the sub-initialization for enhancing the ensemble diversity; and co-
our training strategy, including data augmentation and model each sub-network to learn from each other to further boost views of the same data to increase diversity. This allows B. Co-training strategy
will discuss on how to implement this training strategy in a data-privacy aware manner in FL setting. We can guarantee that $h(x, \epsilon_i), h(x, \epsilon_2), \ldots, h(x, \epsilon_k)$ will produce different views of $x$ for corresponding networks in most cases by applying these random data augmentation operations multiple times.

**Mutual knowledge distillation.** We can further boost the ensemble performance by allowing sub-models to learn from each other, thereby avoiding poor performance caused by overly de-correlated sub-networks [30]. Following the co-training assumptions [65], sub-networks are expected to agree on their predictions of $x$ although they see different views of $x$. In particular, we adopt a natural measure of similarity, the Jensen-Shannon divergence [32] between predicted logits of sub-networks, as the co-training objective

$$
\mathcal{L}_{\text{cot}}(p_1, p_2, \ldots, p_S) = H \left( \frac{1}{S} \sum_{k=1}^{S} p_k \right) - \frac{1}{S} \sum_{k=1}^{S} H(p_k),
$$

where $p_k$ depicts the distribution of sub-model $M_k$’s predicted logits for input $h(x, \epsilon_i)$, and $H(p) = \mathbb{E}[-\log(p)]$ represents the Shannon entropy [66] of a distribution $p$. The global objective function is the sum of the classification losses $\mathcal{L}_{ce}$ of small networks and the co-training loss. During inference, we average the outputs before softmax layers as the final ensemble output.

For instance, if we divide a bottleneck layer in ResNet by a factor of 4, it becomes 4 small blocks as

$$
\begin{bmatrix}
1 \times 1, & 64 \\
3 \times 3, & 64 \\
1 \times 1, & 256
\end{bmatrix}
\rightarrow
\begin{bmatrix}
1 \times 1, & 32 \\
3 \times 3, & 32 \\
1 \times 1, & 128
\end{bmatrix}
\quad (9)
$$

Compared to the original block, each small block in Eq. (9) has only a quarter of the parameters and FLOPs. More details about model division is given in the Appendix.

**Regularization** Since the model capacity degrades after partitioning, the regularization components in networks should be adjusted correspondingly. To this end, we change the scale of dropping regularization linearly based on the assumption that model capacity is linearly dependent on network width. Specifically, we divide the dropout’s dropping probabilities [55] and stochastic depth [62] by $\sqrt{S}$ in the experiments. Due to its unclear intrinsic mechanism, the weight decay value is kept unchanged. It is worth noting that the model partition described above is constrained by a number of factors, such as the number of parameters in the original model must be divisible by $S$, and $\sqrt{S}$ must be an integer. In practice, however, these restrictions are not always satisfied. For those cases, we simply round the values $\frac{\text{param}}{S}$ to the nearest integer numbers.

**Concurrent training:** For a large ensemble network to be trained concurrently on multiple devices, we follow the strategy as shown in Fig. 3. In Sections IV-C and IV-D, we will discuss on how to implement this training strategy in a data-privacy aware manner in FL setting.

**B. Co-training strategy**

After dividing, small networks are trained with different views of the same data to increase diversity. This allows each sub-network to learn from each other to further boost individual performance. In the following, we describe in detail our training strategy, including data augmentation and model initialization for enhancing the ensemble diversity; and co-training loss to mutually exploit knowledge between the sub-models.

**Diversity enhancement.** The diversity of the sub-models is one of the factors affecting the effectiveness of an ensemble learning scheme. In general, an increase in diversity help to capture different types of information, hence enhancing the ensemble model’s efficiency. Diversity across sub-models is achieved in our proposed model using two techniques: diversity initialization and data augmentation. To ensure diversity among sub-models, we first initialize them with different weights [31]. Moreover, during training, each sub-model is also provided with a different abstract representation of the same input image. In particular, for each sub-network $W_i$ in Fig. 3, we use a random data transformer with $\epsilon_i$ as the random seed to generate a separate view $h(x, \epsilon_i)$ of the input image $x$. In practice, different data views are generated by randomness in the data augmentation procedure. Besides the commonly used random resize, crop, and flip, we further introduce random erasing [63] and random augmentation [64] in our algorithm design.

**Partitioning the model:** After dividing, small networks and the co-training loss. During inference, we average the outputs before softmax layers as the final ensemble output.

![Diagram](image)

Fig. 3. Model division and concurrent training on multiple devices. This division and co-training approach will not work in a FL environment since devices cannot directly exchange data. Accordingly, we propose a novel Federated Divide and Co-training paradigm.
Algorithm 2 Federated Divide and Co-training (FedDCT). 
S is the set of all devices, W_C is the model of cluster C, N(C) is the number of data samples owned by devices in cluster C, L is the cutting layer, E is the number of local epochs, and η is the learning rate.

1: **procedure** SERVER EXECUTES
2: initialize \( W_{en} = E\{W_{10}, W_{20}, \ldots, W_{S0}\} \)
3: initialize \( W_{en} = E\{W_{10}, W_{20}, \ldots, W_{S0}\} \)
4: for each round \( t = 0, 1, \ldots \) do
5: \( S_t \leftarrow \) (random set of \( K \) clients from \( S \))
6: \( C_t \leftarrow \) (partition the \( S_t \) into to \( C \) clusters)
7: sends \( W_{en} \) to first device in each cluster
8: sends \( W_{en} \) to first device in each cluster
9: for each cluster \( C \in C \) in parallel do
10: \( W_{en}^C \leftarrow \) FedCoTraining \((C, W_{en}^C, W_{pi}^C)\)
11: \( \triangleright \) Update lower ensemble model and upper ensemble model
12: \( W_{en+1} = \sum_{C \in C} N(C) W_{en+1} \)
13: \( \triangleright \) end for
14: end for

end **procedure**

15: **procedure** FedCoTraining \((C, W_{en}, W_p)\) \( \triangleright \) Run on cluster \( C \) of \( S \) devices
16: for each local epoch from 1 to \( E \) do
17: for each device \( c_i \in C (i = 1, 2, \ldots, S) \) do
18: \( a(W_{en}^m), y \leftarrow \) MainDeviceForward \((c_i, W_{en}^m)\)
19: \( W_{en}^p, dA(W_{en}^p) \leftarrow \) ProxyDevicesUpdate
20: \( (a(W_{en}^m), y, C, W_p) \)
21: \( W_{en}^m \leftarrow \) MainDeviceBackProp \((dA(W_{en}^p))\)
22: \( \triangleright \) Sends \( W_{en}^m \) to device \( c_{i+1} \)
23: end for
24: end for
25: \( \triangleright \) Last device \( c_S \) sends \( W_{en}^m \) to server
26: \( \triangleright \) Each device \( c_i \) sends \( W_{en}^p \) to server
27: end procedure

28: \( \triangleright \) end procedure

29: **procedure** MainDeviceForward \((c_i, W_{en})\) \( \triangleright \) Run on device \( c_i \)
30: for each data \( X_j \in c_i \) do
31: \( \{h(X_j, \epsilon_1), h(X_j, \epsilon_2), \ldots, h(X_j, \epsilon_S)\} \leftarrow \) RandAug \((X_j)\)
32: for each view \( h(X_j, \epsilon_k), k = 1, 2, \ldots, S \) do
33: \( a(W_{en}^m) \leftarrow \) Forward propagation with \( h(X_j, \epsilon_k) \) up to layer \( L \) in \( W_{en}^m \)
34: \( \triangleright \) Produce \( S \) abstract representations
35: \( \triangleright \) Send \( a(W_{en}^m) \) to device \( c_k \)
36: \( \triangleright \) end for
37: \( \triangleright \) end for
38: end procedure

39: \( \triangleright \) end procedure

40: **procedure** ProxyDevicesUpdate \((a(W_{en}^m), y, C, W_p)\)
41: for each device \( c_k \in C, k = 1, 2, \ldots, S \) in parallel do
42: \( p_k \leftarrow \) Forward propagation with \( a(W_{en}^m) \) on \( W_p \)
43: computes prediction
44: \( L_{cut}(p_k, y) \leftarrow \) Loss of \( W_p \) concerning \( a(W_{en}^m), y \)
45: Client \( c_k \) sends \( p_k \) to the server and gets \( L_{cut} \)
46: Backpropagates calculates \( \nabla L_{cut}(W_p, a(W_{en}^m)) \)
47: \( \triangleright \) sends \( dA(W_{en}^m) \) to Main Device for
48: \( \triangleright \) MainDeviceBackProp \((dA(W_p))\)
49: \( W_{en}^m \leftarrow W_{en}^m - \eta \nabla L_{cut}(W_{en}^m, a(W_{en}^m)) \)
50: \( \triangleright \) end for
51: end procedure

52: **procedure** MainDeviceBackProp \((dA(W_p))\)
53: for each Gradient \( dA(W_p), g = 1, 2, \ldots, S \) do
54: Backpropagate, calculates gradients \( \nabla \ell_g(W_{en}^m) \) with \( dA(W_g) \)
55: \( W_{en}^m \leftarrow W_{en}^m - \eta \nabla \ell_g(W_{en}^m) \)
56: \( \triangleright \) end for
57: end procedure

58: \( \triangleright \) end procedure

59: **procedure** FedDCT workflow
60: To elaborate, we illustrate the overall workflow in Fig. 2 as well as summarize the procedures of FedDCT in Algorithm 2. First, the global model is channel-wise divided into an ensemble of \( S \) sub-models regarding its parameters and regularization components. The model division is explained in detail in Section IV-A. Additionally, the model division procedure is modified to comply with FL’s privacy standards. At the beginning of each communication round, \( K \) clients are randomly selected to participate in the training procedure. These \( K \) devices are grouped into \( K/S \) clusters of \( S \) devices. To initialize the network, the server sends appropriate portions of the global model to each participating client. Then, each client updates the ensemble model in parallel using only local data from its clients. Specifically, clients from each cluster collaborate to simultaneously train the large ensemble model without exchanging data in a process we refer to as “Federated Co-training”. Using FedAvg, the server aggregates updated ensemble models from all clusters to generate a new global ensemble model. During testing, each client will be responsible for running the whole ensemble model (by running inference on each sub-model sequentially). We take the mean of outputs before softmax layers of all sub-models as the final output of the model.

D. Federated Divide and Co-training

1) Model division in FL setting: Let us refer to the set of \( S \) devices of a cluster \( C \) as \( c_1, c_2, \ldots, c_S \). As stated previously, the original model \( W \) is divided into an ensemble of \( S \) sub-models \( W_{en} = E\{W_{1}, W_{2}, \ldots, W_{S}\} \), that will be concurrently trained by \( S \) devices. The naive approach to co-training would be to send the training data directly from one client to the others in a cluster in order to update the sub-models simultaneously (see Fig. 3). However, this defies FL’s emphasis on privacy, as data is explicitly shared among clients. To reap the benefits of co-training while ensuring client’s data privacy, we propose a novel federated co-training paradigm. This process is illustrated in Fig. 4.

Initially, we define the cut layer as an intermediate layer of \( W_i \) \((i = 1, \ldots, S)\), which layer-wise split \( W_i \) into two portions:
- Lower sub-model \( W_{m} \), consists of only a few first layers of \( W_i \) which are responsible for extracting the abstract representation of the input.
- Upper sub-model \( W_{p} \), includes the remaining layers of the \( W_i \) which are responsible for making the prediction.

At the beginning of each communication round, we choose the first client (device \( c_1 \) ) in each cluster \( C \in C \) as the main client of that cluster. The remaining clients in the cluster are now considered proxy clients. The server sends \( W_{en} = E\{W_{1}, W_{2}, \ldots, W_{S}\} \) to the main client in each cluster \( k \), and \( W_{pi} \), \((i = 1, \ldots, S)\) to the \( i \)th client in each cluster.

2) Federated Co-training: Given a set of input-output pairs \((X_1, y_1)\), where \( X_1 \) is the data sample of main device \( c_1 \), and \( y_1 \) is the corresponding label of \( X_1 \). Instead of sending \( X_1 \) directly to each device in \( \{c_1, c_2, \ldots, c_S\} \) to complete the training process of \( \{W_1, W_2, \ldots, W_S\} \), \( c_1 \) will generate
Fig. 4. Federated Co-training of an ensemble model across $S$ clients in a cluster of FedDCT. Let $x$ be a training data of main device $c_1$, then $c_1$ first creates $S$ augmented data and run forward propagation through $\{W^m_1, W^m_2, \ldots, W^m_s\}$ to obtain $S$ abstract resentations. These representations are then sent to upper sub-models of other clients for collaborative training.

$S$ augmented versions of $X_1$ and execute forward propagation on $\{W^m_1, W^m_2, \ldots, W^m_s\}$ to produce $S$ unique abstract representations of the same input image. In other words, the main client performs forward propagation until the cut layer of $S$ lower sub-models utilizing $S$ different view of the same input image. The activations (smashed data) at cut layers are then sent to $S - 1$ proxy clients in the cluster, with one abstract representation remaining on the main client.

Each device $c_i \in \{c_1, c_2, \ldots, c_s\}$ then completes the forward propagation on corresponding upper sub-model $W^i_p$ with smashed data $a(W^m_i)$ separately in parallel with other clients, and sends its prediction result $p_i$ to the server. This completes a round of forward propagation without sharing raw data. The server then gathers the predictions $\{p_1, p_2, \ldots, p_S\}$ from all devices, and calculate the cross entropy classification loss of the entire ensemble model as $\sum_{i=1}^S L_{ce}(p_i, y_i)$. Furthermore, to improve the generalization performance, we add a regularization term that allows the devices to mutually learn from each other. This regularization term is designed as the Jensen-Shannon (JS) divergence $\text{JS}d_{\lambda}$ among predicted probabilities. The final objective function with $c_1$ as the main client now becomes

$$\mathcal{F}^{(C)}(W_{en}) = \sum_{i=1}^S L_{ce}(p_i, y_i) + \lambda_{cot}L_{cot}(p_1, p_2, \ldots, p_S).$$

where $p_i$ ($i = 1, \ldots, S$) is a vector representing the output of $W^i_p$ on the data sample of main device $c_1$, $L_{cot}$ is the co-training loss (the details of $L_{cot}$ is given in Section IV-B), $\lambda_{cot}$ is the weight factor of $L_{cot}$ and $L_{ce}$ is the cross entropy classification loss. $\mathcal{F}^{(C)}(W_{en})$ is then sent back to all devices. Each device $c_i$ then performs backpropagation on its upper sub-model $W^i_p$. In other words, gradients are back propagated from the last layer to the cut layer of each sub-model at the corresponding client. The gradients at the cut layers $\nabla \ell_i(W^p_i, a(W^m_i))$ are sent back to the main client, which then complete the rest of backpropagation on $\{W^m_1, W^m_2, \ldots, W^m_s\}$. This completes a round of forward and backward pass without looking at each others raw data. This procedure is repeated for a number of epochs. The main client then transfer the updated $W^m_{en} = \mathcal{E}\{W^m_1, \ldots, W^m_s\}$ to the next client in the cluster ($c_2$ in the case of $c_1$). This client will become the new main client, and training will be repeated with a new main client. This procedure proceed until each client in the cluster finish training as the main client once. The overall final objective function with $k^{th}$ client $c_k$ as the main client is

$$\mathcal{F}^{(C)}(W_{en}) = \sum_{i=1}^S L_{ce}(p_i, y_k) + \lambda_{cot}L_{cot}(p_1, p_2, \ldots, p_S).$$

E. Cluster Aggregation

After the training round, the last main client $c_S$ in each cluster $C \in C$ then sends a updated version $W^m_C$ of global lower ensemble model $W^m_{en}$ to the server, and each device $c_i$ in cluster $C$ sends the corresponding updated upper sub-model $W^i_p$ to the server. The server will then ensemble the received weights from the clients to form corresponding ensemble model of each cluster. Specifically, for each cluster $C$, corresponding upper sub-models $W^p_1, \ldots, W^p_s$ are ensembled to generate an upper ensemble model $W^p_C = \mathcal{E}\{W^p_1, \ldots, W^p_s\}$ of that cluster. The server then merges $W^m_C$ and $W^p_C$ to create an ensemble model $W_C$ for each cluster. The server then aggregates the updated ensemble models of all clusters to produce a new global ensemble model:

$$W_{en} = \frac{1}{N} \sum_{C \in C} N^{(C)}W_C,$$

where $N^{(C)}$ is the cardinality of cluster $C$. This concludes a communication round in FedDCT.

V. EXPERIMENTS AND RESULTS

In this section, we extensively evaluate the performance of FedDCT on image classification tasks. The proposed framework achieves higher performance as well as more stable convergence compared with state-of-the-art FL methods [14], [27], [35] on both standard natural datasets [5] and real-world medical image datasets [67]. In the following, we firstly introduce the four image classification datasets used in our experiments and describe how the data is split over
the devices\textsuperscript{2}. We then describe the setup for the experiments and implementation details of the FedDCT on each dataset. After that, we report and compare our proposed FedDCT with current state-of-the-art FL methods using Top-1 accuracy. Finally, we conduct ablation studies to highlight some important properties of the proposed framework.

\subsection*{A. Datasets and experimental settings}

We evaluate the effectiveness of FedDCT on standard image classification datasets including CIFAR-10 \cite{5}, CIFAR-100 \cite{5}, and two real-world medical imaging datasets that are HAM10000 \cite{67} and VAIPE. For each dataset, we use the official training and validation splits as proposed in the original papers \cite{5}, \cite{67} for all our experiments.

\textbf{CIFAR-10} \cite{5}. The CIFAR-10 dataset consists of 60,000 colour images in 10 classes, with 6,000 images per class. The dataset provides 50,000 training images and 10,000 test images with the size of 32\times32 pixels. On CIFAR-10, we consider this experiment as a balanced and IID setting. In particular, all training examples are divided into 20 clients, each containing 2,500 training examples. CIFAR-10 is available at https://www.cs.toronto.edu/ kriz/cifar.html.

\textbf{CIFAR-100} \cite{5}. To validate FedDCT further on a dataset with a larger number of classes, we conduct experiments on the CIFAR-100. The CIFAR-100 dataset is similar to CIFAR-10, except that it has 100 categories each contains 600 images. The number of images for training and testing per class is 500 and 100, respectively. The training images are equally partitioned into 20 clients, each contains 2,500 samples. This setting also is a balanced IID setting. CIFAR-100 dataset is available at https://www.cs.toronto.edu/ kriz/cifar.html.

\textbf{HAM10000} \cite{67}. HAM10000 ("Human Against Machine with 10,000 training images") is a large-scale database of multi-source dermatoscopic images of common pigmented skin lesions collected from different populations, acquired and stored by different modalities. The main task of this dataset is to identify critical diagnostic categories in the realm of pigmented lesions. In total, the HAM10000 dataset contains 10,015 published dermatoscopic images; each has a dimension of 600 \times 450 pixels (see examples in Fig. 5). As a real-world medical imaging dataset, HAM10000 is a high-class imbalanced dataset, with the minority class are rare diseases. In our experiments, we randomly use 80\% of the images for training and the rest of 20\% for evaluation. Similar to CIFAR-10 and CIFAR-100, we partition the training dataset into 20 clients, each contains about 500 images. The HAM10000 dataset is available for download at https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000.

\textbf{Pill Image Identification Dataset (VAIPE).} To further evaluate the robustness of the proposed method in a real-world setting, we have collected and annotated a large-scale pill image identification dataset. The dataset, which we call VAIPE, is a real-world pill image dataset consisting of 98 pill categories. The main task of the VAIPE dataset is to classify pill images correctly into classes. In our experiments, 8,161 images are used as training examples, and 1,619 images are used for evaluation. The VAIPE dataset contains pill images taken in real-world scenarios under unconstrained environments, including different lighting, backgrounds, and devices. Fig. 6 shows several representative examples from the VAIPE dataset. For VAIPE, we also equally divide the training samples to 20 clients. Experiments on such a dataset allow us to investigate the robustness of FedDCT in real-world scenarios. To make this work reproducible and encourage new advances, we make all images and annotations of the VAIPE dataset publicly available as a part of a bigger dataset that we released on our project website at https://smarthealth.vinuni.edu.vn/resources/.

\subsection*{B. Implementation details}

This section presents our implementation details for experiments across all four datasets. WRN-16-8 \cite{45}, ResNet-110 \cite{2}, and WRN-50-2 \cite{45} are used as baseline networks. To implement these networks, we follow the original implementations described in \cite{2} \cite{45}. For all experiments, the networks first initialize the weights using the Kaiming initialization technique \cite{68}. We then train the networks using Nesterov accelerated SGD optimizer with a momentum of 0.9. We apply the warm-up and cosine learning rate decay policy \cite{69}. To verify the effectiveness of the proposed method, we compare the performance of FedDCT (S = 4) with state-of-the-art FL approaches, including FedProx \cite{14}, FedAvg \cite{35}, SplitFed \cite{27}. To this end, we reproduce these approaches on four datasets. For a fair comparison, we apply the same training methodologies and hyper-parameter settings as described in the original papers \cite{14}, \cite{27}, \cite{35}. We also compare the proposed framework’s performance with the centralized training approach. In all experiments, the number of local epochs in each round is set to 1. In our implementations, all deep networks are implemented and trained using the PyTorch framework (version 1.10.2) on a machine with 8\times NVIDIA GTX 3090 24GB GPU.

\textsuperscript{2}In the following, we use the terms “device” and “client”, interchangeably.
**Implementation details for CIFAR-10.** For the CIFAR-10 dataset, we use WRN-16-8 [45] as the baseline model and train FedDCT (S = 4) for 300 communication rounds. In the training stage, all images are fed into the network with a standard size of 32 × 32 pixels and a batch-size of 128. The initial learning rate was initially set to 0.1 and then gradually decreased using cosine learning rate decay policy [69]. In addition, common data augmentation techniques including random crop, random flipping, normalization, random erasing, mixup, and RandAugment [64] are used during the training stage. We then train FedAvg [35], FedProx [11], and SplitFed [27] using the same training setting as FedDCT. All networks are trained with a 100% participation rate of 20 clients.

**Implementation details for CIFAR-100.** For the CIFAR-100 dataset, we use ResNet-110 [2] as the baseline model and train FedDCT (S = 4) for 650 communication rounds. The network is trained on 32 × 32 images with a batch size of 128. The learning rate scheduler and data augmentation are similar to that used in the CIFAR-10 dataset. Also, FedAvg [35], FedProx [11], and SplitFed [27] then are trained on CIFAR-100 dataset using the same training procedures as the original works [11], [27], [35].

**Implementation details for HAM10000.** For the HAM10000 dataset, we use WRN-50-2 [45] as the baseline model and train FedDCT (S = 4) for 200 communication rounds. The network is trained on 64 × 64 images with a batch size of 128. To reduce overfitting, we use different data augmentation strategies such as random vertical flip, random horizontal flip, random adjust sharpness, random auto contrast, rotation, and center crop. The initial learning rate is initially set to 0.01 and then decreases by the cosine learning rate decay policy [69]. Similarity, we train FedAvg [35], FedProx [11], and SplitFed [27] for 200 communication rounds using the same training setting as the original works [11], [27], [35].

**Implementation details for VAIPE.** For the VAIPE dataset, we also use WRN-50-2 [45] as the baseline model and then train the proposed FedDCT (S = 4) for 350 communication rounds. The proposed framework is trained on pill images with the dimension of 256 × 256 pixels and with a batch size of 32 samples. Data augmentation strategies on the VAIPE dataset are similar to that used in the CIFAR-10, and CIFAR-100 datasets, except that the label smoothing technique is applied. For FedAvg [35], FedProx [14], SplitFed [27], we use 20 clients with 100% participation rate on IID data. All other settings are the same as described in the original works [11], [27], [35].

**C. Accuracy evaluation**

We report in this section the classification performance of FedDCT (S = 4) on CIFAR-10, CIFAR-100, HAM10000, and VAIPE datasets, respectively. We then compare our approach with recent state-of-the-art FL approaches, i.e., FedAvg [35], FedProx [14], and SplitFed [27]. We also provide a comprehensive analysis of the convergence of FedDCT to highlight the effectiveness of our method in terms of reducing the client’s memory requirements and communication costs.

1) **Model accuracy:** Table II shows quantitative results of the proposed framework on CIFAR-10, CIFAR-100, HAM10000, and VAIPE datasets. Experimental results show that FedDCT is able to train large deep CNN models and reach a high level of performance. For instance, when using our FedDCT training strategy, the WRN-16-8 network achieves 95.99% top-1 accuracy on the CIFAR-10 dataset. ResNet-100 achieves 77.38% top-1 accuracy on CIFAR-100 dataset. Meanwhile, WRN-50-2 reports a top-1 accuracy of 78.62% and 97.75% on HAM10000 and VAIPE datasets, respectively. We observe that the high-performing performances are demonstrated consistently on a variety of model architectures and datasets. Fig 7 shows the top-1 accuracy of all learning models on four datasets over communication rounds.

2) **Comparison with the state-of-the-arts:** Table II also provides the performance of the proposed model and previous methods on four image classification datasets. Concerning the CIFAR-10 dataset, FedDCT sets a new state-of-the-art performance with an accuracy of 95.99%. This result significantly outperforms other state-of-the-art FL method including...
SplitFed (91.98% top-1 accuracy), FedAvg (92.12% top-1 accuracy), and FedProx (89.02% top-1 accuracy). Compared to the second-best approach (i.e., FedAvg), our FedDCT surpasses it by 3.87% top-1 accuracy. On the CIFAR-100 dataset, FedDCT obtains a top-1 accuracy of 77.38%. Compared with FedProx, FedAvg, and SplitFed, our method achieves much better performance, surpassing them by at least +10.46%, +3.71%, and +4.46% in top-1 accuracy, respectively. On skin lesions dataset HAM10000, FedDCT reaches a top-1 accuracy of 78.62% and significantly surpasses FedAvg, FedProx and SplitFed by +1.49%, +0.57% and +2.48%, respectively. Concerning the pill image VAIPE dataset, FedDCT also outperforms all other state-of-the-art approaches and sets new state-of-the-art. Specifically, compared to the competing method SplitFed, our FedDCT significantly surpasses it by +5.9% top-1 accuracy while surpassing FedProx and FedAvg by +11.55% and +8.96% top-1 accuracy, respectively. Experimental results on HAM10000 and VAIPE datasets suggest that the proposed FedDCT is a reliable FL approach to solving real-world applications.

3) Convergence analysis: To demonstrate the effectiveness of the proposed FedDCT framework in reducing local updates, we provide the number of communication rounds and speedup relative to the baseline FL methods. As shown in Table III, FedAvg achieves a test accuracy of 80% on the VAIPE dataset after 195 communication rounds (base speed). Meanwhile, the proposed approach FedDCT achieves a similar test accuracy after only 67 communication rounds (0.3 x base). This communication cost is much cheaper than FedProx and SplitFed, which need 221 (1.3 x base) rounds and 138 (0.7 x base) rounds, respectively. Besides, FedDCT reaches an accuracy of 85% in only 86 communication rounds. Meanwhile, FedAvg needs 260 communication rounds. Table III also shows that FedDCT converges much faster than existing approaches such as FedProx and SplitFed. We observed the same convergence behaviors to reach 85%, 90%, and 95% on the test data of VAIPE. In conclusion, our experimental results indicate that the proposed FedDCT requires fewer communication rounds to achieve the same level of accuracy compared to other state-of-the-art techniques.

This superiority in accuracy and convergence rate can be attributed to two things. First, recent research has demonstrated that ensemble model can outperform a single model with same parameters [58] and co-training between sub-models improves ensemble performance (see Section V-E5). Similarly, FedDCT outperforms FedAvg with the same computation cost each communication round (despite training on more views of data, each view is trained on only a sub-model, hence similar total training cost). Furthermore, due to the fact that each cluster of $S$ clients only trains a single ensemble model, the number of models in the aggregation phase is greatly decreased, hence lowering accuracy loss due to aggregation.

### D. Efficiency evaluation

1) Computation cost: Split Learning [29], and SplitFed [27] reduce the client’s computation by offloading the majority of the training workload to the server-side. Both of these approaches assume that the training cost at the server-side is negligible. While this might be true for laboratory settings where the dataset size is small, recent trends indicate that real-world datasets may be exceedingly huge [70], making training them extremely expensive. For example, a single training run of the GPT-3 [71] model will take 355 V100 GPU-years and cost 4.6 million dollars. To evaluate the computational cost caused by the training process at the server-side, we estimate the number of FLOPs (floating-point operations) required by our method and compare it to those of other approaches. We report the results on CIFAR 100 dataset with 60000 images of the size 32 x 32 with 650 communication rounds in Fig. 8(a). As shown, compared to SplitFed and Split Learning, our method requires no training overhead at the server-side.

Furthermore, in the case of SplitFed, where the training on the server side is run in parallel, the amount of VRAM required at the server increases linearly with the number of clients. In particular, SplitFed would require over 400GB of memory while training a ResNet-110 model on CIFAR-100 with batch size of 128 using 100 clients. Compared to other methods, FedDCT does not require the server to have GPUs (see Fig. 8(b)).

2) Client memory requirement: This section studies the memory requirements of FedDCT under various settings of the split factor $S$, and compare them to that of FedAvg. As demonstrated in Fig. 9, the greater the number of the split factor, the less memory required to train the model on-device. Experiments on CIFAR-100 using Resnet-110 model with a batch size of 128 demonstrates that while training the entire model on clients using FedAvg would consume $\approx 4.4$ GB
memory, FedDCT consumes substantially less, for example, less than 1GB when $S = 16$.

3) Communication cost: Let $K$ be the number of clients, $p$ be the total data size of all clients, $Q$ be the size of the smashed layer, $|W|$ be the size of the entire model, and $\beta$ be the ratio of the lower part to the entire model, i.e., $|W^m| = \beta |W|$. Note that, for each complete forward and backward propagation in FedDCT, a client may take one among two roles: main client and proxy client. The main client is in charge of generating the smashed data and performing a local model update on both lower and upper sub-models. In the meanwhile, the proxy clients receive the smashed data from the main client and perform the model update on only the upper sub-model. Accordingly, the communication cost for the owner of the training sample during each training iteration can be calculated as follows:

$$\text{Comm. (main)} : (S-1)\left(\frac{2p}{K}\right)\left(\frac{Q}{S}\right) + 2\beta |W| + 2\left(1 - \beta\right)\frac{|W|}{S}. \quad (14)$$

On the other hand, the communication cost for each proxy client is determined as follows:

$$\text{Comm. (proxy)} : \left(\frac{2p}{K}\right)\left(\frac{Q}{S}\right) + 2\left(1 - \beta\right)\frac{|W|}{S}. \quad (15)$$

During each communication round, each client will be the main client one time and proxy client for $S - 1$ times. Consequently, the total communication cost of each client in one communication round is

$$\text{Comm. (total)} : (S-1)\left(\frac{4p}{K}\right)\left(\frac{Q}{S}\right) + 2|W|. \quad (16)$$

The result of communication efficiency of FedDCT compared to FedAvg is presented in Fig. 10. As shown in the figure, FedDCT has slight communication overhead per epoch compared to FedAvg due to exchange of smashed data and gradients between clients, but this overhead is not noticeable with a large number of clients. Furthermore, as we have observed in Table III, FedDCT achieves targeted accuracies much faster compared to FedAvg, hence requires less total communication round, reducing total communication cost.

E. Ablation studies

In this section, we further conduct additional ablation studies to investigate the critical properties of the proposed FedDCT. Concretely, we conduct a performance comparison between FedDCT and centralized training. We then study the influence of the number of splits on learning performance. In addition, we also study the performance change over the number of clients, on non-IID dataset and the impact of co-training. We report ablation study results on CIFAR-10 and VAIPE datasets.

1) Comparison to centralized training: To further investigate the learning performance of FedDCT, we compare the proposed method and a centralized data training method, where the training data is gathered, and the entire training process executes at the central server [72] on two datasets, CIFAR-10 and VAIPE. In particular, we train WRN-16-8 and WRN-50-2 on training examples of these datasets in a centralized manner. We report the test sets’ performance and compare the centralized training approach and FedDCT $(S=4)$ in Table IV. Specifically, FedDCT $(S=4)$ achieves 95.99% top-1 accuracy on CIFAR-10 and 97.75% on VAIPE. Meanwhile, the centralized training models report an accuracy of 97.31% on CIFAR-10 and 98.58% on VAIPE. These results indicate that our FedDCT reaches almost the sample level of performance compared to centralized training scenarios. Fig. 11 shows the convergence curves of FedDCT and the centralized training approach on the VAIPE dataset.

2) Effect of number of splits: In this experiment, we investigate the effect of the number of splits on the performance of FedDCT. The experiment is conducted on the CIFAR-10 dataset, with the number of splits is set at 2, 4, 8, 16, and 32, respectively. To this end, we use WRN-16-8 as the baseline network. The experimental settings and training methodology are similar to our main experiments described in Section V-B. Learning behaviors of FedDCT with a different number of splits are shown in Fig. 12. We find that the WRN-16-8 achieves the highest top-1 accuracy with the number of splits equal to 16, with the performance of FedDCT continuously increases when the number of splits increases from 2 to 16.
However, a significant drop in performance is observed when we increase the number of splits to 32. Explanation: The sub-networks may be too thin to guarantee sufficient model capacity.

3) Effect of number clients: We study the effect of the number of clients on the performance of FedDCT. The proposed method is trained and evaluated on the CIFAR-10 dataset with a number of clients of 20, 40, 60, 80, and 100, respectively. Fig. 13 shows the effect of various number clients on FedDCT (S=4) performance on CIFAR-10 dataset. We found that the use of 20 clients allows FedDCT to achieve the highest top-1 accuracy (95.87%). We also observe that the model’s performance decreases when the number of clients increases. Our main observation is that usually, the convergence slows down, and performance degrades with the increase in the number of users within the observation window of the global epochs. Furthermore, all our DCML approaches show similar behavior over the various number of users (clients). This behavior is also observed in other FL techniques.

4) Non-IID datasets: To create a non-IID dataset, CIFAR 10 is randomly divided into the given number of clients (see Fig. 14), where each client can have a different number of classes and samples. This type of dataset replicates the real-world scenario where clients can have different types of images. We report the test sets’ performance of FedDCT (S=4) compared to FedAvg, SplitFed and FedProx in Fig. 15. In non-IID settings, FedDCT shows a superior performance compared to other methods, i.e., achieving 94.01% top-1 accuracy.

5) Influences of different co-training components: Table V shows the impacts of varying weight initialization, various weight factor values of co-training loss in Equation 12, and various diverse data perspectives. Using different data transformers (0.30%↑) and co-training loss (0.68%↑) can help the model improve performance. Given the strong baseline, this improvement is also noteworthy. In this work, we simply apply a straightforward co-training strategy and don’t further explore this topic. Other, more complicated co-training or mutual learning techniques do exist. For instance,
to improve the performance of the entire network, MutualNet [57] creates numerous networks with different widths from a single full network, feeds them images at varied resolutions, and trains them jointly in a weight-sharing manner. More elaborate co-training methods are left as future work since we primarily concentrate on demonstrating that co-training can help enhance overall performance.

VI. CONCLUSION AND FUTURE WORKS

Recent works on FL motivated us to find a robust solution for training high-performing, large-scale deep neural networks in edge computing devices. This paper proposed FedDCT, a new approach for training large deep neural networks in FL settings. The proposed learning framework allows dividing large CNN networks into sub-networks and co-training them with local clients in parallel. To the best of our knowledge, this is the first work that enables the training of large deep learning networks in edge computing devices in FL settings. Extensive experiments on various benchmarks show that the proposed FedDCT reduces the number of local updates compared with the state-of-the-art FL methods and makes a new state-of-the-art for the image classification task. We also showed that FedDCT could provide a higher level of performance over baseline protocols for training diverse machine learning models on real-world medical imaging data.

Our work faces a few limitations. First, the proposed method applies to CNN models. However, our dividing and co-training strategy are hard to apply for time series learning architectures such as recurrent neural networks [73] (RNN) or famous segmentation networks such as U-Net and its variants [74]. Second, the slowest client determines the pace of forward propagation and backpropagation in a cluster. Nonetheless, this is an issue faced by FedAvg [35], SplitFed [27], and other variants [29]. Several client selection methods such as [75], [76] have been proposed to solve this problem and can be incorporated into FedDCT, however this is outside the scope of this work. By allowing a client to train only a portion of the model, we assume that the client will be able to complete the training process. Additionally, considering the heterogeneous resources between the devices, an interesting future work is to divide the global model into unequal partitions and assign the larger sub-model to more capable clients. Another interesting direction is to combine the proposed FedDCT framework with state-of-the-art robust aggregation protocols (e.g., [37], [38]) users while ensuring privacy. For instance, if intra-cluster aggregation is carried out first, and then the complete model is transmitted from the cluster head to the server, the communication efficiency may be improved, especially when some users in the cluster are far away from the server [77].

ACKNOWLEDGMENTS

This work was funded by Vingroup Joint Stock Company and supported by Vingroup Innovation Foundation (VINIF) under project code VINIF.2021.DA00128 and also supported by JSPS KAKENHI Grant Numbers JP21K17751. We thank all our collaborators who participated into the collection and annotation of the VAIPE dataset.

APPENDIX

DETAILS OF DIVIDING LARGE MODEL

Our dividing method is robust and can apply to most modern CNNs. In this section, we consider some of the most common architectures. Here \( S \) is the split factor (the number of small networks after dividing).

ResNet For CIFAR-10 and CIFAR-100 datasets, the input channel numbers of the three blocks are as follows

\[
\text{baseline : } [16, 32, 64], \\
S = 2 : [12, 24, 48], \\
S = 4 : [8, 16, 32], \\
S = 8 : [6, 12, 24], \\
S = 16 : [4, 8, 16], \\
S = 32 : [3, 6, 12].
\] (17)

Wide-ResNet The input channel is divided similar to ResNet. If the widen factor is \( f_w \), the new widen factor after division is \( f_w^* = \max \left( \frac{f_w}{\sqrt{S}} + 0.4, 1.0 \right) \). (18)

ResNeXt [53] Assume the original cardinality (groups in convolution) is \( f_c \), and the new cardinality is \( f_c^* \), in which

\[
f_c^* = \max \left( \frac{f_c}{\sqrt{S}}, 1.0 \right).
\] (19)

EfficientNet [78] The number of output channels and blocks in the first convolutional layer of the EfficientNet baseline are as follows

\[
\text{baseline : } [32, 16, 24, 40, 80, 112, 192, 320, 1280], \\
S = 2 : [24, 12, 16, 24, 56, 80, 136, 224, 920], \\
S = 4 : [16, 12, 16, 20, 40, 56, 96, 160, 640].
\] (20)

DenseNet [79] If DenseNet’s growth rate is \( f_g \), then the new growth rate after division \( f_g^* \) is

\[
f_g^* = \frac{1}{2} \times 2 \times \frac{f_g}{\sqrt{S}}.
\] (21)

REFERENCES

[1]. J. Hu et al., “Squeeze-and-excitation networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[2]. K. He et al., “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[3]. J. Deng et al., “Imagenet: A large-scale hierarchical image database,” in Conf. on comp. vision and pattern recogn. IEEE, 2009, pp. 248–255.

[4]. G. Huang et al., “Densely connected convolutional networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[5]. A. Krizhevsky et al., “Cifar-10 (canadian institute for advanced research),” URL: http://www.cs.toronto.edu/~kriz/cifar.html

[6]. A. Paszke et al., “Pytorch: An imperative style, high-performance deep learning library,” in Advances in Neural Information Processing Systems 32. Curran Associates, Inc., 2019, pp. 8024–8035.

[7]. “Antutu,” “global users preferences for android phones, q1 2022,” https://www.antutu.com/en/doc/127152.htm, 2022. [12 June 2022].

[8]. P. Kairouz et al., “Advances and open problems in federated learning,” Foundations and Trends® in Machine Learning, vol. 14, no. 1–2, pp. 1–210, 2021.
K. Bonawitz et al., “Towards federated learning at scale: System design,” Proceedings of Machine Learning and Systems, vol. 1, pp. 374–388, 2019.

Q. Yang et al., “Federated learning,” Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 13, no. 3, pp. 1–207, 2019.

T. Li et al., “Federated learning: Challenges, methods, and future directions,” IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 50–60, 2020.

D. C. Nguyen et al., “Federated learning for internet of things: A comprehensive survey,” IEEE Communications Surveys & Tutorials, 2021.

L. U. Khan et al., “Federated learning for internet of things: Recent advances, taxonomy, and open challenges,” IEEE Communications Surveys & Tutorials, 2021.

T. Li et al., “Federated optimization in heterogeneous networks,” Proceedings of Machine Learning and Systems, vol. 2, pp. 429–450, 2020.

N. S. Sohoni et al., “Low-memory neural network training: A technical report,” arXiv preprint arXiv:1904.10631, 2019.

J. A. Dunnam et al., “Assessment of convolutional neural networks for automated classification of chest radiographs,” Radiology, vol. 290, no. 2, pp. 537–544, 2019.

D. Yi et al., “Optimizing and visualizing deep learning for benign/malignant classification in breast tumors,” arXiv preprint arXiv:1705.08682, 2017.

W. Liu et al., “Ssd: Single shot multibox detector,” in European conference on computer vision. Springer, 2016, pp. 21–37.

K. Hara et al., “Learning spatio-temporal features with 3d residual networks for action recognition,” in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 3154–3160.

Z. Qiu et al., “Learning spatio-temporal representation with pseudo-3d residual networks,” in proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5533–5541.

S. Han et al., “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” arXiv preprint arXiv:1510.00419, 2015.

M. Zhu et al., “To prune, or not to prune: exploring the efficacy of pruning for model compression,” arXiv preprint arXiv:1710.01878, 2017.

M. Wahib et al., “Scaling distributed deep learning workloads beyond the memory capacity with karma,” in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–15.

T. Chen et al., “Training deep nets with sublinear memory cost,” 2016. URL: https://arxiv.org/abs/1604.06174

M. Kusumoto et al., “A graph theoretic framework of recomputation algorithms for memory-efficient backpropagation,” in Advances in Neural Information Processing Systems. H. Wallach et al., Eds., vol. 32. Curran Associates, Inc., 2019.

P. Jain et al., “Checkmate: Breaking the memory wall with optimal tensor rematerialization,” in Proceedings of Machine Learning and Systems, I. Dhillon et al., Eds., vol. 2, 2020, pp. 497–511.

C. Thapa et al., “Splitted: When federated learning meets split learning,” arXiv preprint arXiv:2004.12088, 2020.

C. He et al., “Group knowledge transfer: Federated learning of large cnns at the edge,” Advances in Neural Information Processing Systems, vol. 33, pp. 14198–14209, 2020.

M. G. Piorot et al., “Split learning for collaborative deep learning in healthcare,” arXiv preprint arXiv:1912.12115, 2019.

S. Qiao et al., “Deep co-training for semi-supervised image recognition,” in Proceedings of the european conference on computer vision (eccv), 2018, pp. 135–151, 2020.

H. Li et al., “Research and development of neural network ensembles: a survey,” Artificial Intelligence Review, vol. 49, no. 4, pp. 455–479, 2018.

M. Menéndez et al., “The jensen-shannon divergence,” Journal of the Franklin Institute, vol. 334, no. 2, pp. 307–318, 1997.

G. Hinton et al., “Distilling the knowledge in a neural network,” arXiv preprint arXiv:1503.02531, vol. 2, no. 7, 2015.

Y. Zhang et al., “Deep mutual learning,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4320–4328.

B. McMahan et al., “Communication-efficient learning of deep networks from decentralized data,” in Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282, 2015.

M. Asad et al., “Fedopt: Towards communication efficiency and privacy preservation in federated learning,” Applied Sciences, vol. 10, no. 8, p. 2864, 2020.

H. Wang et al., “Federated learning with matched averaging,” arXiv preprint arXiv:2002.06440, 2020.

J. Wang et al., “Tackling the objective inconsistency problem in heterogeneous federated optimization,” Advances in neural information processing systems, vol. 33, pp. 7611–7623, 2020.

C. He et al., “Fednias: Federated deep learning via neural architecture search,” 2021.

J. Bernstein et al., “signsgd: Compressed optimisation for non-convex problems,” in International Conference on Machine Learning. PMLR, 2018, pp. 560–569.

J. Wangni et al., “Gradient sparsification for communication-efficient distributed optimization,” Advances in Neural Information Processing Systems, vol. 31, 2018.

J. So et al., “Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning,” IEEE Journal on Selected Areas in Information Theory, vol. 2, no. 1, pp. 479–489, 2021.

Y. Lin et al., “Deep gradient compression: Reducing the communication bandwidth for distributed training,” arXiv preprint arXiv:1712.01887, 2017.

Y. Huang et al., “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” in Advances in Neural Information Processing Systems, H. Wallach et al., Eds., vol. 32. Curran Associates, Inc., 2019.

S. Zagoruyko et al., “Wide residual networks,” arXiv preprint arXiv:1605.07146, 2016.

K. Huang et al., “Efficient quantization for neural networks with binary weights and low bitwidth activations,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, 2019, pp. 3854–3861.

Z. Cai et al., “Deep learning with low precision by half-wave gaussian quantization,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5918–5926.

W. Song et al., “Resiliency of deep neural networks under quantization,” arXiv preprint arXiv:1511.06488, 2015.

M. Zhu et al., “To prune, or not to prune: exploring the efficacy of pruning for model compression.” URL: https://arxiv.org/abs/1710.01878

S. Lym et al., “Prunetrain: Fast neural network training by dynamic sparse model reconfiguration,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC ’19, 2019.

P. Micikevicius et al., “Mixed precision training,” 2017. URL: https://arxiv.org/abs/1710.03740

A. Hard et al., “Federated learning for mobile keyboard prediction,” arXiv preprint arXiv:1811.03604, 2018.

S. Xie et al., “Aggregated residual transformations for deep neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

H. Zhang et al., “Resnest: Split-attention networks,” arXiv preprint arXiv:2004.08925, 2020.

N. Srivastava et al., “Dropout: a simple way to prevent neural networks from overfitting,” The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

J. Yu et al., “Slimmable neural networks,” arXiv preprint arXiv:1812.08928, 2018.

T. Yang et al., “Mutualnet: Adaptive convnet via mutual learning from network width and resolution,” in European conference on computer vision. Springer, 2020, pp. 299–315.

D. Kondratyuk et al., “When ensembling smaller models is more efficient than single large models,” arXiv preprint arXiv:2005.00570, 2020.

Q. Guo et al., “Online knowledge distillation via collaborative learning,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11 020–11 029.

S. Zhao et al., “Towards better accuracy-efficiency trade-offs: Divide and co-training,” arXiv preprint arXiv:2011.14660, 2020.

“Conv2d - pytorch 1.12 documentation,” https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html, accessed: 2022-09-25.

G. Huang et al., “Deep networks with stochastic depth,” in European conference on computer vision. Springer, 2016, pp. 646–661.

Z. Zhong et al., “Random erasing data augmentation,” in Proceedings of the AAAI conference on artificial intelligence, vol. 34, no. 07, 2020, pp. 13 001–13 008.

E. D. Cubuk et al., “Randaugment: Practical automated data augmentation with a reduced search space,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020, pp. 702–703.
The Bell Conference on Advanced Video and Signal-Based Surveillance (AVSS).

He is the author of various scientific articles published in about 20 conferences and journals, such as Computer Vision and Image Understanding, Neurocomputing, Scientific Data (Nature), International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Medical Imaging with Deep Learning (MIDL), IEEE International Conference on Image Processing (ICIP), and IEEE International Conference on Computer Vision (ICCV). His research interests include computer vision, machine learning, medical image analysis, and their applications in smart healthcare. He is the author and co-author of 30 scientific articles appeared in about 20 conferences and journals, such as Computer Vision and Image Understanding, Neurocomputing, Scientific Data (Nature), International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Medical Imaging with Deep Learning (MIDL), IEEE International Conference on Image Processing (ICIP), and IEEE International Conference on Computer Vision (ICCV). He is also currently serving as Reviewer for MICCAI, ICCV, CVPR, IET Computer Vision Journal (IET-CVI), IEEE Journal of Biomedical and Health Informatics (JBHI), IEEE Journal of Selected Topics in Signal Processing, and Scientific Reports (Nature). Before joining VinUniversity, he worked at the Vingroup Big Data Institute (VinBigData), as a Research Scientist and the Head of the Fundamental Research Team. With his position, he led several research projects on medical AI, including collecting various types of medical data, managing and annotating data, and developing new AI solutions for medical analysis.

Kok-Seng Wong (Member, IEEE) received his first degree in Computer Science (Software Engineering) from the University of Malaya, Malaysia in 2002, and an M.Sc. (Information Technology) degree from the Malaysia University of Science and Technology (in collaboration with MIT) in 2004. He obtained his Ph.D. from Soongsil University, South Korea, in 2012. He is currently an Associate Professor in the College of Engineering and Computer Science, VinUniversity. Before joining VinUniversity, he taught computer science subjects (undergraduate and postgraduate) in Kazakhstan, Malaysian, and South Korean universities for the past 17 years. Dr. Wong aims to bring principles and techniques from cryptography to the design and implementation of secure and privacy-protected systems. He has published 60 articles in journals and conferences in his research field. To this end, he conducts research that spans the areas of security, data privacy, and AI security while maintaining a strong relevance to the privacy-preserving framework.

Quan Nguyen received his Engineering degree in Biomedical Engineering at Hanoi University of Science of Technology (HUST), Vietnam in 2021, and B.Sc. in Computer Science at HUST in 2022. He is currently a Master Student at Department of Informatics, Technical University Munich and Research Assistant at VinUni-Illinois Smart Health Center (VISHC). His research interests include Privacy-Preserving Collaborative Deep Learning and Edge AI for Computer Vision. Before joining VISHC, he worked as an AI engineer at Viettel High Technology Industries Corporation where he developed Computer Vision Algorithms for smart camera systems, including People detection using Fish-eye Camera, face recognition algorithm for edge devices and high-speed AI Camera for traffic surveillance. He is the author of various scientific articles published in reputable conferences on surveillance systems such as IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS).
Truong Thao Nguyen  Dr. Truong Thao Nguyen received the BE and ME degrees from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2011 and 2014, respectively. He received the Ph.D. in Informatics from the Graduate University for Advanced Studies, Japan in 2018. He is currently working at Digital Architecture Research Center, at National Institute of Advanced Industrial Science and Technology (AIST), where he focuses on the topics of High Performance Computing system, Distributed Deep Learning and beyond.

Minh Do (Fellow, IEEE) was born in Vietnam in 1974. He received the B.Eng. degree in computer engineering from the University of Canberra, Australia, in 1997, and the Dr.Sc. degree in communication systems from the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland, in 2001. Since 2002, he has been on the Faculty of the University of Illinois at Urbana-Champaign (UIUC), where he is currently the Thomas and Margaret Huang Endowed Professor in signal processing and data science with the Department of Electrical and Computer Engineering, and holds affiliate appointments with the Coordinated Science Laboratory, Department of Bioengineering, and the Department of Computer Science, Beckman Institute for Advanced Science and Technology. From 2020 to 2021, he is on leave from UIUC to serve as the Vice Provost for VinUniversity, Vietnam. Dr. Do was a member of several IEEE technical committees on signal processing. He was elected as a fellow of IEEE in 2014 for his contributions to image representation and computational imaging. He received the Silver Medal from the 32nd International Mathematical Olympiad in 1991, the University Medal from the University of Canberra in 1997, the Doctorate Award from the EPFL in 2001, the CAREER Award from the National Science Foundation in 2003, the Xerox Award for Faculty Research from UIUC in 2007, and the Young Author Best Paper Award from IEEE in 2008. He has contributed to several tech-transfer efforts, including as the Co-Founder and the CTO of Personify, and the Chief Scientist of Misfit. He was an Associate Editor of the IEEE Transactions on Image Processing.