Hindawi
Security and Communication Networks
Volume 2022, Article ID 5358873, 14 pages
https://doi.org/10.1155/2022/5358873

Research Article

Research on Intelligent Scheduling Mechanism in Edge Network for Industrial Internet of Things

Zhenzhong Zhang,[1,2] Wei Sun,[1] and Yanliang Yu[3]

[1]*Center of Quantitative Economies, Jilin University, Changchun Jilin 130012, China*
[2]*Zhuhai College of Science and Technology, Zhuhai, Guangdong 519041, China*
[3]*School of Law and Social Work, Dongguan University of Technology, Dongguan, Guangdong 523000, China*

Correspondence should be addressed to Yanliang Yu; yyl3039@email.poe.edu.pl

Received 13 August 2021; Accepted 21 October 2021; Published 5 January 2022

Academic Editor: Xuyun Zhang

With the vigorous development of the Internet of Things, the Internet, cloud computing, and mobile terminals, edge computing has emerged as a new type of Internet of Things technology, which is one of the important components of the Industrial Internet of Things. In the face of large-scale data processing and calculations, traditional cloud computing is facing tremendous pressure, and the demand for new low-latency computing technologies is imminent. As a supplementary expansion of cloud computing technology, mobile edge computing will sink the computing power from the previous cloud to a network edge node. Through the mutual cooperation between computing nodes, the number of nodes that can be calculated is more, the types are more comprehensive, and the computing range is even greater. Broadly, it makes up for the shortcomings of cloud computing technology. Although edge computing technology has many advantages and has certain research and application results, how to allocate a large number of computing tasks and computing resources to computing nodes and how to schedule computing tasks at edge nodes are still challenges for edge computing. In view of the problems encountered by edge computing technology in resource allocation and task scheduling, this paper designs a dynamic task scheduling strategy for edge computing with delay-aware characteristics, which realizes the reasonable utilization of computing resources and is required for edge computing systems. This paper proposes a resource allocation scheme combined with the simulated annealing algorithm, which minimizes the overall performance loss of the system while keeping the system low delay. Finally, it is verified through experiments that the task scheduling and resource allocation methods proposed in this paper can significantly reduce the response delay of the application.

1. Introduction

The rapid development of technologies such as the Internet of Things has brought mankind into a new era of intelligence. The rapid popularization of high-speed Internet has brought about continuous growth in the amount of websites and data. The massive amount of data has promoted the evolution of the entire computing model and also put forward higher requirements on data storage and processing technology [1]. However, due to the disadvantages of traditional cloud computing technology, such as insufficient bandwidth, real-time performance, and high energy consumption, it has been unable to efficiently process massive amounts of data [2]. Therefore, edge computing emerged as a new computing model, which extends data to the edge of the network on the basis of cloud computing to achieve the overall performance of the Internet of Things technology [3, 4].

Edge computing brings convenient services in the face of complex network environments and diverse application services, but it also brings a series of new problems and challenges, such as the configuration of computing resources and task scheduling. Literature [5] puts forward the theory of edge computing on the basis of cloud computing and transmits data to the edge of the Internet to improve the overall availability and scalability of the system; literature [6]

puts forward the theoretical basis of fog computing for the first time, and the technology is also by configuring computing and storage devices at the edge of the Internet to reduce the amount of Internet data transmission, so as to achieve the purpose of reducing latency and saving bandwidth [7]. Compared with fog computing, edge computing technology pays more attention to the collaboration of resources between edge nodes and can handle data upstream of the cloud or downstream of the Internet of Things very well [8]. Resource allocation and task scheduling optimization in edge computing technology is one of the important research issues of this technology, and its implementation plan directly affects the utilization rate of resources and the service experience of users [9]. Literature [10] integrates optimization problems in edge computing scenarios and sorts out a number of optimization indicators according to the optimization scenarios. For the problem of resource optimization and allocation of edge computing, Brogi et al. sorted out the types of optimization algorithms, optimization goals, and constraints [11]. Aiming at the task scheduling problem, literature [12] specifically studied three task scheduling methods. The first method is concurrent, the second is FCFS (first come, first served), and the third method is allocated according to delay priority. In the first method, the acquired tasks are allocated to edge devices for processing, and there is no need to care about the usage of each device. In the second method, the acquired tasks will be processed in sequence according to the entry order. Only when the computing power of the edge node cannot handle the current task will the task be moved to the cloud for processing. In the delayed priority allocation method, the arriving tasks will be scheduled in the order of priority. When the edge computing resources are not enough, the low priority will be processed by the cloud. Research shows that although the number of tasks that can be executed at the same time is the largest in the first method, the equipment utilization is the highest, but because the resources that can be used for computing are limited, each task causes a large delay. In the second method, Since the order of task execution is carried out in order of priority, some tasks with lower priority will be sent to the cloud for execution, so this method cannot cope with some tasks with higher requirements for delay, and it also brings data transmission. Energy consumption is high. For this reason, literature [13] introduced a knapsack algorithm-based symbiosis search scheduling algorithm based on the above research. This method has significantly improved energy consumption, network utilization, and execution cost compared with the traditional knapsack algorithm and the FCFS method.

In order to make full use of the computing power of the edge server and further reduce the response delay of the application, literature [14] proposed a delay-aware application module management method oriented to the edge environment. Te-Yi et al. [15] divided the delay priority of different tasks and used heuristic algorithms to solve the problem of computing resource allocation, thereby improving the efficiency and quality of edge computing. According to the above-mentioned research findings, task scheduling and resource allocation in edge computing technology have attracted the attention of a large number of scholars, but the research on the two aspects of true comprehensive resource allocation and task scheduling needs to be further deepened. This paper studies the problem of resource allocation and task scheduling for edge computing. First, through the realization of collaborative caching between different edge nodes, each edge node caches differentiated data, so as to train and obtain a submodel with greater difference. To achieve a more accurate edge integration model, second, use cache compression records and record sharing to achieve reasonable data distribution scheduling and caching, and finally design and implement the TCP/IP network node cache module in the edge computing framework.

1.1. Principle Analysis of Collaborative Cache for Edge Computing. In order to improve the performance of edge computing, the process of edge computing is first studied. Integrated diversity, that is, the difference between submodels, is the key issue of integrated learning methods. Through research, it is found that if the same submodels are combined, there will be no performance improvement; if there is a performance improvement after the combination, there must be a difference between the submodels. Tumer et al. [16] analyzed the simple soft voting integration method through decision boundary analysis. In order to keep it simple, assuming that all submodels have the same error rate, the θ term is introduced to describe the relationship between the different submodels, and the expected cumulative error after integration is shown in the following formula:

$$\overline{err}(H) = \frac{1+\theta(n-1)}{n} err_i(h_i), i = 1, 2, \ldots, n. \quad (1)$$

In the above formula, $err_i(h_i)$ is the expected error rate of the submodel and n is the size of the integration scale. It can be seen from the formula that if the submodels are independent of each other, namely, $\theta = 0$, the ensemble learning error will be reduced by n times. If each submodel is associated with all other submodels, namely, $\theta = 1$, the performance of the integrated submodel will not be effectively improved. This analysis clearly reveals the importance of different submodels in ensemble learning, and the same conclusion is also applicable to other ensemble methods [17].

However, it is not easy to generate highly diverse submodels [18]. The biggest obstacle is that the submodels are obtained on the same task and the same training set, so there is often a high correlation between the submodels. Many theoretically feasible methods, such as the optimal solution of the weighted average method, are difficult to work in reality, and the situation may even be worse. In fact, the performance of the submodels should not be too bad; otherwise, the combined performance not only will not be improved but also will be reduced, which makes it more challenging to generate diverse submodels. If the performance of the submodel is poor, the cumulative errors after simple soft voting integration will continue to increase, and other integration methods have similar results [19, 20].

If the submodels are independent of each other, the error of ensemble learning will be reduced. If each submodel is related to other submodels, the error of ensemble learning will become larger. This analysis clearly reveals the importance of the different submodels. In this case, it is necessary to implement collaborative caching between different edge nodes, and each edge node caches differentiated data, so as to train a submodel with larger differences and realize a more accurate edge integration model.

1.2. Cache Compression Records and Record Sharing. The compressed record of the cached data plays an important role in the intelligent scheduling of the cache. The efficient compression recording method can record the data information cached by each edge node and realize the exchange and collection of cache information between edge nodes. The cache intelligent scheduling scheme can reasonably schedule the cache according to the data distribution and supports the distributed submodel learning and final integrated learning of each edge node, as shown in Figure 1. In this section, we mainly introduce two parts: cache data record and record sharing.

1.2.1. Cache Data on Edge Computing Nodes and Record Efficiently. The data required for the training of the integrated learning model is collected from the neighboring user terminal equipment and transmitted to the edge node, and the edge node is cached in the LRU mode and is efficiently recorded using the combinable counting bloom filter (CCBF). The specific process is as follows: when the data arrives at a certain edge node, whether the data has been cached by querying the CCBF is judged. If it has been cached, the data is not cached; if it has not been cached, the data is cached using LRU and used CCBF performs high-efficiency compression recording.

The specific operation is as follows: use k hash functions to hash the data received by the edge node into k bit arrays and check whether the corresponding unit of orBarr in CCBF is 1; if it is 1, it means that the data has been cached, so do not proceed. If it is not 1, LRU cache is required. The implementation of LRU adopts the form of a linked list. When caching, it is necessary to determine whether the cache capacity of the edge node is reached. If the cache capacity of the edge node is not reached, the data is cached at the head of the linked list; if the cache capacity has been reached, it is the oldest. The used data is eliminated, and, through the pseudorandom number generator, the bit array corresponding to the unit of each hash function operation in the last insertion operation of the data is cleared to zero, the orBarr array is updated, and the cache record is cleared. Then add the new data to the head of the cache linked list. After adding the data to the cache, use the pseudorandom number generator to correspond to the bit array whose location unit (unit with subscript $Hash_j(d)$) has been set to 1 according to the hash result. Randomly select a bit array $barr_i$ (the i-th bit array of CCBF) in the g bit array, set its subscript as $Hash_j(d)$ to 1, update the OrBarr array, and complete the update of CCBF.

1.2.2. Exchange and Merge Compressed Records of Cached Data with Neighbor Nodes. The compressed representation of cached data (CCBF) is exchanged and merged with neighbors within a certain range in order to obtain a global view of the edge node cached data, and the subsequent cache scheduling process can be guided based on this global view. Once a node receives a compressed record of the neighbor node cache data from the interface, the compressed record will be stored with the name $CCBF_i$, where i is the ID number of the corresponding edge node interface.

Use the ID number of the edge network node to exchange $CCBF_i$, hash the received data into k bit array units through k hash functions, and then query whether the orBarr unit corresponding to the neighbor node is 1; if it is 1, it means that the data has been existing in the cache data of the neighbor node, it is necessary to delete the redundant cache data, set the unit corresponding to the middle bit array of the node to 0, and update $CCBF_i$ the node.

The original compression record of the edge node is merged with the received neighbor nodes. The specific operation is to first determine whether the amount of cached data represented by the merged compression has exceeded the capacity n of CCBF, and then, according to the different bit arrays label, merge each bit array in order, and update the orBarr array to get a global view of the edge network cache data. After merging, a global view of the data compression records cached in the neighbor nodes can be obtained, and this view will be used to guide the neighbor nodes to cache various data subsequently received.

1.3. Differentiated Adaptive Collaborative Caching. The cache intelligent scheduling scheme can reasonably schedule the cache according to the data distribution and supports the distributed submodel learning and final integrated learning of each edge node. In this section, we mainly introduce the differentiated adaptive collaborative caching method for model learning.

1.3.1. Cache Different Data between Neighbor Nodes. When an edge node requests to cache some data, it needs to determine whether the data already exists in the cache of the node and its neighbor nodes according to the global view of

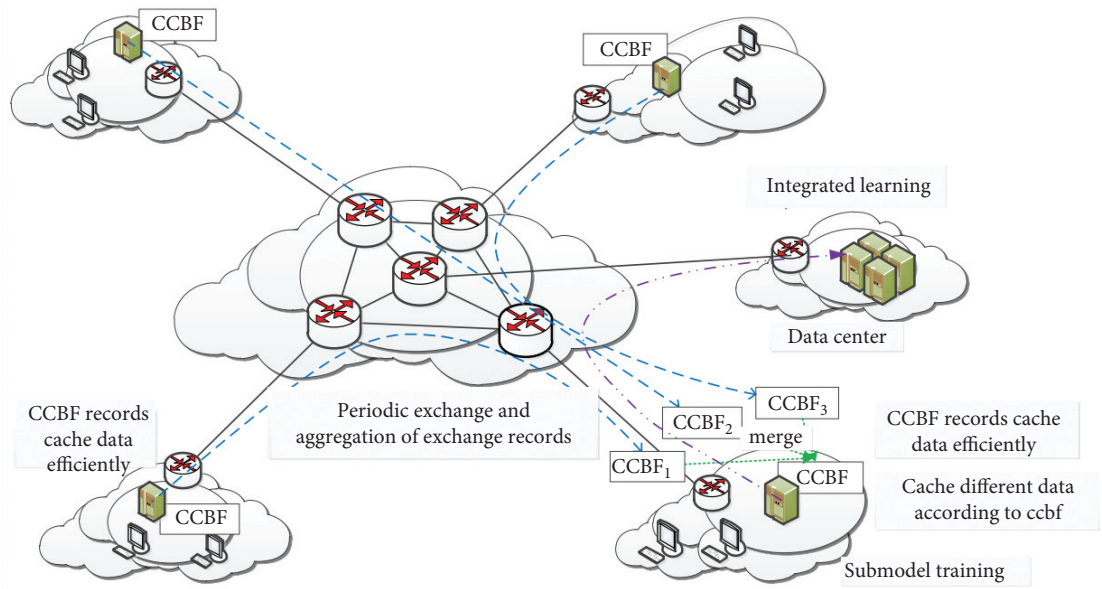

Figure 1: Collaborative caching scheduling method.

the cached data. The specific operation is as follows: query, which represents a global view of data cached in neighbor nodes. Hash the cached data requested by the node k times to obtain k hash results, and check whether the corresponding array orBarr unit is 1; if the corresponding unit in the orBarr array is 1, it means that the data has been cached at this edge. On the network node, the data is ignored, no caching operation is performed, and the following data processing is performed; if the corresponding unit of the array in is 0, it means that there is no compressed record of the data in the cache, indicating the cache of other neighboring nodes. If the data is not included, the data is added to the cache of the edge node, and the compressed record of the data is added to the corresponding node. Through the above operations, it can be ensured that different data can be cached on neighboring nodes for training different submodels, and, at the same time, communication overhead can be reduced through collaborative caching.

1.3.2. Distributed Training of Submodels. The data cached on a node is used to train the local submodel. When the local data is not enough to make the submodel converge, it is necessary to expand the scope of collaboration by requesting differentiated data from other edge nodes. By performing merging of orBarr in $CCBF_i$ of different neighboring nodes' cached data, the obtained cached data records of different neighboring nodes are compared with the cached data records of the local node to obtain the required data compression record $CCBF_i$ and send it to the corresponding edge node. When the corresponding edge node receives the request, it queries the cache of the local node according to $\overline{orBarr}$ and returns the differentiated data to the requesting node. After the requesting node receives the data, it caches the data and updates $CCBF_i$ and $CCBF$ and then inputs the data into the submodel for training. Repeat these processes until the submodel converges.

1.3.3. Integrated Learning. In order to reduce network data transmission traffic and ensure data privacy and security, the training results of the distributed submodels are uploaded to the data center, and the integrated results are obtained by assigning different weights to the output results of each submodel in the data center. The set output result $H(x)$ is shown in the following formula:

$$H(x) = \sum_{i=1}^{n} \omega_i h_i\ (x). \tag{2}$$

In the above formula, ω_i represents the weight of h, usually with the constraints of $\omega_i \geq 0$ and $\sum_{i=1}^{n} \omega_i = 1$.

The weights of these parameters in the submodel are uploaded to the central node, and the central node performs integrated learning. Specifically, for n sub-$h_1, \ldots, h_n$ models, the following methods are used for ensemble learning: $p(x)$ is the distribution of the input, $\epsilon_i(x)$ is the error term, and $C_{ij} = \int (h_i(x) - f(x))(h_j(x) - f(x))p(x)d(x)$.

The optimal weight can be solved by the following formula:

$$\omega = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{n} \omega_i \omega_j C_{ij}. \tag{3}$$

By Lagrangian multiplier method, ω_i is obtained as shown in the following formula:

$$\omega_i = \frac{\sum_{j=1}^{n} C_{ij}^{-1}}{\sum_{k=1}^{n} \sum_{j=1}^{n} \omega_i \omega_j C_{kj}^{-1}}. \tag{4}$$

1.4. Design and Implementation of Node Cache Module in TCP/IP Edge Network. The TCP/IP network node cache module is designed and implemented in the edge integrated learning framework. First, the implementation of the LRU cache module of TCP/IP network nodes is introduced, and then the method of deploying the LRU cache module to the NS-3 edge simulation platform is introduced.

LRU is the abbreviation of "Least Recently Used." It is a commonly used cache replacement algorithm. The cache data that has not been used the most recently is selected to be eliminated. The LRU cache in NS-3 mainly implements the function of caching data on each edge network node. In this section, the LRU cache on the TCP/IP edge network node is designed and implemented.

First the implementation of LRU cache is introduced, followed by the way to implement LRU cache on the NS-3 platform. The specific operations are as follows:

(1) The first step is to construct the code of the LRU cache, where the cache size needs to be set, so the function of LRU cache is used, and the capacity of the cache is set as a parameter of this function. The LRU cache is based on the encapsulated data packet as a unit for caching. The LRU cache mechanism uses the form of a linked list, placing the least recently used at the head of the linked list. It mainly includes three functions, namely, the addition, deletion, and search functions of cached data. The specific operations are as follows:

(1) To increase the cache data (Memory), it is necessary to first determine whether the current cache has reached the capacity of the cache. If the cache capacity has been reached, delete the last one of the linked list, and then store it; if the cache capacity is not reached, store it directly at the head of the cache.

(2) In the process of deleting the cached data (Remove), when the cache capacity is not enough, the last data in the linked list represents the most recently unused data, and it is deleted.

(3) In the process of looking up cached data (Lookup), if the data is not found, -1 is returned; if it is found, the value of the data is returned, and then the data is placed at the head of the cache.

(2) Implement LRU cache on the NS-3 platform.

To implement LRU caching on the NS-3 platform, you need to add a custom LRU caching module to the original module of NS-3.

(1) First, the basic structure of NS-3 is introduced here. src is the source code directory of NS-3, and the directory structure basically corresponds to the compiled module. Each file in the src directory basically corresponds to a module, and the structure of all modules in it is basically the same.

(2) Then, according to the design of NS-3, the implementation of LRU cache is added to NS-3 as a custom module.

1.5. Edge Network Node Deployment. This paper deploys the edge network tree topology as shown in Figure 2. As shown in the figure, it includes a remote data center, a gateway node, four edge computing nodes, and eight terminal devices, which are connected through a gigabit link, and the data transmission between them is point-to-point transmission. Peer-to-peer technology (P2P), also known as peer-to-peer network technology, is a new network technology that relies on the computing power and bandwidth of participants in the network, instead of concentrating all the dependencies on a few servers in the edge network topology. The cache size of each edge computing node is 2000 KB. Each edge computing node can cache data, efficiently record cached data, and perform computing tasks for cached data.

The data required for neural network model training are all released by the terminal equipment node. The terminal device generates the learning data of the model and sends the data to the edge computing node. After the edge computing node receives the data, it first performs data caching and efficient recording and then uses the different data in the collaborative cache to train the submodel and finally sends the training results of the submodel to the data center for integrated learning. Background traffic data is released by remote data center nodes. The data center generates background traffic data and sends the data to edge computing nodes. After the edge computing node receives the data, it caches the data and sends the background traffic data to the terminal device.

1.6. Construction of Edge Network Node Learning Module. In this article, the edge network node learning module is designed and implemented. When deploying a neural network model on the NS-3 edge simulation platform, the difficulty encountered is the joint compilation of the NS-3 platform and OpenNN. So, the method of joint compilation of NS-3 platform and OpenNN is introduced first, and then the process of edge integration learning based on OpenNN design pattern is introduced. The operation steps applied on the simulation platform of NS-3 are shown in Figure 3.

Because the NS-3 simulation platform is compiled with /waf, in the process of compiling the OpenNN neural network library, the newly added library needs to be included in the wscript file, and the corresponding library file needs to be included in the script. The specific operations are as follows:

(1) Put the Eigen folder in OpenNN under the ndnSIM/ns-3 folder, which is the preliminary step of the joint compilation of OpenNN and NS-3. Eigen is a C++ template library for linear operations, supporting matrix and vector operations, numerical analysis, and related algorithms. Because OpenNN contains a lot of matrix operations, you need to use the Eigen library.

(2) Add in the corresponding position in the wscript file under the NS-3 folder:

Def build (bld):

Bld.stlib ("opennn")

Module.uselib = 'opennn.'

Module.source = 'opennn/∗∗/∗.cpp.'

Module.full_headers = 'opennn/∗∗/∗.cpp.'

Such an operation is to add the OpenNN neural network library to the wscript file and include the corresponding header files and source files.

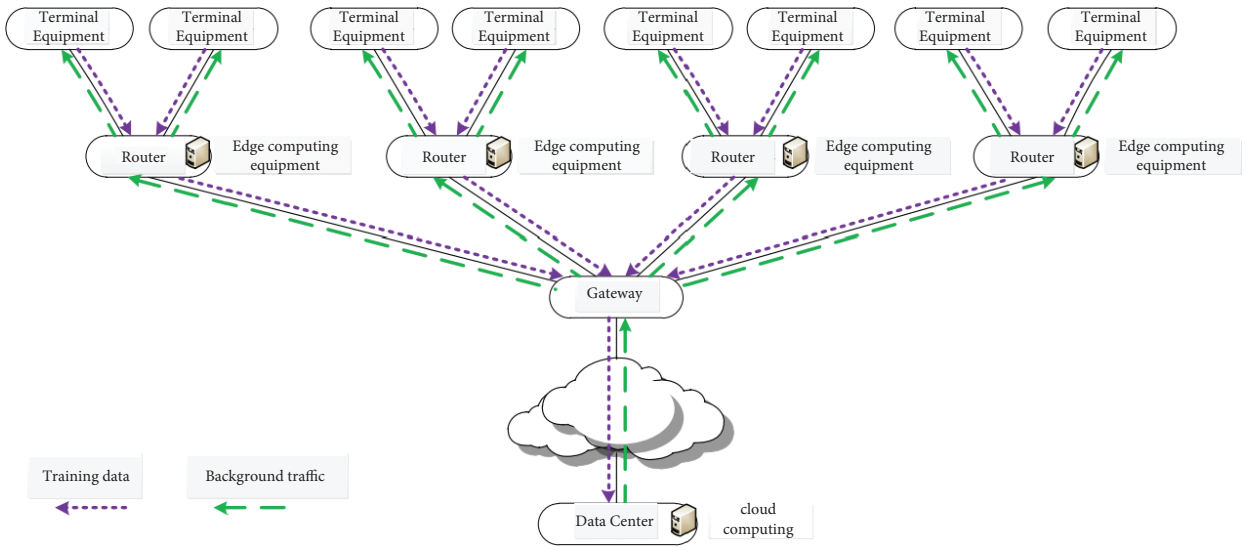

Figure 2: Edge network topology.

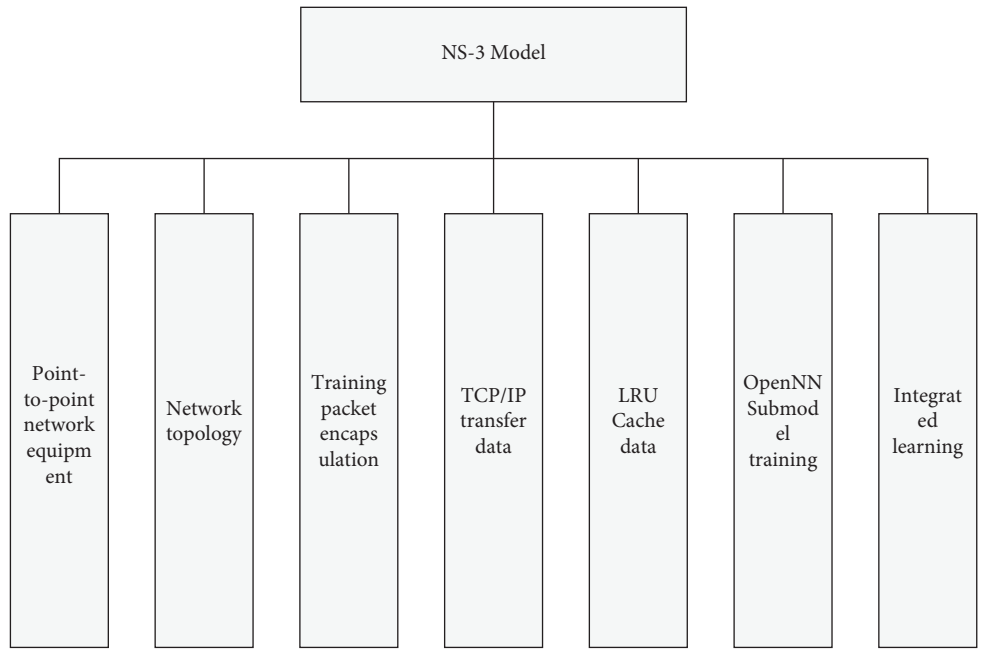

Figure 3: Edge network topology.

(3) Add the following to the header file in the code that needs neural network model training:

#include "../opennn/opennn.h."

Because the opennn.h file contains the header files required by OpenNN, only including this one header file in the source file can include all the required header files.

(4) NS-3 contains a fixed Vector vector. When using the Vector vector defined by the Eigen library in OpenNN, it needs to be distinguished. The method adopted is as follows:

Using namespace OpenNN:

When using the OpenNN model, first include the OpenNN namespace, and then when using the OpenNN Vector vector, add "OpenNN:" in front, that is, OpenNN: : Vector *****. In this way, the Vector vector in NS-3 can be distinguished from the vector in OpenNN.

The ensemble learning process is closely related to different submodels. In the edge integrated learning scenario, different edge nodes often deploy similar models, build submodels by learning the data around the edge nodes, and finally distribute the submodels on different nodes through the central node to form an integrated model. In this case, it is necessary to provide different data for different edge nodes, so as to obtain the training results of different

submodels, so as to achieve a more accurate integrated model.

In this article, the integrated learning method based on the OpenNN design pattern obtains the results by assigning different weights to the output results of each submodel. The set output result is $H(x)$, where $H(x) = \sum_{i=1}^{n} \omega_i h_i(x)$ represents the weight of ω_i, usually with $\omega_i \geq 0$ and $\sum_{i=1}^{n} \omega_i = 1$ constraints.

The weights of these parameters in the submodel are uploaded to the central node, and the central node performs integrated learning. Specifically, for n submodels $h_1, \ldots, h_n$, the following methods are used for ensemble learning.

Assume that the output of each submodel can be written as a true value plus an error term, as shown in the following formula:

$$h_i(x) = f(x) + \epsilon_i(x), i = 1, \ldots, n. \tag{5}$$

The integration error can be expressed as in the following formula:

$$\begin{aligned} \widehat{err}(H) &= \int \left(\sum_{i=1}^{n} \omega_i h_i(x) - f(x) \right)^2 p(x) d(x) \\ &= \int \left(\sum_{i=1}^{n} \omega_i h_i(x) - f(x) \right) \times \left(\sum_{j=1}^{n} \omega_j h_j(x) - f(x) \right) p(x) d(x) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} \omega_i \omega_j C_{ij} \int \left(h_i(x) - f(x)\right)\left(h_j(x) - f(x)\right) p(x) d(x). \end{aligned} \tag{6}$$

In the above formula, $p(x)$ is the distribution of the input and $\epsilon_i(x)$ is the error term. The optimal weight of $C_{ij} = \int (h_i(x) - f(x))(h_j(x) - f(x)) p(x) d(x)$ can be solved by the following formula:

$$C_{ij} = \int \left(h_i(x) - f(x)\right)\left(h_j(x) - f(x)\right) p(x) d(x). \tag{7}$$

By Lagrangian multiplier method, D is obtained by the following formula:

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

2. Experiment

2.1. Lab Environment. The performance of the adaptive collaborative caching scheme was evaluated on the NS-3 platform. The NS-3 platform is a modular, programmable, extensible, open, open-source, and community-supported computer network simulation framework. Connect the neural network library OpenNN (Open Neural Network Library) to NS-3 for experimental simulation. OpenNN is an open-source neural network library for the construction of neural networks. It has a wide range of applications, including functional regression, pattern recognition, time series forecasting, optimal control, optimal shape design, or inverse problems. In this article, all simulation experiments are performed on a local machine. The configuration and environment of the experiment host are as follows:

(1) CPU: Intel Core i7, 3.4 G CPU;

(2) Installed memory (RAM): 16 GB;

(3) Linux operating system: Ubuntu 16.04;

(4) Kernel version: 3.19.

2.2. Dataset. In order to evaluate the performance of the collaborative caching scheme, this paper uses four datasets for learning. Specifically, two text datasets (D1 and D2) are used to train the MLP model. In order to train the VGG model, a tigerface image dataset (D3) and a human face dataset (D4) are applied.

(1) Covertype dataset (D1): this dataset includes the forest vegetation types of Roosevelt National Forest. There are 4 types of soil, corresponding to 7 types of vegetation. The 581,012 data-item forest vegetation is divided into four soil types. The number of data items for different soil types is uneven. The number of type 4 is less than 3,000, and the number of type 5 is close to 10,000. The quantity of any other type is greater than 10,000.

(2) Healthy elderly dataset (D2): the sequential exercise data of 14 healthy elderly aged 66 to 86 years who used sensors to identify clinical environmental activities. Participants were assigned to two clinical room environments (S1 and S2). S1 (Clinical Room (1)) and S2 (Clinical Room (2)) are equipped with different sensor receiving numbers and positions. The number of data items is 75128, which is divided into 6 different behaviors on average.

(3) Reid-tigerface dataset (D3): Atrw Reid-tigerface image captured. After the picture is edited, the image resolution is adjusted to 128×128. There are 500 tigers in total, each of which has 10 photos. According to the active region (Russia Far East and Northern India), the dataset is divided into two scenarios.

(4) Casia-face dataset (D4): obtain face images of human faces. After the picture is edited, the image resolution is adjusted to 128×128. There are 500 people in total,

and each of them has 10 facial photos. According to the shooting angle (front position and side 45°), the dataset is divided into two scenes.

2.3. Validation Model. This paper implements the two following learning models: the multilayer perceptron (MLP) model is used to train two text datasets, and the Visual Geometry Group (VGG) network model is used to train two image datasets. The model is introduced in detail.

2.3.1. Multilayer Perceptron (MLP) Model. MLP is a feedforward artificial neural network that can map multiple input data to output data. Each layer of MLP is a fully connected layer. This paper implements a six-layer MLP model, including an input layer, four hidden layers, and an output layer. The following describes the internal structure of the multilayer perceptron.

Neurons can be combined into a neural network. The structure of a neural network refers to the number, arrangement, and connectivity of neurons. Any kind of network structure can be represented by a directed label graph, where nodes represent neurons, and edges represent connections between neurons. The edge labels represent the parameters of the neuron and indicate the inflow of the neuron. Most neural networks, even biological neural networks, present a hierarchical structure. In this case, the working layer is the basis for determining the structure of the neural network. Therefore, a neural network usually consists of a set of perception nodes that constitute the input layer, one or more hidden layers of neurons, and a set of neurons that constitute the output layer. As mentioned above, the characteristic neuron model of the multilayer perceptron is the perceptron. On the other hand, the multilayer perceptron has a feedforward network structure. The feedforward structure does not contain cycles; that is, the structure of the feedforward neural network can be expressed as an acyclic graph. Therefore, the neurons in the feedforward neural network are divided into a series of layer $h+1$ neurons $L^{(1)}, \ldots, L^{(h)}, L^{(h+1)}$, so that the neurons in any layer are only connected to the neurons in the next layer. The input layer is composed of n external inputs, not a neuron layer; the hidden layer $L^{(1)}, \ldots, L^{(h)}$, respectively, contains a hidden neuron in $s^{(1)}, \ldots, s^{(h)}$; the output layer $L^{(h+1)}$ is composed of m output neurons. Figure 4 shows the network structure of the multilayer perceptron. There are n inputs, h hidden layers, $s^{(i)}$ neurons, and $i = 1, \ldots, h$ and neurons are in the output layer. In this chapter, the superscript is used to identify the layer.

The multilayer perceptron neural network can be regarded as a parameterized function space V from input $X \subset \mathbb{R}^n$ to output $Y \subset \mathbb{R}^m$. The element form of V is y: $X \longrightarrow Y$. They are parameterized by neural parameters, which can be combined in a d-dimensional vector $\zeta = (\zeta_1), \ldots, (\zeta_d)$. Therefore, the dimension of the function space V is d.

For the first hidden layer L (1), by formula (9), the combined function is obtained by adding the dot product of the weight and the input to the deviation, thereby obtaining

$$c^{(1)} = b^{(1)} + w^{(1)} \cdot x. \tag{9}$$

According to formula (10), the output of this layer $a^{(1)}$ is obtained by the combination of conversion and activation function:

$$y^{(1)} = a^{(1)}\left(c^{(1)}\right). \tag{10}$$

Similarly, for the last hidden layer, the combined function is given by the following formula:

$$c^{(h)} = b^{(h)} + w^{(h)} \cdot y^{(h-1)}. \tag{11}$$

The output of this layer is found by formula (12) by using the activation function:

$$y^{(h)} = a^{(h)}\left(c^{(h)}\right). \tag{12}$$

The output of the neural network is obtained by transforming the output of the last hidden layer by the neurons in the output layer F. Therefore, the combined form of the output layer is shown in the following formula:

$$c^{(h+1)} = b^{(h+1)} + w^{(h+1)} \cdot y^{(h)}. \tag{13}$$

The output of the output layer is transformed by formula (14) through the combination of the layer and activation into

$$y^{(h+1)} = a^{(h+1)}\left(c^{(h+1)}\right). \tag{14}$$

Combining the above equations, an explicit expression of the multilayer perceptron function is obtained in the following form:

$$y = a^{(h+1)}\left(b^{(h+1)} + w^{(h+1)} \cdot a^{(h)}\left(b^{(h)} + w^{(h)} \cdot a^{(h-1)}\left(\ldots a^{(1)}\left(b^{(1)} + w^{(1)} \cdot x\right)\right)\right)\right). \tag{15}$$

In this way, the multilayer perceptron function is represented by formula (16) as the composition of the layer output function:

$$y = y^{(h+1)} \circ y^{(h)} \circ \ldots \circ y^{(1)}. \tag{16}$$

Multilayer perceptron can be regarded as a function of multiple variables formed by the superposition and addition of functions of one variable. Different activation functions produce different function families, and multilayer perceptrons can define these function families. Similarly, different neural parameter sets cause different elements in the function space defined by a particular multilayer perceptron.

2.3.2. Visual Group Network (VGG) Model. VGG is a deep convolutional neural network for computer vision. The implementation of this paper includes 5 convolutional

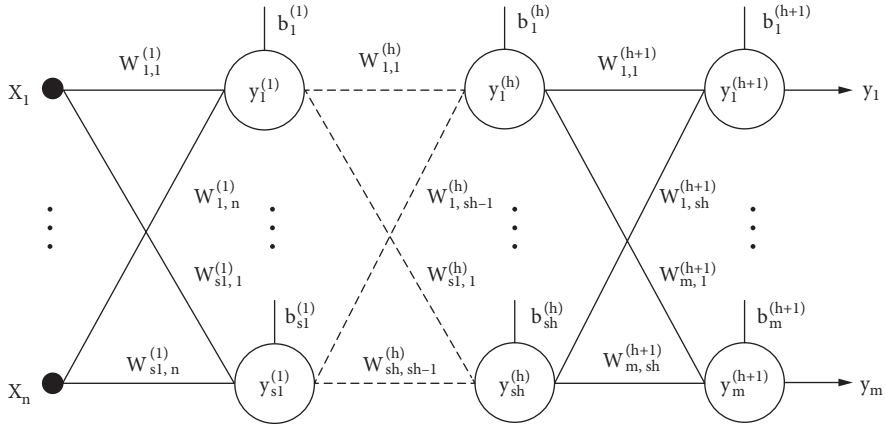

Figure 4: Multilayer perceptron.

blocks, each of which consists of 2–4 convolutional layers. At the same time, a maximum pooling layer is connected to the end of each block to reduce the size of the picture. The number of convolution kernels in each block is the same, and the number of convolution kernels in the later block is larger. For five convolution blocks, each layer contains 64-128-256-512 convolution kernels, and, in this article, 10 convolutional layers are used and 4 pooling layers are alternately performed, and the specific arrangement is shown in Figure 5. Among them, the convolution layer uses a 3∗3 convolution kernel, the activation function uses the ReLU activation function, that is, $F(x) = \max(0, x)$, and the training algorithm uses the Adam algorithm that can adaptively adjust the learning rate. The following describes the Adam optimization algorithm.

The Adam optimization algorithm is an extension of the stochastic gradient descent algorithm. Recently, it is widely used in deep learning applications, especially tasks such as computer vision and natural language processing. Adam is different from the classic stochastic gradient descent method. Stochastic gradient descent maintains a single learning rate (called α) for all weight updates, and the learning rate does not change during the training process. The Adam optimization algorithm maintains a learning rate for each network weight (parameter) and adjusts it individually as the learning expands. This method calculates the adaptive learning rate of different parameters from the budget of the first and second moments of the gradient. The following describes the Adam parameter configuration:

α: it is called the learning rate or step size. It controls the weight update rate (such as 0.001). A larger value (such as 0.3) will have faster initial learning before the learning rate is updated, while a smaller value (such as 1.0E-5) will make the training converge to better performance.

β_1: it is the exponential decay rate of the first moment estimation (such as 0.9).

β_2: it is the exponential decay rate of the second moment estimation (such as 0.999). This hyperparameter should be set to a number close to 1 in sparse gradients (such as in NLP or computer vision tasks).

ε: this parameter is a very small number, which is to prevent division by zero in implementation (such as 10E-8).

2.4. Classification Accuracy of Neural Network Model. Table 1 describes the classification accuracy of the MLP and VGG models trained on different schemes. For different training models and datasets, the collaborative caching scheme and the centralized scheme both achieve similar high performance in accuracy. This is because the collaborative caching solution can provide more valuable training data to support model training, while the centralized solution can collect all training data to support model training. On the contrary, the solution of periodically requesting cached data cannot provide enough training data in a short time, which affects the training of the edge nodes by the submodel, thereby reducing the performance of the integrated result.

2.5. Training Delay of Neural Network Model. Figure 6 describes the learning delay of different models under the three schemes. It can be seen from the figure that both MLP and VGG can use cooperative caching to achieve rapid convergence. There is a maximum difference of 7000 seconds in the learning delay between the periodic request cached data scheme and the collaborative cache scheme. Within one to two hours, the collaborative caching solution provided enough cached data items for the submodel learning and integration process. Since the centralized solution collects all training data to support model training, the centralized model learning delay is less than that of the solution that periodically requests cached data. On the other hand, the centralized transmission delay is large, which also reduces the efficiency of centralized model learning.

2.6. Network Data Transmission Traffic Load. The network data transmission traffic load is shown in Figure 7. It can be seen from the figure that regardless of the model or dataset, the network data transmission traffic load of the collaborative caching scheme is always the smallest, and more

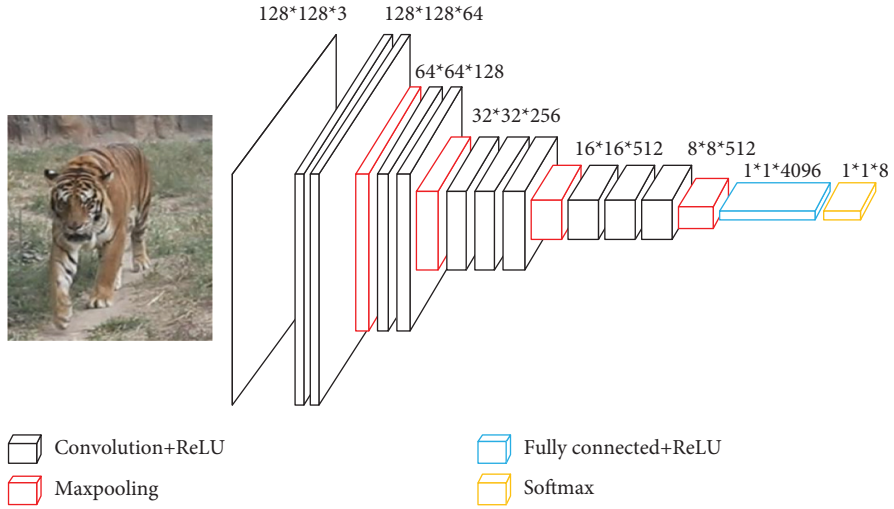

Figure 5: VGG model.

Table 1: Classification accuracy of neural network model.

Method	MLP		VGG	
	D1	D2	D3	D4
Centralized	0.848	0.968	0.917	0.923
Periodically requested cache	0.789	0.947	0.827	0.852
Cooperative caching	0.847	0.968	0.917	0.923

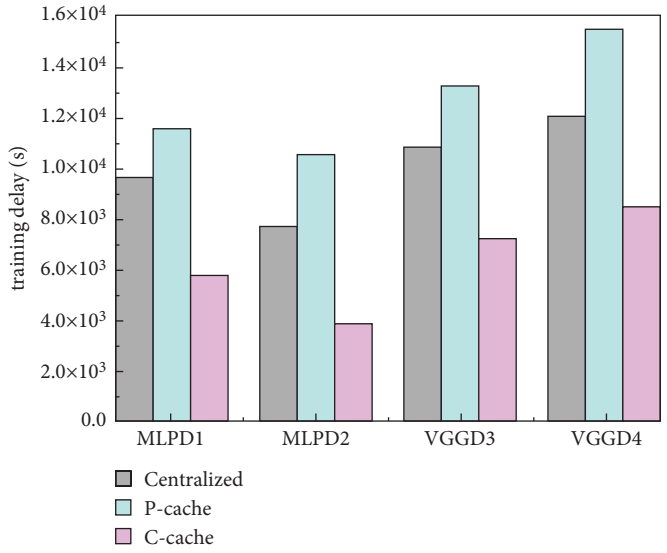

Figure 6: Training latency of neural network model.

powerful models such as VGG will consume more communication resources. The network data transmission traffic load of the centralized solution is twice that of the collaborative cache solution. Because all learning data needs to be sent to the data center, the network data transmission traffic load of the centralized solution is the largest. In addition, data request and cooperative caching are beneficial to the cooperative caching scheme in terms of transmission overhead. More valuable data is cached on edge nodes, thereby reducing redundant data transmission between different edge nodes and reducing the transmission traffic load in the network.

2.7. Cache Hit Rate. Since the centralized scheme trains the model in the data center and does not cache the data at the edge nodes, we only compare the cache hit rates of Centralized, P-cache, and the proposed C-cache. The local learning hit rate is shown in Figure 8. The overall learning hit rate is shown in Figure 9. The local learning hit rate of

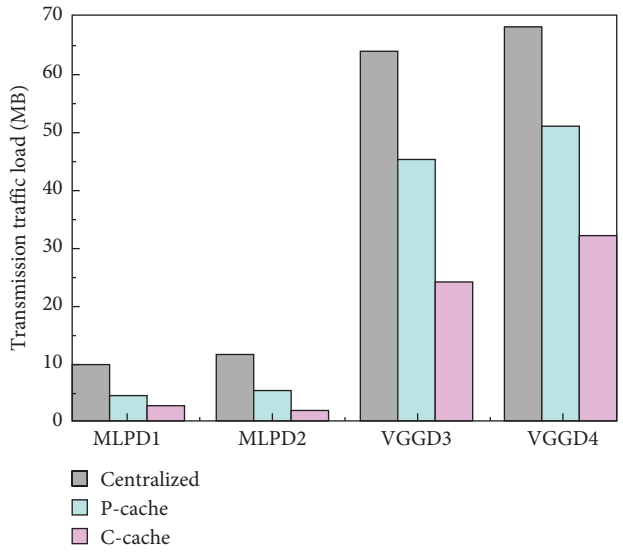

Figure 7: Network data transmission overhead.

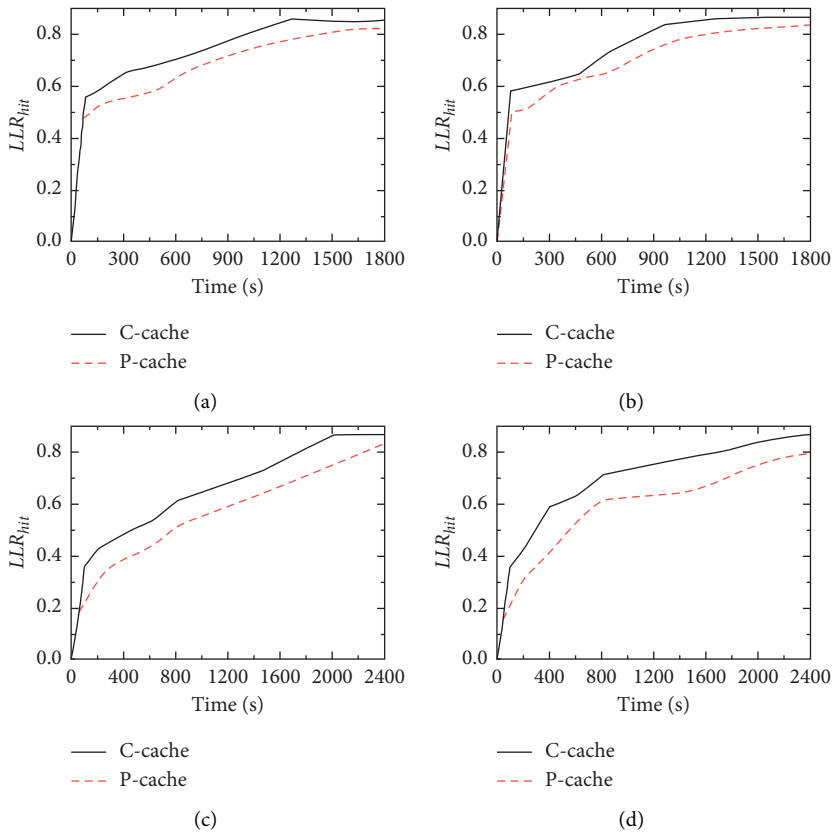

Figure 8: Local learning hit ratio. (a) LLR_{hit} during training MLP on D1. (b) LLR_{hit} during training MLP on D2. (c) LLR_{hit} during training VGG on D3. (d) LLR_{hit} during training VGG on D4.

C-cache and P-cache is increased to the maximum stable values of 0.87 and 0.85, respectively. The global learning hit rate of C-cache and P-cache is increased to the maximum stable values of 0.83 and 0.81, respectively, and the learning data is generated and cached at different edge nodes.

Figure 10 depicts the hit rate of background traffic data. The cache hit rate of background traffic data first increases with the passage of time. When the learning data increases, more background traffic data is switched from the cache of edge computing nodes. Therefore, the cache hit rates of

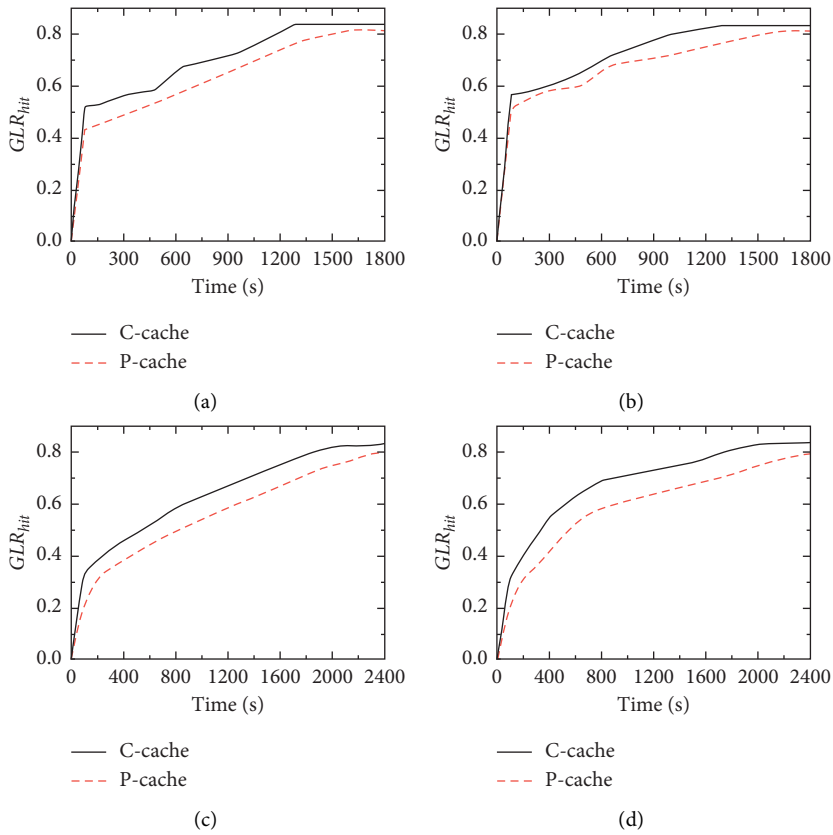

Figure 9: Global learning hit ratio. (a) GLR_{hit} during training MLP on D1. (b) GLR_{hit} during training MLP on D2. (c) GLR_{hit} during training VGG on D3. (d) GLR_{hit} during training VGG on D4.

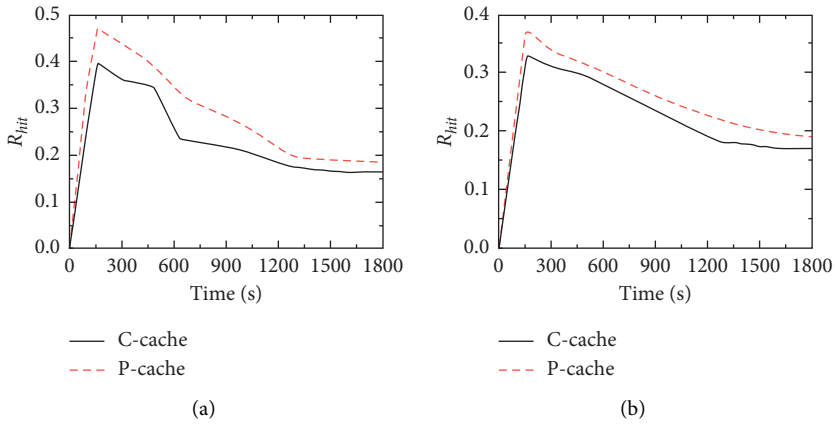

Figure 10: Continued.

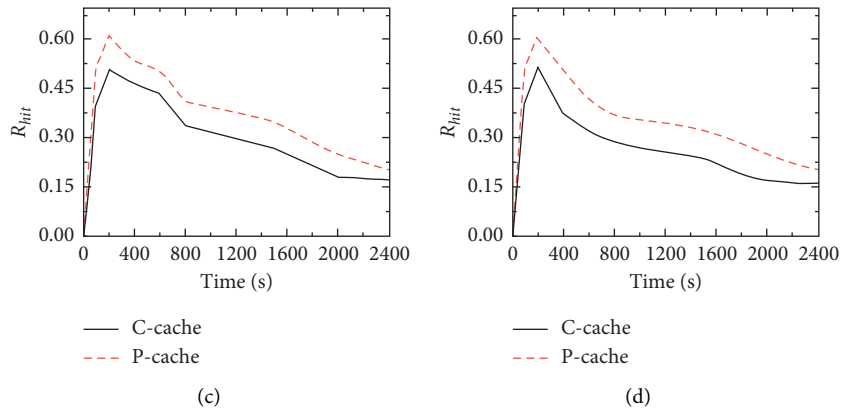

Figure 10: Background traffic data hit ratio. (a) R_{hit} during training MLP on D1. (b) R_{hit} during training MLP on D2. (c) R_{hit} during training VGG on D3. (d) R_{hit} during training VGG on D4.

C-cache and P-cache middle and background traffic data are reduced to 0.17 and 0.19, respectively. For different training models and datasets, the cache hit rate under C-cache decreases faster than that under P-cache. This is because C-cache can use learning data better than P-cache and reserves less available cache space for caching background traffic data.

3. Conclusion

As a complementary extension of cloud computing technology, mobile edge computing will reduce the computing power of the previous cloud to the edge nodes of the network. Through the cooperation between computing nodes, the number of nodes can be calculated, the type can be more comprehensive, and the calculation range can be larger. The emergence of mobile edge computing makes up for the shortcomings of cloud computing technology. Aiming at the problem of network edge cache computing and intelligent scheduling of resource allocation, in order to reduce network traffic load and delay, a simulation framework is finally established within the framework of effective recording cache data collaboration and edge computing learning framework to verify the network collaborative cache solution proposed in this paper. A large number of simulation results show that the collaborative caching scheme proposed in edge network can significantly reduce the learning delay and transmission cost of ensemble learning. In future studies, more datasets and more complex integrated learning models can be used to further improve the experiment. The edge integrated learning framework designed in this paper can be further optimized.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): a vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] Setting the standard[Z]. https://www.itu.int/en/ITU-T/wtsa16/Documents AVTSASna pshotReport. pdf.

[3] X. Xu, Z. Fang, J. Zhang et al., "Edge content caching with deep spatiotemporal residual network for IoV in smart city," *ACM Transactions on Sensor Networks*, vol. 17, no. 3, pp. 1–33, 2021.

[4] M. Chiang and T. Zhang, "Fog and IoT: an overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[5] X. Xu, Z. Fang, L. Qi, X. Zhang, Q. He, and X. Zhou, "TripRes," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 17, no. 2, pp. 1–21, 2021.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the 1st Edition MCC Workshop Mobile Cloud Comput*, pp. 13–16, NY, USA, 2012.

[7] W. Shi, J. Cao, Q. Zhang et al., "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[8] X. Xu, D. Zhu, X. Yang, S. Wang, L. Qi, and W. Dou, "Concurrent practical byzantine fault tolerance for integration of blockchain and supply chain," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1–17, 2021.

[9] W. Wen, C. Xu, F. Yan et al., "Terngrad: ternary gradients to reduce communication in distributed deep learning," in *Proceedings of the Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1509–1519, Long Beach California, USA, December 2017.

[10] H. Hussain, S. U. R. Malik, and A. Hameed, "A survey on resource allocation in high performance distributed computing systems," *Parallel Computing*, vol. 39, no. 11, pp. 709–736, 2013.

[11] J. Bellendorf and Z. Á Mann, "Classification of optimization problems in fog computing," *Future Generation Computer Systems*, vol. 107, no. 1, pp. 158–176, 2020.

[12] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog: state of the art and open challenges," *Software: Practice and Experience*, vol. 1, no. 1, pp. 1–8, 2019.

[13] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[14] D. Rahbari and M. Nickray, "Scheduling of fog networks with optimized knapsack by symbiotic organisms search," in *Proceedings of the 2017 21st Conference of Open Innovations Association (FRUCT)*, pp. 278–283, Helsinki, Finland, November 2017.

[15] T. Zhang, J. Deng, and J. Wang, "Progressive damage analysis (PDA) of carbon fiber plates with out-of-plane fold under pressure," *Computer Modeling in Engineering and Sciences*, vol. 124, no. 2, pp. 545–559, 2020.

[16] M. Redowan, R. Kotagiri, and B. Rajkumar, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology*, vol. 19, no. 1, pp. 1–21, 2018.

[17] T. Y. Kan, Y. Chiang, and H. Y. Wei, "Task offloading and resource allocation in mobile-edge computing system," in *Proceedings of the 2018 27th Wireless and Optical Communication*, pp. 1–4, Hualien, Taiwan, May 2018.

[18] T. Zheng, Y. Chang, and S. Zhang, "Quantum risk assessment model based on two three-qubit GHZ states," *Computer Modeling in Engineering and Sciences*, vol. 124, no. 2, pp. 573–584, 2020.

[19] T. Kagan and J. Ghosh, "Theoretical foundations of linear and order statistics combiners for neural pattern classifiers," *IEEE Transactions on Neural Networks*, vol. 7, pp. 1–35, 1996.

[20] Y. Qin, D. Wu, Z. Xu, J. Tian, and Y. Zhang, "Adaptive in-network collaborative caching for enhanced ensemble deep learning at edge," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–14, Article ID 9285802, 2021.