PervasiveFL: Pervasive Federated Learning for Heterogeneous IoT Systems

Jun Xia, *Graduate Student Member, IEEE*, Tian Liu, *Graduate Student Member, IEEE*, Zhiwei Ling, Ting Wang, *Senior Member, IEEE*, Xin Fu, *Senior Member, IEEE*, and Mingsong Chen, *Senior Member, IEEE*

***Abstract*—Federated learning (FL) has been recognized as a promising collaborative on-device machine learning method in the design of Internet of Things (IoT) systems. However, most existing FL methods fail to deal with IoT applications that contain a variety of IoT devices equipped with different types of neural network (NN) models. This is because traditional FL methods assume that local models on devices should have the same architecture as the global model on cloud. To address this problem, we propose a novel framework named PervasiveFL that enables efficient and effective FL among heterogeneous IoT devices. Without modifying original local models, PervasiveFL installs one lightweight NN model named modellet on each device. By using the deep mutual learning (DML) and our entropy-based decision gating (EDG) method, modellets and local models can selectively learn from each other through soft labels using locally captured data. Meanwhile, since modellets are of the same architecture, the learned knowledge by modellets can be shared among devices in a traditional FL manner. In this way, PervasiveFL can be pervasively applied to any heterogeneous IoT system. Comprehensive experimental results on four well-known datasets show that PervasiveFL can not only pervasively enable FL among heterogeneous devices within a large-scale IoT system, but also significantly enhance the inference accuracy of heterogeneous IoT devices with low communication overhead.**

***Index Terms*—Deep mutual learning (DML), federated learning (FL), Internet of Things (IoT), model heterogeneity, neural network (NN).**

I. Introduction

ALONG with the increasing popularity of artificial intelligence (AI), more and more devices of AI Internet of Things (AIoT) systems (e.g., autonomous driving, commercial surveillance, industrial control, and medical monitoring) are equipped with deep neural network (DNN) models for the purposes of accurate sensing and intelligent control [1], [2]. However, due to both the restricted access to global data and limited classification capabilities of local models, the inference quality of AIoT devices cannot be guaranteed, especially when AIoT systems are deployed within a dynamic and uncertain environment. To enhance the inference performance of AIoT devices, we are witnessing an increasing number of modern large-scale AIoT systems [3], [4] that resort to the enormous computing power of cloud computing to quickly figure out inference results. Although such cloud-based AIoT systems are promising in managing a large number of devices under a global system view, due to the privacy issues, most of them cannot share the local data of a device to other devices in the same system, thus the inference capability of AIoT devices is still greatly restricted in practice.

Due to the merit of central model training on decentralized device data without compromising user privacy, federated learning (FL) [5] allows knowledge sharing among devices and is becoming an emerging collaborative AI paradigm in Internet of things (IoT) design. Instead of uploading local data, FL methods only send the gradients of DNN models to the cloud for aggregation. As thus, the inference capability of all the involved AIoT devices are improved, while the privacy of devices can be guaranteed. Fig. 1 presents an overview of the FL architecture for an AIoT application, where all the AIoT devices are connected to a cloud server. Note that the upload operation of classical FL assumes that the local device models

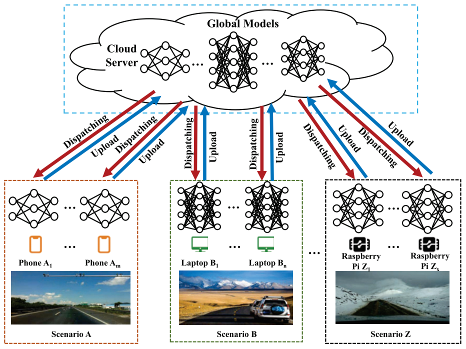


Fig. 1. Overview of the FL architecture for AIoT applications.

Manuscript received 5 August 2022; accepted 5 August 2022. Date of publication 20 October 2022; date of current version 24 October 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB2101300; in part by the Natural Science Foundation of China under Grant 61872147; in part by the Shanghai Trusted Industry Internet Software Collaborative Innovation Center; and in part by the Fundamental Research Funds for the Central Universities under Grant YBNLTS2022-008. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2022 and appears as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding author: Mingsong Chen.)*

Jun Xia, Tian Liu, Zhiwei Ling, Ting Wang, and Mingsong Chen are with the MoE Engineering Research Center of Hardware/Software Co-Design Technology and Application, East China Normal University, Shanghai 200062, China (e-mail: mschen@sei.ecnu.edu.cn).

Xin Fu is with the ECE Department, University of Houston, Houston, TX 77204 USA.

Digital Object Identifier 10.1109/TCAD.2022.3197491

have the same architecture as the global model on the cloud to aggregate. However, this assumption is too ideal for modern AIoT systems, which typically comprise a variety of devices equipped with heterogeneous DNN models. The violation of the assumption strongly hinders the deployment of FL on AIoT systems. Therefore, how to break through the barrier of model heterogeneity and enable effective knowledge sharing among devices is becoming a major bottleneck in the design of FL framework for AIoT systems.

To address the model heterogeneity problem in FL, various methods (e.g., FedPer [6], FedDF [7], and HeteroFL [8]) have been investigated. However, most of them focus on the knowledge sharing by uploading local models or soft labels rather than gradients to the cloud, thus the user privacy cannot be guaranteed [9]. Worse still, some of them make unrealistic architecture-oriented assumptions about DNN implementations in practice. For example, HeteroFL requires that the parameters of its local models should be a subset of its global model parameters. However, sharing the same part of the neural network (NN) model is not efficient for learning valuable knowledge in the FL scenario. FedMD requires that a client in FL uses a global public dataset for knowledge distillation.

To enable secure FL for IoT devices equipped with heterogeneous DNN models, this article proposes a novel lightweight cloud-based FL framework named *PervasiveFL*, which enables the training of heterogeneous device models without losing any assumptions. Moreover, PervasiveFL can be easily implemented and allows pervasive FL on large-scale heterogeneous IoT systems for both independent and identically distributed (IID) and non-IID scenarios. This article makes the following three major contributions.

1) We introduce the concept of *modellet* that acts as an omnipotent portal for FL. Based on the deep mutual learning (DML) [10], our approach allows the mutual learning between modellets and local models on devices. Since modellets in an AIoT system are of the same architecture, they enable the FL-like knowledge sharing among heterogeneous devices with different types of local models.
2) Based on our proposed entropy-based loss functions and the entropy-based decision gating (EDG) method, we develop an effective ensemble scheme that supports selective knowledge sharing between modellets and local models, thus maximizing the benefits of modellets in various FL scenarios.
3) We conduct comprehensive experiments on four well-known datasets from different fields. Experimental results show that PervasiveFL can accommodate large-scale IoT systems involving various types of local models. Compared with both nonFL local training methods and state-of-the-art FL approaches, PervasiveFL can not only achieve better inference performance, but also have smaller FL computation and communication overhead.

The remainder of this article is organized as follows. Section II introduces the related work on FL for IoT systems. Section III introduces the preliminaries of FL and DML. Section IV details our proposed PervasiveFL approach. Section V presents the performance evaluation results on four datasets. Finally, Section VI concludes this article.

II. Related Work

Although FL enables collaborative AI among IoT devices, it suffers from the problem of heterogeneity, which can be classified into three categories, i.e., device heterogeneity, data heterogeneity, and model heterogeneity. The device heterogeneity refers to the heterogeneity caused by the difference in hardware resources (e.g., CPU, memory, and network), which strongly affects the training and transmission time of individual devices. To address this issue, various cloud-edge-based scheduling methods have been proposed to optimize the response time of devices for efficient gradient aggregation on cloud. For example, Liu *et al.* [11] presented a client-edge-cloud hierarchical FL system that combines the advantages of both the cloud and edge servers. Their proposed HierFAVG algorithm allows partial model aggregation on edge servers, thus the model can be trained more quickly and better communication–computation tradeoffs can be achieved.

The data heterogeneity considers the scenarios where IoT devices are distributed in open environments. In this case, the distributions of training data captured by devices are different, which strongly affects FL convergence performance. To further understand this problem, Li *et al.* [12] conducted the theoretical analysis of convergence bounds for FedAvg, which shows that heterogeneity of training data and partial device participation can slow down the convergence. To speedup FL convergence, Wang *et al.* [13] proposed an experience-driven control framework that intelligently chooses client devices to participate in each round of FL to counterbalance the bias introduced by non-IID data.

Model heterogeneity has been widely investigated by various non-FL methods [14], [15] to improve the inference capabilities of device models individually. However, the model heterogeneity problem is still a major bottleneck in FL design. Although existing transfer learning and knowledge distillation-based FL approaches (e.g., FedDF [7], MIFL [16], FedGEN [17], and MOON [18]) can achieve knowledge sharing among heterogeneous devices, most of them need to upload local models or soft labels to the cloud. Consequently, the user privacy can be easily leaked and the communication overhead is formidable for these methods. As an alternative technology, FedPer [6] and HeteroFL [8] do not rely on transfer learning and knowledge distillation for knowledge sharing. However, both of them are based on non-negligible assumptions on the structure of models or parameters, which significantly restricts the use of FL in IoT design.

To the best of our knowledge, PervasiveFL is the first attempt that combines the benefits of DML and EDG to address the problem of model heterogeneity on IoT devices with any types of DNN models. Compared with state-of-the-art FL methods, due to the small size of modellets, the implementation of PervasiveFL for large-scale IoT systems is simpler and the computation and communication overhead is much lower.

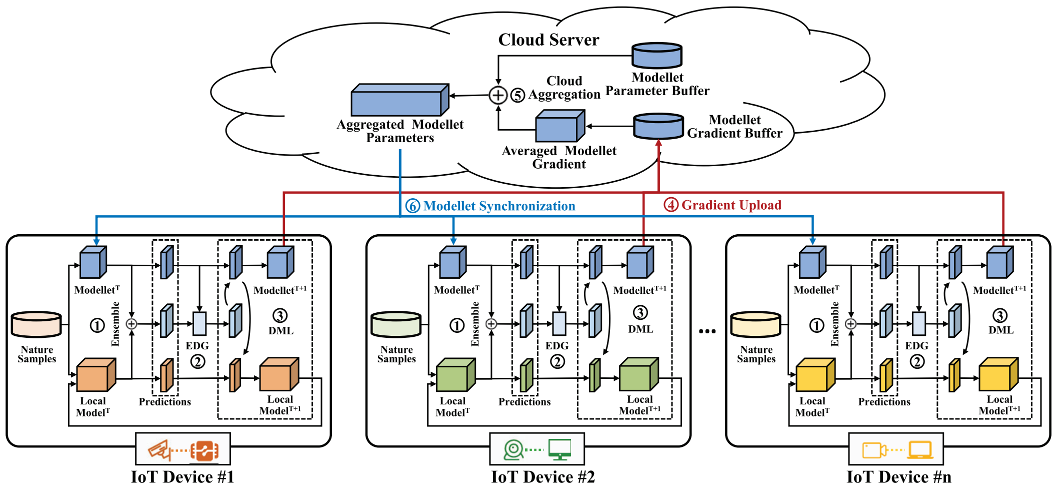


Fig. 2. Architecture and workflow of PervasiveFL training.

III. Preliminaries

A. Federated Learning

Along with the prosperity of distributed machine learning technologies [19], [20], privacy-aware FL is proposed to effectively solve the problem of data silos, where all the involved IoT devices can achieve knowledge sharing without leaking data privacy. Assume that there are a total of K devices in an AIoT system, and in the t-th FL communication round there are N ($N \leq K$) devices selected. At the end of the t-th communication round, we need to update each local model of the selected devices as follows:

$$\mathbb{L}_{t+1}^{k} \leftarrow \mathbb{W}_t + \eta \nabla \mathbb{L}_t^k \tag{1}$$

where $\mathbb{L}_{t+1}^{k}$ denotes the local model of the kth selected device in round $t+1$, and $\mathbb{W}_t$ represents the global model obtained in round t. Here, η is the learning rate and $\nabla \mathbb{L}_t^k$ is the gradient obtained from $\mathbb{L}_t^k$. To protect data privacy of devices, at the end of each communication round, each selected device only uploads its model gradients (i.e., weight differences) rather than its newly updated model to the cloud for aggregation. In each FL communication round, once finishing the gathering of gradients from all the selected devices, the classic FL method (i.e., Fedavg [21]) needs to update global model parameters as follows:

$$\mathbb{W}_{t+1} \leftarrow \mathbb{W}_t + \frac{\sum_{i=1}^{N} \nabla \mathbb{L}_t^i}{N} \tag{2}$$

where $((\sum_{i=1}^{N} \nabla \mathbb{L}_t^i)/N)$ denotes the average gradient of N participating devices in communication round t.

B. Deep Mutual Learning

The DML method is inspired by the fact that soft labels contain more knowledge than hard labels [10], [22]. Suppose that there are two NN models (i.e., $\mathbb{W}$ and $\mathbb{S}$) participating in DML, whose training data and corresponding labels are x and y, respectively. The loss function of DML is defined as follows:

$$\mathfrak{L}_{\mathbb{W}}(x, y, \mathcal{P}_{\mathbb{W}}, \mathcal{P}_{\mathbb{S}}) = \mathfrak{L}_{\text{Hard}}^{\mathbb{W}}(x, y) + \mathfrak{L}_{\text{Soft}}^{\mathbb{W}}(\mathcal{P}_{\mathbb{W}}, \mathcal{P}_{\mathbb{S}}) \tag{3}$$

where $\mathfrak{L}_{\text{Hard}}^{\mathbb{W}}(x, y)$ is the cross-entropy loss function and $\mathfrak{L}_{\text{Soft}}^{\mathbb{W}}(\mathcal{P}_{\mathbb{W}}, \mathcal{P}_{\mathbb{S}})$ is an imitative loss function that can unify classification probabilities among different students. Based on the same training data, the two models exchange their soft labels in each training epoch during the mutual learning process of DML. In this way, the inference performance of all the participating models can be significantly improved.

IV. Our PervasiveFL Framework

This section details the design and implementation of our PervasiveFL framework. First, it introduces the architecture and workflow of PervasiveFL. Then it presents the modeling of loss functions used in PervasiveFL. Finally, it describes the implementation of PervasiveFL in detail and proves the convergence of modellets in PervasiveFL.

A. Architecture and Workflow of PervasiveFL

PervasiveFL focuses on the sharing of benign knowledge among heterogeneous device models, which can not only improve the inference performance of both local models and modellets but also reduce both computation and communication overhead.

1) Architecture of PervasiveFL: Different from traditional FL methods that require homogeneous local device models PervasiveFL allows FL based on heterogeneous local models so as to best fit for the resource capacities of devices. In order to achieve this goal, PervasiveFL introduces a lightweight NN model, named modellet, to eclipse the heterogeneity of the local models on devices. Fig. 2 presents the architecture of PervasiveFL together with its workflow. As illustrated in the figure, the PervasiveFL framework consists of two parts: 1) the cloud server that mainly focuses on the aggregation of modellets and 2) heterogeneous edge IoT devices that concentrate on the training of local models and modellets based on their locally collected data. Note that PervasiveFL adopts the same FL framework as the classic FedAvg.

In PervasiveFL, each IoT device is equipped with one local model and one modellet, where the local model is gradually trained using the locally captured data (i.e., nature samples). Different from local models which are heterogeneous, the modellets installed on all devices are homogeneous. Note that it is not required that modellets are installed on devices initially. Instead, it needs to install a modellet for knowledge sharing only when a device wants to join the PervasiveFL to enhance its inference capability. In PervasiveFL, the modellet acts like a portal of its host device to enable FL among devices. As a small-size NN model, it can teach the local model with the shared knowledge from other devices based on DML. Meanwhile, it can propagate the newly learned knowledge by the local model to other devices in an FL manner. Note that there is no requirement that modellets need to be pretrained before installation. The first usable version of a modellet on some device can be downloaded from the cloud server or be locally trained from scratch based on DML. As for the cloud server, it is responsible for three tasks: 1) collecting the gradients of modellets trained on distributed devices and saving it in the modellet gradient buffer; 2) globally averaging the collected modellet gradients to form a newly aggregated modellet; and 3) dispatching the aggregated modellet to all the AIoT devices for further local training or inference.

2) Workflow of PervasiveFL: In PervasiveFL, the cloud server and all the connected AIoT devices work closely to enable the global learning via heterogeneous device models. Before PervasiveFL starts, all the devices requesting to participate in PervasiveFL should complete the installation and initialization of a modellet. Afterward, PervasiveFL will perform local training on AIoT devices, model aggregation in the cloud, and model synchronization between the AIoT devices and the cloud. As shown in Fig. 2, the PervasiveFL training workflow involves six steps as follows.

Step 1 (Ensemble): At the beginning of local training, each selected device makes inferences from its local data based on the *ensemble model*, whose predictions (i.e., soft labels) are the average of predictions made by the pair of its corresponding modellet and local model.

Step 2 (EDG): To judge the quality of knowledge that can be shared between modellets and local models, we proposed the EDG method, which compares the entropy of predictions [see details in (8)] between modellets and ensemble models.

Step 3 (DML): Unlike traditional DML, in PervasiveFL we find that it is unwise to conduct mutual learning "equally" between modellets and local models based on locally captured data. This is because in non-IID scenarios the knowledge structure (i.e., compositions and distributions) of modellets are quite different from the ones of local models. During the DML, if a modellet learns the knowledge from its local model counterpart without any filtering, the knowledge structure of such a modellet could be degraded due to the biased device data. To address this issue, our approach uses the ensemble model together with our proposed EDG method to enable knowledge filtering, which can ensure the quality of the learned knowledge by modellets from local models. Based on the entropy definition [see (8)], PervasiveFL computes the entropy of predictions for both modellets and ensemble models. If the soft label entropy of some ensemble model is smaller than that of its corresponding modellet, the DML will be conducted between the modellet and the ensemble model rather than its local model. Note that in our approach, local models need to learn all the knowledge from the modellets.

Step 4 (Gradient Upload): At the end of local training, an AIoT device will upload its modellet gradients to the cloud server, which are stored in the gradient buffer as shown in Fig. 2. Such collected gradients will then be averaged to derive an aggregated modellet. Since the size of a modellet is small, in this article we do not take any communication optimization methods (e.g., gradient compression [23]) into account. Moreover, when uploading model gradients to the cloud, there could be some risk of privacy leak. Note that since our approach uses the same FL framework as FedAvg, existing privacy protection methods [24], [25] for FL can be easily integrated into PervasiveFL to address this issue.

Step 5 (Cloud Aggregation) When modellet gradients of all the selected devices are received, this step will average such gradients and use this information to form a new modellet.

Step 6 (Modellet Synchronization): The aggregated modellet on the cloud server will be dispatched to all the selected AIoT devices for the next epoch training.

PervasiveFL repeats all the above six steps until the convergence of both modellets and local models. During the DML process, a local model and its corresponding modellet need to selectively learn from each other in each round of PervasiveFL. Note that at the end of DML, the test accuracy of modellets and local models could diverge significantly. However, this will not affect the convergence of both modellets and local models. This is because in DML the mutual learning processes between modellets and local models are based on local device data. If we do not take DML into account, both the modellet-based FL method and the pure local training method will converge eventually. Since the DML scheme can enhance the learning efficacy of both modellets and local models on local device data, when DML is applied, the convergence of both modellets and local models will be accelerated.

B. Loss Function Modeling

Loss functions play an important role in guiding the model training through measuring the difference between model prediction values and corresponding label values. On the device side of PervasiveFL, there are two different types of models on each device, i.e., the local model and the modellet. In PervasiveFL, we resort to DML to enable the knowledge sharing between the two models on each device. To fully make the full use of DML, in PervasiveFL we adopt two different loss functions for both the modellets and local models, respectively.

Inspired by the work in [10], PervasiveFL uses the following loss function for a modellet $\mathbb{M}$ to enable mutual learning on some device:

$$\mathfrak{L}_{\mathbb{M}}(x, y, \mathcal{P}_{\mathbb{M}}, \mathcal{P}_{\mathbb{D}}) = \mathfrak{L}^{\mathbb{M}}_{\text{Hard}}(x, y) + \mathfrak{L}^{\mathbb{M}}_{\text{Soft}}\left(\mathcal{P}_{\mathbb{M}}, \frac{\mathcal{P}_{\mathbb{M}} + \mathcal{P}_{\mathbb{D}}}{2}\right) \quad (4)$$

where $\mathfrak{L}_{\mathbb{M}}$ represents the overall loss function of deep mutual training for modellet $\mathbb{M}$, x indicates the original sample values, y denotes the classification labels, and $\mathbb{D}$ refers to the local model in DML. $\mathcal{P}_{\mathbb{M}}$ and $\mathcal{P}_{\mathbb{D}}$ represent the probability distributions of $\mathbb{M}$ and $\mathbb{D}$ for the same batch of data, respectively. $\mathfrak{L}^{\mathbb{M}}_{\text{Hard}}(x, y)$ denotes the cross-entropy loss function for $\mathbb{M}$, which measures the difference between the maximum values in model's probability distributions and actual label values. $\mathfrak{L}^{\mathbb{M}}_{\text{Soft}}(\mathcal{P}_{\mathbb{M}}, \mathcal{P}_{\mathbb{D}})$ is a loss function for $\mathbb{M}$ that indicates the Kullback–Leibler (KL) divergence during training, where the loss is calculated as the probability distribution difference between $\mathbb{M}$ and its ensemble model for the same data batch.

According to the findings in [10], regardless of the difference of heterogeneous NN models deployed on different devices, the DML approach can be used to facilitate local models and modellets to learn from each other through mutual knowledge transfer, which can greatly improve the performance of both models. To achieve this goal, we optimize the modellet parameters (i.e., $w_{\mathbb{M}}$) on a device based on (5), i.e.,

$$w_{\mathbb{M}} = \operatorname{argmin}_{w_{\mathbb{M}}} \mathbb{E}_x\{\mathfrak{L}_{\mathbb{M}}(x, y, \mathcal{P}_{\mathbb{M}}, \mathcal{P}_{\mathbb{D}})\}. \tag{5}$$

To achieve globally optimized modellets in an FL manner, we design the following loss function for modellets as a whole

$$\mathfrak{L}^{\text{Fed}}_{\mathbb{M}} = \sum_{i=1}^{N} \frac{\tau_{x^i} \cdot \mathfrak{L}_{\mathbb{M}}\left(x^i, y^i, \mathcal{P}^i_{\mathbb{M}}, \mathcal{P}^i_{\mathbb{D}}\right)}{\sum_{j=1}^{N} \tau_{x^j}} \tag{6}$$

where τ_x means the data size of x, x^i indicates the nature samples on the i-th device, $\mathfrak{L}^{\text{Fed}}_{\mathbb{M}}$ equals to the weighted average of all the loss function values of modellets on all the N devices. As for the local model on each device, its loss function is formulated as follows:

$$\mathfrak{L}_{\mathbb{D}}(x, y, \mathcal{P}_{\mathbb{M}}, \mathcal{P}_{\mathbb{D}}) = \mathfrak{L}^{\mathbb{D}}_{\text{Hard}}(x, y) + \mathfrak{L}^{\mathbb{D}}_{\text{Soft}}(\mathcal{P}_{\mathbb{D}}, \mathcal{P}_{\mathbb{M}}) \tag{7}$$

where the definitions of $\mathfrak{L}^{\mathbb{D}}_{\text{Hard}}(x, y)$ and $\mathfrak{L}^{\mathbb{D}}_{\text{Soft}}(\mathcal{P}_{\mathbb{D}}, \mathcal{P}_{\mathbb{M}})$ are similar to the ones in (4).

C. Entropy-Based Decision Gating

Inspired by the entropy-based classification method presented in [26] that uses the entropy of soft labels to judge the prediction confidence of some classifier, we use the soft label entropy of both the modellet and ensemble model to determine the usefulness of the current training data batch to the modellet. Based on the classic definition, we calculate the entropy using the following equation:

$$\text{entropy}(y) = \sum_{c \in \mathcal{C}} y_C \cdot \log(y_C) \tag{8}$$

where $\mathcal{C}$ and y are class labels and predictions for corresponding classes, respectively. By using this equation, we can calculate the soft label entropy of both modellets and ensemble models for the comparison purpose. If the modellet entropy is higher than the ensemble model entropy, the prediction confidence of the modellet is lower than the ensemble model. In this case, our EDG-based DML method will use the soft labels of an ensemble model to train its corresponding modellet. Otherwise, the modellet will be trained by the hard labels of its local natural samples.

Algorithm 1: Local Training Procedure (Devices)

```
Input: i) $S$, the cloud server; ii) $k$, index of device; iii)
       $E$, number of training epochs; iv) $\mathbb{M}$, modellet; v)
       $\mathbb{D}$, local device model.
Output: $w_{\mathbb{M}}$, $w_{\mathbb{D}}$
1  $(x, y) = Collect()$;
2  for $e \leftarrow 1$ to $E$ do
3      $(\mathcal{V}^{\mathbb{M}}, \mathcal{P}^{\mathbb{M}}) \leftarrow \mathbb{M}(x)$;
4      $(\mathcal{V}^{\mathbb{D}}, \mathcal{P}^{\mathbb{D}}) \leftarrow \mathbb{D}(x)$;
5      $\mathcal{P}^{En} \leftarrow (\mathcal{P}^{\mathbb{D}} + P^{\mathbb{M}})/2$;
6      $\mathfrak{L}^{\mathbb{M}}_{Hard} \leftarrow CrossEntropy(y, \mathcal{V}^{\mathbb{M}})$;
7      $\mathfrak{L}^{\mathbb{D}}_{Hard} \leftarrow CrossEntropy(y, \mathcal{V}^{\mathbb{D}})$;
8      $\mathfrak{L}^{\mathbb{M}}_{Soft} \leftarrow KLLoss(\mathcal{P}^{\mathbb{M}}, \mathcal{P}^{En})$;
9      $\mathfrak{L}^{\mathbb{D}}_{Soft} \leftarrow KLLoss(\mathcal{P}^{\mathbb{D}}, \mathcal{P}^{\mathbb{M}})$;
10     $\mathfrak{L}_{\mathbb{M}} \leftarrow \mathfrak{L}^{\mathbb{M}}_{Hard}$;
11     if $entropy(\mathcal{P}^{\mathbb{M}}) > entropy(\mathcal{P}^{En})$ then
12         $\mathfrak{L}_{\mathbb{M}} \leftarrow \mathfrak{L}^{\mathbb{M}}_{Hard} + \mathfrak{L}^{\mathbb{M}}_{Soft}$;
13     end
14     $\mathfrak{L}_{\mathbb{D}} \leftarrow \mathfrak{L}^{\mathbb{D}}_{Hard} + \mathfrak{L}^{\mathbb{D}}_{Soft}$;
15     $\nabla w^e_{\mathbb{M}} \leftarrow SGD.get\_gradients(\mathfrak{L}_{\mathbb{M}}, w_{\mathbb{M}})$;
16     $\nabla w^e_{\mathbb{D}} \leftarrow SGD.get\_gradients(\mathfrak{L}_{\mathbb{D}}, w_{\mathbb{D}})$;
17     $w_{\mathbb{D}} \leftarrow Update(\nabla w^e_{\mathbb{D}}, w_{\mathbb{D}})$;
18     $Send(\nabla w^e_{\mathbb{M}}, k, S)$;
19     $w_{\mathbb{M}} \leftarrow Receive()$;
20 end
```

D. Implementation of PervasiveFL

This section presents the implementation of PervasiveFL on devices and cloud, respectively. Algorithm 1 describes the local training procedure on devices and Algorithm 2 depicts the model aggregation procedure on the cloud server.

1) Implementation of Local Training: Assume that there are N devices involved in the training process of PervasiveFL, and the index of the current device is k. Algorithm 1 details the local training procedure for a device. Note that this procedure will be invoked on all the participating devices if a new request of local training is launched in PervasiveFL.

In line 1, the device collects a set of labeled nature samples for the purpose of local training. Here, we use x and y to denote the collected training data and their labels, respectively. Lines 2–20 conduct E epochs of local training, where each epoch indicates one interaction with the cloud server for the modellet update. Lines 3 and 4 apply the training dataset on both modellet $\mathbb{M}$ and local model $\mathbb{D}$, and figure out their corresponding model prediction results (i.e., $\mathcal{V}^{\mathbb{M}}$, $\mathcal{V}^{\mathbb{D}}$) and soft labels (i.e., $\mathcal{P}^{\mathbb{M}}$, $\mathcal{P}^{\mathbb{D}}$), respectively. Line 5 calculates the ensemble predictions by both the modellet and local model. Based on the obtained model prediction results and soft labels, lines 6 and 7 calculate the cross-entropy losses, and lines 8 and 9 calculate the KL divergence losses for both the modellet and local model. Lines 10–14 calculate the total losses

Algorithm 2: Training Procedure (Cloud)

```
Input: i) $N$, number of devices; ii) $E$, number of
       training epochs; iii) $\mathbb{M}$, global modellet.
1  while true do
2      for $e \leftarrow 1$ to $E$ do
3          for $k \leftarrow 1$ to $N$ do
4              $\nabla w_{\mathbb{M}}^{e,k} \leftarrow Receive(k)$;
5          end
6          $\nabla w_{\mathbb{M}}^{e} \leftarrow \frac{1}{N}\sum_{k=1}^{N}\nabla w_{\mathbb{M}}^{e,k}$;
7          $w_{\mathbb{M}} \leftarrow SGD.get\_weights(w_{\mathbb{M}}, \nabla w_{\mathbb{M}}^{e})$;
8          for $k \leftarrow 1$ to $N$ do
9              $Send(k, w_{\mathbb{M}})$;
10         end
11     end
12 end
```

for both $\mathbb{M}$ and $\mathbb{D}$ based on our EDG method. Lines 15 and 16 figure out the gradients for both $\mathbb{M}$ and $\mathbb{D}$ using the classic stochastic gradient descent (SGD) method, where $w_{\mathbb{M}}$ and $w_{\mathbb{D}}$ denote the parameters of $\mathbb{M}$ and $\mathbb{D}$, respectively. Line 17 updates the parameters of $\mathbb{D}$ based on the newly obtained gradients from line 16. For the knowledge sharing, line 18 sends the gradients of $\mathbb{M}$ to the cloud server S. Line 19 waits for the aggregated modellet information from the server, and uses it for the update of its local modellet. Note that *Receive()* is a blocking function.

2) Implementation of Cloud Aggregation: Algorithm 2 presents the PervasiveFL training process on the cloud server, including averaging gradients and model aggregation.

As shown in Algorithm 2, the cloud server periodically conducts the PervasiveFL training, where each training involves E epochs (lines 2–11). In the algorithm, lines 3–5 try to receive all the modellet gradients from the N heterogeneous devices, and line 6 calculates the average of all the received gradients. Note that the function *Receive()* in line 4 is a blocking function. Line 7 performs the modellet aggregation by applying the averaged gradients to the modellet obtained in previous training round. Lines 8–10 dispatch the newly aggregated modellet to all the N devices in an asynchronous way, i.e., the function *Send()* is nonblocking.

During the PervasiveFL training, it is possible that some local IoT device crashes due to some failure. To deal with such a scenario where some local IoT device does not answer, we can set a timeout for each device. If a device does not reply in time for a given timeout, we can safely remove it from the FL group. Note that PervasiveFL allows to hot-plug IoT devices. Since PervasiveFL adopts the same FL framework as the classic FedAvg, any solutions to address the problems in FL, such as device crashes and training synchronization are also suitable for PervasiveFL.

E. Proof of PervasiveFL Convergence

In our approach, we assume that the soft labels generated by local models will not change (or the change is negligible) after a specific number of training rounds. This is because the number of training data on a device is limited and the sizes of local models are typically larger than the modellet size. Note that the local model training involves two loss functions, i.e., cross-entropy loss for the training on nature samples, and KL divergence loss for the distribution differences between modellet soft labels and local model soft labels. Here, the contribution of KL divergence loss to the overall loss is much smaller than the one of cross-entropy loss. Therefore, after a certain number of local training, the local models will converge earlier than the modellet. Since the local device model fits the local data better than the modellet, we assume that the entropy of local model soft labels is smaller than that of the modellet soft labels. Similarly, we can assume that within a device the entropy of ensemble model soft labels is lower than that of modellet soft labels in most cases since the ensemble model soft labels are the average of both the modellet soft labels and local model soft labels. Based on the following six assumptions, we can prove the modellet convergence of PervasiveFL.

Assumption 1: After a certain number of rounds, Algorithm 1 will always execute the *if* branch for any device.

Assumption 2: After a certain number of rounds, soft labels generated by local models do not change for all the devices.

The cross-entropy loss computes the distance between predictions of modellet and hard labels. Therefore, cross-entropy loss only depends on modellet w. The KL divergence loss computes the difference between the predictions of modellet and that of local ensemble model (i.e., the average predictions of modellet and local model). Due to Assumption 2, the predictions of local models are fixed. Therefore, the KL divergence loss only depends on modellet w.

According to Assumptions 1 and 2, after a certain number of training rounds, we can achieve the optimization objective is defined as follows:

$$\min_{w}\left\{\Phi(w) = \sum_{k=1}^{N}\Theta_k(f_k(w) + \rho_k(w))\right\} \tag{9}$$

where N is the number of participating devices in PervasiveFL, Θ_k is the weight of the k-th device such that $\Theta_k \geq 0$ and $\sum_{k=1}^{N}\Theta_k = 1$. $f_k(w)$ and $\rho_k(w)$ are two loss functions, where $f_k(w)$ denotes the distance between some hard label and its target, and $\rho_k(w)$ denotes the difference between probability distributions of $\mathbb{M}$ and $((\mathbb{D}+\mathbb{M})/2)$ predictions. Here, $\mathbb{M}$ is the modellet and $\mathbb{D}$ is the local model, where the probability distribution of $\mathbb{D}$'s predictions is fixed. Similar to [12], we use the following four assumptions based on the two loss function sets, i.e., $\bigcup_{i=1}^{N}\{f_i\}$ and $\bigcup_{i=1}^{N}\{\rho_i\}$.

Assumption 3: Assume that the elements in the two function sets (i.e., $f_1,\ldots,f_N$ and $\rho_1,\ldots,\rho_N$) are *M-smooth*, i.e., for all $\mathbb{A}$ and $\mathbb{B}$, $f_k(\mathbb{A}) - f_k(\mathbb{B}) - (\mathbb{A}-\mathbb{B})^T\nabla f_k(\mathbb{B}) \leq (M/2)||\mathbb{A}-\mathbb{B}||^2$, $\rho_k(\mathbb{A}) - \rho_k(\mathbb{B}) - (\mathbb{A}-\mathbb{B})^T\nabla\rho_k(\mathbb{B}) \leq (M/2)||\mathbb{A}-\mathbb{B}||^2$.

Assumption 4: Assume that the elements in the two function sets (i.e., $f_1,\ldots,f_N$ and $\rho_1,\ldots,\rho_N$) are *μ-strongly convex*, i.e., for all $\mathbb{A}$ and $\mathbb{B}$, $f_k(\mathbb{A}) - f_k(\mathbb{B}) - (\mathbb{A}-\mathbb{B})^T\nabla f_k(\mathbb{B}) \geq (\mu/2)||\mathbb{A}-\mathbb{B}||^2$, $\rho_k(\mathbb{A}) - \rho_k(\mathbb{B}) - (\mathbb{A}-\mathbb{B})^T\nabla\rho_k(\mathbb{B}) \geq (\mu/2)||\mathbb{A}-\mathbb{B}||^2$.

TABLE I
MODEL SETTINGS FOR DIFFERENT DATASETS

Model Type	**CIFAR10**		**CIFAR100**		**ImageNet-10**		**Shakespeare**	
	Configuration	**Size (M)**	**Configuration**	**Size (M)**	**Configuration**	**Size (M)**	**Configuration**	**Size (M)**
Modellet	ResNet20_CIFAR10	0.87	ResNet20_CIFAR100	0.95	MobileNetV2	8.71	CharLSTM-64	0.23
Small	ResNet56_CIFAR10	2.33	ResNet56_CIFAR100	2.42	ResNet18	42.70	CharLSTM-128	0.71
Middle	ResNet101_CIFAR10	4.17	ResNet101_CIFAR100	4.26	ResNet50	90.00	CharLSTM-256	3.3
Large	MobileNetV2	8.71	MobileNetV2	9.23	ResNet101	162.00	CharLSTM-512	12.9

Assumption 5: Let ξ_t^k be the samples that are randomly sampled from local data of the *kth* device in a uniform way. For the modellet in each device, the variance of its stochastic gradients is bounded in each communication round, i.e., for each $k \in \{1, \ldots, N\}$, $\mathbb{E}||\nabla f_k(w_t^k, \xi_t^k) - \nabla f_k(w_t^k)||^2 \leq \alpha_k^2$ and $\mathbb{E}||\nabla \rho_k(w_t^k, \xi_t^k) - \nabla \rho_k(w_t^k)||^2 \leq \beta_k^2$.

Assumption 6: The expected squared norm of stochastic gradients is upper bounded, i.e., for each $t \in \{1, \ldots, T\}$ and each $k \in \{1, \ldots, N\}$, $\mathbb{E}||\nabla f_k(w_t^k, \xi_t^k)||^2 \leq G_1^2$ and $\mathbb{E}||\nabla \rho_k(w_t^k, \xi_t^k)||^2 \leq G_2^2$.

Based on above six assumptions, we analyze the convergence rate of modellet in PervasiveFL with full device participation as follows. In our approach, the model update for modellets is defined as follows:

$$\begin{aligned} v_{t+1}^k &= w_t^k - \eta_t\left(\nabla f_k\left(w_t^k, \xi_t^k\right) + \nabla \rho_k\left(w_t^k, \xi_t^k\right)\right) \\ w_{t+1}^k &= \begin{cases} v_{t+1}^k, & \text{if} \quad T \nmid (t+1) \\ \sum_{k+1}^N \Theta_k v_{t+1}^k, & \text{if} \quad T \mid (t+1) \end{cases} \end{aligned} \tag{10}$$

where w_t^k is the local model of the k-th device at the t-th SGD step, v_{t+1}^k is the immediate result of w_t^k after executing one SGD update, and η_t is the learning rate of t-th round. T is a local SGD step within one training round. If T can be divided by $t+1$, all the devices are activated. Similar to [12], we define two virtual sequences as follows:

$$\overline{v}_t = \sum_{k=1}^N \Theta_k v_t^k, \quad \overline{w}_t = \sum_{k=1}^N \Theta_k w_t^k. \tag{11}$$

By combining (10) and (11), we always have $\overline{v}_t = \overline{w}_t$. We define $\overline{\Delta}_t = \sum_{k=1}^N \Theta_k[\nabla f_k(w_t^k) + \nabla \rho_k(w_t^k)]$ and $\Delta_t = \sum_{k=1}^N \Theta_k[\nabla f_k(w_t^k, \xi_t^k) + \nabla \rho_k(w_t^k, \xi_t^k)]$, where Δ_t is the gradient and $\overline{\Delta}_t$ is the expectation of gradient. Therefore, $\overline{v}_{t+1} = \overline{w}_t - \eta_t \Delta_t$ and $\mathbb{E}[\Delta_t] = \overline{\Delta}_t$.

Let $\Phi^\star$ be the optimal value of the loss function (9), and $w^\star$ be its corresponding parameter. According to the lemmas presented in [12], [27], we can get the inequality as follows:

$$\mathbb{E}\Phi(\overline{w}_t) - \Phi^\star \leq \frac{M}{\mu(\gamma + t - 1)}\left(\frac{2\mathcal{Z}}{\mu} + \frac{\mu\gamma}{2}\mathbb{E}||\overline{w}_1 - w^\star||^2\right) \tag{12}$$

where $\mathcal{Z} = 32(T-1)^2(G_1^2 + G_2^2)\sum_{k=1}^N \Theta_k^2(\alpha_k^2 + \beta_k^2) + 8M\Gamma$ and γ is a constant defined in [12]. The right-hand side of the inequality (12) decreases as t increases. Therefore, we prove the convergence of PervasiveFL.

V. EXPERIMENTS

To evaluate the effectiveness of our approach, we implemented our PervasiveFL framework using Pytorch (version 1.4.0). Aiming at eclipsing the impact of randomness during PervasiveFL, we set the participation ratio of devices in each round to 1. We set the mini-batch size of both DML and modellet-based FL to 64. All the experiments were conducted on an Ubuntu workstation (with Intel i9 CPU, 16-GB memory, and GTX3080 GPU) and ten Nvidia Jetson Nano boards (with ARM Cortex-A57 processor and 4-GB memory). Note that the ten boards were used to emulate partial involved devices with heterogeneous models, while the other remaining devices (only for the scalability analysis in Section V-C1) were simulated on the workstation. The Jetson Nano boards connect to the cloud server (i.e., the workstation) via a WiFi environment.

A. Experimental Settings

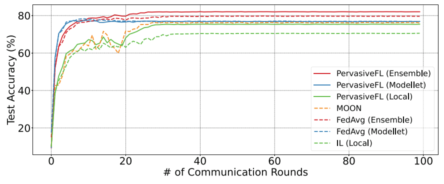
1) Dataset Settings: To comprehensively justify the "pervasiveness" of our approach, we considered two kinds of datasets: 1) three image datasets, i.e., CIFAR10, CIFAR100 [28], and ImageNet-10 [29] and 2) one text dataset, i.e., Shakespeare [30]. For the image datasets, both CIFAR10 and CIFAR100 have a training set of 500 00 images and a test set of 100 00 images, respectively. Based on the first ten categories of ImageNet, ImageNet-10 has a training set of 130 00 images and a test set of 500 images. To facilitate the training of image datasets, the data augmentation includes horizontal flips and random crops from images that are padded by four pixels on each side, where the missing pixels of an image are filled with the reflections of its original version. Since one complex IoT scenario may involve a large quantity of devices, to avoid the overfitting problem in PervasiveFL caused by insufficient training data for individual devices, we expanded the PervasiveFL training datasets using the generative adversarial network (GAN) method [31] in the three image datasets. As a text dataset, Shakespeare involves 1129 users, where each user has an average of 3734.2 training samples. Note that since the samples in Shakespeare follow a non-IID distribution naturally, we did not consider the IID performance of PervasiveFL on Shakespeare in the experiment.

2) Model Settings: Since different datasets require different DNN models and devices with different sizes can accommodate DNN models with different scales, in this experiment, we considered two kinds of heterogeneity (i.e., model type and memory size) for DNN models under the same PervasiveFL framework. Table I shows the model settings for the four datasets. We assumed that there are three types of models involved in one IoT system, i.e., *small*, *middle*, and *large*, which are of different sizes. For example, in CIFAR10, the small model has a size of 2.33M, the middle model has a size of 4.17M, and the large model has a size of 8.71M. Note that these three models are heterogeneous with different architectures (i.e., ResNet56_CIFAR10, ResNet101_CIFAR10 [32], and MobileNetV2 [33]). In each experiment, we used a lightweight model to act as the modellet for PervasiveFL. For instance, in the experiment on ImageNet-10 the large model has a size of 162M while the modellet only has a size of 8.71M. Typically, IoT devices equipped with large models have large memories to capture more data. To mimic this scenario, in the experiments on image datasets, we assumed that devices containing large, middle, and small models have 150 00, 100 00, and 5000 images for PervasiveFL training, respectively, and there is no overlap between images on different devices. For Shakespeare, we assumed that there are 1129 devices, where each device hosts the same number of text samples as specified by the dataset. For the image datasets, we set learning rates of all the involved local models and modellets to 0.1, which will be reduced by 90% in every 30 and 80 epochs for IID and non-IID scenarios, respectively. For Shakespeare, we set learning rates of all the involved models to 1.4, which will be reduced by 90% in every 80 epochs. Note that to show the effectiveness of PervasiveFL, we considered one state-of-the-art FL method (i.e., MOON [18]), whose framework is built on top of the modellets shown in Table I for different datasets.

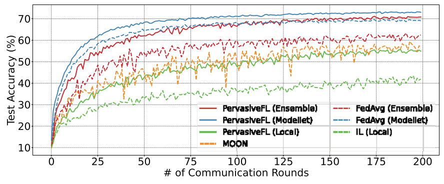
B. Performance Evaluation

1) Experimental Results of CIFAR10: We conducted both IID and non-IID experiments on CIFAR10 dataset. For the non-IID scenario, we assumed that each device captures 80% images randomly from one single category and the other 20% images randomly from all the other categories. To verify the scalability of PervasiveFL, we considered three heterogeneous IoT systems with 10, 30, and 50 devices, respectively. For each of these three systems, we set the percentage of devices with small, middle, and large models to 40%, 30%, and 30%, respectively. Note that we tested each local model using a test dataset with 100 00 images. Due to the limited space, this section only presents the detailed results for the system with ten devices. Please refer to Fig. 10 for the results of the other two systems.

Fig. 3 presents the average inference results of the ten devices for both IID and non-IID scenarios. Here, *PervasiveFL(Ensemble)* indicates our PervasiveFL approach, whose inference results come from the ensemble of both modellets and local models. *PervasiveFL(Modellet)* denotes the case where the inference results come from the modellets of PervasiveFL, while *PervasiveFL(Local)* specifies that the inference results come from the local models of PervasiveFL. Although *FedAvg(Ensemble)* also makes inferences based on the ensemble model, its training process is totally different from *PervasiveFL(Ensemble)*, where the local models and modellets are trained independently without using DML. Similarly, *FedAvg(modellet)* conducts FL based on modellets and uses modellets for inference. *IL(Local)* indicates the nonFL case without modellets, where only local models are used for inference. Note that in the subfigure, we use different colors to indicate the pair of inference results using our



(a)



(b)

Fig. 3. Inference performance comparison for CIFAR10. (a) PervasiveFL versus FedAvg (IID). (b) PervasiveFL versus FedAvg (Non-IID).

PervasiveFL method and its counterpart with different inference models (i.e., *Ensemble*, *Modellet*, and *Local*). To show the effectiveness of our approach, we also present the results of MOON [18] with the same settings.

From Fig. 3, we can find that *PervasiveFL(Ensemble)* can achieve the best results in the IID scenario. Compared with *IL(Local)*, *PervasiveFL(Ensemble)* achieves more than 10% improvement. Furthermore, we can find that our PervasiveFL methods always outperform their counterparts. For instance, *PervasiveFL(Ensemble)* outperforms *FedAvg(Ensemble)* by 3%.[1] In other words, the mutual learning between modellets and local models can enhance the inference capabilities of both kinds of models. The comparison between *PervasiveFL(Local)* and *IL(Local)* proves that local models can benefit from modellets to learn the knowledge shared among devices. Note that *PervasiveFL(Ensemble)* outperforms MOON by 5.1% in the IID scenario. The reason why *PervasiveFL(Ensemble)* beats *PervasiveFL(Modellet)* here is mainly because in the IID scenario the training data are uniformly distributed on devices. In this case, the accuracy difference between *PervasiveFL(Modellet)* and *IL(Local)* is small, since the local models and modellets have the same knowledge composition and distribution. Therefore, with the synergy between modellets and local models by selective DML, *PervasiveFL(Ensemble)* can achieve the best FL performance for IID scenarios.

From Fig. 3(b), we can observe that *PervasiveFL(Modellet)* achieves the best inference performance in the non-IID scenario, where it outperforms *IL(Local)* by 31%. Similar to the IID case, our PervasiveFL methods always outperform their counterparts in the non-IID case, which again shows the superiority of PervasiveFL. Note that *PervasiveFL(Modellet)* outperforms MOON by 19.8% in the non-IID scenario. Unlike IID scenarios, in non-IID scenarios *PervasiveFL(Modellet)* can achieve the best performance. This is because the distributions of the data on local devices are totally different from the distribution of overall device data. In other

[1]In this article, we denote test accuracy improvements using absolute change.

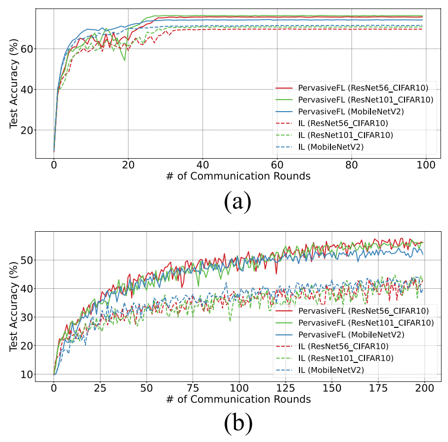


Fig. 4. Impacts on heterogeneous models for CIFAR10. (a) PervasiveFL versus IL (IID). (b) PervasiveFL versus IL (Non-IID).

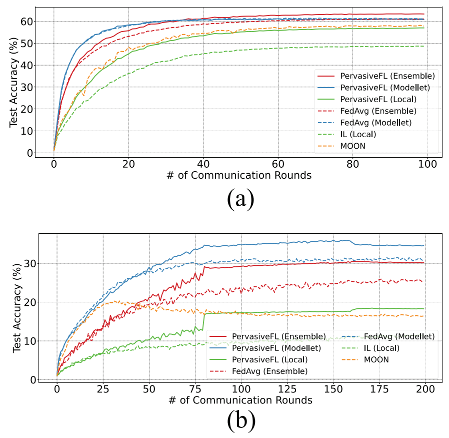


Fig. 5. Comparison results for CIFAR100. (a) PervasiveFL versus FedAvg (IID). (b) PervasiveFL versus FedAvg (Non-IID).

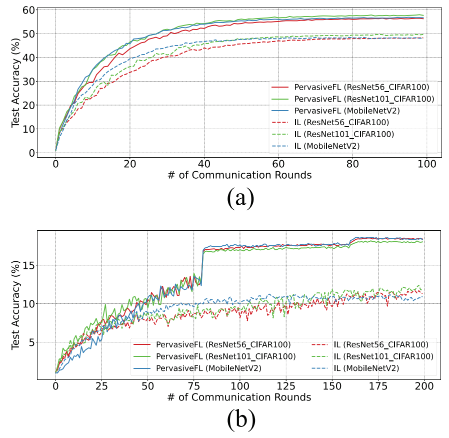


Fig. 6. Impacts on heterogeneous models for CIFAR100. (a) PervasiveFL versus IL (IID). (b) PervasiveFL versus IL (Non-IID).

words, the knowledge composition and distribution of a modellet are quite different from the ones of local models. However, since the test set has the same data distribution as the training set and the modellet learns from the overall training data, *PervasiveFL(Modellet)* can achieve the best performance. Although the modellet can share some new knowledge to local models, *PervasiveFL(Ensemble)* cannot beat *PervasiveFL(Modellet)* due to its biased knowledge from local models. Note that even in this case, the modellet can still help to improve the generalization of local models.

In order to investigate the impact of PervasiveFL on different kinds of heterogeneous IoT devices within a system, Fig. 4 presents the average inference accuracy for different types of devices in IID and non-IID scenarios, respectively. We use the notations *PervasiveFL(X)* and *IL(X)* to indicate the averaged inference results of all the local models of type *x* in *PervasiveFL(Local)* and *IL(Local)*, respectively. For example, we use *PervasiveFL(ResNet56_CIFAR10)* to denote the average inference results of the four devices equipped with small local models. Note that for fair comparison, all the inference results in Fig. 4 are produced by the local models of devices. From this figure, we can find that PervasiveFL outperforms IL for all kinds of local models, since PervasiveFL enables the sharing of knowledge among devices via modellets. Meanwhile, the DML between modellets and local models can practically enhance their inference capabilities.

2) Experimental Results of CIFAR100: With the same experimental settings as CIFAR10 dataset, we investigated the inference performance of PervasiveFL on CIFAR100 dataset.

Based on the inference comparison results shown in Fig. 5, we can find that PervasiveFL methods always outperform their competitors. Although in the non-IID scenario *PervasiveFL(Modellet)* and *FedAvg(Modellet)* have the similar inference performance, it does not mean that DML has no effect, since *PervasiveFL(Local)* outperforms *IL(Local)* by 8.3%. Note that in the non-IID scenario, *PervasiveFL(Modellet)* outperforms *FedAvg(Modellet)* by 3.9%, and *PervasiveFL(Local)* outperforms *IL(Local)* by 6.9%. It means that by DML the inference capabilities of both modellets and local models are enhanced. Note that *PervasiveFL(Modellet)* outperforms MOON, especially for the non-IID scenario.

Fig. 6 presents the impacts of PervasiveFL on local models. From this figure, we can clearly observe the benefits of PervasiveFL to various kinds of local models in both IID and non-IID scenarios.

3) Experimental Results of ImageNet-10: Based on the same experimental settings as previous two experiments, this experiment adopted modellets based on MobileNetV2. From Fig. 7, we can observe that the test accuracy trend here is similar to the ones of CIFAR10 and CIFAR100 datasets, which again prove the superiority of our proposed PervasiveFL framework. From Fig. 7(b), we can find that *PervasiveFL(Modellet)* outperforms *FedAvg(Modellet)* by 3.2%. Since the overall test accuracy of *PervasiveFL(Modellet)* is only 41.4%, such improvement is significant.

Fig. 8 shows the impacts of PervasiveFL on the inference improvements of local models by DML. For example, *PervasiveFL(ResNet101)* outperforms *IL(ResNet101)* by 3.2% in IID scenario, while *PervasiveFL(ResNet18)* outperforms *IL(ResNet18)* by 5.5% in non-IID scenario.

4) Experimental Results of Shakespeare: Since the text dataset Shakespeare is naturally non-IID, this experiment only

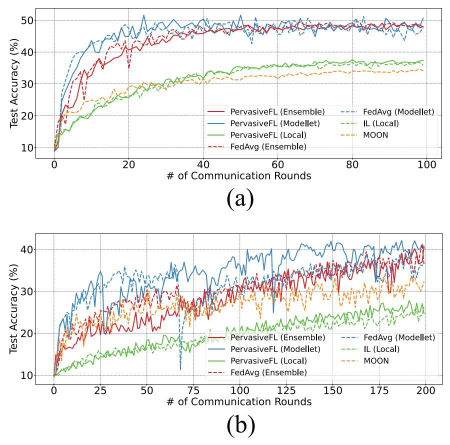


Fig. 7. Comparison results for ImageNet-10. (a) PervasiveFL versus FedAvg (IID). (b) PervasiveFL versus FedAvg (Non-IID).

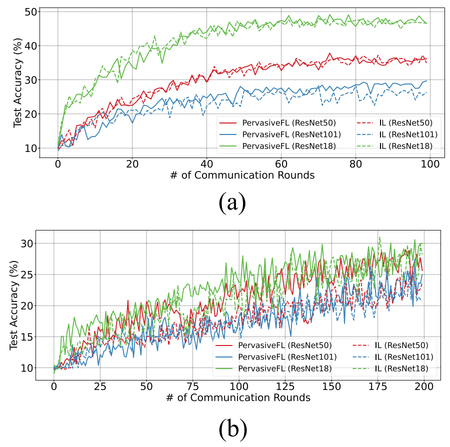


Fig. 8. Impacts on heterogeneous models for ImageNet-10. (a) PervasiveFL versus IL (IID). (b) PervasiveFL versus IL (Non-IID).

considers the non-IID scenario with 1129 devices. Fig. 9 shows both the performance comparison results and the impacts of PervasiveFL on local models. From Fig. 9(a), we can find that our *PervasiveFL* methods outperform all of their counterparts, and can achieve the similar performance as the one of MOON. In this case, *PervasiveFL(Local)* improves *IL(Local)* by 6.1%, indicating that the modellets can greatly benefit local models by DML in practice. In Fig. 9(b), we can find that the classification performance of all the three types of local models are improved significantly.

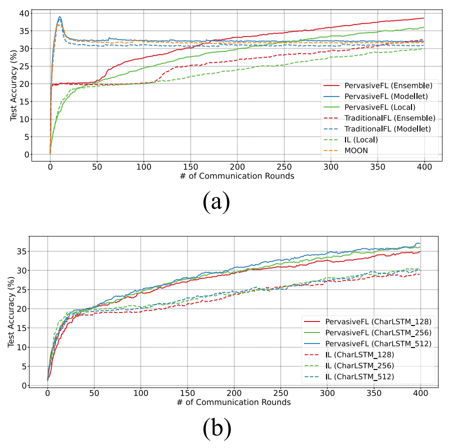


Fig. 9. Effectiveness of PervasiveFL on Shakespeare. (a) PervasiveFL versus FedAvg (non-IID). (b) PervasiveFL versus IL.

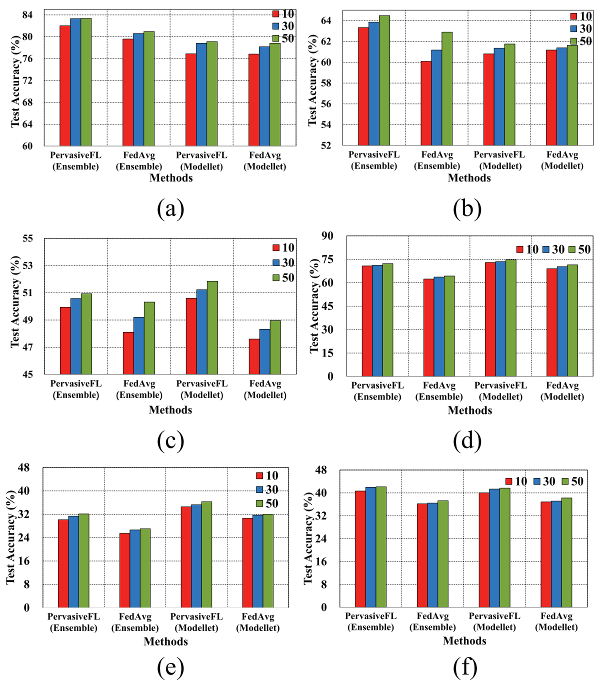
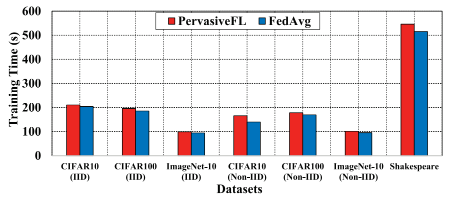


Fig. 10. Impact of the number of IoT devices on PervasiveFL. (a) CIFAR10 (IID). (b) CIFAR100 (IID). (c) ImageNet-10 (IID). (d) CIFAR10 (Non-IID). (e) CIFAR100 (Non-IID). (f) ImageNet-10 (Non-IID).

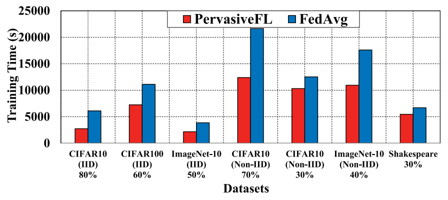
C. Discussions

1) Scalability Analysis: As more and more heterogeneous devices are integrated into complex IoT systems, the scalability plays an important role in the deployment of PervasiveFL. By using the same experimental settings used in the previous section, Fig. 10 compares the inference accuracy of IoT systems with different scales in both IID and non-IID scenarios. Note that since the Shakespeare dataset fixes the number of users (i.e., 1129 users), we did not investigate its scalability here. For each image dataset, we constructed three IoT systems with 10, 30, and 50 devices based on Table I, respectively. We assumed that for the three systems PervasiveFL converges at the 100th and 200th round for IID scenarios and non-IID scenarios, respectively. To avoid jitters, for each case we used the average of the last ten test accuracy values for the inference performance comparison. From Fig. 10, we can find that when more devices are involved in PervasiveFL, the overall inference accuracy will increase in both IID and non-IID scenarios. Our PervasiveFL methods always outperform their counterparts for all the three image datasets.

2) Computation Overhead: To evaluate the computation overhead of components (i.e., modellets and DML) introduced by PervasiveFL in both IID and non-IID scenarios, we conducted various experiments to investigate their impacts on the overall training time. Fig. 11 shows the evaluation results from



(a)



(b)

Fig. 11. Analysis of PervasiveFL computation overhead. (a) Average training time per round. (b) Training time to achieve same accuracy.

two perceptiveness: 1) average training time (on workstation) per FL training round and 2) training time to achieve a specific test accuracy.

From Fig. 11(a), we can find that one PervasiveFL round needs slightly more training time than one FedAvg round. However, as shown in Fig. 11(b), PervasiveFL needs much less training time to achieve a specific test accuracy. For example, to achieve a test accuracy of 80% for CIFAR10, PervasiveFL only needs only 13 rounds, while FedAvg requires 30 rounds. This is because our DML-based method enables selective learning that can share high-quality knowledge between modellets and local models, thus accelerating the overall convergence process. Note that before the training it is hard to figure out the number of epochs for an existing FL method to achieve convergence. However, based on the observations in Section V-B and Fig. 11, PervasiveFL can quickly converge to a higher test accuracy. Since the current version of PervasiveFL is built on top of FedAvg, any existing FL convergence optimization methods can be easily applied on PervasiveFL to further improve the convergence.

3) Communication Overhead: Since PervasiveFL shares knowledge among devices via small-scale modellets, the communication overhead of PervasiveFL is much smaller than traditional FL approaches. After finishing the local training of a batch of images, our approach only needs to send the gradients of modellets for aggregation. The size of the proposed method is much smaller than that of local models (see Table I). As an example for ImageNet-10, the size of modellet gradients is 8.71M, which needs 1.65 s on average for one round of communication including both modellet uploading and dispatching operations. However, the gradient size of the large model is 162M, which requires 30.96 s for one round of communication. Evidently, the overall PervasiveFL communication overhead is significantly smaller than the ones of traditional FL methods.

VI. Conclusion

This article proposed a novel framework named PervasiveFL that enables effective and scalable FL on various heterogeneous devices with different kinds of models. By installing a lightweight DNN model on each device, PervasiveFL allows mutual selective learning between the modellet and local models on each device by using DML and our proposed EDG method. Meanwhile, since all the modellets in PervasiveFL are of the same structure, they can be used to conduct the FL-style knowledge sharing among devices. In this way, PervasiveFL enables "pervasive" FL on a large set of heterogeneous IoT devices with different types of local models. Comprehensive experiments on well-known datasets demonstrate the effectiveness of PervasiveFL from the perspectives of inference performance and scalability.

References

[1] F. Samie, L. Bauer, and J. Henkel, "IoT technologies for embedded computing: A survey," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES + ISSS)*, 2016, pp. 1–10.
[2] K. Bhardwaj, W. Chen, and R. Marculescu, "New directions in distributed deep learning: Bringing the network at forefront of IoT design," in *Proc. Des. Autom. Conf. (DAC)*, 2020, pp. 1–6.
[3] O. Bringmann *et al.*, "Automated HW/SW co-design for edge AI: State, challenges and steps ahead: Special session paper," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES + ISSS)*, 2021, pp. 11–20.
[4] X. Zhang, M. Hu, J. Xia, T. Wei, M. Chen, and S. Hu, "Efficient federated learning for cloud-based AIoT applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2211–2223, Nov. 2021.
[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2017, pp. 1273–1282.
[6] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," 2019, *arXiv:1912.00818*.
[7] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 2351–2363.
[8] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.
[9] Q. Zhang, B. Gu, C. Deng, and H. Huang, "Secure bilevel asynchronous vertical federated learning with backward updating," in *Proc. AAAI Conf. Artif. (AAAI)*, 2021, pp. 10896–10904.
[10] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4320–4328.
[11] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. Int. Conf. Commun. (ICC)*, 2020, pp. 1–6.
[12] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedAvg on non-IID data," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–26.
[13] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. Int. Conf. Comput. Commun. (INFOCOM)*, 2020, pp. 1698–1707.
[14] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 6402–6413.
[15] X. Lan, X. Zhu, and S. Gong, "Knowledge distillation by on-the-fly native ensemble," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 7528–7538.
[16] M. P. Uddin, Y. Xiang, X. Lu, J. Yearwood, and L. Gao, "Mutual information driven federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1526–1538, Jul. 2021.
[17] Z. Zhu, J. Hong, and J. Zhou, "Data-free knowledge distillation for heterogeneous federated learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 12878–12889.
[18] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 10713–10722.
[19] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Robustly executing DNNs in IoT systems using coded distributed computing," in *Proc. Des. Autom. Conf. (DAC)*, 2019, pp. 1–2.

[20] L. Nagalapatti and R. Narayanam, “Game of gradients: Mitigating irrelevant clients in federated learning,” in *Proc. AAAI*, 2021, pp. 9046–9054.

[21] T. Yang *et al.*, “Applied federated learning: Improving Google keyboard query suggestions,” 2018, *arXiv:1812.02903*.

[22] X. Cheng, Z. Rao, Y. Chen, and Q. Zhang, “Explaining knowledge distillation by quantifying the knowledge,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 12922–12932.

[23] Y. Bai *et al.*, “Gradient compression supercharged high-performance data parallel DNN training,” in *Proc. ACM Symp. Oper. Syst. Principles (SOSP)*, 2021, pp. 359–375.

[24] W. Zheng, L. Yan, C. Gou, and F.-Y. Wang, “Federated meta-learning for fraudulent credit card detection,” in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2020, pp. 4654–4660.

[25] M. Park, J. Foulds, K. Chaudhuri, and M. Welling, “Variational bayes in private settings (VIPS),” *J. Artif. Intell. Res.*, vol. 68, pp. 109–157, May 2020.

[26] S. Teerapittayanon, B. McDanel, and H. T. Kung, “ BranchyNet: Fast inference via early exiting from deep neural networks,” in *Proc. Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464–2469.

[27] H. Kim and C. Tepedelenlioglu, “Performance bounds on average error rates using the AM-GM inequality and their applications in relay networks,” *IEEE Trans. Wireless Commun.*, vol. 11, no. 8, pp. 2986–2995, Aug. 2012.

[28] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

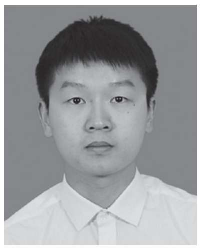
[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2012, pp. 1106–1114.

[30] S. Caldas *et al.*, “LEAF: A benchmark for federated settings,” 2019, *arXiv:1812.01097*.

[31] I. J. Goodfellow *et al.*, “Generative adversarial nets,” in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2014, pp. 2672–2680.

[32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE CVPR*, 2016, pp. 770–778.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510–4520.



Jun Xia (Graduate Student Member, IEEE) received the B.S. degree from the Department of Computer Science and Technology, Hainan University, Haikou, Hainan, China, in 2016, and the M.E. degree from the Department of Computer Science and Technology, Jiangnan University, Wuxi, China, in 2019. He is currently pursuing the Ph.D. degree with the Software Engineering Institute, East China Normal University, Shanghai, China.

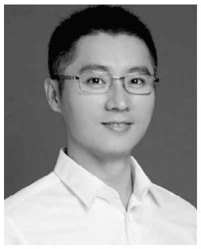
His research interests are in the areas of AIoT applications and trustworthy computing.

Tian Liu (Graduate Student Member, IEEE) received the B.S. and M.E. degrees from the Department of Computer Science and Technology, Hohai University, Nanjing, China, in 2011 and 2014, respectively, and the Engineer degree from the Department of Information and Statistic, Polytech’Lille, Villeneuve-d’Ascq, France, in 2012. He is currently pursuing the Ph.D. degree with the Software Engineering Institute, East China Normal University, Shanghai, China.

His research interests are in the area of federated learning and cloud computing.

Zhiwei Ling received the B.S. degree from the Department of Education Information Technology, East China Normal University, Shanghai, China, in 2021, where he is currently pursuing the master’s degree with Software Engineering Institute.

His research interests are in the area of federated learning, machine learning, and Internet of Things.



Ting Wang (Senior Member, IEEE)received the Ph.D. degree in computer science and engineering from the Hong Kong University of Science and Technology, Hong Kong, China, in 2015.

He is currently an Associate Professor with the Software Engineering Institute, East China Normal University (ECNU), Shanghai, China. Prior to joining ECNU in 2020, he worked with the Bell Labs, Shanghai, as a Research Scientist from 2015 to 2016, and with Huawei, Shenzhen, China, as a Senior Engineer from 2016 to 2020. His research interests include cloud/edge computing, federated learning, data center networks, machine learning, and AI-aided intelligent networking.

Xin Fu (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the University of Florida, Gainesville, FL, USA, in 2009.

She was an NSF Computing Innovation Fellow with the Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL, USA, from 2009 to 2010. From 2010 to 2014, she was an Assistant Professor with the Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS, USA. She is currently an Associate Professor with the Electrical and Computer Engineering Department, University of Houston, Houston, TX, USA. Her research interests include high-performance computing, machine learning, energy-efficient computing, and mobile computing.

Mingsong Chen (Senior Member, IEEE) received the B.S. and M.E. degrees from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, FL, USA, in 2010.

He is currently a Professor with the Software Engineering Institute, East China Normal University, Shanghai, China, and serves as the Director of the Engineering Research Center of Software/Hardware Co-Design Technology and Application affiliated to the Ministry of Education, China, and the Vice Director of Technical Committee of Embedded Systems of China Computer Federation. His research interests are in the area of real-time computing, design automation of cyber–physical systems, EDA, parallel and distributed systems, and formal verification techniques.