EcoFed: Efficient Communication for DNN Partitioning-based Federated Learning

Di Wu, Rehmat Ullah, Philip Rodgers, Peter Kilpatrick, Ivor Spence, and Blesson Varghese

Abstract—Efficiently running federated learning (FL) on resource-constrained devices is challenging since they are required to train computationally intensive deep neural networks (DNN) independently. DNN partitioning-based FL (DPFL) has been proposed as one mechanism to accelerate training where the layers of a DNN (or computation) are offloaded from the device to the server. However, this creates significant communication overheads since the intermediate activation and gradient need to be transferred between the device and the server during training. While current research reduces the communication introduced by DNN partitioning using local loss-based methods, we demonstrate that these methods are ineffective in improving the overall efficiency (communication overhead and training speed) of a DPFL system. This is because they suffer from accuracy degradation and ignore the communication costs incurred when transferring the activation from the device to the server. This article proposes EcoFed – a communication efficient framework for DPFL systems. EcoFed eliminates the transmission of the gradient by developing pre-trained initialization of the DNN model on the device for the first time. This reduces the accuracy degradation seen in local loss-based methods. In addition, EcoFed proposes a novel replay buffer mechanism and implements a quantization-based compression technique to reduce the transmission of the activation. It is experimentally demonstrated that EcoFed can reduce the communication cost by up to 133× and accelerate training by up to 21× when compared to classic FL. Compared to vanilla DPFL, EcoFed achieves a 16× communication reduction and 2.86× training time speed-up. EcoFed is available from https://github.com/blessonvar/EcoFed.

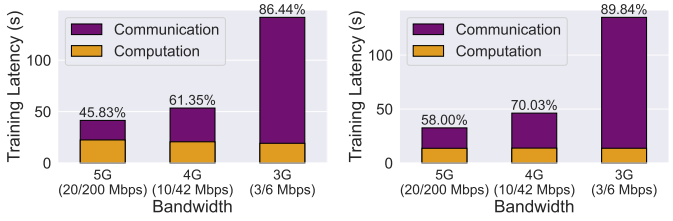
Index Terms—Edge computing, Federated learning, DNN partitioning, communication efficiency

1 Introduction

Federated learning (FL) is a privacy-preserving machine learning paradigm that facilitates collaborative training without transferring raw data from participating devices to a server [1]–[3]. However, running FL training on resource constrained devices is challenging since training deep neural networks (DNN), which is computationally expensive, is solely run on devices. This is a known bottleneck [4]–[6].

DNN partitioning-based FL (***DPFL***) in which the DNN is partitioned across the device and server has been developed to surmount the challenge of running FL on resource constrained devices [7]–[9]. In DPFL, an entire DNN model is partitioned into two parts – a device-side model and a server-side model. The first few layers of the DNN are deployed on the device-side for training. The remaining layers are offloaded to a server that has more computational resources than the device. The computational burden on the device is alleviated as it only trains a few layers of the entire model. Consequently, the training time is reduced.

Although DPFL reduces the computational burden on a device compared to FL, it incurs additional communication overheads. This is because the outputs of the activation generated by the device-side model in a forward pass and the corresponding gradients calculated during backpropagation need to be transferred between the devices and the server.



(a) VGG11 (b) ResNet9

Fig. 1: Computation and communication latency in DPFL training under typical (upload/download) network bandwidth. Numerical value above the bars is the percentage of communication latency.

Figure 1 shows the computation and communication latency incurred with DPFL training for the partitioning point with minimum latency[1]. The communication overhead in DPFL is nearly 46% (58% for ResNet9) of the overall training time under 5G conditions and around 86% (90% for ResNet9) for 3G bandwidth. In addition, this inefficiency leads to a substantial increase in: (i) the total volume of communication that is directly proportional to the data transferred across all devices and the server, and (ii) the communication frequency due to the need for transmitting activations and gradients during each forward and backward pass.

Split Federated Learning (SFL), which we refer to as *vanilla DPFL*, is the first FL work that partitions the DNN across the device and the server [10]. However, the communication overheads introduced by partitioning are not considered there. Recent DPFL methods [8], [9], which we

- *D. Wu and B. Varghese are with the School of Computer Science, University of St Andrews, UK. Corresponding author: dw217@st-andrews.ac.uk.*
- *R. Ullah is with the Cardiff School of Technologies, Cardiff Metropolitan University, UK.*
- *P. Rodgers is with Rakuten Mobile, Inc., Japan.*
- *P. Kilpatrick and I. Spence are with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, UK.*

1. Refer to Section 4.2 for the experiment configuration

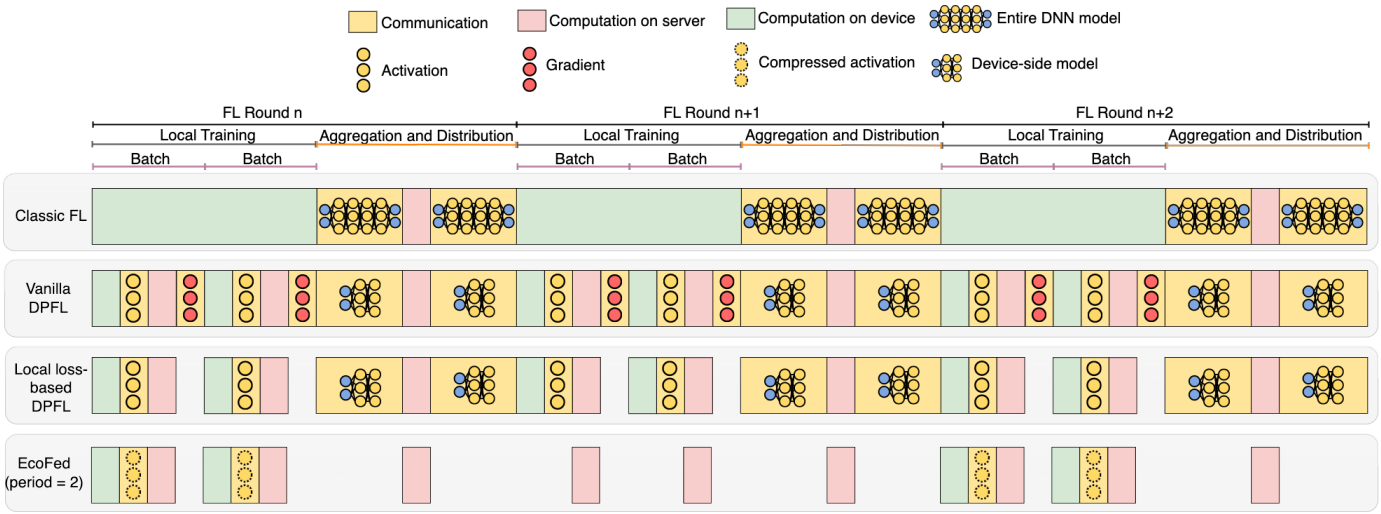


Fig. 2: The training pipeline of classic FL, vanilla DPFL, local loss-based DPFL and EcoFed for three rounds of training. Classic FL transfers the entire model from the devices to the server at the end of each round. Vanilla DPFL only needs to upload a partitioned device-side model at the end of each round. However, Vanilla DPFL transfers the activation and gradient for each batch sample. Local loss-based DPFL reduces the communication by half since the gradients are computed locally. EcoFed reduces communication further as it transfers the activation only periodically (for example, once in two rounds) and further compresses the size of the activations.

refer to as *local loss-based DPFL*, use local loss generated by an auxiliary network to train the device-side model instead of transferring and using the gradient from the server. Local loss-based DPFL can reduce half of the communication cost using device-side auxiliary networks since only the activation needs to be sent from the devices to the server.

Although local loss-based DPFL methods eliminate the need to send the gradient, we demonstrate that they cannot improve the efficiency of vanilla DPFL due to two significant issues (Section 4). Firstly, *the volume of communication required to achieve the target accuracy is not reduced and even increases due to poor learning performance as indicated by lower final accuracy and convergence speed*. The poor learning performance of local loss-based DPFL methods is referred to as 'accuracy degradation' and is caused by training using local loss instead of end-to-end training. We also demonstrate that the use of local error signals will lower the final accuracy and convergence speed (Section 4.3). As a result, the communication cost to achieve the target accuracy is similar to or even higher than vanilla DPFL (Section 4.4).

Secondly, *under limited network bandwidth conditions, the training time does not significantly decrease since the upload bandwidth is usually lower than the download bandwidth.* Local loss-based DPFL methods do not consider the transfer of activations, which is significant - half of the communication volume - and has a high frequency per iteration. Therefore, when network bandwidth is limited as seen in resource constrained environments, it is not feasible to accelerate training using current local loss-based DPFL methods (Section 4.5).

In this article, we propose EcoFed, a communication efficient framework for DPFL on resource-constrained devices. Figure 2 illustrates the training pipeline of classic FL, vanilla DPFL, local loss-based DPFL and EcoFed. EcoFed only transfers the activation periodically (for example, once every two rounds) and further reduces the size of the activation, thereby reducing the overall frequency and volume of communication. Pre-trained initialization of the device-side model is employed in EcoFed to reduce accuracy degradation caused by local loss-based methods. The frequency of transferring activations is reduced by proposing a replay buffer mechanism in which the server-side model is periodically trained by making use of cached activations instead of regularly transferring the activation from the devices to the server. Moreover, EcoFed compresses the activation using a lightweight quantization technique to further reduce the size of the data transferred and the corresponding buffer. Two DNN models and datasets are considered in our evaluations by comparing EcoFed against four baselines, including classical FL, vanilla DPFL and two state-of-the-art local loss-based DPFL methods. EcoFed improves the test accuracy compared to the baselines while reducing the communication volume by up to 16× and thus accelerates training by up to 2.86× compared to other DPFL methods.

The ***research contributions*** of this article are:

1) Identifying the limitations of local loss-based DPFL approaches by systematically exploring the accuracy, communication size and training latency of DPFL methods on resource constrained devices.

2) Designing, developing and evaluating EcoFed, the first framework to effectively reduce communication overheads of DPFL by proposing novel approaches that optimize the forward and backward passes in DPFL.

3) Proposing novel techniques that use pre-trained initialization on the device-side to eliminate the need for transferring gradients from the server to the device without significant accuracy degradation and a replay buffer along with quantization for reducing the frequency and volume of activations transferred to the server in DPFL.

The rest of this article is organized as follows: Section 2 provides an overview and the underlying methods of the EcoFed framework. Section 3 theoretically analyzes convergence and the computation and communication cost of

EcoFed. Section 4 evaluates EcoFed against the baselines. Section 5 presents related work and Section 6 concludes this article.

2 EcoFed Framework

This section firstly provides an overview of the proposed EcoFed framework (Section 2.1). Then the underlying techniques, namely *Pre-trained Initialization* (Section 2.2) and *Replay Buffer* (Section 2.3) are presented followed by the proposed algorithm of EcoFed (Section 2.4).

2.1 Overview

Figure 3 provides an overview of the EcoFed modules that operate across resource constrained devices and servers (either cloud or edge servers). The underlying techniques used by the modules are discussed in the next sub-sections.

When FL training begins, the *Initializer* (❶) module on the server determines the training scheme (the configurations of the DNN models) and initializes the weights. The *Initializer* also splits the model (to device-side and server-side models) for each participating device. The device-side models are sent to the devices and the corresponding server-side models are sent to the server.

In vanilla DPFL, after configuration, the training starts on the device-side model for each device. The *Device Trainer* (the training engine on devices, ❸) will first generate activation outputs of the device-side model. The outputs and labels of the corresponding data samples are sent to the server. The *Server Trainer* (the training engine on the server, ❼) trains the server-side model using the activation outputs received from the device and generates corresponding gradients. The gradients are sent back to each device to update the device-side model. The above steps are repeated for each batch sample in vanilla DPFL training.

However, in EcoFed, before generating and sending the activation outputs of the device-side model to the server, the *Activation Switch* (❷) will determine whether the outputs of the device-side model are required to be sent to the server or if the server can use the buffer with the cached activation to train the server-side model. If the activation outputs are required to be sent, then they will be further compressed using the quantization technique implemented by the *Compressor* (❹). The compressed activation and labels of the corresponding samples will then be sent to the server. On the server, the compressed data will be firstly used to update the *Replay Buffer* (❺) and reconstructed by the *Decompressor* (❻) for training the server-side models. The *Server Trainer* only needs to calculate and update the gradients of the server-side models without sending the gradient back to each device for training the device-side models.

After completing the above steps, the *Aggregator* (❽) will collect updated weights from each device for aggregation and for generating new global weights using the *FederatedAveraging* (FedAvg) algorithm [11].

EcoFed reduces the communication cost in vanilla DPFL by eliminating the transmission of the gradient, adjusting the communication frequency of the activation and compressing the activation data. The *Initializer* uses pre-trained weights to initialize the device-side model and freezes them during training, thus eliminating the need for transmitting the gradient. In addition, the *Activation Switch* periodically uploads the output activation of the device-side model to the edge servers to update the *Replay Buffer*. When the edge server does not receive the activation from devices, it will continue training the server-side models using the activations that were cached in the *Replay Buffer*. The *Compressor* and *Decompressor* modules are underpinned by quantization and dequantization techniques presented in the literature [12], [13].

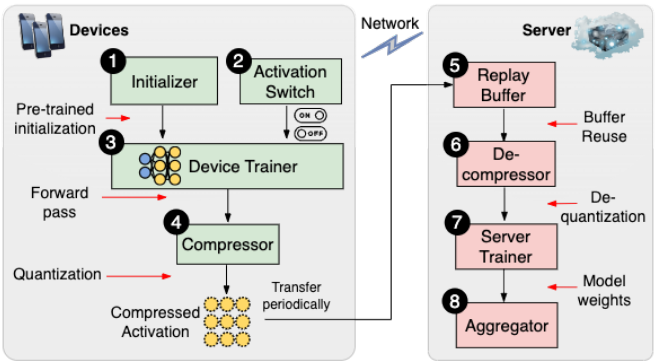


Fig. 3: EcoFed modules on the device and server.

2.2 Pre-trained Initialization

Existing local loss-based DPFL incorporates local error signals to train the device-side model using an auxiliary network [8], [9]. The auxiliary network consists of a few lightweight hidden layers, for example, fully connected layers that map the output of the device-side model to the same dimensions as the ground truth labels. Although the local error signals eliminate the need to transfer global gradients from the server to devices, this approach adversely impacts accuracy and convergence rate since the device-side model and the server-side model are decoupled and trained by different error signals, namely local and global error signals. We empirically demonstrate this in Section 4.3. A similar accuracy loss is reported in the literature for greedy layer-wise learning. In this method, the use of an auxiliary network results in accuracy degradation when compared to end-to-end training that uses global loss calculated with the entire model [14]. As a result, the communication required to achieve the target accuracy is effectively not reduced and even increases due to the lower accuracy and slower convergence speed of local loss-based methods.

To address the above, EcoFed adopts pre-trained initialization of the device-side model ($\boldsymbol{w}_C$). For each device k, the device-side model is initialized with pre-trained weights ($\boldsymbol{w}_{C,k}$), which are the partial weights of the entire model trained on a pre-training dataset (e.g., ImageNet [15]). We empirically study the impact of local loss on generating activation outputs, which are used to train the server-side model. We identify that local loss is not a satisfactory criterion for representing a 'good' activation output for training the server-side model with pre-trained initialization (Section 4.3). Therefore, during FL training, we freeze the weights of the device-side model ($\boldsymbol{w}_{C,k}$), which runs the first

few layers of the model that learn general features. These layers are not specific to a particular task, and therefore, the weights of the initial layers can be transferred from a pre-trained model [16], [17].

Benefits of pre-trained initialization: There are four benefits to pre-trained initialization and freezing weights of the device-side model $\boldsymbol{w}_{C,k}$ in DPFL. They are: (i) Communication during training is reduced since the need for the devices to receive gradients from the edge server is eliminated; (ii) Accuracy loss caused by using local error signals is reduced; (iii) The computational workload on resource constrained devices is reduced since the gradients of $\boldsymbol{w}_{C,k}$ do not need to be calculated and updated on the devices; (iv) The device-side model ($\boldsymbol{w}_{C,k}$) and the respective activation outputs are not significantly changed during each training round, making it possible to use cached activations from a buffer to train server-side models.

2.3 Replay Buffer

EcoFed introduces a *Replay Buffer* on the edge server, i.e., the server uses the activation cached in the buffer, which is obtained from a previous training round, to train the server-side model in a given round. EcoFed caches and updates the buffer periodically. If the transmission of activations is switched off in a given round, then EcoFed will reuse the cached activations. Therefore, there are two modes of training, namely EcoFed with and without the buffer.

Periodic transfer: The *Activation Switch* controls the transfer frequency of the activation in the forward pass from the device-side to the server-side model. The frequency is controlled by the interval between successive activation transfers, which is denoted as ρ. Consider, for example, $\rho = 2$. In this case, the activation outputs of the devices will only be transferred every second round and the buffer is cached with the activations when it is sent. During the other rounds, the edge server will use the cached activations to train the server-side models. It is worth noting that the activations ($\boldsymbol{a}_k^t$) for each round will change due to different participating clients, data batches and data augmentations. Therefore, the buffer on the edge servers needs to be periodically updated during the training. Thus, ρ is a hyper-parameter that will affect the model performance and communication cost of EcoFed, which is further considered in Section 4.3.

Reducing the buffer size: A potential issue that must be mitigated is that a large buffer may be required if large or many activations are transferred. The maximum size of the buffer required will be the size of all activations transferred from all devices. However, it is practical to establish a maximum buffer size and implement periodic updates to the buffer to efficiently manage its storage capacity. In addition, EcoFed stores the compressed activation outputs ($\boldsymbol{z}$) instead of the original activations ($\boldsymbol{a}_k^t$). The activation is compressed by the *Compressor* module of EcoFed using a lightweight 8-bit linear quantization technique [18] denoted as function $Q(.)$. The output ($\boldsymbol{a}_k^t$) is quantized from 32 bits to 8 bits before sending it to the server. The compressed activation ($\boldsymbol{z}_k^t$) is then cached in the *Replay Buffer*.

Algorithm 1: Partitioning-based training in EcoFed

1 **Input:** Pre-trained $\boldsymbol{w}_C^*$ and data $\{D_k\}_{k=1}^K$
2 **Output:** $\boldsymbol{w}^*$
3 **for** *each device* $k \in K$ *in parallel* **do**
4 Download $\boldsymbol{w}_C^*$ to device k;
5 Initialize and freeze $\boldsymbol{w}_{C,k}^*$; //Pre-trained initialization
6 **end**
7 **for** *each round (each device and corresponding worker* $k \in K$ *in parallel)* **do**
8 Initialize $\boldsymbol{w}_{S,k}$ on the server;
9 **for** *each round* $t \in T$ **do**
10 **if** *Activation switch is on* ($t \mod \rho == 0$) **then**
11 Forward on $\boldsymbol{w}_{C,k}^*$ to get $\boldsymbol{a}_k^t$;
12 $\boldsymbol{z}_k^t \leftarrow Q(\boldsymbol{a}_k^t)$; //Quantization
13 Send $\boldsymbol{z}_k^t$ to the server;
14 update buffer with $\boldsymbol{z}_k^t$;
15 **end**
16 **else**
17 $\boldsymbol{z}_k^t \leftarrow$ buffer; //Replay Buffer
18 **end**
19 $\hat{\boldsymbol{a}}_k^t \leftarrow Q^{-1}(\boldsymbol{z}_k^t)$; //Dequantization
20 Forward on $\boldsymbol{w}_{S,k}^t$;
21 Calculate loss $\ell_{S,k}^t$ and gradients $\nabla \ell_{S,k}^t(\boldsymbol{w}_{S,k}^t)$;
22 $\boldsymbol{w}_{S,k}^t \leftarrow \boldsymbol{w}_{S,k}^t - \eta \nabla \ell_{S,k}^t(\boldsymbol{w}_{S,k}^t)$;
23 **end**
24 $\boldsymbol{w}_{S,k} \leftarrow \boldsymbol{w}_{S,k}^t$;
25 $\boldsymbol{w}_S \leftarrow \Sigma_{k=1}^K \frac{|D_k|}{|D|} \boldsymbol{w}_{S,k}$; //FedAvg
26 **end**
27 $\boldsymbol{w}_S^* \leftarrow \boldsymbol{w}_S$;
28 $\boldsymbol{w}^* \leftarrow \{\boldsymbol{w}_C^*, \boldsymbol{w}_S^*\}$; //Concatenation of $\boldsymbol{w}_C^*$ and $\boldsymbol{w}_S^*$
29 **return** $\boldsymbol{w}^*$

2.4 Proposed Algorithm

Algorithm 1 shows the steps of partitioning-based training in EcoFed by employing pre-trained initialization and a replay buffer with compression using quantization.

We first present the notation used. The collection of K devices is denoted as $\{k\}_{k=1}^K$. Each device generates its own data D_k. The data from all devices is denoted as D: $\{D_k\}_{k=1}^K$. The number of samples in D_k is denoted as $|D_k|$ and the total number of samples is $|D|$. Let $\boldsymbol{w}$ be the entire model, which will be partitioned as the device-side model ($\boldsymbol{w}_C$) and server-side model ($\boldsymbol{w}_S$). $\boldsymbol{w}_{C,k}$ and $\boldsymbol{w}_{S,k}$ are the models of the k^{th} device. The superscript t is used to represent a training round t in a total of T rounds. $\boldsymbol{a}_k$ is the intermediate activation generated by $\boldsymbol{w}_{C,k}$. $\ell_{S,k}(.)$ is the loss function of the server-side model of the k^{th} device.

EcoFed first prepares the pre-trained weights of the device-side model ($\boldsymbol{w}_C^*$). The weights of $\boldsymbol{w}_C^*$ will be frozen during training to eliminate the transfer of the gradient from the server-side to device-side model $\boldsymbol{w}_C$ (Lines 4 and 5). Then, the server-side model of each device will be trained independently with K parallel workers.

For each round t, if activation transfer is switched on (i.e. $t \mod \rho == 0$), then the output activation of $\boldsymbol{w}_{C,k}^*$ is generated, denoted as $\boldsymbol{a}_k^t$, and compressed to 8 bits, denoted

as $\boldsymbol{z}_t^k$. The compressed activation is uploaded to the edge server and is used to update the corresponding buffer (Lines 10-15). Alternatively, the edge server retrieves $\boldsymbol{z}_t^k$ directly from the buffer (Line 17). Using $\boldsymbol{z}_t^k$, EcoFed will firstly dequantize $\boldsymbol{z}_t^k$ and provide the output $\hat{\boldsymbol{a}}_k^t$ to the server-side model for the training of $\boldsymbol{w}_{S,k}$ (Lines 19-22).

Once a training round is completed by an edge server, it will send each $\boldsymbol{w}_{S,k}$ to the cloud. The cloud aggregates a new global server-side model (Lines 24-26). Finally, after T rounds of training, the cloud obtains the optimal server-side model ($\boldsymbol{w}_S^*$) and concatenates it with $\boldsymbol{w}_C^*$ into a complete model $\boldsymbol{w}*$ (Lines 28-29).

3 Convergence and Cost Analysis

In this section, we firstly analyze EcoFed convergence (Section 3.1). Then the computation and communication costs of EcoFed on the device-side are compared against classic FL, vanilla DPFL and local loss-based DPFL (Section 3.2).

3.1 Convergence Analysis

We follow the convergence analysis presented in the literature [8], [19], [20] to analyze the convergence of Algorithm 1. We assume the following:

Assumption 1 – *The server-side objective functions are L-smooth, i.e.,* $\|\nabla F_S(\boldsymbol{u}) - \nabla F_S(\boldsymbol{v})\| \leq L\|\boldsymbol{u} - \boldsymbol{v}\|, \forall \boldsymbol{u}, \forall \boldsymbol{v}$.

Assumption 2 – *The squared norm of the stochastic gradient has an upper bound for the server-side object function, i.e.,* $\|\nabla F_S(\boldsymbol{a}_k^t; \boldsymbol{w}_{S,k}^t)\|^2 \leq G, \forall k, \forall t$.

Assumption 3 – *The learning rate* η_t *satisfies* $\sum_t \eta_t = \infty$ *and* $\sum_t \eta_t^2 < \infty$.

In each global round t, the output activation of the k_{th} device-side model is equal to $\boldsymbol{a}_k^t = \boldsymbol{w}_{C,k}^t(\boldsymbol{x})$. We assume $\boldsymbol{a}_k^t$ follows a probability distribution of $p_k^t(\boldsymbol{a})$, which is determined by $\boldsymbol{w}_C^t$ and D_k. In EcoFed, the device-side model $\boldsymbol{w}_{C,k}^t$ is initialized with pre-trained weights and frozen during the training, thus fixed as $\boldsymbol{w}_{C,k}^*$. EcoFed uses the quantization function $Q(.)$ and dequantization function $Q^{-1}(.)$ along with the *Replay Buffer* to train the server-side models, denoted as $\hat{\boldsymbol{a}}_k^t \triangleq Q^{-1}(Q(\boldsymbol{a}_k^t))$. We define the probability distributions of the *Replay Buffer* as $q_k^t(\boldsymbol{a})$.

To analyze the impact of *Replay Buffer* and quantization on the convergence, we further define two types of errors: buffer and quantization errors. Buffer error is defined as the distance of gradients between the original distribution $p_k^t(\boldsymbol{a})$ and buffer distribution $q_k^t(\boldsymbol{a})$, denoted as $\delta_k \triangleq \int \left\|\nabla \ell(a_k^t; \boldsymbol{w}_S)\right\| \left\|(q_k^t(\boldsymbol{a}) - p_k^t(\boldsymbol{a}))\right\| da$. The $\ell(.)$ is the loss function (e.g. cross entropy loss for classification). Quantization error is defined as $\int \left\|\nabla \ell(\hat{a_k^t}; \boldsymbol{w}_S) - \nabla \ell(a_k^t; \boldsymbol{w}_S)\right\| \left\|q_k^t(a)\right\| da$, which is the sum of gradient errors over $q_k^t(\boldsymbol{a})$ due to quantization. We include a further assumption specific to EcoFed as follows:

Assumption 4 – *The buffer error and the quantization error have upper bounds, i.e.,* $\delta_k^t \leq H_1$ *and* $\varepsilon_k^t \leq H_2, \forall k, \forall t$.

Convergence of device-side model: Since $\boldsymbol{w}_C$ is fixed during training, its convergence is not considered.

Convergence of server-side model: Let $\frac{1}{\Gamma_T} \triangleq \sum_{t=0}^{T-1} \eta_t$. *Following on from Assumption 1, Assumption 2 and Assumption 4, Algorithm 1 ensures the following:*

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[\|\nabla F_S(\boldsymbol{w}_S^t)\|^2] \leq \frac{4(F_S(\boldsymbol{w}_S^0) - F_S(\boldsymbol{w}_S^*))}{3\Gamma_T} + \frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t(\sqrt{G}+1)(H_1+H_2) + \frac{L}{2}\eta_t^2 G\right) \quad (1)$$

where $\nabla F_S(\boldsymbol{w}_S^t) \triangleq \frac{1}{K}\sum_{k=1}^K \nabla F_{S,k}(\boldsymbol{w}_S^t)$. $F_{S,k}(.)$ is the objective function of the k^{th} server-side model. $\boldsymbol{w}_S^0$ is the initial server-side weights and $\boldsymbol{w}_S^*$ is the optimal server-side weights. The detailed proof is provided in Appendix A.

Based on Assumption 3, it is noted that with increasing T, the right-hand side of Equation 1 convergences to zero. Thus, Equation 1 guarantees that the proposed algorithm of EcoFed converges to a stationary point with increasing T.

Differences between the convergence of local loss-based DPFL and EcoFed: Equation 1 is similar to the convergence analysis presented in the literature [8]. The server-side model converges as follows:

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t \mathbb{E}[\|\nabla F_S(\boldsymbol{w}_S^t)\|^2] \leq \frac{4(F_S(\boldsymbol{w}_S^0) - F_S(\boldsymbol{w}_S^*))}{3\Gamma_T} + G\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t \frac{1}{K}\sum_{k=1}^K \left(d_k^t\right) + \frac{L}{2}\eta_t^2\right) \quad (3)$$

where $d_k^t \triangleq \int \|p_k^t(\boldsymbol{a}) - p_k^*(\boldsymbol{a})\| d\boldsymbol{a}$ which is defined as the distance between the probability distribution of activation $\boldsymbol{a}_k^t$ and $\boldsymbol{a}_k^*$. It is worth noting that $p_k^t(\boldsymbol{a})$ keeps changing during FL training since $\boldsymbol{w}_{C,k}^t$ is updated by the local error signals. Therefore, in local loss-based DPFL, the changing distance (d_k^t) of the probability distribution of $\boldsymbol{a}_k^t$ caused by updating $\boldsymbol{w}_{C,k}^t$ affects the convergence of the server-side model as shown in Equation 3. However, in EcoFed, $d_k^t = 0$ since $\boldsymbol{w}_{C,k}^t$ is fixed during training. In addition, the *Replay Buffer* and quantization error affects convergence behaviour, which is δ_k^t and ε_k^t (bounded by H_1 and H_2) as shown in Equation 1. Different ρ values and quantization techniques used by the *Compressor* module will determine the values of H_1 and H_2, thus affecting the convergence of EcoFed.

3.2 Cost Analysis

Table 1 compares the computation and communication costs of EcoFed against classic FL, vanilla DPFL and local loss-based DPFL. We use $|.|$ to denote either the computation or communication workload of a given model or an activation. We distinguish two modes in EcoFed, namely EcoFed without buffer and EcoFed with buffer. In classic FL, the entire model ($\boldsymbol{w}$) is computed on each device ($|\boldsymbol{w}|$). At the end of each round, $\boldsymbol{w}$ is uploaded to the cloud and then the newly aggregated $\boldsymbol{w}$ is downloaded to the device ($2|\boldsymbol{w}|$). For vanilla DPFL, each device trains only the device-side model, $\boldsymbol{w}_C$ ($|\boldsymbol{w}_C|$) and $\boldsymbol{w}_C$ is transferred at the end of each round ($2|\boldsymbol{w}_C|$). The activation and gradient of each data sample will be communicated with an overhead of $2|D_k||\boldsymbol{a}_k|$. For local loss-based DPFL, each device also needs to train $\boldsymbol{w}_C$ ($|\boldsymbol{w}_C|$). In addition, $\boldsymbol{w}_C$ is uploaded and downloaded at the end of each round ($2|\boldsymbol{w}_C|$) but only the activation is transferred during each training round ($|D_k||\boldsymbol{a}_k|$).

TABLE 1: Computation and communication costs on the device for each round.

Methods	Computation	Communication
FL	$\|\boldsymbol{w}\|$	$2\|\boldsymbol{w}\|$
Vanilla DPFL	$\|\boldsymbol{w}_C\|$	$2\|\boldsymbol{w}_C\| + 2\|D_k\|\|\boldsymbol{a}_k\|$
Local loss-based DPFL	$\|\boldsymbol{w}_C\|$	$2\|\boldsymbol{w}_C\| + \|D_k\|\|\boldsymbol{a}_k\|$
EcoFed **w/o buffer**	$\frac{1}{2}\|\boldsymbol{w}_C\|$	$\|D_k\|\|\boldsymbol{z}_k\|$ 2
EcoFed **w buffer**	0	0

However, for EcoFed without using the Replay Buffer (indicated as EcoFed w/o buffer; when the buffer is used indicated as EcoFed w buffer), the device only computes the forward pass on $\boldsymbol{w}_C$, ($\frac{1}{2}|\boldsymbol{w}_C|$). In addition, the communication overhead is reduced to $|D_k||\boldsymbol{z}_k|$ where $\boldsymbol{z}_k$ is the compressed activation [2]. There are no computation or communication costs during training when using the buffer.

4 Evaluation

In this section, we evaluate EcoFed against four baselines. The test environment, the evaluation setup and the results obtained from the experimental studies are considered. The results highlight that EcoFed improves accuracy compared to state-of-the-art methods and eliminates accuracy degradation caused by local error signals in local loss-based DPFL. In addition, EcoFed significantly reduces communication costs and accelerates training.

4.1 Test Environment

Datasets and Models: Two image classification datasets, namely CIFAR-10 [21] and CIFAR-100 [21] are used [3] We follow a similar method reported in the literature [11] to partition data across devices. For an independent and identically distributed (I.I.D.) setting, the training set is initially divided into 500 shards for CIFAR-10 and 5000 shards for CIFAR-100. We randomly assign 5 shards and 50 shards to each device for CIFAR-10 and CIFAR-100, respectively. In the non-I.I.D. setting, we first sort the dataset based on their labels, dividing it into 500 shards for CIFAR-10 and 5000 shards for CIFAR-100. Then, we randomly allocate 5 shards and 50 shards to each device for CIFAR-10 and CIFAR-100, respectively. Consequently, each device will have training samples of up to half of all available classes. The test dataset is available on the server and will be used to test model performance after each round of training.

We train two popular convolutional neural networks, namely VGG11 [24] (plain convolutional neural network) and ResNet9 [25] (residual convolutional neural network). In terms of the DNN partitioning point (PP) and auxiliary networks used on devices, we follow the same configuration used in the local loss-based DPFL literature [8], which corresponds to PP2 detailed in Appendix B.3. The architectures of VGG11, ResNet9 and the auxiliary network along with the device-side and server-side model partitions are shown in Table 2.

2. There is one exception - in the first round of training, devices need to download $\boldsymbol{w}_C$. Therefore, the communication cost for the first round is $|\boldsymbol{w}_C| + |D_k||\boldsymbol{z}_k|$.

3. We do not report learning performance on MNIST [22] and FMNIST [23] as they are simple datasets and do not highlight the differences between the methods we evaluate.

TABLE 2: Models for evaluation. Convolution layers denoted as C followed by the no. of filters; filter size is 3×3 for all convolution layers except for downsampling convolution which is 1×1; Max Pooling layer is MP; Fully Connected layer is FC; and Residual Block is RB including two convolution layers, a max pooling layer and downsampling convolution layers; number following is no. of output channels.

Model	Device	Server
VGG11	C64-MP-C128-MP	C256-C256-MP-C512-C512-MP-C512-C512-FC4096-FC4096-FC10
ResNet9	C64-MP-C128-MP	RB256-RB512-RB512-FC10
Auxiliary Network	FC10	N/A

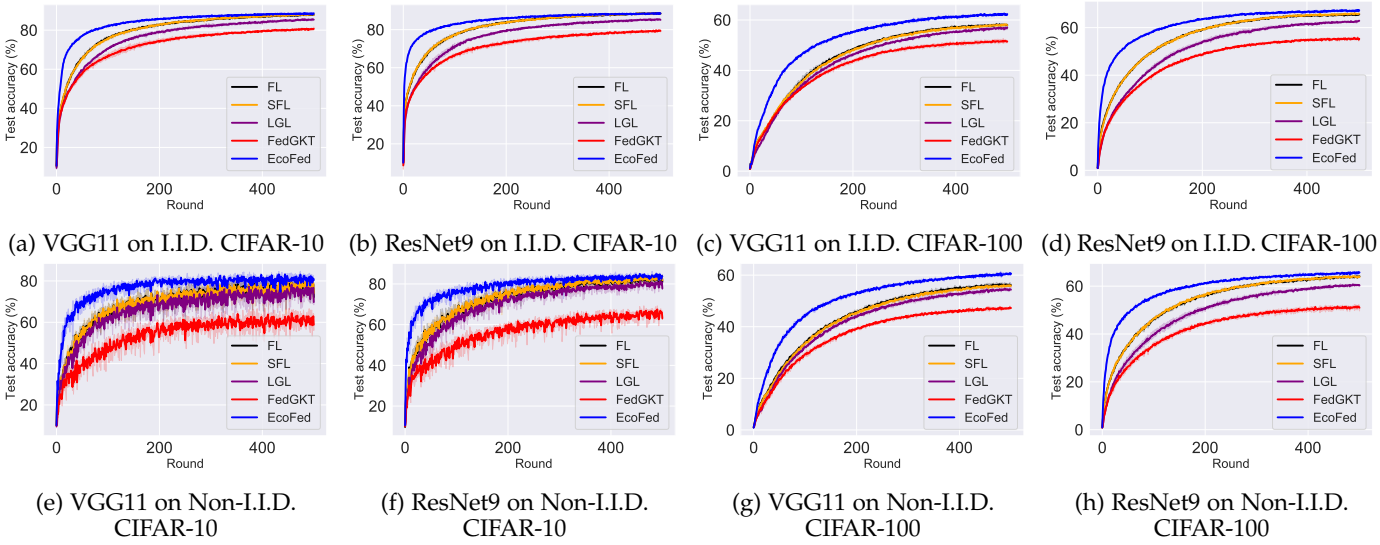
FL Training hyper-parameters: At the beginning of each FL round, the server randomly selects 20 devices out of 100 devices (sampling ratio of 0.2) for participating in the current training round. The standard FedAvg [11] aggregation method is used by the *Aggregator* for all approaches. We adopt the same data augmentation and learning schedules for all methods for fair comparisons (horizontal flip with a probability of 0.5 and the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01). We set a total of 500 rounds for training on all datasets.

Testbed: To evaluate system performance (i.e. communication cost and training latency), we build a prototype with an edge server and five resource constrained devices. The edge server has a 2.5GHz Intel i7 8-core CPU and 16GB RAM. Five Raspberry Pi 4 Model B single-board computers with 1.5GHz quad-core ARM Cortex-A53 CPU are used as resource constrained devices. All devices and server run PyTorch as the training engine. The server and devices are connected via a router. We use Linux `tc` commands to emulate different network (upload/download) bandwidths: 5G (20/200 Mbps), 4G (10/42 Mbps) and 3G (3/6 Mbps) as indicated in Appendix B.1.

We also employ a simulation-based testbed comprising a 2GHz AMD EPYC 7713P 64-Core CPU with 252GB RAM and two Nvidia A6000 GPUs. This testbed enables us to rapidly carry out the evaluation of learning performance.

4.2 Evaluation Setup

Baselines: We consider four baselines, namely classic FL, SFL (vanilla DPFL) and two state-of-the-art local loss-based DPFL methods: (1) **Classic FL** trains the entire model locally on-device. (2) **SFL (vanilla DPFL)** is the first DPFL approach. The model is partitioned into device-side and server-side models, sent to the devices and the server for training, respectively. The activation and gradient are transferred between devices and the server for each data batch. This method does not optimize communication. (3) **Local generated loss (LGL)** [8] introduces a locally generated loss on devices to calculate gradients for training the device-side model, thus reducing the need for sending gradients from the server. (4) **FedGKT** [9] incorporates local loss on devices but also uses probabilistic predictions, called soft labels [26], of the server-side models. These are periodically



(a) VGG11 on I.I.D. CIFAR-10 (b) ResNet9 on I.I.D. CIFAR-10 (c) VGG11 on I.I.D. CIFAR-100 (d) ResNet9 on I.I.D. CIFAR-100

(e) VGG11 on Non-I.I.D. CIFAR-10 (f) ResNet9 on Non-I.I.D. CIFAR-10 (g) VGG11 on Non-I.I.D. CIFAR-100 (h) ResNet9 on Non-I.I.D. CIFAR-100

Fig. 4: Test accuracy curves of EcoFed and the baselines using VGG11 and ResNet9 in I.I.D. and Non-I.I.D. settings for CIFAR-10 and CIFAR-100 datasets.

TABLE 3: The highest test accuracy of EcoFed compared to the baselines for VGG11 and ResNet9 on two datasets on I.I.D. and Non-I.I.D. distribution. The results are an average of three independent runs with different random seeds.

Methods	CIFAR-10				CIFAR-100			
	I.I.D.		Non-I.I.D.		I.I.D.		Non-I.I.D.	
	VGG11	ResNet9	VGG11	ResNet9	VGG11	ResNet9	VGG11	ResNet9
FL	88.29%	88.95%	82.01%	84.52%	58.48%	65.87%	56.79%	64.29%
SFL	88.31%	88.81%	81.38%	84.56%	58.5%	66.32%	56.47%	64.42%
LGL	85.76%	85.65%	79.66%	82.18%	57.22%	63.01%	54.88%	61%
FedGKT	81.09%	79.82%	66.39%	68.71%	52.03%	55.92%	47.74%	51.81%
EcoFed	**88.87%**	**88.81%**	**84.47%**	**85.82%**	**62.81%**	**67.61%**	**61.1%**	**66.08%**

transferred to the devices and vice versa. The soft labels distil the training of device-side and server-side models[4].

EcoFed is implemented with the following configuration. We adopt the pre-trained weights of the device-side model from the respective pre-trained model trained on the ImageNet dataset [15] and freeze them during the training. $\rho = 2$ for the *Replay Buffer* (the buffer will be updated every two rounds). In addition, a linear 8-bit quantization implementation is adopted for the *Compressor*.

Evaluation Metrics: Two sets of metrics are used:

1) **Metrics on learning performance**: (i) *Accuracy* is evaluated on the global test data. We record the highest test accuracy achieved during entire rounds of training for each baseline. The results are an average of three independent runs with different random seeds.

2) **Metrics on system performance**: (i) *Communication cost* is the measured communication overhead for one round. The communication cost versus test accuracy curve is presented to evaluate the efficiency of communication for achieving a target accuracy; (ii) *Training latency* is the wall-clock time of one training round for each baseline given a network bandwidth.

4.3 Accuracy

The test accuracy curves of the five methods, including EcoFed and the four baseline methods, on the two datasets (both I.I.D. and Non-I.I.D. setting) using the two DNN models are shown in Figure 4. In addition, the highest test accuracy is reported in Table 3. The results show that EcoFed usually outperforms all baselines across all datasets and the two model architectures (except for FL on I.I.D. CIFAR-10 for ResNet9). In detail, EcoFed achieves up to a 4.63% increase in accuracy on Non-I.I.D. CIFAR-100 for VGG11 compared to SFL (4.31% compared to FL). In addition, EcoFed significantly improves the accuracy by up to 6.22% on Non-I.I.D CIFAR-100 for VGG11 compared to LGL and by up to 18.08% on Non-I.I.D CIFAR-10 for VGG11 compared to FedGKT, respectively.

SFL offloads parts of the training computation to the server but fundamentally shares the same algorithm as FL. As a result, their accuracy and learning curves are similar. LGL and FedGKT introduce local error signals to reduce communication. However, although there is a reduction in communication, there is a significant loss in accuracy and slower convergence compared to FL and SFL. EcoFed, on the other hand, does not have an accuracy degradation seen in LGL and FedGKT, while also achieving a more substantial reduction in communication costs compared to local loss-based DPFL methods (Section 4.4).

4.3.1 Impact of pre-trained initialization and freezing weights on the device-side model

To obtain a better understanding of accuracy achieved in Figure 4 and Table 3, we further investigate accuracy degradation in local loss-based DPFL methods and the effects of

4. The implementation of FedGKT for our experiments is a variant that we developed based on vanilla DPFL in order to compare it to other baselines when using the same aggregation algorithm (FedAvg).

TABLE 4: The highest test accuracy of FedGKT (bi-direction) and FedGKT (uni-direction) on CIFAR-10. The results are an average of three independent runs with different random seeds.

FedGKT	I.I.D.		Non-I.I.D.	
	VGG11	ResNet9	VGG11	ResNet9
Bidirection	81.09%	79.82%	66.39%	68.71%
Unidirection	83.18%	83.42%	75.89%	77.68%

TABLE 5: The highest test accuracy of EcoFed compared to the baselines on CIFAR-10 with pre-trained initialization on the device-side model. The results are an average of three independent runs with different random seeds.

Methods	I.I.D.		Non-I.I.D.	
	VGG11	ResNet9	VGG11	ResNet9
FL	89.48%	89.55%	84.81%	86.06%
SFL	89.44%	89.57%	85.02%	86.19%
LGL	88.06%	88.29%	83.44%	85.04%
FedGKT	83.59%	82.35%	72.73%	74.36%
EcoFed	**88.87%**	**88.81%**	**84.47%**	**85.82%**

using pre-trained initialization and freezing weights on $\boldsymbol{w}_c$ in EcoFed. We answer the following three questions[5]:

Can local error signals generated on the device degrade accuracy? From Table 3 and Figure 4, it is noted that local loss-based DPFL methods suffer a higher accuracy loss than the other methods. Even when compared to FL and SFL, LGL and FedGKT consistently achieve lower accuracy. In addition, local loss-based DPFL has a slower convergence rate as demonstrated in Figure 4 resulting from separately optimizing the device-side model and the server-side model by inconsistent gradient signals.

We also note that FedGKT has a relatively lower accuracy compared to LGL. In FedGKT, knowledge distillation is carried out both on device-side and server-side models. We demonstrate that the distillation of device-side soft labels on server-side training has a negative impact on accuracy. Table 4 shows the accuracy results of bidirectional FedGKT and unidirectional FedGKT (only the server-side soft labels are used for distilling the device-side model). We observe a significant accuracy improvement when removing distillation from device-side soft labels when training the server-side model, shown as unidirectional FedGKT. This further demonstrates that local training results in accuracy degradation of the server-side model.

Can pre-trained initialization on the device-side model also improve accuracy for other baselines? We further investigate the impact of pre-trained initialization of the device-side model for FL, SFL and local loss-based DPFL methods. Table 5 shows the highest test accuracy achieved by each method when we apply pre-trained initialization for the device-side model. The results indicate that pre-trained initialization can significantly improve the accuracy of FL and SFL, specifically in the Non-I.I.D. setting. This finding has also been observed in recent research [27], [28].

Surprisingly, the results demonstrate that pre-trained initialization can also mitigate accuracy degradation caused by local error signals in the local loss-based DPFL methods, which to the best of our knowledge, has never been reported before. When all methods adopt pre-training initialization, EcoFed still outperforms all local loss-based DPFL methods. Compared to FL and SFL, EcoFed achieves competitive accuracy performance (less than 1% loss) with considerable communication reduction by up to 114× (Section 4.4).

5. We show the results for only CIFAR-10 due to the limitation of space; the same conclusions were observed for CIFAR-100.

TABLE 6: The highest test accuracy of device-side model of trainable $\boldsymbol{w}_c$ and frozen $\boldsymbol{w}_c^*$ in LGL (on CIFAR-10) and the highest test accuracy of respective server-side models under $\boldsymbol{w}_c$ and $\boldsymbol{w}_c^*$. The results are an average of three independent runs with different random seeds.

LGL		I.I.D.		Non-I.I.D.	
		VGG11	ResNet9	VGG11	ResNet9
Device	Trainable $\boldsymbol{w}_c$	78.61%	78.68%	76.78%	76.77%
	Frozen $\boldsymbol{w}_c^*$	70.6%	70.72%	69.46%	67.89%
Server	$\boldsymbol{w}_s$ under $\boldsymbol{w}_c$	87.82%	88.04%	83.05%	84.9%
	$\boldsymbol{w}_s$ under $\boldsymbol{w}_c^*$	88.97%	89.34%	84.4%	85.77%

Does training device-side models with local loss in the context of pre-trained initialization improve the accuracy of the server-side model? In LGL and FedGKT, the device-side model is trained by local error signals to avoid receiving the gradients. However, for pre-trained initialization, we investigate whether device-side training by local loss can improve the training of the server-side model after applying pre-trained initialization on the device-side model.

To this end, the test accuracy of device-side model ($\boldsymbol{w}_c$) and server-side model ($\boldsymbol{w}_s$) of LGL are recorded on CIFAR-10. We compare the test accuracy of LGL (with pre-trained initialization) with trainable $\boldsymbol{w}_c$ (device-side model) and frozen $\boldsymbol{w}_c^*$. Table 6 shows that the trainable $\boldsymbol{w}_c$ has significantly higher test accuracy than the frozen $\boldsymbol{w}_c^*$ across I.I.D. and non-I.I.D. settings. However, it is surprising that the server-side model ($\boldsymbol{w}_s$), when the device-side model is frozen (denoted as $\boldsymbol{w}_s$ under $\boldsymbol{w}_c^*$), achieves higher accuracy compared to the server-side model with a trainable device-side model (denoted as $\boldsymbol{w}_s$ under $\boldsymbol{w}_c$).

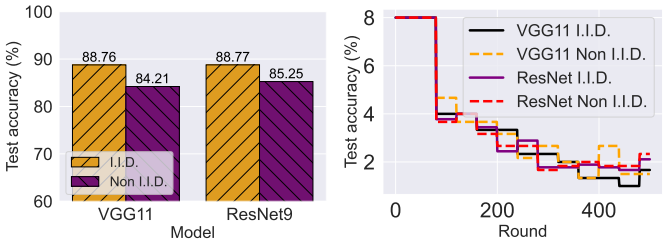
The results suggest that training the device-side model can indeed significantly improve the accuracy of the device-side model, but it does not necessarily improve the accuracy of the server-side model and may even degrade the accuracy of the server-side model. However, the accuracy of the server-side model is what we ultimately aim to achieve. It is thus inferred that local training on the device-side model is not required for improving the accuracy performance of server-side model. We conjecture that the local error signals generated by the local auxiliary network are not optimal for the training performance of server-side models. Given the above observations, EcoFed freezes the device-side weights when pre-trained initialization is adopted.

4.3.2 *Impact of ρ and quantization on accuracy*

We investigate the accuracy performance of EcoFed under different hyper-parameter settings. ρ controls the update frequency of the *Replay Buffer* and quantization is adopted to reduce the size of transferred data and consequently, the memory required by the cached buffer. The impact of ρ and quantization on test accuracy for CIFAR-10 is shown in Figure 6. The accuracy gradually decreases as ρ increases since the update frequency is reduced. Accuracy is less sensitive to quantization since (near) similar accuracy is achieved with or without quantization for the same ρ value.

TABLE 7: Heuristic rules for dynamic ρ.

Period	$\Delta_{acc} \in (10^{-1}, \infty)$	$\Delta_{acc} \in (10^{-2}, 10^{-1}]$	$\Delta_{acc} \in (10^{-3}, 10^{-2}]$	$\Delta_{acc} \in (-\infty, 10^{3}]$
ρ	8	4	2	1



(a) Accuracy (b) Period

Fig. 5: Dynamic ρ values based on Table 7 in EcoFed. The results are an average of three independent runs with different random seeds.

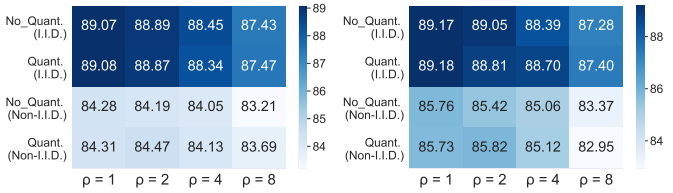
It is worth noting that when $\rho = 1$, EcoFed updates the cached buffer every round. To achieve higher accuracy, the buffer must be updated more frequently (smaller ρ).

Dynamic control strategy for ρ: We explore dynamic control of ρ as accuracy is sensitive to ρ. We use a heuristic-based strategy and monitor accuracy improvement every 40 rounds. The accuracy growth rate, denoted by Δ_{acc}, is calculated. The value of ρ for the next 40 rounds is set based on the predefined heuristic rules shown in Table 7. Figure 5 presents the highest accuracy and the corresponding change of ρ during training. In general, ρ gradually decreases as training accuracy approaches a plateau in the later stages. We calculate the average value of ρ with dynamic control, which is $\rho = 3.5$ throughout the entire training process. In addition, it is observed that the highest accuracy using dynamic control is also obtained between fixed periods with $\rho = 2$ and $\rho = 4$.

4.3.3 *Impact of pre-training dataset*

Pre-trained weights obtained using large-scale datasets (e.g., ImageNet) for the device-side model can be downloaded from open-source repositories, such as Hugging Face[6] and PyTorch Model Zoo[7]. This approach saves substantial pre-training time when using large datasets. We have also explored the impact on accuracy when training from scratch on a small pre-training dataset, e.g., Tiny-ImageNet [29]. The training overhead for this is minimal compared to FL training since the server usually has more resources than devices. In addition, given the challenges of collecting user data for pre-training, we have carried out pre-training on synthetic data, such as CIFAR-5m [30] and SIP-17 [31]. Please refer to Appendix B.2 for a description of pre-training on these datasets. Table 8 shows the accuracy results on CIFAR-10 when using pre-trained weights on four different types of pre-training datasets. In general, EcoFed demonstrates robust generalization across various pre-training datasets. It achieves a higher level of accuracy on natural datasets and maintains similar accuracy, even on small-scale datasets

6. https://huggingface.co/
7. https://pytorch.org/serve/model_zoo.html



(a) For VGG11 (b) For ResNet9

Fig. 6: Accuracy for different ρ values with or without quantization in EcoFed. The results are an average of three independent runs with different random seeds.

TABLE 8: The highest test accuracy of EcoFed on CIFAR-10 with different pre-training datasets. We use **L** and **S** to denote large-scale or small-scale datasets, and **Nat** and **Syn** to denote whether the data is natural or synthetic. The results are an average of three independent runs with different random seeds.

Pre-training datasets	I.I.D.		Non-I.I.D.	
	VGG11	ResNet9	VGG11	ResNet9
ImageNet (L, Nat)	88.87%	88.81%	84.47%	85.82%
Tiny-ImageNet (S, Nat)	89.32%	88.99%	85.66%	85.86%
CIFAR-5m (S, Syn)	87.92%	87.94%	83.61%	83.6%
SIP-17 (S, Syn)	86.81%	86.98%	81.45%	83.49%

like Tiny-ImageNet. EcoFed exhibits a small decrease in accuracy compared to SFL (up to 0.96%) for CIFAR-5m. However, it still outperforms local loss-based methods such as LGL (by up to 3.95%). When dealing with the more challenging SIP-17 dataset with domain shifts, EcoFed still achieves high performance with a small accuracy loss (up to 1.83%) compared to SFL. In addition, it outperforms local loss-based methods, such as LGL (up to 1.79%).

4.4 Communication Cost

Communication cost for one training round: The communication overhead of EcoFed and the four baseline methods are compared for one training round on the CIFAR-10 dataset as shown in Table 9[8].

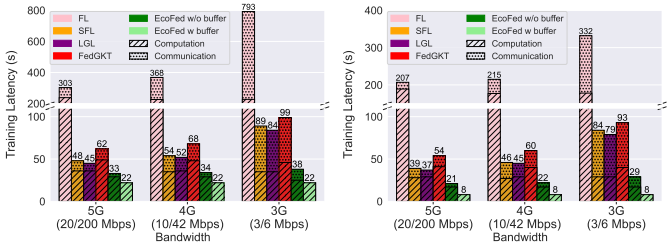
Compared to classic FL, other DPFL methods have a smaller communication overhead; for example, the communication cost is reduced by 8.27× (SFL) and 15.55× (LGL and FedGKT) when training VGG11. The reason is that DPFL methods only need to transfer the device-side model between the devices and the server, which is usually smaller than the entire model. EcoFed (with buffer) achieves a further reduction in communication cost of 66.62× on VGG11 and 18.18× on ResNet9. When using the buffer, EcoFed fundamentally eliminates the need for communication. In terms of the overall communication costs for all rounds of FL, EcoFed ($\rho = 2$) can reduce communication by 133.25× on VGG11 and 36.36× on ResNet9 compared to classic FL.

Compared to the other DPFL methods, EcoFed also reduces the communication cost significantly. EcoFed without buffer reduces the communication cost by 8.05×, 4.29× and 4.29× on both VGG11 and ResNet9. LGL and FedGKT can reduce the communication cost by half when compared to

8. For evaluation of communication cost and training latency of one training round, we only report the results on the CIFAR-10 dataset as system performance is independent of datasets.

TABLE 9: Communication cost for one training round.

Methods	Communication cost	
	VGG11	ResNet9
FL	5.13 GB	1.4 GB
SFL	0.62 GB	0.62 GB
LGL	0.33 GB	0.33 GB
FedGKT	0.33 GB	0.33 GB
EcoFed **w/o buffer**	**0.077 GB**	**0.077 GB**
EcoFed **w buffer**	**0 GB**	**0 GB**



(a) For VGG11 (b) For ResNet9

Fig. 7: Latency of one training round for VGG11 and ResNet9 under different network conditions for PP2. The results are an average of three independent runs.

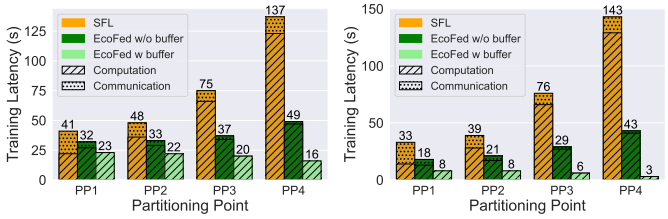
SFL as they only require the activations to be transferred from devices to the server. However, due to the cached buffer technique in EcoFed, the average communication per round is further reduced by a factor of ρ. For instance, in our experiments with $\rho = 2$, the average communication per round is 0.0385 GB, which is 16.1×, 8.57× and 8.57× lower than SFL, LGL and FedGKT, respectively.

Communication cost vs test accuracy: The cumulative communication cost for all training rounds on the two datasets is considered. The communication cost after each training round is recorded. Figure 9 highlights the communication cost incurred to achieve a target test accuracy.

For instance, given a test accuracy target of 80% on I.I.D. CIFAR-10 for VGG11, FL, SFL, LGL and FedGKT will require 729GB, 90GB, 71GB and 145GB of data transfers, respectively. Similarly, for a 50% test accuracy target on I.I.D. CIFAR-100 the communication costs for VGG11 are 1121GB, 139GB, 110GB and 156GB. However, EcoFed will only require 4GB on I.I.D. CIFAR-10 and 7GB on I.I.D. CIFAR-100 to achieve the target test accuracy. For ResNet9, FL, SFL, and LGL will require 169GB, 78GB, and 67GB of data to be transferred (150GB, 65GB, 64GB and 100GB on I.I.D. CIFAR-100), respectively. FedGKT fails to achieve 80% test accuracy on I.I.D. CIFAR-10 and requires 100GB on I.I.D. CIFAR-100 for 50% test accuracy. In contrast, EcoFed has a low volume of data transfer (4GB on I.I.D. CIFAR-10 and 3GB on I.I.D. CIFAR-100) to achieve the target test accuracy.

For the Non-I.I.D. setting on both datasets, EcoFed has higher communication efficiency to reach a target test accuracy when compared to all baselines. It is worth noting that although the communication of LGL and FedGKT is reduced by half per round (since gradient transfers are eliminated) compared to SFL, the volume of communication required to achieve the same level of accuracy does not significantly decrease, and in some cases, such as ResNet9 on CIFAR-100 increases. This is because of accuracy degradation and slower convergence using local error signals.

EcoFed reduces communication cost by 133.25× on VGG11 (36.36× on ResNet9) compared to classic FL. Compared to SFL (vanilla DPFL), EcoFed improves communication by up to 16.1×. In short, classic FL, SFL (vanilla DPFL) and local loss-based DPFL incur significant communication costs. In contrast, EcoFed has significantly lower communication costs and higher communication efficiency.



(a) For VGG11 (b) For ResNet9

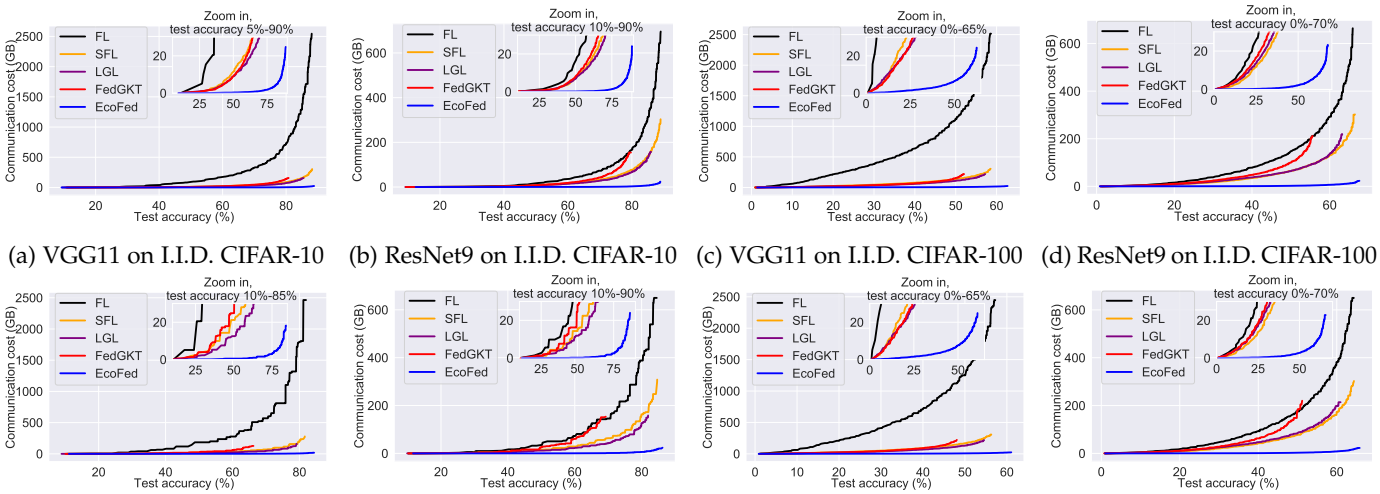
Fig. 8: Latency of one training round for VGG11 and ResNet9 under different partitioning points in 5G conditions. The results are an average of three independent runs.

4.5 Training Latency

EcoFed improves the communication efficiency and eliminates device-side gradient computation resulting in a speed-up of the overall training latency. The average training time for one round of EcoFed is compared against the baseline methods to quantify this benefit. Figure 7a and Figure 7b highlight that compared to classic FL, EcoFed achieves a 9.33× and 10.06× speed-up for VGG11 and ResNet9 without using buffer (under 5G conditions), respectively. In addition, compared to SFL (vanilla DPFL), EcoFed achieves a speed-up of 1.47× and 1.9× on VGG11 and ResNet9 without using the buffer (for 5G). Compared to local loss-based DPFL methods (LGL and FedGKT) with communication optimization, there is an improvement of training latency of about 1.38× and 1.91× on VGG11 and 1.83× and 2.62× on ResNet9 without using the buffer.

We further consider the impact of network bandwidth, which will be a bottleneck for devices that operate in environments that have limited network capabilities (e.g. mobile phones with 4G or 3G signal). Therefore, we evaluated EcoFed and each baseline under different network bandwidth conditions. When the network bandwidth is limited to 4G (10 Mbps and 42 Mbps for upload and download) and 3G (3 Mbps and 6 Mbps for upload and download), the training latency of FL, SFL, LGL and FedGKT are high due to the increase in the communication costs for transferring the model and intermediate activation and gradients. It is worth noting that local loss-based methods (i.e. LGL and FedGKT) have a similar training latency compared to non-optimized vanilla DPFL (i.e. SFL) since they still incur large communication costs due to transferring activations. In addition, the upload bandwidth for sending activations is typically much lower than the download bandwidth used for sending gradients, and it has not been considered in local loss-based DPFL. This highlights the importance of reducing communication costs when transferring activations.

In contrast, EcoFed has only a small increase in training latency, resulting in a 21.08× and 2.38× speed-up on VGG11 and 11.26× and 2.86× speed-up on ResNet9 in



(a) VGG11 on I.I.D. CIFAR-10 (b) ResNet9 on I.I.D. CIFAR-10 (c) VGG11 on I.I.D. CIFAR-100 (d) ResNet9 on I.I.D. CIFAR-100

(e) VGG11 on Non-I.I.D. CIFAR-10 (f) ResNet9 on Non-I.I.D. CIFAR-10 (g) VGG11 on Non-I.I.D. CIFAR-100 (h) ResNet9 on Non-I.I.D. CIFAR-100

Fig. 9: Communication cost versus test accuracy of VGG11 and ResNet9 for the I.I.D and Non-I.I.D. settings on CIFAR-10 and CIFAR-100 datasets.

a 3G network, compared to FL and SFL, respectively. In addition, when EcoFed employs the buffer for training (half of the entire rounds with $\rho = 2$), the training latency is fundamentally independent of network bandwidths. This validates the low bandwidth requirements of EcoFed.

Latency under different partitioning points: Variants of vanilla DPFL dynamically adjust the PP to minimize training latency [7], [32], [33]. We investigated the impact on training latency for different PPs in EcoFed. We considered four PPs (PP1, PP2, PP3 and PP4); refer to the complete configuration in Appendix B.3. We compared EcoFed with SFL for different PPs using 5G network bandwidth. We report the overall latency and its individual components of computation and communication. Please refer to Appendix B.4 for more results on accuracy performance.

Figure8 illustrates the latency of SFL and EcoFed under different PPs. As the partition point is placed deeper in the model, the overall training latency gradually increases since more computations are performed on the device. In addition, the communication latency becomes smaller as activations and gradients in the later layers are smaller in size. Overall, EcoFed achieves significant acceleration across all partition points without using the buffer. It results in speed-ups of 1.31×, 1.47×, 2.03×, and 2.83× for VGG11 (and 1.8×, 1.9×, 2.6×, and 3.36× for ResNet9) at PP1, PP2, PP3, and PP4, respectively. When the partition point is at the initial layers of the model (PP1 and PP2), EcoFed minimizes latency by reducing network communication latency. When the partition point is at the later layers of the model (PP3 and PP4), EcoFed reduces latency by eliminating gradient computation on the device. The lowest overall latency is achieved for PP1, where EcoFed achieves 1.8× speed-up compared to SFL. When the buffer is used there is no computation on the device. In this case, the training latency of EcoFed is further reduced and gradually decreases when the PP is set at later layers since less computation is offloaded to the server. In summary, EcoFed can further accelerate training latency for different partitioning points.

5 Related Work

In this section, we consider techniques for reducing communication in FL and the literature on DPFL methods. We also discuss pre-training in FL and layer-wise learning.

Reducing Communication in FL: Communication reduction techniques in FL can be grouped into two categories, depending on whether they (i) reduce the frequency of communication or (ii) compress the size of the transferred data. Under the first category, a key technique is to increase the interval between model aggregation, thus reducing the communication frequency from the device to the server and vice-versa [34], [35]. Under the second category, compression approaches, such as quantization and sparsification, are employed to minimize the size of models (updated weights) in each round of communication [36], [37]. By incorporating distillation, the communication overhead of transferring model parameters in traditional FL can be eliminated [38]. However, it is the above focus on the communication of the updated models at the end of each FL round, rather than the communication costs introduced due to DNN partitioning during training, that our article considers.

DPFL: Existing research on DPFL can be categorized as vanilla DPFL and local loss-based DPFL based on how they optimize the communication. SFL [10] is the first DPFL work that combines FL and split learning [39] such that the device-side models are independently trained by receiving gradients from the server. The server-side models are trained by collecting activations from devices in parallel. In addition, dynamic partitioning strategies for DPFL [7], [32], [33] and pipeline scheduling [40] have been considered to optimize performance. However, these methods are considered vanilla DPFL as they do not consider the communication overhead introduced by DNN partitioning.

Recently DPFL approaches have been optimized by computing local loss on the device-side to reduce the communication cost [8], [9]. In these approaches, the device-side model is trained with local error signals generated by an

TABLE 10: Comparing FL, vanilla DPFL, local loss-based DPFL and EcoFed.

	FL	Vanilla DPFL (e.g., [7], [10], [32], [33])	Local loss-based DPFL (e.g., [8], [9])	EcoFed
Offers device-side acceleration	✗	✓	✓	✓
Reduces communication cost for DNN partitioning	n/a	✗	✓	✓
Optimizes communication arising from activation transfers	n/a	✗	✗	✓
Has low accuracy degradation	n/a	n/a	✗	✓

additional auxiliary network. This eliminates the need for transferring the gradient from the server and making use of it on the device. However, the training of device-side model with local error signals is sub-optimal, and thus detrimental to accuracy. In addition, the communication costs to transfer the activation are not considered. We highlight the differences between classic FL, vanilla DPFL, local loss-based DPFL and EcoFed in Table 10.

Pre-training in FL: Pre-training a model is rarely investigated in the literature on FL. Instead, the model is usually trained from random weights. Recent work has demonstrated that pre-training can close the accuracy gap between FL and centralized learning, specifically in the non-IID setting [27], [28]. EcoFed utilizes pre-training on the device-side and presents an approach to reduce the accuracy loss inherent to local loss-based DPFL methods; both of these are considered for the first time.

Layer-wise Learning: Another line of related research is layer-wise learning in which each DNN layer is independently trained using auxiliary networks [14], [19], [41]. Since these approaches train using local loss in a resource-rich environment (with centralized servers that have an abundance of resources), the computation overheads of each layer and communication costs to transfer intermediate data are rarely considered. EcoFed can be considered as a special case of parallel block-wise learning. However, the device-side model is deployed in a resource-constrained environment on devices with limited computation and communication resources, which is challenging.

6 Conclusion

In this article, we present EcoFed, a communication-efficient DNN partitioning-based federated learning (DPFL) framework. DPFL partitions a DNN and offloads some of the layers of the DNN (or computation) from a resource constrained device to the server. EcoFed proposes pre-trained initialization to eliminate the transmission of the gradient from the server-side model to device-side model in DPFL for the first time and designs a novel replay buffer mechanism with quantization-based compression to further reduce the communication cost incurred by transferring activation. In other words, EcoFed proposes a unique way of carrying out the forward and backward passes for efficiently training in resource constrained environments. We comprehensively evaluate EcoFed and demonstrate that EcoFed can reduce the accuracy degradation caused by state-of-the-art local loss-based DPFL methods while significantly improving the communication efficiency and training speed compared to classical FL and state-of-the-art DPFL methods.

7 Acknowledgement

This work was sponsored by Rakuten Mobile, Japan.

References

[1] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[2] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated Learning for Emoji Prediction in a Mobile Keyboard," *arXiv preprint arXiv:1906.04329*, 2019.

[3] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated Learning for Keyword Spotting," in *2019 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019, pp. 6341–6345.

[4] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things," in *International Symposium on Reliable Distributed Systems*, 2020, pp. 91–100.

[5] C. Wang, Y. Yang, and P. Zhou, "Towards Efficient Scheduling of Federated Mobile Devices under Computational and Statistical Heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, 2021.

[6] Z. Xu, F. Yu, J. Xiong, and X. Chen, "Helios: Heterogeneity-Aware Federated Learning with Dynamically Balanced Collaboration," in *ACM/IEEE Design Automation Conference*, 2021, pp. 997–1002.

[7] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "FedAdapt: Adaptive Offloading for IoT Devices in Federated Learning," *IEEE Internet of Things Journal*, 2022.

[8] D.-J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating Federated Learning with Split Learning on Locally Generated Losses," in *International Conference on Machine Learning Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2021.

[9] C. He, M. Annavaram, and S. Avestimehr, "Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[10] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When Federated Learning Meets Split Learning," in *AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[12] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit Quantization of Neural Networks for Efficient Inference," in *IEEE/CVF International Conference on Computer Vision Workshop*, 2019, pp. 3009–3018.

[14] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Greedy Layerwise Learning Can Scale To Imagenet," in *International Conference on Machine Learning*, 2019, pp. 583–593.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[16] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable are Features in Deep Neural Networks?" *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[17] M. Huh, P. Agrawal, and A. A. Efros, "What Makes ImageNet Good for Transfer Learning?" *arXiv preprint arXiv:1608.08614*, 2016.

[18] D. Wu, Q. Tang, Y. Zhao, M. Zhang, Y. Fu, and D. Zhang, "EasyQuant: Post-Training Quantization via Scale Optimization," *arXiv preprint arXiv:2006.16669*, 2020.

[19] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled Greedy Learning of CNNs," in *International Conference on Machine Learning*, 2020, pp. 736–745.

[20] J. Wang, H. Qi, A. S. Rawat, S. Reddi, S. Waghmare, F. X. Yu, and G. Joshi, "FedLite: A Scalable Approach for Federated Learning on Resource-constrained Clients," *arXiv preprint arXiv:2201.11865*, 2022.

[21] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009, [Online; accessed 24-October-2023].

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-Mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[24] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556*, 2014.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[26] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the Knowledge in A Neural Network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[27] H.-Y. Chen, C.-H. Tu, Z. Li, H.-W. Shen, and W.-L. Chao, "On Pre-Training for Federated Learning," *arXiv preprint arXiv:2206.11488*, 2022.

[28] J. Nguyen, K. Malik, M. Sanjabi, and M. Rabbat, "Where to Begin? Exploring the Impact of Pre-Training and Initialization in Federated Learning," *arXiv preprint arXiv:2206.15387*, 2022.

[29] Y. Le and X. Yang, "Tiny Imagenet Visual Recognition Challenge," http://cs231n.stanford.edu/reports/2015/pdfs/yle_project.pdf, 2015, [Online; accessed 24-October-2023].

[30] P. Nakkiran, B. Neyshabur, and H. Sedghi, "The Deep Bootstrap Framework: Good Online Learners are Good Offline Generalizers," *arXiv preprint arXiv:2010.08127*, 2020.

[31] X. Zhu, T. Bilal, P. Mårtensson, L. Hanson, M. Björkman, and A. Maki, "Towards Sim-to-Real Industrial Parts Classification With Synthetic Dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4453–4462.

[32] X. Deng, J. Li, C. Ma, K. Wei, L. Shi, M. Ding, and W. Chen, "Low-Latency Federated Learning With DNN Partition in Distributed Industrial IoT Networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 3, pp. 755–775, 2022.

[33] P. He, C. Lan, Y. Cui, R. Wang, and D. Wu, "Accelerated Federated Learning with Dynamic Model Partitioning for H-IoT," in *2023 IEEE Wireless Communications and Networking Conference*, 2023, pp. 1–6.

[34] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[35] Q. Wu, X. Chen, T. Ouyang, Z. Zhou, X. Zhang, S. Yang, and J. Zhang, "HiFlash: Communication-Efficient Hierarchical Federated Learning with Adaptive Staleness Control and Heterogeneity-aware Client-Edge Association," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[36] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedpPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2020, pp. 2021–2031.

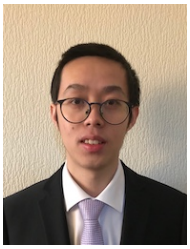
[37] P. Han, S. Wang, and K. K. Leung, "Adaptive Gradient Sparsification for Efficient Federated Learning: An Online Learning Approach," in *IEEE International Conference on Distributed Computing Systems*, 2020, pp. 300–310.

[38] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto, "Distillation-based Semi-supervised Federated Learning for Communication-efficient Collaborative Training with Non-iid Private Data," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 191–205, 2021.

[39] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split Learning for Health: Distributed Deep Learning Without Sharing Raw Patient Data," *arXiv:1812.00564*, 2018.

[40] Z. Zhang, P. Rodgers, P. Kilpatrick, I. Spence, and B. Varghese, "PipeLearn: Pipeline Parallelism for Collaborative Machine Learning," *arXiv preprint arXiv:2302.12803*, 2022.

[41] A. Nøkland and L. H. Eidnes, "Training Neural Networks with Local Error Signals," in *International Conference on Machine Learning*, 2019, pp. 4839–4850.



Di Wu is currently pursuing a PhD degree in computer science at University of St Andrews, UK. He received a B.S. degree in Information System and Information Management from Northeast Forestry University, China in 2015, and an M.S. degree in Data Science from University of Southampton, UK in 2018. His major interests are in the areas of federated learning, distributed machine learning, edge computing, model compression, and Internet-of-Things.

Rehmat Ullah earned a PhD degree in electronics and computer engineering from Hongik University, South Korea. He is currently an Assistant Professor at the Cardiff School of Technologies, Cardiff Metropolitan University, UK. Previously, he worked as a research fellow at University of St Andrews, UK and as an assistant professor at Gachon University, South Korea. His research interests are in edge computing, information centric networking and 5G evolution and beyond with a recent focus on federated learning for edge computing systems. More information is available from www.rehmatkhan.com.

Philip Rodgers received the PhD degree in Computer Science from the University of Strathclyde, UK in 2019. He is currently a Research Scientist at the Rakuten Mobile Innovation Studio, Japan. His research interests are in distributed algorithms for artificial intelligence.

Peter Kilpatrick received the PhD degree in Computer Science in 1985, and the BSc degree in Mathematics and Computer Science in 1981. He is currently a Reader in Computer Science at Queen's University Belfast. His interests include parallel programming models and cloud and edge computing.

Ivor Spence received the PhD degree in computer science from Queen's University Belfast, UK, where he did research on code generation. He is currently a Reader in computer science at Queen's University Belfast where he leads the artificial intelligence (AI) research theme. His research is primarily on heterogeneous computing systems for AI.

Blesson Varghese received the PhD degree in Computer Science from the University of Reading, UK on international scholarships. He is a Reader in Computer Science at the University of St Andrews, UK, and the Principal Investigator of the Edge Computing Hub. He is a previous Royal Society Short Industry Fellow. His interests include distributed systems that span the cloud-edge-device continuum and edge intelligence applications. More information is available from www.blessonv.com.

APPENDIX A
CONVERGENCE OF THE SERVER-SIDE MODEL

This Appendix presents the proof to demonstrate the convergence of the server-side model ($\boldsymbol{w}_S$).

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[\|\nabla F_S(\boldsymbol{w}_S^t)\|^2] \leq \frac{4(F_S(\boldsymbol{w}_S^0)-F_S(\boldsymbol{w}_S^*))}{3\Gamma_T} + \frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t\left(\sqrt{G}+1\right)(H_1+H_2)+\frac{L}{2}\eta_t^2 G\right) \quad (1)$$

Based on Assumption 1 presented in the article, we have

$$F_S(\boldsymbol{w}_S^{t+1}) \leq F_S(\boldsymbol{w}_S^t) + \nabla F_S(\boldsymbol{w}_S^t)^T(\boldsymbol{w}_S^{t+1}-\boldsymbol{w}_S^t) + \frac{L}{2}\|\boldsymbol{w}_S^{t+1}-\boldsymbol{w}_S^t\|^2 \quad (2)$$

Also, note that we have

$$\boldsymbol{w}_S^{t+1} = \boldsymbol{w}_S^t - \eta_t\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t) \quad (3)$$

When substituting Equation 3 in Equation 2 we have

$$F_S(\boldsymbol{w}_S^{t+1}) \leq F_S(\boldsymbol{w}_S^t) - \eta_t\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right) + \frac{L}{2}\eta_t^2\left\|\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\|^2 \quad (4)$$

By taking the expectation from both sides of Equation 4 we have

$$\mathbb{E}[F_S(\boldsymbol{w}_S^{t+1})] \leq \mathbb{E}[F_S(\boldsymbol{w}_S^t)] - \eta_t\underbrace{\mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right)\right]}_{B_1} + \frac{L}{2}\eta_t^2\underbrace{\mathbb{E}\left[\left\|\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\|^2\right]}_{B_2} \quad (5)$$

Next, we will find the lower bound of B_1 and upper bound of B_2. We define X as

$$X = \frac{1}{K}\sum_{k=1}^{K}(\nabla F_{S,k}(\boldsymbol{w}_S^t;\boldsymbol{a}_k^t) - \nabla F_{S,k}(\boldsymbol{w}_S^t;\hat{\boldsymbol{a}}_k^t)) \quad (6)$$

which is the difference in average gradient using $\boldsymbol{a}_k^t$ and $\hat{\boldsymbol{a}}_k^t$. For simplicity, we use the following abbreviation.

$$X = \frac{1}{K}\sum_{k=1}^{K}(\nabla F_{S,k}(\boldsymbol{w}_S^t) - \hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)) \quad (7)$$

Substituting X for B_1 we have

$$B_1 = \mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right)\right] \quad (8)$$

$$= \mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(X+\frac{1}{K}\sum_{k=1}^{K}\hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)\right)\right] \quad (9)$$

$$\geq \mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)\right)\right] - \|\mathbb{E}[\nabla F_S(\boldsymbol{w}_S^t)^T X]\| \quad (10)$$

$$= \underbrace{\mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right)\right]}_{C_1} + \underbrace{\mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}(\hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)-\nabla F_{S,k}(\boldsymbol{w}_S^t))\right)\right]}_{C_2} - \underbrace{\|\mathbb{E}[\nabla F_S(\boldsymbol{w}_S^t)^T X]\|}_{C_3} \quad (11)$$

C_1 is defined as

$$C_1 = \mathbb{E}[\|\nabla F_{S,k}(\boldsymbol{w}_S^t)\|^2] \quad (12)$$

When considering C_2

$$\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}(\hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)-\nabla F_{S,k}(\boldsymbol{w}_S^t))\right) \quad (13)$$

$$\geq -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}\left\|\hat{\nabla} F_{S,k}(\boldsymbol{w}_S^t)-\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\| \quad (14)$$

$$= -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}\left\|\nabla F_{S,k}(\boldsymbol{w}_S^t;\hat{\boldsymbol{a}}_k^t)-\nabla F_{S,k}(\boldsymbol{w}_S^t;a_k^t)\right\| \quad (15)$$

$$= -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}\left\|\int\nabla\ell(\hat{a}_k^t;\boldsymbol{w}_S)q_k^t(a)da - \int\nabla\ell(a_k^t;\boldsymbol{w}_S)p_k^t(a)da\right\| \quad (16)$$

$$\geq -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}\left\|\int(\nabla\ell(\hat{a_k^t};\boldsymbol{w}_S)-\nabla\ell(a_k^t;\boldsymbol{w}_S))q_k^t(a)da\right\| + \left\|\int\nabla\ell(a_k^t;\boldsymbol{w}_S)(q_k^t(a)-p_k^t(a))da\right\| \quad (17)$$

$$\geq -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}\int\left\|\nabla\ell(\hat{a_k^t};\boldsymbol{w}_S)-\nabla\ell(a_k^t;\boldsymbol{w}_S)\right\|\left\|q_k^t(a)\right\|da + \int\left\|\nabla\ell(a_k^t;\boldsymbol{w}_S)\right\|\left\|(q_k^t(a)-p_k^t(a))\right\|da \quad (18)$$

$$\geq -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}(\varepsilon_k^t+\delta_k^t) \quad (19)$$

Based on Assumption 4 presented in the article, we have

$$C_2 \geq -\mathbb{E}\left[\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}(\varepsilon_k^t + \delta_k^t)\right] \tag{20}$$

$$\geq -\sqrt{G}\frac{1}{K}\sum_{k=1}^{K}(H_1 + H_2) \tag{21}$$

$$\geq -\sqrt{G}(H_1 + H_2) \tag{22}$$

We consider C_3 based on $\mathbb{E}[U^T V] \leq \frac{1}{4}\mathbb{E}[\|U\|^2] + \mathbb{E}[\|V\|^2], \forall U, \forall V$. Then, we have

$$C_3 = \|\mathbb{E}[\nabla F_S(\boldsymbol{w}_S^t)^T X]\| \tag{23}$$

$$\leq \frac{1}{4}\mathbb{E}[\|\nabla F_S(\boldsymbol{w}_S^t)\|^2] + \mathbb{E}[\|X\|^2] \tag{24}$$

$$\leq \frac{1}{4}\mathbb{E}[\|\nabla F_S(\boldsymbol{w}_S^t)\|^2] + H_1 + H_2 \tag{25}$$

where Equation 25 can be derived by following similar steps as shown in Equation 13 to Equation 19.
By using the results of C_1, C_2 and C_3 we have

$$B_1 = \mathbb{E}\left[\nabla F_S(\boldsymbol{w}_S^t)^T\left(\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right)\right] \tag{26}$$

$$\geq \frac{3}{4}\mathbb{E}[\|\nabla F_{S,k}(\boldsymbol{w}_S^t)\|^2] - (\sqrt{G}+1)(H_1 + H_2) \tag{27}$$

Now we consider B_2

$$B_2 = \mathbb{E}\left[\left\|\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\|^2\right] \tag{28}$$

$$= \left\|\frac{1}{K}\sum_{k=1}^{K}\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\|^2 \tag{29}$$

$$\underset{a}{\leq} \frac{1}{K}\sum_{k=1}^{K}\left\|\nabla F_{S,k}(\boldsymbol{w}_S^t)\right\|^2 \tag{30}$$

$$\underset{b}{\leq} G \tag{31}$$

where a is obtained from the Cauchy-Schwarz inequality and b is obtained from Assumption 2 presented in the article.

By substituting the results of Equation 27 and Equation 31, we have

$$\mathbb{E}[F_S(\boldsymbol{w}_S^{t+1})] \leq \mathbb{E}[F_S(\boldsymbol{w}_S^t)] - \frac{3}{4}\eta_t\mathbb{E}[\|\nabla F_S, k(\boldsymbol{w}_S^t)\|^2] + \eta_t(\sqrt{G}+1)(H_1 + H_2) + \frac{L}{2}\eta_t^2 G \tag{32}$$

By summing up Equation 32 for all global rounds $t = 0, 1, \ldots T-1$, we have

$$\mathbb{E}[F_S(\boldsymbol{w}_S^T)] \leq \mathbb{E}[F_S(\boldsymbol{w}_S^0)] - \frac{3}{4}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[\|\nabla F_S, k(\boldsymbol{w}_S^t)\|^2] + \sum_{t=0}^{T-1}\left(\eta_t(\sqrt{G}+1)(H_1 + H_2) + \frac{L}{2}\eta_t^2 G\right) \tag{33}$$

Finally, from $F_S(\boldsymbol{w}_S^*) \leq \mathbb{E}[F_S(\boldsymbol{w}_S^T)]$ we obtain

$$\frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\eta_t\mathbb{E}[\|\nabla F_{S,k}(\boldsymbol{w}_S^t)\|^2] \leq \frac{4(F_S(\boldsymbol{w}_S^0) - F_S(\boldsymbol{w}_S^*))}{3\Gamma_T} + \frac{1}{\Gamma_T}\sum_{t=0}^{T-1}\left(\eta_t(\sqrt{G}+1)(H_1 + H_2) + \frac{L}{2}\eta_t^2 G\right) \tag{34}$$

which completes the proof.

APPENDIX B
NETWORK, PRE-TRAINING DATASET AND PARTITIONING POINTS

B.1 Network bandwidth

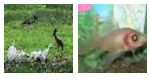
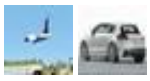
For all experiments in EcoFed involving different network bandwidth settings, we used real-world data as shown in Table 11. We carried out tests on network conditions of 3G HSPA+, 4G LTE-Advanced, and 5G, denoted as 3G, 4G, and 5G, respectively, in this article.

TABLE 11: Typical network bandwidths available in the UK.

Network type	Upload bandwidths (Mbps)	Download bandwidths (Mbps)
3G	0.4	3
3G HSPA+	3	6
4G LTE	5	20
4G LTE-Advanced	10	42
Home Broadband Wi-Fi	11	60
5G	20	200

B.2 Pre-training dataset

TABLE 12: Summary of pre-training datasets and training costs. We report the costs for pre-training the VGG11 model as pre-training time in seconds and the proportion relative to the total training time of EcoFed in brackets

Dataset	#Class	#Samples	Visual examples	Pre-training cost
ImageNet	1000	1.2M		n/a
Tiny-ImageNet	200	100K		1117s (8%)
CIFAR-5m	10	50K		684s (5%)
SIP-17	15	18K		463s (3%)

In this article, we present four different pre-training datasets, namely ImageNet, Tiny-ImageNet, CIFAR-5m, and synthetic industrial parts dataset (SIP-17) shown in Table 12.

ImageNet [15] is a popular large-scale dataset (1.2M samples with 1000 classes) that has been widely used for pre-training on various tasks [17].

Tiny-ImageNet [29] is a sampled version of ImageNet, containing 100K images from 200 classes, each with 500 training images.

CIFAR-5m [30] is a dataset of 5 million synthetic images. It was generated by sampling from the denoising diffusion probabilistic model (DDPM) [30]. We further sampled the original CIFAR-5m dataset and reduced it to 50K samples, with 5,000 samples per class, to construct a small-scale synthetic dataset that is equal to the size of CIFAR-10.

SIP-17 [31], consists of 15 individual objects and 2 assembly objects such as crosses and gears, each with 1,200 synthetic images. We used all 15 of the single objects for the pre-training. The SIP-17 dataset is not only synthetic but also exhibits greater domain shift compared to CIFAR-10.

Regarding the pre-training cost, we report the training time of VGG11 model using a single A6000 GPU. For the CIFAR-5m dataset with the same size as CIFAR-10, it took 684 seconds which is equivalent to 5% of the total training time of EcoFed on our Raspberry Pi prototype.

B.3 Partitioning point

In this article, we employed four different partitioning points (PP) as illustrated in Table 14. EcoFed does not optimize partitioning points in DPFL to minimize training latency, but its efficient communication techniques can significantly enhance the performance over a vanilla DPFL system at all partitioning points. In this article, we adopted a fixed partitioning point, i.e., PP2, in the majority of our experiments to align with the configuration of LGL [8]. However, we also investigate the latency performance of EcoFed for all partitioning points.

B.4 Accuracy obtained for different partitioning points

TABLE 13: The highest test accuracy of EcoFed on CIFAR-10 for different partitioning points using pre-trained ImageNet weights. The results are an average of three independent runs with different random seeds.

Partitioning points	I.I.D.		Non-I.I.D.	
	VGG11	**ResNet9**	**VGG11**	**ResNet9**
PP1	88.36%	88.59%	83.26%	84.34%
PP2	88.87%	88.81%	84.47%	85.82%
PP3	92.6%	92.26%	90.74%	90.42%
PP4	92.34%	91.96%	90.88%	89.44%

We tested the accuracy of EcoFed for different partition points with pre-trained ImageNet weights on the CIFAR-10 dataset. Overall, EcoFed achieves high accuracy for different partitioning points. It is worth noting that there is a significant accuracy increase with EcoFed when the partition point is moved to later layers (i.e., PP3 and PP4). This indicates that initializing and freezing more layers of pre-trained weights significantly improves the training accuracy of federated learning, which is also observed in prior literature [27].

TABLE 14: Different partitioning points (PP) of VGG11 and ResNet9 used for evaluation. Convolution layers denoted as C followed by the no. of filters; filter size is 3×3 for all convolution layers except for the downsampling convolution where filter size is 1×1; Max Pooling layer is MP; Fully Connected layer is FC; and Residual Block is RB including two convolution layers, a max pooling layer and downsampling convolution layers; number followed is no. of output channels. We represent communication size as $channels \times width \times height$

PP	Device		Server		Communication size	
	VGG11	**ResNet9**	**VGG11**	**ResNet9**	**Activation**	**Gradient**
PP1	C64-MP	C64-MP	C128-MP-C256-C256-MP-C512-C512-MP-C512-C512-FC4096-FC4096-FC10	C128-MP-RB256-RB512-RB512-FC10	$64 \times 16 \times 16$	$64 \times 16 \times 16$
PP2	C64-MP-C128-MP	C64-MP-C128-MP	C256-C256-MP-C512-C512-MP-C512-C512-FC4096-FC4096-FC10	RB256-RB512-RB512-FC10	$128 \times 8 \times 8$	$128 \times 8 \times 8$
PP3	C64-MP-C128-MP-C256-C256-MP	C64-MP-C128-MP-RB256	C512-C512-MP-C512-C512-FC4096-FC4096-FC10	RB512-RB512-FC10	$256 \times 4 \times 4$	$256 \times 4 \times 4$
PP4	C64-MP-C128-MP-C256-C256-MP-C512-C512-MP	C64-MP-C128-MP-RB256-RB512	C512-C512-FC4096-FC4096-FC10	RB512-FC10	$512 \times 2 \times 2$	$512 \times 2 \times 2$