Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 9285802, 14 pages
https://doi.org/10.1155/2021/9285802

Research Article

Adaptive In-Network Collaborative Caching for Enhanced Ensemble Deep Learning at Edge

Yana Qin,[1,2] **Danye Wu**,[3] **Zhiwei Xu**,[1,2] **Jie Tian**,[4] **and Yujun Zhang**[2]

[1]*College of Data Science and Application, Inner Mongolia University of Technology, Hohhot 100080, China*
[2]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
[3]*Samsung R&D Institute of China, Beijing, China*
[4]*Department of Computer Science, New Jersey Institute of Technology, 323 Dr Martin Luther King Jr Blvd, Newark, NJ 07102, USA*

Correspondence should be addressed to Zhiwei Xu; xuzhiwei2001@ict.ac.cn

Received 5 April 2021; Revised 17 August 2021; Accepted 4 September 2021; Published 26 September 2021

Academic Editor: Wenyu Zhang

To enhance the quality and speed of data processing and protect the privacy and security of the data, edge computing has been extensively applied to support data-intensive intelligent processing services at edge. Among these data-intensive services, ensemble learning-based services can, in natural, leverage the distributed computation and storage resources at edge devices to achieve efficient data collection, processing, and analysis. Collaborative caching has been applied in edge computing to support services close to the data source, in order to take the limited resources at edge devices to support high-performance ensemble learning solutions. To achieve this goal, we propose an adaptive in-network collaborative caching scheme for ensemble learning at edge. First, an efficient data representation structure is proposed to record cached data among different nodes. In addition, we design a collaboration scheme to facilitate edge nodes to cache valuable data for local ensemble learning, by scheduling local caching according to a summarization of data representations from different edge nodes. Our extensive simulations demonstrate the high performance of the proposed collaborative caching scheme, which significantly reduces the learning latency and the transmission overhead.

1. Introduction

With the breakthrough of artificial intelligence (AI), we are witnessing a booming increase in AI-based applications and services [1]. The existing intelligent applications are computation intensive. To process a huge number of data in time at the edge of the network, edge computing has rapidly developed in recently. Edge computing [2] takes a part of the resources and memory from the data center and puts it at the edge of the network to be closer to end users, which can reduce the network transmission delay, protect user's privacy, and improve the network experience of end users.

The rapid uptake of edge computing applications and services pose considerable challenges on networking resources [3]. Fulfilling these challenges is difficult due to the conventional networking infrastructure. The extensive application of deep neural network models at edge makes this problem more serious. Neural network models learn relationships among a huge number of training data [4]. Meanwhile, this type of complex nonlinear models is sensitive to initial conditions, both in terms of the initial random weights and in terms of the statistical noise in the training data [5]. This nature of the learning algorithm means that different neural network models are trained; it may learn a new group of features from inputs to outputs, which have different performances in practice.

To meet the stringent demands of emerging edge intelligence applications such as smart home, smart city, and virtual reality in 5 G/6G networks, distributed learning has been applied in practice. Its learning performance, however, depends on the level of synchronizing the parameter updates from neighbors, and significant asynchronous updates from

various nodes could severely compromise the learning quality and the convergence speed [6]. Ensemble learning [7, 8] provides a feasible way to handle the variance of neural network learning at different nodes. Ensemble learning schemes train multiple models and combine their outputs to alleviate the variance of model learning processes [9]. Nguyen et al. proposed an edge learning architecture with long short-term memory (LSTM) and ensemble learning [10]. Wang and Nakachi developed a secure face recognition framework to orchestrate sparse coding in edge networks from the ensemble learning perspective [11]. We can achieve a high-quality ensemble neural network model by combining different neural network models to outperform other models.

Although ensemble learning can enhance the capability of edge deep learning, there is no performance improvement if the similar individual submodels are combined [12]. Tumer and Ghosh [13] analyzed simple soft vote ensemble methods by decision boundary analysis, revealing the importance of difference among submodels. The same conclusion is also applicable to other ensemble methods. However, it is not easy to generate individual submodels with high diversity. A significant challenge comes up that submodels are obtained on similar training data, so they are often highly correlated [14].

To make all submodels different from each other, we target on an adaptive collaborative caching scheme to guarantee submodels learning different data and consequently being diverse. With comprehensive study of collaborative caching at edge, we propose a compact recording structure of the cached data, maximizing the difference among the data cached in different nodes. Besides the caching locality considered in the conventional caching schemes [15], it is also important to consider the diversity of the data cached to train submodels. In this way, we improve the performance of the ensemble learning.

1.1. Contributions. Our main contributions are summarized as follows:

(1) To efficiently record cached data among collaborative nodes, we study data recording and exchanging among edge nodes and introduce a compact representation structure, Combinable Counting Bloom Filter (CCBF)

(2) We design an adaptive collaborative caching scheme based on CCBF, based on which a high-performance ensemble deep learning at edge is proposed

(3) Comprehensive evaluation of the proposed scheme is performed on NS-3-based simulations over real-world deep learning models and data

1.2. Organization. In Section 2, we study the related work. A compact data structure to collect the information of the cached data is present in Section 3. Based on this data structure, we propose an adaptive collaborative caching scheme for ensemble learning in Section 4. In detail, we exchange the information of the cached data, elevate these data to learn local knowledge, and achieve a high-performance ensemble result. The performance of our design is evaluated in Section 5. Finally, we conclude the work in Section 6.

2. Related Work

In recent years, ensemble learning at the edge has been used in all kinds of applications [14, 16]. Meanwhile, due to the storage limitation of each edge node, edge nodes always collaborate in data collection and model training [17–19].

2.1. Ensemble Learning. We investigate relevant works in recent years, and we find that the performance can be effectively improved with the introduction of ensemble mechanism. To optimize the determination process of deep classification model structure and the combination of multimodal feature abstractions, Yin et al. [20] proposed multiple-fusion layer-based ensemble classifier of stacked autoencoder (MESAE) for recognizing emotions, in which deep learning is used for guiding autoencoder ensemble. Moreover, based on the assumption that different convolutional neural network (CNN) architectures learn different levels of semantic representations, Kumar et al. [21] developed a new feature extractor by ensembling CNNs that were initialized on a large dataset of natural images. Experiment showed that the ensemble of CNNs can extract features with a higher quality, compared with traditional CNNs. Xiao [22] proposed an ensemble learning method to improve the robustness in traffic incident detection. Galicia et al. [23] presented ensemble models for forecasting big data time series. Liu et al. [24] applied ensemble convolutional neural network models with different architectures for visual traffic surveillance systems. Liu et al. [25] designed an ensemble transfer learning framework which used AdaBoost to adjust the weights of the source data and target data, this method achieved good performance on UCI datasets when the training data are insufficient. Chen et al. [26] proposed an ensemble network architecture for deep reinforcement learning, in order to solve the problem of existing ensemble algorithms in reinforcement learning.

2.2. Collaborative Caching. Collaborative caching has been applied in the ensemble learning field to collect sufficient data for submodels' training and high-quality ensemble result achievement. Amer et al. [27] stated the role of wireless caching in low-latency wireless networks and characterized the network average delay on a per request basis from the global network perspective. Li et al. [28] proposed a cache-aware task-scheduling method in edge computing. First, an integrated utility function is derived with respect to the data chunk transmission cost, caching value, and cache replacement penalty. Data chunks are cached at optimal edge servers to maximize the integrated utility value. After placing the caches, a cache locality-based task scheduling method is presented. Chien et al. [29] proposed a collaborative cache mechanism in multiple Remote Radio Heads (RRHs) to multiple Baseband Units (BBUs). In addition, they use Q-learning to design the cache

mechanism and propose an action selection strategy for the cache problem. Through reinforcement learning to find the appropriate cache state, Ndikumana et al. [30] proposed collaborative cache allocation and computation offloading, where the MEC servers collaborate for executing computation tasks and data caching. Tang et al. [31] proposed caching mechanisms in collaborative edge-cloud computing architecture, which can implement the caching paradigm in cloud for frequent n-hop neighbor activity regions. Khan et al. [32] proposed reversing the way in which node connectivity is used for the placement of content in caching networks and introduced a Low-Centrality High-Popularity (LoCHiP) caching algorithm that populates poorly connected nodes with popular content. Wei et al. [33] presented a system that automatically parallelizes serial imperative ML programs on distributed caching. The system makes a static dependence analysis to determine when dependence-preserving parallelization is effective and maps a computational process to a distributed schedule.

2.3. Problems and Our Insight. Although existing caching approaches can facilitate ensemble learning, enhanced collaborative caching should be studied. Actually, the ensembling learning process is highly related to different submodels. As analyzed by Tumer and Ghosh [13], if the submodels are independent of each other, the error of ensemble learning will be reduced. If each submodel is correlated with all others, the error of ensemble learning becomes larger. This analysis clearly reveals the importance of different submodels [12].

To differentiate all submodels as much as possible, we take different data to train various submodels and improve the performance of ensemble learning: (1) an efficient way to record the cached data items is highly required, (2) exchange the record of the cached data among different edge nodes, and (3) schedule the edge caching according to the records and train different submodels for high-quality ensemble learning.

3. Composable Counting Bloom Filter

To make the valuable data remain on edge nodes, we need to exchange the compact records of the cached data. However, the most popular compact recording method, Counting Bloom Filter (CBF) [34], only supports inserting, deleting, and querying on data and cannot support the combination operation of multiple filters which will be used to summarize the exchanged compact records of the cached data. Therefore, to meet the aforementioned requirement, we introduce a Composable Counting Bloom Filter (CCBF) in this section.

3.1. Design Structure. To support the dynamic update of the record of the cached data, as well as the combination of multiple compact records of the cached data, we design a new structure for the proposed CCBF on the basis of the basic Bloom Filter. We can combine multiple basic Bloom filters by performing bitwise OR on these filters, but cannot combine CBFs in the similar way since CBF aggregates the information of the inserted data into its counters. Based on the above observation, we can stack several basic Bloom Filters to build a Counting Bloom Filter which can support updating and merging operations simultaneously.

The detailed structure of CCBF is shown in Figure 1. The CCBF has k hash functions $(h_1, h_2, \ldots, h_k)$ and consists of the following two components:

(1) *G bit arrays* (barr_i, $i = 1, 2, \ldots, g$): the bit arrays are used to replace the counterarrays in CBF to support counting operations of the inserted data items, the size of which is equal to m. g is set based on the requirement of counting.

(2) *orBarr*: the aggregation result of g bit arrays, by performing bitwise OR, is used to enhance the query efficiency and facilitate the data caching among edge nodes.

CCBF not only supports the insert, query, and delete of items but also supports the combination of multiple CCBFs. These operations will be described one by one in Section 3.2.

3.2. Related Operations. According to the needs of exchanging and updating cached data records, we have implemented the functions of inserting, querying, deleting, and combination in CCBF. In this part, we introduce the related operations.

3.2.1. Inserting. To insert a data item into CCBF, we use the pseudorandom integer generator to generate a random matrix to perform an efficient and nonrepeated insert operation. The specific inserting operation is shown in Algorithm 1, and the procedures are shown as follows:

(1) A random matrix ($\text{matrix}[g][m]$) of size $g \times m$ is constructed by using a pseudorandom integer generator with different seeds on different columns. For each column, the value of each cell is different from the others and belongs to a range from 1 to g.

(2) Hash the cached data k times to get k hash results ($\{p_j\}$) (line 3).

(3) Use the RandChoice function to search the p_jth column of $\text{matrix}[g][m]$ for the next available bit array, according to the number of arrays whose p_jth cells are used (line 4).

(4) Set the p_jth cell in barr_i to be 1 and update orBarr (line 5–7).

Notice that we will check if the bit arrays used to record the k hash results meets the following condition:

$$\forall \text{barr}_i[p_j] = 1, \quad j \in 1, 2, \ldots, k. \tag{1}$$

It means that this data item has been inserted, and this insert operation will be abandoned.

3.2.2. Querying. We include an additional array orBarr to support efficient membership query operations, which can directly check whether the corresponding cells of orBarr are

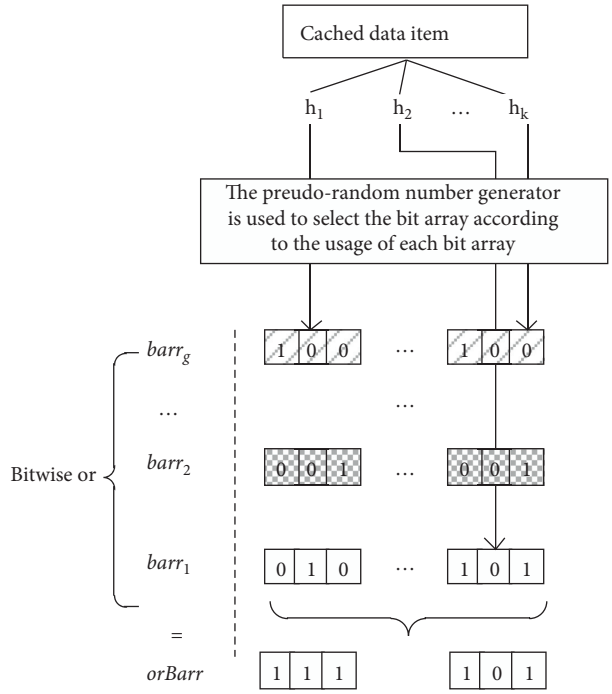

Figure 1: Structure of Composable Counting Bloom Filter.

Require: d
(1) barrSet = CCBF.GetAllBarr;
(2) **for** $j = 0$ to CCBF.$k - 1$ **do**
(3) $p_j = \text{Hash}_j(d)$;
(4) $\text{barr}_i = \text{RandChoice}(\text{barrSet}, p_j, \text{Counter}(p_j))$;
//Function RandChoice searches the p_jth column of matrix$[g][m]$
for an available bit array, according to the number of arrays whose p_jth cells are used;
(5) $\text{barr}_i[p_j] = 1$;
(6) **end for**
(7) Update orBarr;

Algorithm 1: CCBF.insert(d).

set to be 1. The specific query operation is shown in Algorithm 2:

(1) Hash the queried data k times to get j hash results (p_j) (line 2).
(2) Check if orBarr$[p_j]$ is 1. If it is 1, the data is inserted before. Conversely, there is no corresponding data (line 3–7).

3.2.3. Deleting. The calculation process of the delete operation is similar to that of the insert operation. The key operation steps are briefly listed as the follows:

(1) Confirm whether the item exists in this CCBF by performing a query operation on this item (see Algorithm 2)
(2) Locate the bit arrays used in the last inserting operation according to the random matrix, matrix$[g][m]$
(3) Clear the corresponding cells in these bit arrays, and update orBarr

3.2.4. Combination. CCBF supports not only inserting, querying, and deleting data items but also the combination of multiple CCBFs. The combination operation of multiple CCBFs is equivalent to combining the data items inserted in these CCBFs. The random matrix matrix$[g][m]$ generated in CCBF can ensure the bit array selected in a fixed sequence, and thus, the repeated data inserting can be neglected. The specific combination operation is shown in Algorithm 3, and the combination procedure is listed as the follows:

(1) Determine whether the number of items in the compacted representation after merging has exceeded the capacity of CCBF (n) (line 1–3)
(2) Combine bit arrays one by one by bitwise OR (line 6–13)

```
Require: d;
Ensure: Query result (true or false);
(1)  for j = 0 to CCBF.k − 1 do
(2)    p_j = Hash_j(d)
(3)    if CCBF.orBarr[p_j] ≠ 1 then
(4)      return false
(5)    end if
(6)  end for
(7)  return true
(8)  End
```

Algorithm 2: CCBF.query(d).

```
Require: Two CCBFs, CCBF, and another;
 (1) if CCBF.Size + other.Size > CCBF.n then
 (2)   return error;
 (3) end if
 (4) barrSet = CCBF.GetAllBarr;
 (5) otherbarrSet = other.GetAllBarr;
 (6) for j = 0 to CCBF.g − 1 do
 (7)   barr = barrSet.first;
 (8)   otherbarr = otherbarrSet.first;
 (9)   barr.bitwiseOr(otherbarr);
(10)   barrSet.Remove(barr);
(11)   otherbarrSet.Remove(otherbarr);
(12) end for
(13) CCBF.orBarr.bitwiseOr(other.orBarr);
```

Algorithm 3: CCBF.combine(other).

CCBF can record the cached data in each edge node through a compact way. By inserting the data items in CCBF and exchanging and combining CCBFs among edge nodes, the cache of each edge node can be scheduled to store diverse data, which facilitates and obtains different submodels on these data and achieve an high-performance ensemble learning result.

4. Ensemble Learning Based on Collaborative Caching

In this section, we propose an edge ensemble learning scheme based on adaptive collaborative caching. In this scheme, CCBF is used to record the cached data information, so as to realize the exchange and collection of cache information between edge nodes. Also, the scheme can reasonably schedule the cache according to the data distribution and support the submodels' learning and final ensemble learning of each edge node.

4.1. Learning Strategy. Ensemble learning [35] is the process by which multiple models, such as classifiers, are generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model. In order to improve the performance of ensemble learning at edge, we first study the ensemble learning process. The decision boundary analysis results of the simple soft voting ensemble method are as follows.

To remain simple, it is assumed that all submodels have the same error rate. We use θ to describe the relationship between different submodels. The expectation error $H(x)$ [36] of ensemble learning is

$$\overline{\text{err}}(H(x)) = \sum_{i=1}^{n} \frac{1+\theta(n-1)}{n} \text{err}_i(h_i(x)), \tag{2}$$

where $\text{err}_i(h_i(x))$ is the expectation error rate of a submodel, $h_i(x)$ is the ith submodel, and n is the size of the ensemble scale. Formula (2) shows that if the submodels are independent of each other, *i.e.*, $\theta = 0$, the error of ensemble learning will be reduced by n times. If each submodel is correlated with all the others, *i.e.*, $\theta = 1$, the performance of ensemble submodels will not be effectively improved. This analysis clearly reveals the importance of different submodels in ensemble learning, and the same conclusion applies to other ensemble approaches [12]. In edge ensemble learning scenarios, different edge nodes often deploy similar models, build submodels by learning the data around the edge nodes, and eventually form an ensemble model by distributing the submodels on different nodes by the central node. For this case, it is necessary to provide different data

for different edge nodes to get different training results of submodels, so as to achieve a more accurate ensemble model.

4.2. Leveraging Collaborative Caching to Facilitate Ensemble Learning Process. According to the analysis in Section 4.1, we leverage a collaborative caching scheme to support different submodels' learning of ensemble learning. In detail, as shown in Figure 2, the process can be divided into the following five phases.

4.2.1. Efficiently Recording the Cached Data on an Edge Computing Node. Data is collected from neighboring end devices, which will be cached and recorded in the compact way by CCBF (see Section 3) on edge computing nodes.

4.2.2. Exchanging Compact Representation of Cached Data with Neighbors. The compact representation (CCBF) of the cached data is exchanged among neighbors in a range, which adapts to the performance improvement of the submodel training in step 4. Once a neighbor receives a representation from an interface, this representation will be stored, with a name of CCBF_l, where l is the id of the corresponding interface. In addition, the representation is combined with an aggregated representation on this neighbor, CCBF_g, and gains a global view about the data cached in the neighbors, which will be used to guide the neighbor to cache various data received subsequently.

4.2.3. Caching Different Data among Neighbors. When the neighbor requests to cache some data, it first needs to check whether this data has already existed in the neighbors. It queries CCBF_g that represents the global view about the data cached in the neighbors. If the record of this data is found in CCBF_g, which indicates that the data has already existed in the cache of other neighbors, the data no longer need be stored in the neighbor's cache. And, if this record does not exist in CCBF_g, indicating that the caches of other neighbors do not contain this data; this data can be added to the cache and a record is added to the corresponding compaction record. The above operation ensures that different data can be cached at neighbors for training different submodels in ensemble learning, while reducing the communication overhead by collaborative caching.

4.2.4. Submodel Training. The data cached on one node is used to train the local submodel. When the local data is not enough to make the submodel converge, we need to enlarge the collaborative range by requesting differentiated data from the other edge nodes. We compare the cached data records from different neighbors obtained in step 2, CCBF_l, with the local cache record by performing merge on orBarr of different CCBF_l from different neighbors, obtain the required data compact representation $\hat{\mathrm{orBarr}}$, and send it to the corresponding edge node. When the corresponding edge node receives the request, it queries the local cache according to $\hat{\mathrm{orBarr}}$ and returns the differentiated data to the requesting node. After the requesting node receives the data, it caches the data and updates CCBF_l and CCBF_g and then inputs the data into the submodel for training. The procedures are repeated until the submodel converges. When the loss error of the submodel is less than threshold, it is considered to be convergent.

4.2.5. Enhanced Ensemble Process. The ensemble method obtains the result by attaching different weights to the output result of each submodel. The ensemble output result $H(x)$ is

$$H(x) = \sum_{i=1}^{n} \omega_i h_i(x), \tag{3}$$

where ω_i denotes the weight of $h_i(x)$, usually with parameters $\omega_i \geq 0$ and $\sum_{i=1}^{n} \omega_i = 1$. These parameter weights from the submodel are uploaded to the central node, which conducts ensemble learning in an enhanced way. Specifically, for n submodels $(h_1, \ldots, h_n)$, the following method is adopted for ensemble learning.

Suppose the output of each submodel can be written as the true value plus an error term:

$$h_i(x) = f(x) + \varepsilon_i(x), \quad i = 1, \ldots, n. \tag{4}$$

The ensemble error can be expressed as [37]

$$\begin{aligned} \widehat{\mathrm{err}}(H) &= \int \left(\sum_{i=1}^{n} \omega_i h_i(x) - f(x) \right)^2 p(x)\mathrm{d}x \\ &= \int \left(\sum_{i=1}^{n} \omega_i h_i(x) - f(x) \right) \times \left(\sum_{j=1}^{n} \omega_j h_j(x) - f(x) \right) p(x)\mathrm{d}x \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} \omega_i \omega_j C_{ij}, \end{aligned} \tag{5}$$

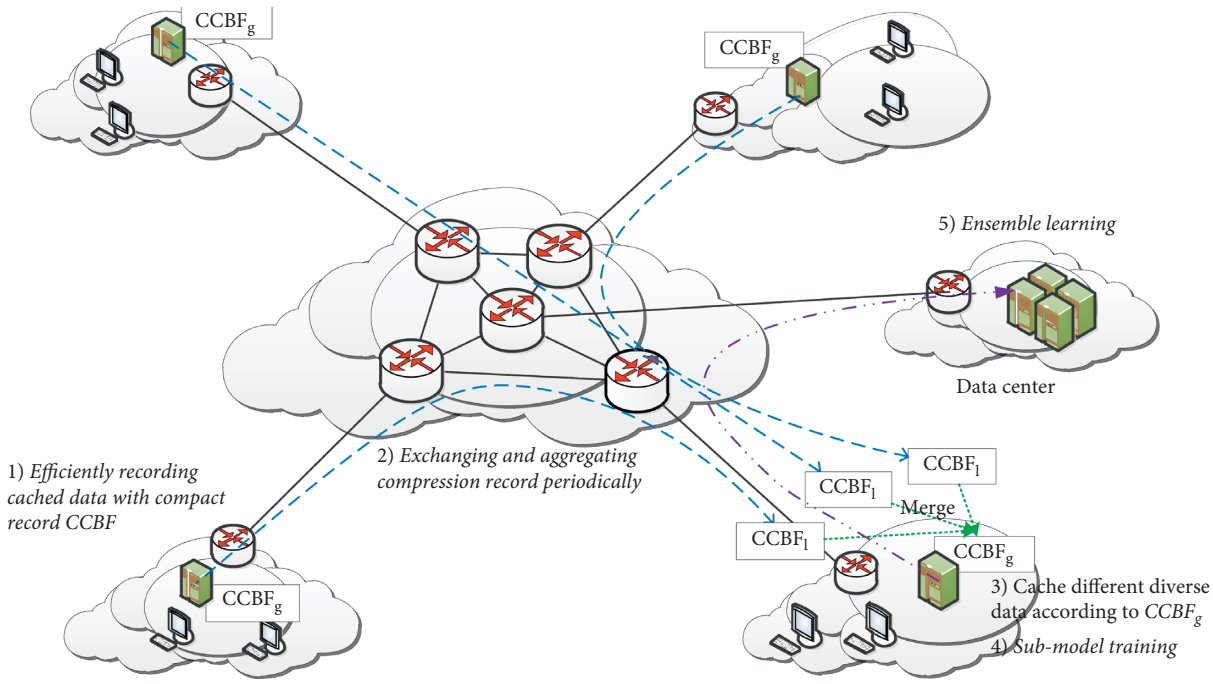

Figure 2: Adaptive in-network collaborative caching process.

where $p(x)$ is the input distribution, $\varepsilon_i(x)$ is an error term, and

$$C_{ij} = \int \left(h_i(x) - f(x)\right)\left(h_j(x) - f(x)\right)p(x)\mathrm{d}x. \tag{6}$$

The optimal weight can be solved in the following ways:

$$W = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{n} \omega_i \omega_j C_{ij}, \tag{7}$$

where W is the set of ω_i when the ensemble error is the smallest.

By means of the Lagrangian multiplier, we get ω_i is

$$\omega_i = \frac{\sum_{j=1}^{n} C_{ij}^{-1}}{\sum_{k=1}^{n}\sum_{j=1}^{n} C_{kj}^{-1}}. \tag{8}$$

According to equation (8), we can get the optimal weight in the ensemble process.

5. Performance Evaluation

In this section, we conduct experimental simulations of two learning models on four different datasets to evaluate the performance of the proposed collaborative caching scheme for ensemble learning at edge.

5.1. Implementation. We evaluate the performance of our adaptive collaborative caching scheme on Ns-3 platform [38]. It is a modular, programmable, extensible, open, open-source, and community-supported simulation framework for computer networks. We connect neural networks' library OpenNN (Open Neural Networks Library) [39] to Ns-3 for experimental simulations. OpenNN is an open-source neural network library to facilitate the building of neural networks. It has found a wide range of applications, which include function regression, pattern recognition, time series prediction, optimal control, optimal shape design, or inverse problems. All simulations are performed on a local machine, equipped with an Intel Core i7, 3.4 G CPU, and 16G RAM, running Ubuntu 16.04 with kernel version 3.19.

As shown in Figure 3, we use a general topology for edge networks. It includes a remote data center, a gateway node, 4 edge computing nodes, and 8 end devices, connected by Gigabit links. The cache size of edge computing nodes is 2,000 KB. Edge computing nodes can cache data and efficiently record cached data and perform cached data computational tasks.

End devices generate the learning data of models and send data to edge computing nodes. The distribution of learning data is random, and the data of each category is sent to edge computing nodes randomly and identically. In our simulations, TCP/IP protocol and LRU (Least Recently Used) strategy are used as the transmission protocol and the data caching strategy. After receiving the data, edge computing nodes first carry out data caching and efficient recording, then use different data of cooperative caching to train submodels, and finally send the training results of submodels to the data center for ensemble learning.

To simulate the real network environment, the data center generates other traffic data in the network, that is, background traffic data, and sends data to the edge computing nodes. Among the used background traffic data, a regular message dataset [40] is collected on Twitter, which includes five famous events: Ottawa Shooting, Charlie Hebdo, Germanwings crash, Sydney Siege, and Ferguson. The background traffic data obeys Zipf(0.8) distribution. After receiving the data, edge computing nodes carry out

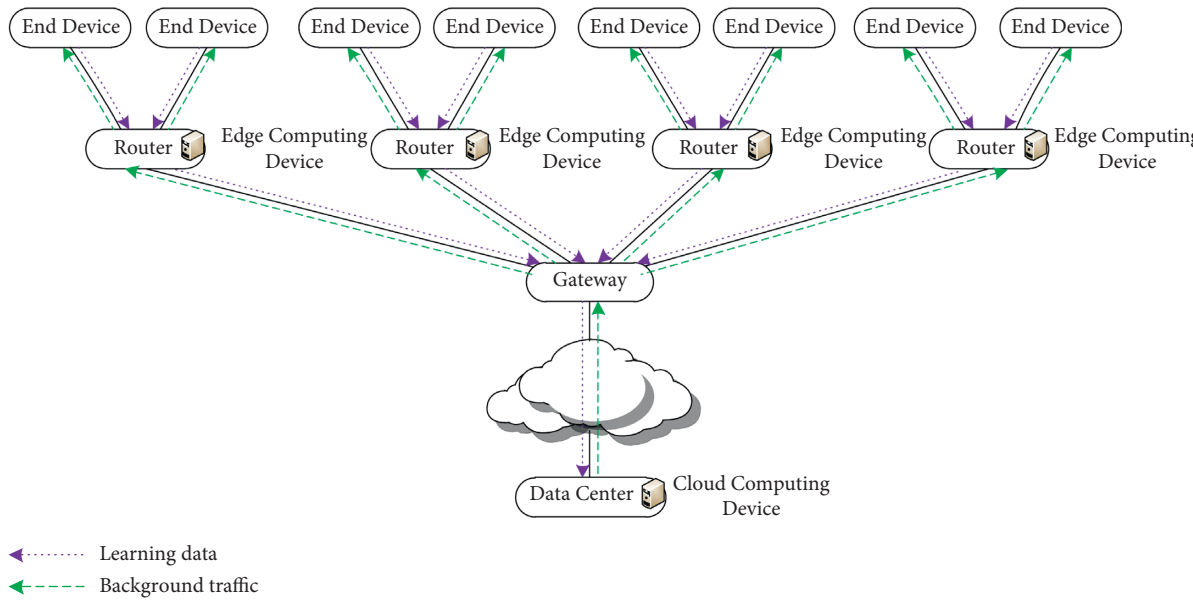

Figure 3: Simulation topology.

data caching and send background traffic data to end devices.

Two types of learning models are deployed on these edge nodes and the data center server. Correspondingly, four different datasets (D1, D2, D3, and D4) are used to compare the proposed adaptive collaborative caching scheme (C-cache) with two baseline schemes implemented as follows:

(i) **Centralized:** in the model training process, the data center requests training data from edge nodes to train ensemble learning models on the server

(ii) **P-cache:** a caching mechanism in collaborative edge-cloud computing architecture [31] is proposed, in which edge computing nodes periodically request cached data for submodels' learning, and data center server performs ensemble learning

To evaluate the performance of the collaborative caching scheme, we use four datasets to learn. Specifically, two text datasets (D1, D2) are used to train multilayer perceptron (MLP) model. To train geometry group network (VGG) model, we apply an image dataset (D3) of tiger faces and a dataset (D4) of human faces.

(i) Covertype dataset [41] (D1): this dataset includes forest vegetation types in the Roosevelt National Forest. There are four types of soil (Rawah, Neota, Comanche Peak, and Cache la Poudre), corresponding to 7 types of vegetation. The number of data items' forest vegetation is 581012, which are identically categorized into 4 different soil types.

(ii) Healthy Old People dataset [42] (D2): this includes sequential motion data from 14 healthy older people aged 66 to 86 years old using sensors for the recognition of activities in clinical environments. Participants were allocated in two clinical room settings (S1 and S2). S1 (Room1) and S2 (Room2) are set with different numbers and locations of the sensor receiver. The number of data items is 75,128, which are identically categorized into 4 different behaviors.

(iii) Atrw Reid-tigerface dataset [43] (D3): the images of Atrw Reid-tiger face are captured. After clipping, the image resolution is adjusted to 128×128. Each one of 500 tigers has 10 photos. According to the areas of activities, the Russian Far East region and the northern region of India, the dataset is separated into two scenarios.

(iv) Casia-face dataset [44] (D4): the face images of human face are captured. After clipping, the image resolution is adjusted to 128×128. Each one of 500 persons has 10 face pictures. According to the angle of the photograph taken, the position of the front and the rhombic 45 degrees, the dataset is separated into two scenarios.

We implement the following two learning models, among which Nadam algorithm [45] is used, which can adaptively adjust the learning rate of the model. The learning rate (step size) is initialized to 0.01.

(i) Multilayer perceptron (MLP) model: MLP is a feedforward artificial neural network that maps multiple input data to outputs. The layers of MLP are fully connected. We implement a six-layer MLP model, including an input layer, four hidden layers, and an output layer.

(ii) Visual geometry group network (VGG) model: VGG is a deep convolutional neural network used in computer vision. Our implementation includes 5 convolutional blocks with each consisting of 2–4 convolutional layers. For the five convolutional

blocks, each of their layers contains 64-128-256-512-512 convolution kernels, respectively.

5.2. Evaluation Metrics. To evaluate the performance of adaptive collaborative caching scheme for ensemble learning at edge, we use three metrics: hit ratio, latency, and accuracy.

5.2.1. Hit Ratio of Collaborative Caching. The concept of hit ratio is defined for any two adjacent level of memory in memory hierarchy. The performance of cache is measured in terms of hit ratio. If a data item requested by edge computing nodes is found in the cache, it is called a hit. Hit ratio is the number of hit data divided by total data items and consists of local hit ratio, global learning hit ratio, and global background hit ratio.

(1) Local learning hit ratio ($\mathrm{LLR_{hit}}$) represents how many local cached data can be used to train a submodel. For example, if 30 pieces of data items in the local cache can be used for training the model and the total amount of cached data items is 100, then the $\mathrm{LLR_{hit}}$ is $30/100 = 0.3$. $\mathrm{LLR_{hit}}$ is the ratio of the learning data in the local cache and the overall cached data:

$$\mathrm{LLR_{hit}} = \frac{N_l}{N_c}, \tag{9}$$

where N_l is the number of data items for training a submodel in the local cache and N_c is the total amount of locally cached data items.

(2) Global learning hit ratio ($\mathrm{GLR_{hit}}$) represents how many data items in edge nodes can be used to train submodels. $\mathrm{GLR_{hit}}$ is the ratio of the learning data items in the global cache and the overall cached data items, calculated as

$$\mathrm{GLR_{hit}} = \frac{N_g}{N_{gc}}, \tag{10}$$

where N_g is the number of data items for training submodels in the global cache and N_{gc} is the total amount of globally cached data items.

(3) Background hit ratio (R_{hit}) is the ratio of the background traffic data items in the global cache and the overall cached data items. Background traffic refers to the flow of data packet exchange between application program and network periodically or intermittently when there is no specific interaction. R_{hit} is calculated as

$$R_{\mathrm{hit}} = \frac{N_b}{N_{gc}}, \tag{11}$$

where N_b is the number of background traffic data items.

5.2.2. Transmission Overhead and Learning Latency.

(1) The data transmission overhead is the size of data requested to support model training among edge computing nodes

(2) Learning latency is a time period how long the training model converges

5.2.3. Learning Accuracy. The accuracy (Acc) of model training on a dataset is defined as

$$\mathrm{Acc} = \frac{\sum_{i=1}^{n} T_i}{\sum_{i=1}^{n} (T_i + F_i)}, \tag{12}$$

where i is the number of categories, n is the total number of categories in the classification, T_i is the number of items that is correctly classified, and F_i is the number of items that is incorrectly classified.

5.3. Evaluation Results

5.3.1. Hit Ratio of Collaborative Caching. Since the centralized scheme trains models in the data center without caching data in edge nodes, we only compare the cache hit ratio of the baseline, P-cache, and proposed C-cache. Local learning hit ratio is depicted in Figures 4 and 5. Global learning hit ratio is depicted in Figures 6 and 7. The local learning hit ratios of C-cache and P-cache increase to their maximum stable value of 0.87 and 0.85. The global learning hit ratios of C-cache and P-cache increase to their maximum stable value of 0.83 and 0.81, while the learning data are generated and cached in different edge nodes.

Figures 8 and 9 depict the hit ratio of background traffic data. The cache hit ratio of background traffic data first increases over time, and when the learning data increases, more background traffic data are switched out from the caches of edge computing nodes. Consequently, the cache hit ratios of background traffic data in C-cache and P-cache decrease to 0.17 and 0.19. Regarding different training models and datasets, the cache hit ratio under C-cache declines faster than that under P-cache. This is because C-cache can use learning data better than P-cache, and less available cache space is reserved for caching background traffic data.

5.3.2. Transmission Overhead and Learning Latency. To evaluate the communication and time overhead of the proposed scheme, we compare two baselines, centralized and P-cache, with our C-cache in terms of transmission overhead and learning latency. Among them, the transmission overhead is depicted in Figure 10. No matter which models or datasets are taken into consideration, C-cache always has the least transmission overhead. More powerful model such as VGG will consume more communication resource, while the transmission overhead of the

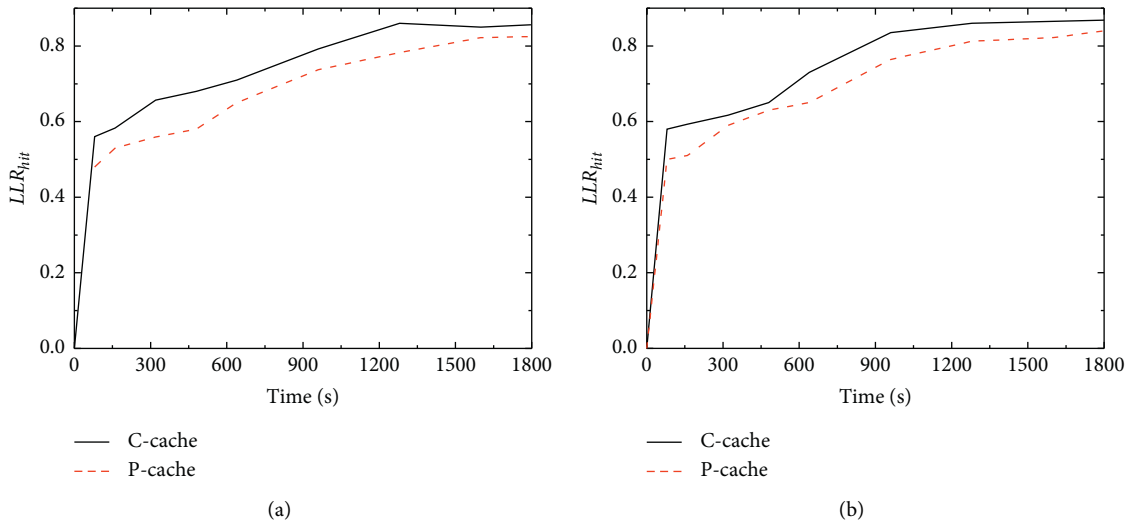

Figure 4: Local hit ratio of MLP. (a) LLR_{hit} during training MLP on D1. (b) LLR_{hit} during training MLP on D2.

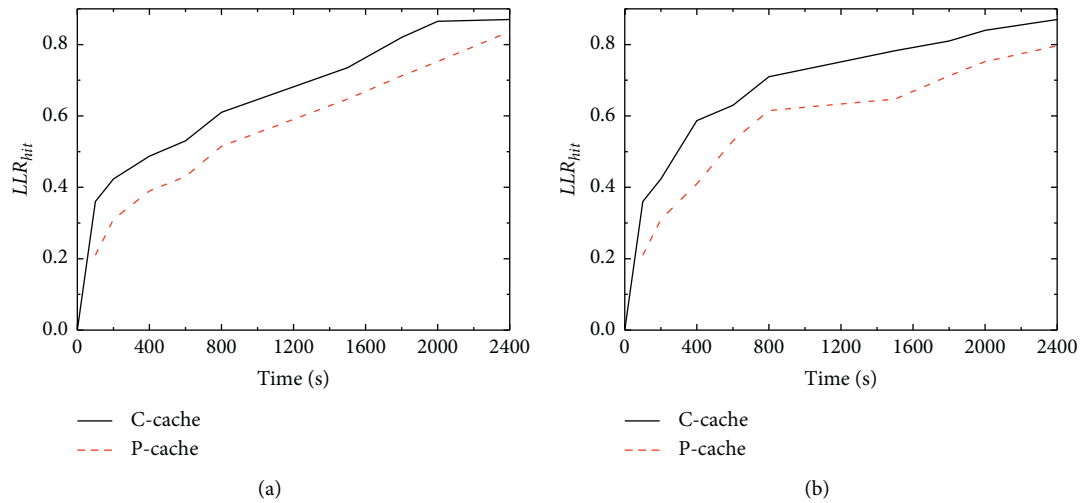

Figure 5: Local hit ratio of VGG. (a) LLR_{hit} during training VGG on D3. (b) LLR_{hit} during training VGG on D4.

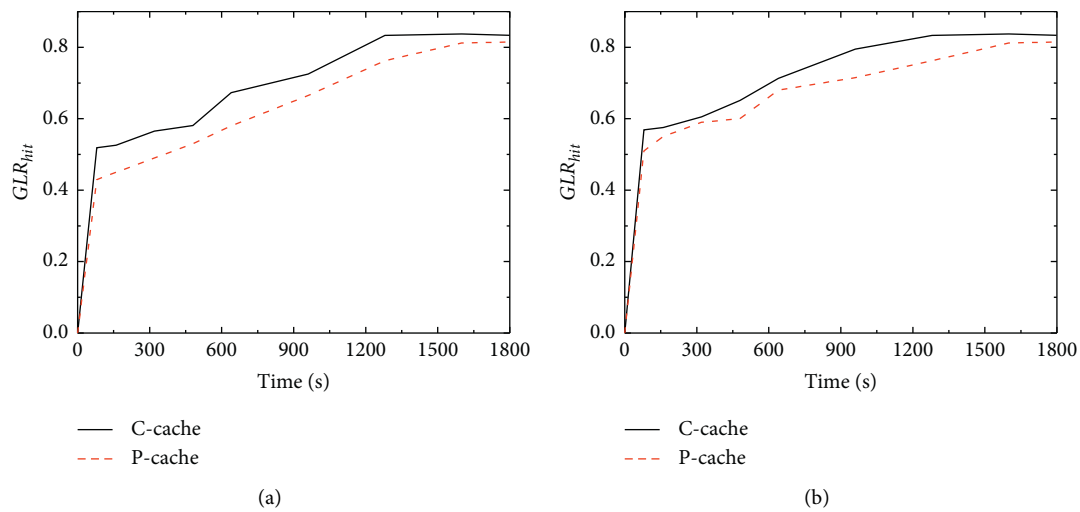

Figure 6: Global learning hit ratio of MLP. (a) GLR_{hit} during training MLP on D1. (b) GLR_{hit} during training MLP on D2.

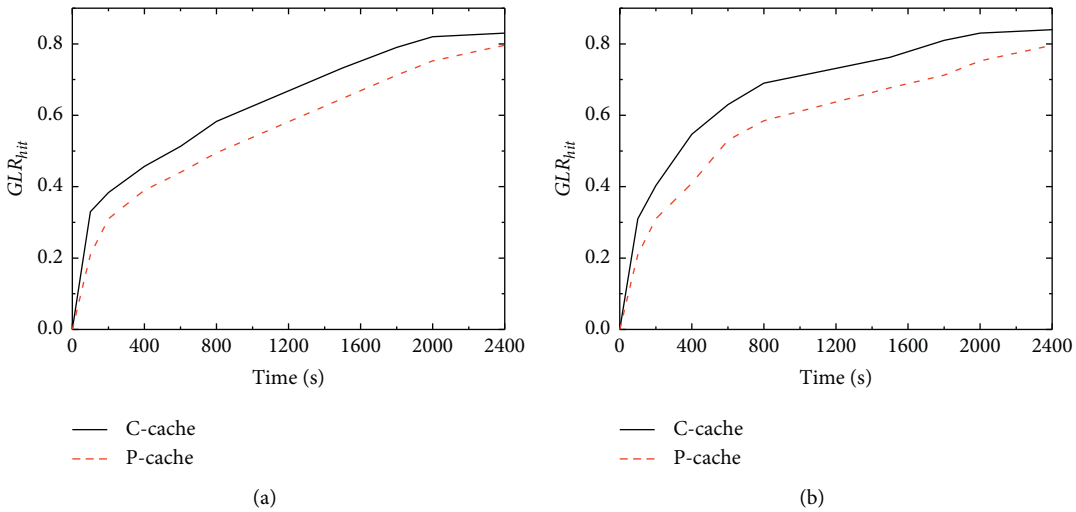

Figure 7: Global learning hit ratio of VGG. (a) GLR_{hit} during training VGG on D3. (b) GLR_{hit} during training VGG on D4.

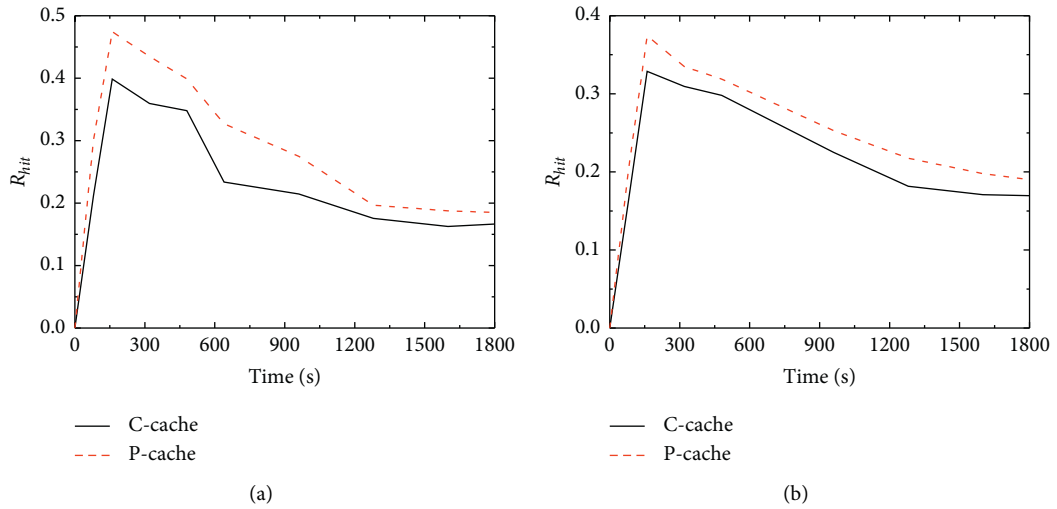

Figure 8: Global background hit ratio of MLP. (a) R_{hit} during training MLP on D1. (b) R_{hit} during training MLP on D2.

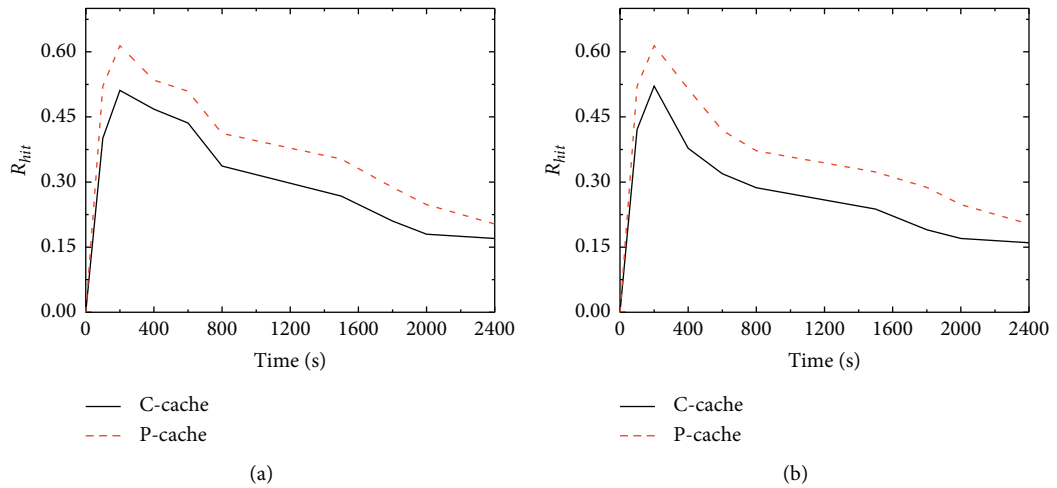

Figure 9: Global background hit ratio of VGG. (a) R_{hit} during training VGG on D3. (b) R_{hit} during training VGG on D4.

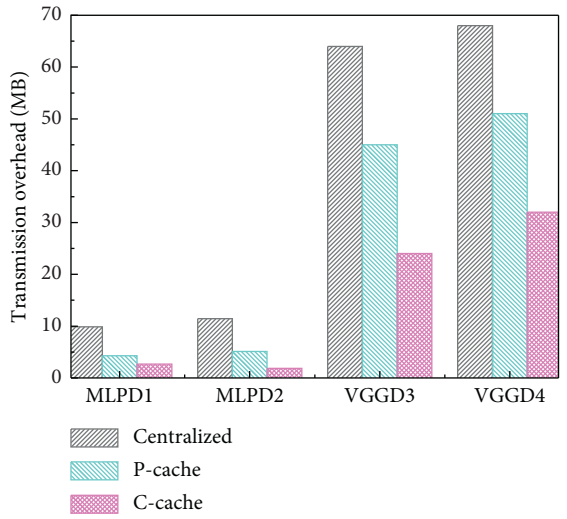

Figure 10: Transmission overhead.

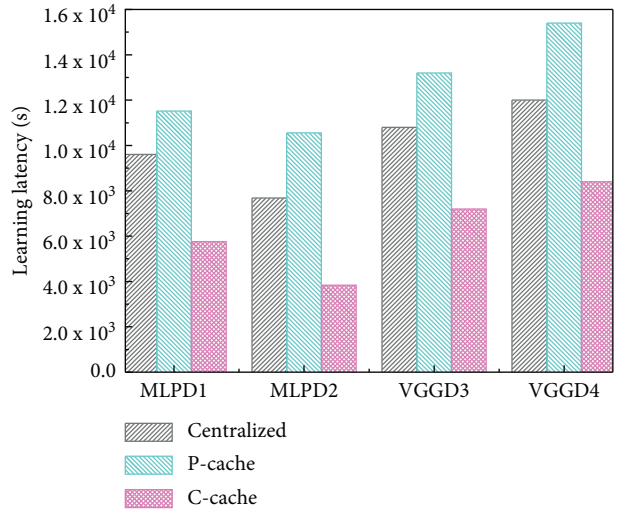

Figure 11: Learning latency.

Table 1: Learning accuracy comparison

Scheme	MLP		VGG	
	D1	D2	D3	D4
Centralized	**0.848**	**0.968**	**0.917**	**0.923**
P-cache	0.789	0.947	0.827	0.852
C-cache	0.847	**0.968**	**0.917**	**0.923**

centralized scheme is twice as much as that of C-cache. All learning data need to be sent to the data center, which makes the centralized scheme have the largest transmission overhead. In addition, the rational data requests and collaborative caching benefit C-cache regarding the transmission overhead. Valuable data are cached on edge nodes and thus reduce redundant data transmission among different edge nodes.

Figure 11 depicts the learning latency of different models on the three schemes. Both MLP and VGG can take advantage of the collaborative caching of C-cache to achieve fast convergence. The maximum difference between learning latency on P-cache and C-cache is 7000 s. Within one or two hours, C-cache provides sufficient data items for submodel learning and their ensemble process. Since the centralized scheme collects all of the training data to support model training, the model learning latency on centralized is less than P-cache. On the contrary, large transmission latency of centralized also degrades the model learning efficiency on centralized.

5.3.3. Learning Accuracy. Table 1 depicts the accuracy of MLP and VGG models trained on different schemes. Regarding different training models and datasets, the C-cache and the centralized scheme achieve the similar high performance in accuracy. This is because C-cache can provide more valuable training data to support model training, while the centralized scheme collects all the training data to support model training. On the contrary, P-cache cannot provide sufficient training data in a short time period, which affects the submodels' training on edge nodes and thus degrades the performance of the ensemble results.

6. Conclusion

In this paper, we propose an adaptive in-network collaborative caching scheme to support efficient ensemble learning at edge. In this scheme, edge nodes collaborate to obtain cached data items with different features as much as possible and train submodels with large differences, thus effectively improving the performance of ensemble learning. The extensive simulations demonstrate that our proposed collaborative caching scheme in edge network can significantly reduce learning latency and transmission overhead for ensemble learning.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

This paper has been submitted to a preprint website as per the following link: https://arxiv.org/abs/2010.12899 [46].

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2020-1-18), the National Science Foundation of China (61962045 and 61962044), the Strategic Priority Research Program of Chinese Academy of Sciences (XDC02030500), the Science and Technology Planning Project of Inner Mongolia Autonomous Region (2019GG372), the Key Technologies RD Program of Inner Mongolia Autonomous Region (2020GG0094), the Science Research Project of Inner Mongolia University of Technology (BS201934), and the Visiting Scholar Project of China Scholarship Council (201908150030 and 201904910802).

References

[1] R. Mitchell, J. Michalski, and T. Carbonell, *An Artificial Intelligence Approach*, Springer, Berlin, Germany, 2013.

[2] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: the confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2019.

[3] H. Li, K. Ota, and M. Dong, "Learning iot in edge: deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[4] J. Chen and X. Ran, "Deep learning with edge computing: a review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[5] S. Z. Li, *Encyclopedia of Biometrics: I-Z*, Vol. 2, Springer Science & Business Media, Berlin, Germany, 2009.

[6] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[7] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[8] Z. Chen, Q. He, L. Liu, D. Lan, H.-M. Chung, and Z. Mao, "An artificial intelligence perspective on mobile edge computing," in *Proceedings of 2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pp. 100–106, IEEE, Tianjin, China, August 2019.

[9] Z. Zhou, *Ensemble Learning: Basics and Algorithms*, Electronic Industry Press, London, UK, 2020.

[10] T.-V. Nguyen, N.-N. Dao, V. D. Tuong, W. Noh, and S. Cho, "User-aware and flexible proactive caching using lstm and ensemble learning in iot-mec networks," *IEEE Internet of Things Journal*, vol. 2021, p. 1, 2021.

[11] Y. Wang and T. Nakachi, "Secure face recognition in edge and cloud networks: from the ensemble learning perspective," in *Proceedings of ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2393–2397, IEEE, Barcelona, Spain, May 2020.

[12] Z.-H. Zhou, "Ensemble learning," *Encyclopedia of biometrics*, vol. 1, pp. 270–273, 2009.

[13] K. Tumer and J. Ghosh, "Theoretical foundations of linear and order statistics combiners for neural pattern classifiers," *IEEE Transactions on Neural Networks*, vol. 1996, 1996.

[14] S. Wang, T. Tuor, T. Salonidis et al., "When edge meets learning: adaptive control for resource-constrained distributed machine learning," in *Proceedings of IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 63–71, IEEE, Honolulu, HI, USA, April 2018.

[15] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, "Toward efficient content delivery for automated driving services: an edge computing solution," *IEEE Network*, vol. 32, no. 1, pp. 80–86, 2018.

[16] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

[17] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: a lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 24, pp. 693–701, 2011.

[18] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," *Advances in Neural Information Processing Systems*, vol. 1, no. 1, pp. 685–693, 2015.

[19] W. Wen, C. Xu, F. Yan et al., "Terngrad: ternary gradients to reduce communication in distributed deep learning," *Advances in Neural Information Processing Systems*, vol. 2017, pp. 1509–1519, 2017.

[20] Z. Yin, M. Zhao, Y. Wang, J. Yang, and J. Zhang, "Recognition of emotions using multimodal physiological signals and an

ensemble deep learning model," *Computer Methods and Programs in Biomedicine*, vol. 140, pp. 93–110, 2017.

[21] A. Kumar, J. Kim, D. Lyndon, M. Fulham, and D. Feng, "An ensemble of fine-tuned convolutional neural networks for medical image classification," *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 31–40, 2016.

[22] J. Xiao, "Svm and knn ensemble learning for traffic incident detection," *Physica A: Statistical Mechanics and Its Applications*, vol. 517, pp. 29–35, 2019.

[23] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez, "Multi-step forecasting for big data time series based on ensemble learning," *Knowledge-Based Systems*, vol. 163, pp. 830–841, 2019.

[24] W. Liu, M. Zhang, Z. Luo, and Y. Cai, "An ensemble deep learning method for vehicle type classification on visual traffic surveillance sensors," *IEEE Access*, vol. 5, pp. 24417–24425, 2017.

[25] X. Liu, Z. Liu, G. Wang, Z. Cai, and H. Zhang, "Ensemble transfer learning algorithm," *IEEE Access*, vol. 6, pp. 2389–2396, 2017.

[26] X.-l. Chen, L. Cao, C.-x. Li, Z.-x. Xu, and J. Lai, "Ensemble network architecture for deep reinforcement learning," *Mathematical Problems in Engineering*, vol. 2018, Article ID 2129393, 6 pages, 2018.

[27] R. Amer, M. M. Butt, and N. Marchetti, "Caching at the edge in low latency wireless networks," *Wireless Automation as an Enabler for the Next Industrial Revolution*, Wiely, Hoboken, NJ, USA, pp. 209–240, 2020.

[28] C. Li, J. Tang, H. Tang, and Y. Luo, "Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment," *Future Generation Computer Systems*, vol. 95, pp. 249–264, 2019.

[29] W.-C. Chien, H.-Y. Weng, and C.-F. Lai, "Q-learning based collaborative cache allocation in mobile edge computing," *Future Generation Computer Systems*, vol. 102, pp. 603–610, 2020.

[30] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *Proceedings of 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 366–369, IEEE, Seoul. Korea, September 2017.

[31] J. Tang, Z. Zhou, X. Xue, and G. Wang, "Using collaborative edge-cloud cache for search in internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 922–936, 2019.

[32] J. A. Khan, C. Westphal, J. Garcia-Luna-Aceves, and Y. Ghamri-Doudane, "Lochip: a distributed collaborative cache management scheme at the network edge," in *Proceedings of NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7, IEEE, Budapest, Hungary, April 2020.

[33] J. Wei, G. A. Gibson, P. B. Gibbons, and E. P. Xing, "Automating dependence-aware parallelization of machine learning training on distributed shared memory," in *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–17, Dresden, Germany, March 2019.

[34] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[35] T. G. Dietterich, "Ensemble learning," *The handbook of brain theory and neural networks*, vol. 2, pp. 110–125, 2002.

[36] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, CRC Press, Boca Raton, FL, USA, 2012.

[37] M. Perrone and L. Cooper, "When networks disagree: ensemble methods for neural networks," *Neural Networks for Speech and Image Processing*, Chapman & Hall, Boca Raton, FL, USA, 1991.

[38] G. Carneiro, "Ns-3: network simulator 3," in *Proceedings of UTM Lab Meeting*, vol. 20, pp. 4-5, Pisa, Italy, 2010.

[39] R. Lopez, *Open Nn: An Open Source Neural Networks C++ Library*, Artificial Intelligence Techniques, Ltd., Salamanca, Spain, 2019.

[40] A. Zubiaga, M. Liakata, and R. Procter, "Learning reporting dynamics during breaking news for rumour detection in social media," arXiv preprint arXiv:1610.07363.

[41] Forest covertype, https://kdd.ics.uci.edu/databases/covertype/covertype.data.htm.

[42] R. L. S. Torres, D. C. Ranasinghe, Q. Shi, and A. P. Sample, "Sensor enabled wearable rfid technology for mitigating the risk of falls near beds," in *Proceedings of 2013 IEEE International Conference on RFID (RFID)*, pp. 191–198, IEEE, Johar Bahru, Malaysia, April 2013.

[43] S. Schneider, G. W. Taylor, and S. C. Kremer, "Similarity learning networks for animal individual re-identification-beyond the capabilities of a human observer," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision Workshops*, pp. 44–52, Snowmass village, CO, USA, March 2020.

[44] S. Sarhan, S. Alhassan, and S. Elmougy, "Multimodal biometric systems: a comparative study," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 443–457, 2017.

[45] A. E. Ibor, F. A. Oladeji, O. B. Okunoye, and C. O. Uwadia, "Novel adaptive cyberattack prediction model using an enhanced genetic algorithm and deep learning (adacdeep)," *Information Security Journal: A Global Perspective*, vol. 2021, pp. 1–20, 2021.

[46] Y. Qin, D. Wu, Z. Xu, J. Tian, and Y. Zhang, "Adaptive in-network collaborative caching for enhanced ensemble deep learning at edge," arXiv preprint arXiv:2010.12899.