The Design and Implementation of Secure Distributed Image Classification Model Training System for Heterogenous Edge Computing

Cong Cheng[1], Huan Dai[2(✉)], Lingzhi Li[1(✉)], Jin Wang[1], and Fei Gu[1]

[1] School of Computer Science and Technology, Soochow University, Suzhou 215006, China
lilingzhi@suda.edu.cn
[2] School of Electronics and Information Engineering, Suzhou University of Science and Technology, Suzhou 215009, China
daihuanjob@163.com

Abstract. Deep learning provides many new and efficient solutions for edge computing. We study training image classification models on edge devices in this paper. Although there have been many researches on deep learning in edge computing. Most of them did not consider the impact of the limited service capabilities of edge devices, the problem of straggler and insecurity of training data on the system. We design a new distributed computing system to train image classification models on edge devices. To be more specific, we vectorize the convolutional neural network (CNN) to transform it to a lot of matrix multiplications. These matrix multiplications can be arbitrarily cut into many smaller matrix multiplications suitable for computing on edge devices. Besides, our system utilizes codes to ensure the stability and security of distributed matrix multiplications on edge devices. In the performance evaluation, we test the performance of matrix multiplications and a CNN model training in our system with uncoded and coded strategies. The evaluation results show that the system with code strategies perform better than with uncoded strategies on the edge devices having the problem of straggler. In summary, we design a secure distributed image classification model training system for heterogenous edge computing.

Keywords: Edge devices · Distributed computing · Deep learning

This work was supported in part by the National Natural Science Foundation of China (62072321, 61672370, 61702354), "Six Talent Peak Project" of Jiangsu Province (XYDXX-084), CERNET Innovation Project (NGII20190314), China Postdoctoral Science Foundation (2020M671597), Jiangsu Postdoctoral Research Foundation (2020Z100) Scientific Research Project of Suzhou University of Science and Technology (XKZ2017004), Postgraduate Research and Practice Innovation Program of Jiangsu Province (SJCX19_0801), Tang Scholar of Soochow University and the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 349, pp. 179–198, 2021.
https://doi.org/10.1007/978-3-030-67537-0_12

1 Introduction

In recent years, edge computing is developing steadily and getting popular. The computing systems and applications based on edge computing are emerging in endlessly. They have provided lots of solutions for many practical problems. For instance, edge computing has outstanding performance in smart health [1], transportation system [6] and object detection [19]. Meanwhile, edge computing is superior to traditional cloud computing in four aspects: latency, data security, scalability and reliability [5]. Computing at the edge instead of computing on the cloud, which has lower latency and facilitates the protection of data security. Hence, we decide to train the image classification model on edge devices.

As well as edge computing, deep learning is booming. Nowadays, efficient image processing technologies are based on deep learning. And it has wide applications in practice. For example, deep learning helps analyze medical images [14,17], construct image [2,3,9,10,16] and classify image [8]. We train the image classification model using convolutional neural networks (CNNs) on edge devices. Because CNNs are important and popular networks of deep learning. They are mainly composed of the convolutional layer, pooling layer and fully connected layer. And convolutional layer and fully connected layer are primary computing layers in CNNs [4].

In general, the edge devices are responsible for collecting data and the cloud utilizes it to train the deep learning model as we can see in Fig. 1. However, this scheme has several drawbacks. First, all the collected data needs to be transmitted to the cloud through the network, which requires a large amount of bandwidth resources. Second, once the cloud is attacked, the whole system will not work well because the architecture of it is cloud-centric. Third, the security of data is easily threatened in the process of uploading to the cloud. Finally, the computing resources of cloud are expensive. In contrast, training CNN models on edge devices has the following advantages: low latency, cheaper computing resources. To sum up, it is necessary to train the deep learning models on edge devices.

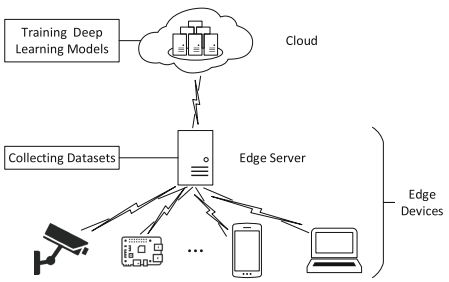


Fig. 1. The edge devices are responsible for collecting datasets and the cloud is responsible for training deep learning models.

However, training the deep learning model on the edge devices has several challenges [5]. First, communication latency may affect the time it takes for the master node to transmit data to the worker nodes. Second, network bandwidth determines the amount of data transmitted from the mater node to the worker nodes per unit time. Third, the compute capability of the worker nodes may affects the computing power of the entire computing system. The computing power of the edge devices is weaker than the cloud. And training a CNN model in a single edge device is scarcely possible. Therefore, we have to use distributed computing method to combine multiple edge devices. Data parallelism and model parallelism are two existed methods of distributed deep learning. Data parallelism is to copy the untrained model to many computing equipment so that they have the same untrained model, and then divide the dataset to them for parallel training. So data parallelism can accelerate the training speed of a CNN model because the dataset is consumed in parallel. However, data parallelism requires the subordinate computing equipment to be able to independently train the entire model. Obviously, ordinary edge devices cannot meet this requirement. Model parallelism is the process of cutting huge models into different parts and deploying them to many computing equipment so that a huge model which cannot be trained on a single computer can be trained in the distributed environment. But the problem of straggler easier occurs to edge devices, which makes the model training using model parallelism method failure. Hence, the two distributed deep learning methods above are not suitable for edge devices. So, we have to design a new distributed computing method to train the CNN models on edge devices.

We transform the computing load in convolutional and fully connected layers to many subtasks, which is equivalent to divide the big task that training a CNN model to lots of subtasks. The convolution transforming to matrix multiplication mentioned in [4,18] can help vectorize CNN. The computing load of training a CNN model transforms to lots of matrix multiplications with it. Then these matrix multiplications can be cut arbitrarily to fit the operation on an edge device. Hence, we can train the CNN models on edge devices.

Nevertheless, distributed computing on the edge devices has a few drawbacks need to be an addressed. The problem of straggler and data security problem are the main issues of distributed computing on edge devices. The edge worker devices distributed computing tasks may not timely return all computed results due to the problem of straggler. And the security of computing data could be threatened when there are edge worker devices being attacked. So we utilize the redundancy coding strategy to handle these problems. After the computing tasks which are matrix multiplications have been coded, we get redundancy of the tasks. Then the redundant computing tasks are assigned to edge worker nodes to deal with the negative impact of the straggler problem. Meanwhile, the details of them are protected because the computing tasks are coded.

Overall, the contributions of this paper are as follows:

- We find a new CNN distributed training method, which is more suitable for running on edge devices with limited service capabilities and unstable states than data parallel and model parallel training methods.
- In this paper, we design and implement a distributed computing system running on edge servers and edge work nodes. It effectively combines and utilizes various computing resources of the edge, which further enhances the computing power of the edge.
- In addition, we apply coded strategies in our distributed computing system. Then we conduct comparative experiments on edge devices simulated by distributed processes with different latencies. The experimental results show that the distributed computing system using coded strategies can not only protect the computing data, but also deal with the negative impact caused by the problem of stragglers.

2 Related Work

2.1 Machine Learning for Edge Computing

There have been many researches on the cooperation between edge computing and machine learning. [21] proposed a scheme for distributed training of machine learning models based on gradient descent on the multiple edge nodes. The training data is collected and stored by the edge node and used for the training of the machine learning model on the edge node instead of being sent to the central location for training the model, which saves network bandwidth. However, [21] does not consider the impact of the security of training data and the limited service capabilities of edge nodes on the system. [19] designed a distributed and efficient target detection system in which end devices, edge servers and cloud cooperate with each other and have a clear division of labor. The end devices are responsible for data collection, compression and transmission to edge server, where the edge servers train the local model, and the cloud server is responsible for aggregating the global model and updating the model on the edge servers. This division of labor makes the entire system more efficient. But, the security of training data is also not taken into consideration in [19]. [13] proposed a deployment strategy of deploying the lower layers of CNN on the edge server and the higher layers on the cloud. Compared with deploying all layers of CNN in the cloud, deploying the lower layers of CNN on edge servers close to the data source can save network bandwidth and ease the computing pressure of the cloud. However, it only uses the computing resources of the edge server, and does not make full use of the computing resources of the edge worker nodes under the edge server. In addition, the security of training data has not been taken seriously.

2.2 Coded Distributed Matrix Computing

There are a lot of matrix computing in computer applications. Coded distributed matrix convolution is studied in [7], which uses the MDS coding strategy to complete the matrix convolution within the deadline on the worker nodes with the

problem of stragglers. However, the limited number of worker nodes is not considered in [7], so it may achieve its theoretical effects in practical applications. Coded distributed matrix multiplication is also studied in [15], which compares the performance of the uncoded, task replication, and coded strategies in a distributed computing environment with the problem of straggler. The experimental results show that coded strategies are better than other strategies under the same experimental environment, and the rateless coded strategy is superior than the MDS code strategy in terms of experimental performance and time complexity. On this basis, we apply the rateless coding strategy and MDS coding strategy to our CNN distributed training system instead of simple matrix multiplication.

2.3 Coded Distributed Machine Learning

The distributed machine learning is already a mature technology. In order to deal with the problem of straggler, code strategies are introduced in distributed machine learning. The linear regression algorithm based on coding distributed computing is studied in [12], which can therefore run well on the distributed computing system with the problem of straggler. However, the linear regression algorithm is just a simple algorithm in machine learning. For deep learning algorithms with non-linear operations like CNN, training methods in [12] cannot work. In this paper, we design a new distributed training method that can be used for CNN model training to provide the problem a solution.

3 Design of the Secure Distributed Image Classification Models Training System Based on Coding

In order to train CNN models on edge devices, we design a secure distributed training system for an image classification model that can run on edge devices. Firstly, we introduce the system framework and show the components of the system and their general functions from a macro perspective. Secondly, we introduce the vectorization of CNN, which converts the complex computation of CNN into matrix multiplication so that we can perform the coded distributed matrix multiplication on this basis. Thirdly, we introduce several distributed computing strategies which are applied to distributed computing systems and help deal with the problem of straggler. Finally, we introduce the design of an edge distributed computing system which is an important part of our system.

3.1 System Framework

From the Fig. 2, we can see the detailed framework of the secure distributed image classification models training system. It consists of an edge server and a number of edge worker nodes. In the left part of the figure is the network topology of our system that every edge worker node independently communicates with the edge server. Specifically, the edge worker nodes obtain the task from the edge server and then return the task result to the edge server in this network topology. In the right part of the figure, we can see what the edge server is responsible for and will be introduced in detail next.

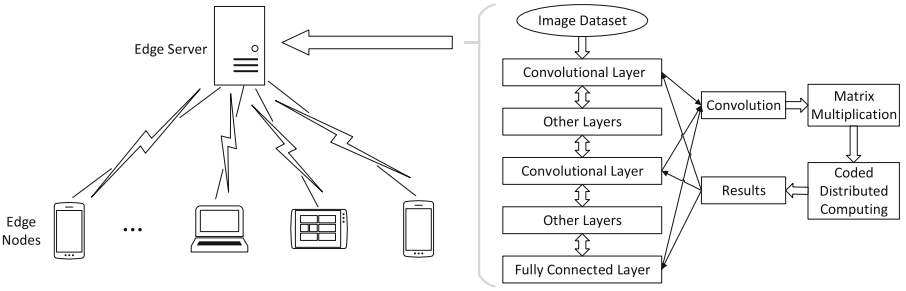


Fig. 2. The framework of the secure distributed image classification models training system.

Edge Server. The edge server maintains a CNN model on it. So it is responsible for updating the parameters of convolutional and fully connected layers. In addition to this, the edge server also do jobs of making coded computing tasks based on matrix multiplication and integrating the returned task results. The method of convolution transforming to matrix multiplication which is introduced in 3.2 helps edge server extract computing tasks from the CNNs. Then, these computing tasks are distributed to the edge worker nodes. Finally, the results computed by edge worker nodes are collected by the edge server and then a round of coded distributed computing ends.

Edge Worker Nodes. There are many kinds of edge worker nodes, such as raspberry pies and smart phones. Nevertheless, they all have the ability of communicating with the edge server and have a few computing power. Edge worker nodes constitute the basis of edge computing.

3.2 Vectorization of Convolutional Neural Network.

Although CNNs have different layers, the computational load is mainly concentrated in the convolutional layer and the fully connected layer. And the total computational load of convolutional layers and fully connected layers is nearly equal to the computational load of CNN. For example, the VGGNet which is a classical CNN. In the Table 1 of [20], we get the configurations of VGGNet. As can be seen from the configurations, the parameters of the CNN are concentrated in the convolutional layer and the fully connected layer. So the entire computational load of VGGNet is composed of the computational load of the convolutional layer and fully connected layer obviously. Meanwhile, the distributed training of the CNN can be transformed into the distributed execution of the computational load of the convolutional layer and the fully connected layer.

Our system is based on the vectorized CNN and vectorization has been proved to accelerate the speed of training CNN models [4]. As mentioned above, the convolutional layer and the fully connected layer are the main computational load layers in CNNs and the parameters and bias that need to be updated only

exist in them. Therefore, the vectorization of the convolutional neural network is the vectorization of the convolutional layer and the fully connected layer.

As shown in Fig. 3, there are forward and backward propagation of CNN on the edge server. In the forward propagation, the input image is processed by the convolutional layer and the fully connected layer to obtain the predicted value. The deviation between the predicted value of the image and the label value becomes the product of the forward propagation i.e. the gradient. In the backward propagation, the gradient is passed back to update the parameters in CNN.

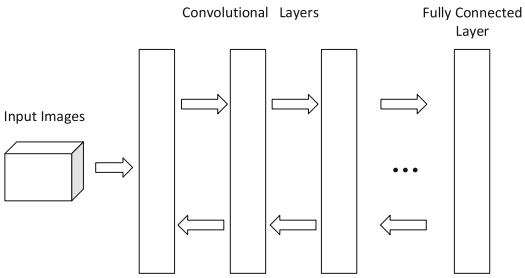


Fig. 3. Forward and backward propagation of CNN on the edge server.

The forward and backward propagation in a convolutional layer can be converted to the forward and backward propagation in a fully connected layer by the method of convolution transforming to matrix multiplication. Next we will introduce the vectorization of forward propagation and the backward propagation in CNNs, i.e. the vectorization of the forward propagation and the backward propagation in the fully connected layer.

Convolution Transforms into Matrix Multiplication Scheme. It is hard to implement a distributed CNNs training system if we do not transform the convolution. Then, a method in [4] can transform convolution to matrix multiplication, which has been proved to speed up image convolution operation in practical application. And it brings us another advantage that we can do distributed computation easier because the image convolution is transformed to matrix multiplication.

An example is shown in Fig. 4, which uses a unique way of matrix unfolding to implement that image convolution turns to matrix multiplication. As we can see, four 2×2 matrices are derived from the 3×3 image matrix after the convolution kernel window scans the image matrix with step size 1. Then, they respectively unfold to a single row and convolution kernel unfold to a single col. Finally, four rows make up a big matrix which multiplies with convolution kernel col. And then the result of the matrix multiplication is converted to the result of the convolution operation.

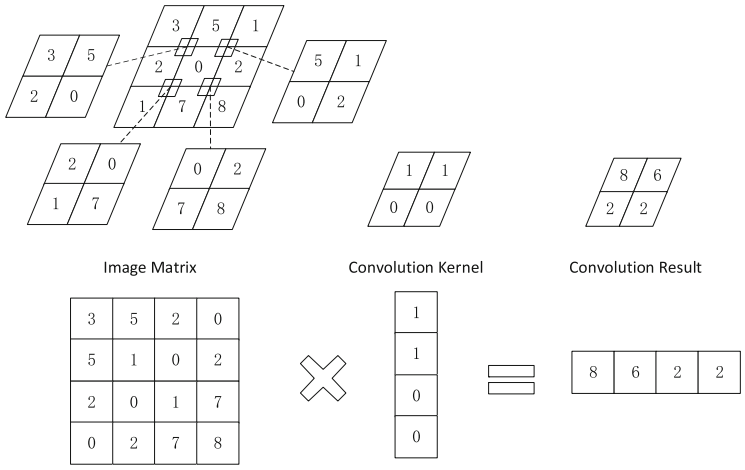


Fig. 4. Example of convolution transforming into matrix multiplication.

Vectorization of Forward and Backward Propagation. In the [4], there is an initial introduction to vectorization of forward propagation and backward propagation, the following is a more detailed vectorization of forward and backward propagation that we have improved on this basis. The forward propagation of the CNNs is to calculate the predicted value of the training samples. The vectorization process of the forward propagation is as follows:

$$Z^{[l]} = A^{[l-1]}.W^{[l]} + b^{[l]}, \tag{1}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}). \tag{2}$$

In Eq. 1, $W^{[l]}$ is the parameter matrix of the lth layer of CNN and $b^{[l]}$ is the bias of the lth layer of CNN. $A^{[l-1]}$ is the output of $l-1$th layer and also the input of lth layer. In Eq. 2, $g^{[l]}$ is the activation function of lth layer.

The backward propagation of the CNNs is to transmit the difference between the predicted value and the real value of the training sample back to convolutional layers and fully connected layers of the CNNs and update the parameters in them. The vectorization process of backward propagation is as follows:

$$\mathrm{d}Z^{[l]} = \mathrm{d}A^{[l]}.[g^{[l]}(Z^{[l]})]',\ \mathrm{d}A^{[l-1]} = \mathrm{d}Z^{[l]}.W^{[l]T}, \tag{3}$$

$$\mathrm{d}W^{[l]} = A^{[l-1]T}.\mathrm{d}Z^{[l]}, \tag{4}$$

$$\mathrm{d}b^{[l]} = sum(\mathrm{d}Z^{[l]}), \tag{5}$$

$$W^{[l]} := W^{[l]} - \alpha \mathrm{d}W^{[l]},\ b^{[l]} := b^{[l]} - \alpha \mathrm{d}b^{[l]}. \tag{6}$$

In the equations above, d[ ] is the gradient of [ ] and [ ]$'$ is the derived function of [ ] and [ ]^T is the transposition of [ ] during backward propagation. In Eq. 5, $\mathrm{d}b^{[l]}$ is the sum of $\mathrm{d}Z^{[l]}$ in rows. In Eq. 6, α is the learning rate of the CNNs.

3.3 Distributed Matrix Multiplication Strategies

The Uncoded Strategy. As a comparison benchmark, we keep the simplest strategy that the uncoded strategy. When do the distributed matrix multiplication $A \times B$ with uncoded strategy, we cut the matrix A evenly along the rows into n matrix blocks $\{a_1, a_2, \ldots, a_n\}$. Then, we distribute n tasks $\{a_1 \times B, a_2 \times B, \ldots, a_n \times B\}$ to p workers. In this strategy, the matrix multiplication $A \times B$ has been completed when the master node received all n computation results.

The r-Replication Strategy. In the r-replication strategy, we do the multiplication of matrix A with matrix B. The matrix A is repeated r times, $A = A_1 = A_2 = \cdots = A_r$. So, the computing task $A \times B$ is also repeated r times. Then we get r computing tasks $\{A_1 \times B, A_2 \times B, \cdots, A_r \times B\}$. In distributed computing, the $A \times B$ task is divided into n subtasks and the total number of subtasks is rn. When the rn subtasks are distributed to worker nodes and at least n results returned or at most $rn - n + 1$ results returned, the result of $A \times B$ can be got.

The *(m,n)* MDS Code Strategy. In the MDS code strategy, we use the Vandermonde matrix to encode the matrixes which come from CNNs. Any n row vectors of the $m \times n$ Vandermonde matrix can form an $n \times n$ invertible matrix. The process of matrix encoding and returned result decoding is as follow:

$$E_{m\times t} = M_{m\times n} \times A_{n\times t}, \tag{7}$$

$$R_{m\times s} = E_{m\times t} \times B_{t\times s}, \tag{8}$$

$$O_{n\times s} = M^{-1}_{n\times n} \times R_{n\times s}. \tag{9}$$

The matrix $M_{m\times n}$ in Eq. 7 is a Vandermmonde matrix, which can encode the matrix $A_{n\times t}$ redundancy to get redundant encoded matrix $E_{m\times t}$. Then the redundant encoded matrix $E_{m\times t}$ multiplied by the matrix $B_{t\times s}$ in the distributed environment. In the equation 9, the matrix $R_{n\times s}$ from the matrix $R_{m\times s}$ can be decoded with the inverse of its encoding matrix to get the result of $A_{n\times s} \times B_{s\times t}$ finally.

The LT-Code Strategy. Luby Transform (LT) code is one of rateless codes that the number of encoded matrix blocks is not fixed and it increasing with the need of decoding end for decoding out the whole result. Comparing with the MDS code, the LT code has simpler encoding way based on $\oplus$. The operation $\oplus$ is exclusive or. In our system, we use addition and subtraction operations instead of the exclusive or operation and they have the same performance in practice.

In the encoding phase, a degree d is selected to represent the number of original matrix blocks participating in the encoding. The d original matrix blocks are selected from n original matrix blocks randomly and the degree $d \in \{1, 2, \ldots, n\}$. The choice of degree d conforms to the probability distribution $\rho(d)$ which is the robust soliton degree distribution and can be found in [15]. The set of the subscripts of d original matrix blocks is S_d and $S_d \subseteq \{1, 2, \ldots, n\}$. The process

Algorithm 1: Matrix Encoding Algorithm Using LT Code

Input:
The original matrix blocks M;
The number of original matrix blocks n_m;
The robust soliton degree distribution $\rho(d)$;
The redundancy ratio r;
Output:
The encoded matrix blocks E;
The number of encoded matrix blocks n_e;

```
n_e = r × n_m;
for i = 0 to n_e do
    d ← get a degree randomly according to ρ(d);
    for j = 1 to d do
        m ← pick an original matrix block randomly from M;
        e = e + m;
    E[i] = e;
return E, n_e.
```

of encoding is shown in Algorithm 1. In the first step of Algorithm 1, we get a degree d according to the robust soliton degree distribution $\rho(d)$. For the original n matrix blocks, the set of values of degree d is $\{1, 2, \ldots, n\}$. Then, we pick d matrix blocks from the original matrix blocks and get the sum of them. The sum of the d matrix blocks is an encoded matrix block. By repeating the above operation, we can theoretically obtain an endless stream of encoded matrix blocks. However, we set a redundant parameter r to determine the number of encoded matrix blocks produced in practice. So, we get rn encoded matrix blocks finally.

In the decoding phase, the decoding starts with blocks of degree 1 and the neighbor blocks of them as we can see in Fig. 5. With the decoding carries on, there are more and more blocks of degree 1. And the blocks of degree 1 are the decoded blocks actually. When all original blocks are decoded, the decoding phase is finished.

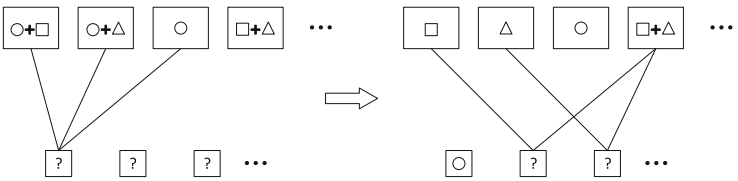


Fig. 5. The LT-Code decoding process.

3.4 The Scheme of Coded Strategies Protecting Data Security

In addition to maintaining the forward and backward propagation of CNNs, the edge server is also responsible for distributing compute tasks to edge worker

nodes and collecting computed results from them. After vectorizing the CNNs, we get lots of large matrix multiplications. In a forward propagation of convolutional and fully connected layers, the image data are transform to matrix A and the weight parameters are transform to matrix B. So, protecting the security of the matrix A is equivalent to protecting the security of the image data. As we can see from Fig. 6, matrix A multiplying with matrix B is done by distributed computing according to our design. Firstly, divide the matrix A into many small matrix blocks evenly and then encode them using coding matrix. The size of the small matrix block is determined according to the size of the largest matrix multiplication that the worst-performing edge worker node can undertake. Secondly, make computing tasks. Thirdly, assign the computing tasks to edge worker nodes. Finally, collect the results returned from edge worker nodes and combine them into the result of $A \times B$.

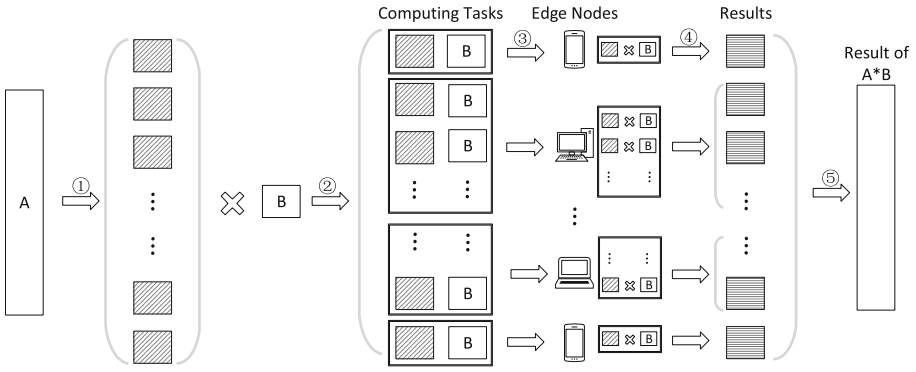


Fig. 6. Make, distribute computing tasks and collect results.

Matrix Redundancy Encoding. The redundant encoding has been proven to perform well in distributed computing [12]. To cope with the problem of stragglers, redundant encoding of the original matrix is inevitable. Because it encodes the original matrix to ensure data security and makes the original matrix redundant at the same time. We use the MDS code strategy as an example. The detailed process is shown in Fig. 7. The Vandermonde matrix can encode the original matrix redundancy in the MDS code strategy, but the encoding vector in the Vandermonde matrix will increase as the size of the encoded matrix increase, which also means that the computational load getting larger. In order to reduce the computational load and low the complexity of encoding, we uniformly partition the original matrix to n smaller blocks at first. Next, the large matrix is encoded with redundancy based on these matrix blocks. The $m \times n(m > n)$ coding matrix is used to encode n small matrix blocks. Finally, we get m encoded matrix blocks.

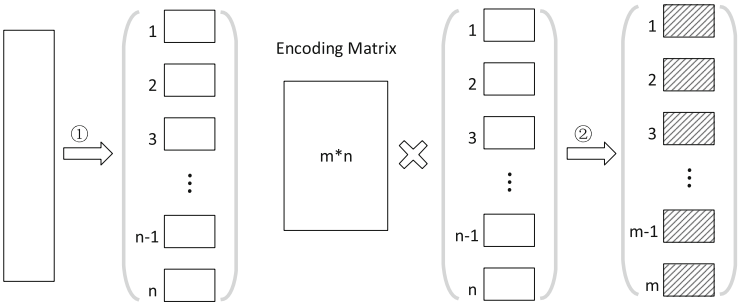


Fig. 7. Divide the big matrix into n blocks, then use the $m \times n(m > n)$ Vandermonde matrix to encode the matrix blocks with redundancy.

Computed Results Decoding. Then the m encoded matrix blocks are distributed to edge worker nodes for computing results. As the edge server has received n computed results from edge worker nodes, the original matrix multiplication result can be decoded out. As we can see in Fig. 8, the decoding matrix is the inverse matrix of the matrix composed of the encoding vectors of n task results. Then, the final result can be obtained when the decoding matrix is left multiplied by the returned computed results.

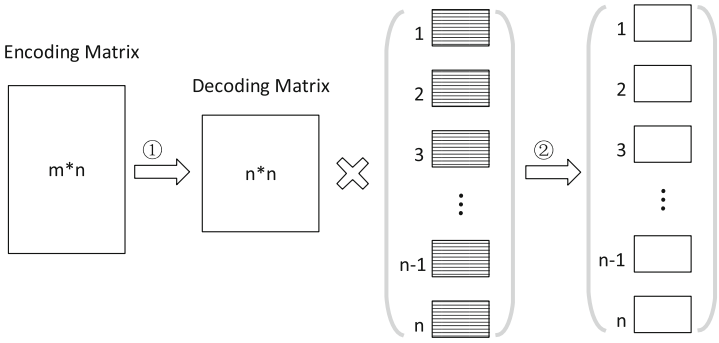


Fig. 8. The process of decoding n result matrix blocks.

Data Security Protecting. In this paper, we protect the security of image data during the transmission from the edge server to the edge worker nodes and when the edge worker nodes perform computing tasks. As we can see in the Fig. 6, the matrix A that needs protection is transformed from the image data. If matrix A is directly transmitted in plaintext from the edge server to the edge worker nodes, the content of matrix A is vulnerable to be leaked during transmission and calculation on the edge worker nodes. Therefore, we use the coding matrix

to encode the matrix blocks of matrix A. After that, the content of the encoded matrix blocks has been protected during transmission and calculation. Even if the attacker collects all the encoded matrix blocks by intercepting the data packets of the transmission or attacking the edge worker nodes, he cannot get the content of matrix A. Because the coding matrix is stored in the edge server and it does not participate in the calculation on the transmission and edge worker nodes. In the end, we protect the security of matrix A, which is to protect the security of image data.

3.5 Design of Edge Distributed Computing System

In our system, we do the matrix multiplications with distributed computing. Hence, the scheme of distributing and recycling computing tasks plays an important role in the secure distributed image classification model training system. Because the edge worker nodes have different performance at work, and it is difficult for an edge server to actively contact all edge worker nodes. How to assign the computing tasks to edge worker nodes determines the performance of the entire system. As we can see in Fig. 9, we create two pools on the edge server: the task pool and the result pool. Due to the two pools, we divide a round of computation into three specific steps. In the first step, every edge worker node randomly gets a task from the task pool. In the second step, after completing the computation of the task, the edge worker node puts the result into the result pool. Finally, the edge server takes all results out from the result pool and integrates them into a final result.

We consider the different performance of the edge worker nodes in the edge distributed computing system. Although these edge worker nodes get computing tasks of the same size meanwhile, they return computed results at different times. And it may lead to the problem of straggler. Previously, we introduced redundant coding of computing tasks to address the problem of straggler in distributed computing. Specifically, the final result can be successfully decoded if first n computed results in $m(m > n)$ computing tasks are returned to the result pool. At the same time, we find that if the return of the first n computed results is accelerated, the time to complete this round of calculation will be shorter. The edge distributed computing system we designed can do this job. It has three advantages:

Advantage 1: The computing tasks are properly assigned. In the edge server, we create two pools: the task pool and the result pool. Hence, the edge worker nodes are not assigned tasks passively, but actively takes them from the task pool. Under this mechanism, the more powerful the edge worker node is, the more computing tasks it can accomplish. From a macro point of view, the edge worker nodes with powerful ability undertake a lot of work, while the edge worker nodes with weak ability undertake little work. This reasonable and healthy way of the division of labor makes our system run efficiently and thus speeds up the return of the first n computed results.

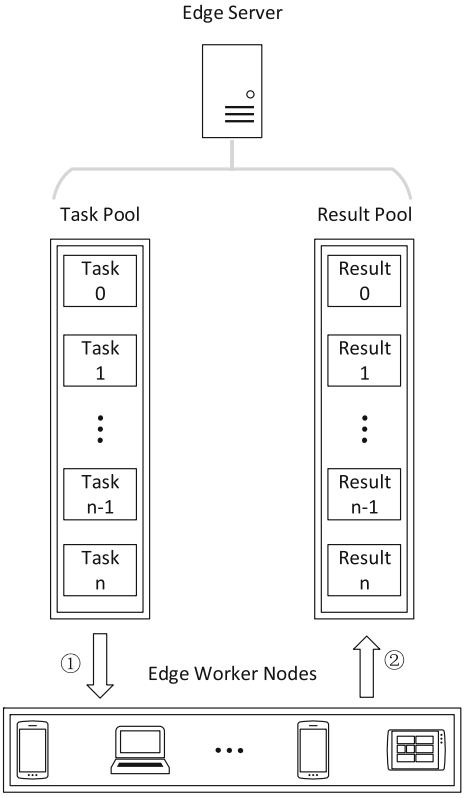


Fig. 9. The edge distributed computing system.

Advantage 2: Easy implementation of the system. The system does not need to set tasks of different sizes for edge worker nodes with different performance. And there is no need to determine which edge worker nodes are the straggler nodes. In the traditional distributed computing system, the edge worker nodes with powerful ability are assigned large computing tasks for reducing the time it takes to complete the all computing tasks. In our system, the edge worker nodes actively fetch tasks from the task pool. So, we just need to partition the original task evenly at the beginning of a distributed computing that the tasks in the task pool are the same size.

Advantage 3: The contribution of straggler edge worker nodes is not wasted. The straggler edge worker nodes take more time to return the computed results than others. However, it does not mean that their work must be abandoned. Especially in our system, the original computing task is divided into many subtasks. During a round of distributed computing, the powerful nodes do lots of tasks and the straggler nodes do few tasks rather than nothing. This design of the edge distributed computing system successfully utilizes the computational

resources existing in the straggler nodes and increases the computing resources of the whole system.

4 Performance Evaluation

In this section, we compare the performance of distributed matrix multiplication strategies in the edge distributed computing system which is running on edge devices.

4.1 Experimental Settings

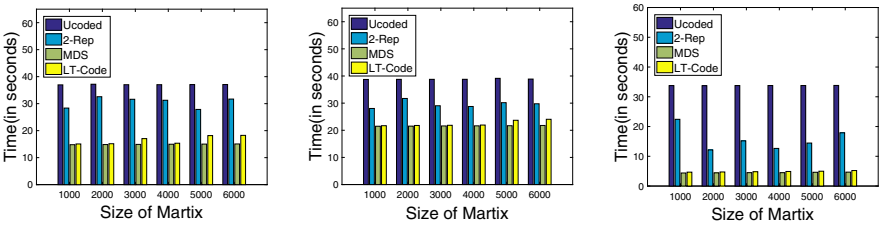
In uncoded distributed matrix multiplication strategies, we have uncoded strategy and 2-replication strategy. Relatively, we have the MDS strategy and the LT-code strategy in coded strategies. Before the experiments, we have set a delay model to simulate the latency on the edge worker nodes during the distributed computing. In this paper, we use the distributed processes to simulate the edge worker nodes. In order to simulate the problem of stragglers of edge worker nodes, we add latency T to the processes using a $Sleep()$ function. The latency T is a set $\{t_1, t_2, \ldots, t_n\}$ relative to n distributed processes. Then, we make the latency T sequentially complies with the uniform, normal and exponential distribution to get three different kinds of latency sets. For testing the performance of the edge distributed computing system, we perform the coded distributed matrix multiplication experiments on it in the first place. Finally, we conduct a distributed training experiment of an image classification model on it to evaluate the performance of our secure distributed image classification model training system.

4.2 Coded Distributed Matrix Multiplication Experiments

In the first place, we need to evaluate the performance of the secure edge distributed computing system above. Uncoded schemes and coded schemes in the system are compared. Moreover, the performance of different coded schemes are also compared. Then, we make the computing tasks for the evaluation of the system and design serval experimental schemes of matrix multiplication. Before each round of distributed computations, we split the matrix A into ten blocks $\{a_1, a_2, \ldots, a_{10}\}$ along the horizontal direction. So, we get 10 original tasks $\{a_1 \times B, a_2 \times B, \ldots, a_{10} \times B\}$.

The 10 original tasks were processed by uncoded, 2-replication, MDS and LT-code strategies. Then, we put them in the task pool. In the scheme A, we test the computing time required for computing tasks of different sizes using uncoded, 2-Replication, MDS code, and LT-code strategies on the ten distributed processes. And, the latency set on the distributed processes obeys uniform, normal or exponential distribution. In the scheme B, we test the performance of the system completing a same computing task when the latency sets that obey uniform, normal or exponential distribution.

Scheme A: We consider different sizes of matrix multiplication computing on the 10 distributed processes which have latencies that obey uniform, normal or exponential distributions. The 6 kinds of matrix multiplications are from 1000×1000 matrix A multiplying with 1000×10 matrix B to 6000×1000 matrix A multiplying with 1000×10 matrix B. So we have six sizes of computing tasks carrying on the 10 distributed processes with 3 kinds of distributed latencies.



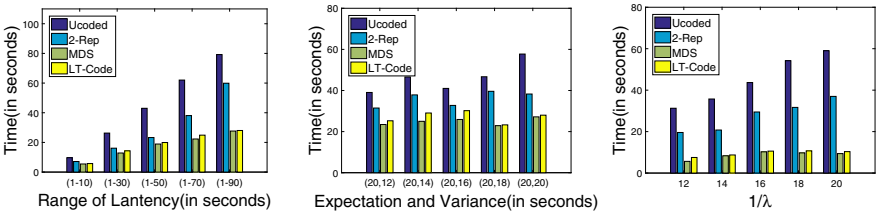
(a) Uniform distribution. (b) Normal distribution. (c) Exponential distribution.

Fig. 10. The matrix multiplications of different sizes carrying on the 10 distributed processes with the latencies that obey the uniform, exponential and normal distribution.

As we can see in Fig. 10(a), Fig. 10(b) and Fig. 10(c), the system with the coded distributed matrix multiplication strategies takes significantly less time to complete the six computing tasks of different sizes than the system with uncoded and 2-Replication strategies whether the latency on the 10 distributed processes obeys uniform, normal or exponential distribution. Among coded strategies, the performance of MDS strategy and LT-code strategy are close. We can also see that the 2-Replication strategy outperforms the uncoded strategy in our experiments. The experiments of the scheme A show that in the distributed computing environment with the problem of straggler, the 2-Replication strategy with backup computing tasks can alleviate the problem of straggler, but the actual performance is inferior to coded strategies.

Scheme B: We consider the multiplication of a 6000×1000 matrix A with 1000×10 matrix B computing on the 10 distributed processes with different latencies. The ranges of values of the latencies that obey uniform distribution are from range between 1 and 10 seconds to range between 1 and 90 seconds. The normal distribution latencies have same expectation 20 and different variances from 12 to 20. And the exponential distribution latencies have different expectations from 12 to 20.

Figure 11(a) shows that the edge distributed computing system adopting different distributed computing strategies takes different times to complete a same computing task when the latencies of 5 ranges on 10 distributed processes obey uniform distribution. From the Fig. 11(a), we can see that coded schemes



(a) Uniform distribution. (b) Normal distribution. (c) Exponential distribution.

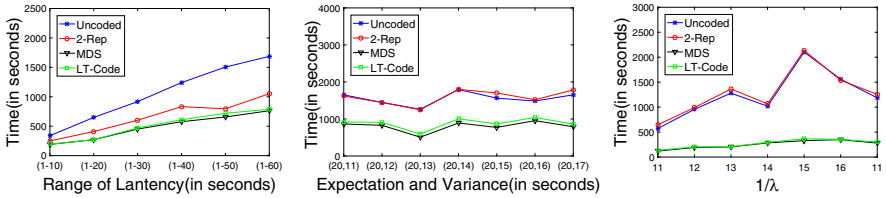
Fig. 11. The multiplication of 6000×1000 matrix A with 1000×10 matrix B carrying on the 10 distributed processes with the latencies that obey the uniform, exponential and normal distribution and have different ranges, different variances and different expectations.

are obviously superior to the uncoded schemes in time spent no matter which range of latencies is set on the distributed processes. And with the range of latencies increases, the performance gap between uncoded and coded schemes is widening, which shows the stability of coded schemes from another point of view. In the Fig. 11(b), the latencies we set on the distributed processes follow normal distribution. And these latency sets have same expectation and different variances. In this condition, we still find that the coded schemes are more excellent than the uncoded ones. In the Fig. 11(c), the latencies on the distributed processes obey exponential distribution. The latencies have different expectations from 12 to 20. As the expectation of the latencies increases, the time taken by the uncoded schemes to complete a same computing task increases linearly. The difference is that the coded schemes are less affected by rising expectations of the latencies.

4.3 Secure Distributed Image Classification Model Training Experiments

We train the vectorized CNN model in the edge distributed computing system and use uncoded schemes and coded schemes respectively. As alluded to above, the coded schemes can deal with the problem of stragglers of edge worker nodes and protect the security of the dataset. For evaluating the performance of our system ,we consider using the MNIST dataset [11] to train a VGGNet [20] model. Besides, we set three kinds of latency sets on the ten distributed processes. Three kinds of latencies respectively obey uniform, normal and exponential distribution. When model training runs out of a batch of images which are 50 images, we record the time spent by our system.

In the Fig. 12(a), the latency sets on the 10 distributed processes obey normal distribution. Whether they are uncoded schemes or coded schemes, the time spent by the system increases as the latency range increases as we can see from the Fig. 12(a). The coded schemes still can bring less time cost to our system. In the coded schemes, MDS strategy and LT-code strategy have comparable per-



(a) Uniform distribution. (b) Normal distribution. (c) Exponential distribution.

Fig. 12. Training VGGNet using the MNIST dataset on the 10 distributed processes with the latencies that obey the uniform, exponential and normal distribution and have different ranges, different variances and different expectations.

formance. Then, the coded schemes also have better performance than uncoded schemes in Fig. 12(b) where the latency sets obey normal distribution. In the Fig. 12(c), the latency sets on 10 distributed processes obey exponential distribution. With the increasing of expectation, coded schemes perform more stable than uncoded schemes. The MDS strategy and LT-code strategy bring our system less time cost than uncoded strategy and 2-replication strategy.

The coded schemes not only spends less time in distributed model training than the uncoded schemes, but also improves the stability of the system. As the range of latency increases, the time spent in the coding schemes increase more slowly than the time spent in the uncoded schemes in Fig. 12(a). Similarly, we can also see from Fig. 12(c) that as the expectation of the latency increases, the time spent curve of the coded schemes fluctuates less and is smoother than that of the uncoded schemes. In summary, the coded schemes can improve the stability of the system when the image classification model is trained on heterogeneous edge worker nodes with different latencies.

5 Conclusion

In this paper, we designed a secure distributed image classification models training system for heterogenous edge computing. The system can run on edge devices and can deal with the problem of straggler of edge worker nodes. Besides, it has the ability to protect the privacy of computing data because our system adopts coded strategies. It brings two advantages. First, we use many idle edge devices to perform task computing to reduce the cost of the system. Second, we have implemented the secure training of CNN models on edge devices. The datasets collected at the edge no longer need to be uploaded to the cloud, which avoids the security problem during transmission and saves the transmission bandwidth. After the experiment, we demonstrated that the system in this paper can deal with the problem of straggler and can run smoothly on edge devices.

References

1. Abdellatif, A.A., Mohamed, A., Chiasserini, C.F., Tlili, M., Erbad, A.: Edge computing for smart health: context-aware approaches, opportunities, and challenges. IEEE Netw. **33**(3), 196–203 (2019)
2. Bahrami, K., Shi, F., Rekik, I., Shen, D.: Convolutional neural network for reconstruction of 7T-like images from 3T MRI using appearance and anatomical features. In: Carneiro, G., et al. (eds.) LABELS/DLMIA -2016. LNCS, vol. 10008, pp. 39–47. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46976-8_5
3. Benou, A., Veksler, R., Friedman, A., Riklin Raviv, T.: De-noising of contrast-enhanced MRI sequences by an ensemble of expert deep neural networks. In: Carneiro, G., et al. (eds.) LABELS/DLMIA -2016. LNCS, vol. 10008, pp. 95–110. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46976-8_11
4. Chellapilla, K., Puri, S., Simard, P.: High performance convolutional neural networks for document processing (2006)
5. Chen, J., Ran, X.: Deep learning with edge computing: a review. Proc. IEEE **107**(8), 1655–1674 (2019)
6. Chen, Y., Zhang, Y., Maharjan, S., Alam, M., Wu, T.: Deep learning for secure mobile edge computing in cyber-physical transportation systems. IEEE Netw. **33**(4), 36–41 (2019)
7. Dutta, S., Cadambe, V., Grover, P.: Coded convolution for parallel and distributed computing within a deadline. In: IEEE International Symposium on Information Theory (ISIT), pp. 2403–2407. IEEE (2017)
8. Ghafoorian, M., et al.: Deep multi-scale location-aware 3D convolutional neural networks for automated detection of lacunes of presumed vascular origin. NeuroImage Clin. **14**, 391–399 (2017)
9. Golkov, V., et al.: q-space deep learning: twelve-fold shorter and model-free diffusion MRI scans. IEEE Trans. Med. Imaging **35**(5), 1344–1351 (2016)
10. Hoffmann, N., Koch, E., Steiner, G., Petersohn, U., Kirsch, M.: Learning thermal process representations for intraoperative analysis of cortical perfusion during ischemic strokes. In: Carneiro, G., et al. (eds.) LABELS/DLMIA -2016. LNCS, vol. 10008, pp. 152–160. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46976-8_16
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
12. Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., Ramchandran, K.: Speeding up distributed machine learning using codes. IEEE Trans. Inf. Theory **64**(3), 1514–1529 (2017)
13. Li, H., Ota, K., Dong, M.: Learning IoT in edge: deep learning for the internet of things with edge computing. IEEE Network **32**(1), 96–101 (2018)
14. Litjens, G., et al.: A survey on deep learning in medical image analysis. Med. Image Anal. **42**, 60–88 (2017)
15. Mallick, A., Chaudhari, M., Joshi, G.: Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. arXiv preprint arXiv:1804.10331 (2018)
16. Nie, D., Cao, X., Gao, Y., Wang, L., Shen, D.: Estimating CT image from MRI data using 3D fully convolutional networks. In: Carneiro, G., et al. (eds.) LABELS/DLMIA -2016. LNCS, vol. 10008, pp. 170–178. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46976-8_18

17. Pouyanfar, S., et al.: A survey on deep learning: algorithms, techniques, and applications. ACM Comput. Surv. (CSUR) **51**(5), 92 (2019)
18. Ren, J.S., Xu, L.: On vectorization of deep convolutional neural networks for vision tasks. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
19. Ren, J., Guo, Y., Zhang, D., Liu, Q., Zhang, Y.: Distributed and efficient object detection in edge computing: challenges and solutions. IEEE Network **32**(6), 137–143 (2018)
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Computer Science (2014)
21. Wang, S., et al.: When edge meets learning: adaptive control for resource-constrained distributed machine learning. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 63–71. IEEE (2018)