Computer Communications 256 (2026) 108574

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A cloud–edge collaborative incremental learning architecture with adaptive parameter freezing based on knowledge distillation

Jiaqi Fei [a], Mengdie Qin [a], Xiaogang Wang [a,*], Hui Guo [a], Ziqi Zhu [a], Jian Cao [b], Rajkumar Buyya [c]

[a] School of Electronic and Information, Shanghai Dianji University, Pudong New Area District, Shanghai, China
[b] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Minghang District, Shanghai, China
[c] Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, Melbourne, Australia

ARTICLE INFO

Keywords:
Distributed computing
Cloud–edge collaboration
Knowledge distillation
Adaptive parameter freezing
Deep neural network splitting

ABSTRACT

The continuous improvement in model structure of deep neural networks enables more refined and efficient models to perform runtime inference tasks in resource-constrained edge scenarios. However, before the models are deployed to edge sides, large-scale pre-training tasks still need to be undertaken by cloud sides. The current compromise is to adopt the cloud–edge collaborative inference method that training large models is in cloud sides, while model inference is deployed in edge sides by using the pre-trained parameters. This makes edge sides rarely undergo incremental parameter updates in time, causing a decline in the inference performance of edge sides. Toward this end, we propose a novel cloud–edge collaborative incremental learning architecture with the adaptive frozen layer knowledge distillation (AFLKD), i.e., a knowledge distillation-based training method for edge networks which enables edge devices to perform fast and efficient inference tasks while ensuring reliable accuracy. Firstly, a robust deep neural network model is pre-trained on the cloud side. Subsequently, the principle of knowledge distillation is employed to treat the cloud model as a teacher model so as to assist the training of the edge (student) model, thereby ensuring the performance of the edge model. Finally, to reduce training consumption while ensuring training effectiveness, the adaptive layer freezing strategy and the feature map splitting parallel inference are devised. During the training, we share some shallow layers from the cloud network with the edge side, and the number of frozen layers is adaptively adjusted based on changes in the distillation temperature to achieve better incremental training results. Training experiments conducted on datasets of different types and sizes show that the proposed strategy achieves accuracy improvements of 1.3%, 1.9%, and 6.7% using the CIFAR10, FASHION-MNIST, and CIFAR100 datasets, respectively, compared to current related methods. In addition, compared with the non-split case, the inference time is reduced by 32.11% through splitting the feature maps.

1. Introduction

In recent years, deep neural networks (DNNs) are widely employed in various industries and research fields. Network models have been constantly streamlined and their performance gradually improved. The ongoing refinement of these models has made it possible to deploy DNNs for high-speed inference tasks on user terminal devices [1]. However, deploying DNN models on these edge devices inevitably brings some challenges. For instance, the resources of edge devices are very limited, making it difficult to deploy a DNN model with reliable inference accuracy and less inference time by using the available resources. In cloud computing architecture [2,3], users can deploy large and complex DNN models on resource-rich cloud servers, input data is collected from end sensor devices and transmitted to the cloud server's networks for inference, finally, end user devices receive the results from the cloud servers. Despite addressing the resource limitations of end devices, the data transmission between end devices and cloud servers incurs significant time overhead and communication costs. Latterly, the development of edge computing provides a solution to these challenges. Edge computing significantly reduces latency and improves communication quality by deploying computational resources and services on edge nodes close to the users.

* Corresponding author.
E-mail addresses: feijq@st.sdju.edu.cn (J. Fei), qinmd@st.sdju.edu.cn (M. Qin), wangxg@sdju.edu.cn (X. Wang), guoh@st.sdju.edu.cn (H. Guo), zhuzq@st.sdju.edu.cn (Z. Zhu), cao-jian@cs.sjtu.edu.cn (J. Cao), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.comcom.2026.108574
Received 23 November 2025; Received in revised form 23 March 2026; Accepted 22 May 2026
Available online 27 May 2026

Nevertheless, the introduction of complex models such as current large language models [4] has greatly increased the parameters of cloud-deployed models, making it impractical to deploy cloud models on edge nodes. To address the issue of overly large models, layer [5] or semantic partitioning (splitting) [6] methods are typically employed. Both methods have their own characteristics, layer splitting ensures inference accuracy and distributes resource consumption across nodes, but it does not enhance inference speed, while semantic partitioning improves inference speed but at the cost of accuracy. Additionally, these two methods focus on the decomposition of the network structures, and do not establish strong interaction with cloud and user edge devices.

A good compromise is to adopt the inference method of cloud–edge collaboration, which can solve the above problems well. Ding et al. [7] propose a cloud–edge collaborative framework that integrates cloud, edge and mobile devices. This framework uses the cloud to assist in training and updating the edge network by newly generated data from the user terminals, providing users with improved services on mobile devices through deploying the updated high-performance models. However, during the training phase of the edge network, this framework opts for freezing certain shallow model layers and fine-tuning a few deep model layers, which shortens the response time but results in mediocre performance for the edge model in some complex tasks, failing to meet the accuracy demands of inference tasks.

In response to the above problems, this paper proposes a new cloud–edge collaborative incremental learning architecture with the adaptive frozen layer knowledge distillation (AFLKD) that employs the knowledge distillation method, treating the cloud network as a teacher model while the edge network as a student model. Under the premise of freezing the shallow network layers, more subsequent network layers are subjected to distillation learning. Due to the effectiveness of knowledge distillation in model compression and small-sample training [8], the teacher (cloud) network can fully transfer its knowledge to the student (edge) network, enabling small-sample training and update at the edge network to ensure good performance even in complex inference tasks. Additionally, this method also considers the number of layers to freeze. Different numbers of frozen layers result in varying effects during training. The optimal number of frozen layers is adaptively inferred based on the role of temperature in knowledge distillation and its practical significance. To accelerate the inference time, we also use feature map splitting to release the computation on the edge network, thereby reducing resource pressure while maintaining inference accuracy.

In conclusion, this paper makes the following contributions.

- A fresh cloud–edge collaborative incremental learning architecture based on knowledge distillation is put forward. Therein, the cloud and edge network models are respectively treated as the teacher and student network models, and the incremental training runs on edge nodes. This approach allows the cloud network to assist in training the edge network, leading to improved model accuracy in scenarios with small samples.
- An adaptive frozen layer knowledge distillation (AFLKD) method is designed to freeze some DNN layers, so as to reduce the computational pressure on edge devices during the incremental training process. The number of frozen layers is not fixed but adjusted based on the analogous temperature to achieve adaptive adjustment of the frozen DNN layers. Through this method, the accuracy of network model can be automatically optimized more effectively during training.
- An efficient method involving the feature map splitting is employed to partition the DNN network and deployed on different edge devices for the parallel inference. This approach effectively reduces resource consumption on edge devices, alleviates operational workloads of devices, and accelerates inference speed. Training and inference experiments on different types and sizes of datasets validate that the proposed method has better accuracy and parallel processing efficiency.

The rest of the paper is organized as follows. Section 2 describes the related work. In Section 3, the basic cloud–edge collaboration architecture and formula are constructed. Section 4 elaborates the incremental update principle and corresponding algorithms of the proposed method. Section 5 gives a parallel accelerated inference method based on the feature map splitting idea. In Section 6, training and inference experiments are illustrated. Finally, Section 7 draws conclusion and gives future work.

2. Related work

2.1. Cloud computing in DNNs

In cloud computing, Alizadeh et al. [9] propose a resource allocation autonomous system based on a clipped double deep Q-learning (CDDQL) algorithm and meta-heuristic particle swarm optimization (PSO), which achieves significant improvements in response time, task completion, resource utilization and energy consumption. Xu et al. [10] present an efficient supervised learning-based deep neural network (es-DNN) method for cloud workload prediction. This method uses a sliding window to convert multivariate data into supervised learning time series, and employs Gated Recurrent Units (GRUs) for accurate predictions. After utilizing various methods to optimize the performance and efficiency of cloud computing, its application in industrial production has become widespread. For example, in defect detection [11–13], cloud computing has been used to reduce inspection time and improve detection accuracy. Cloud computing also typically focuses on resource scheduling issues [14,15]. Proper resource scheduling can decrease resource pressure throughout the training process. Although recent optimizations in cloud computing have led to excellent performance in terms of accuracy and energy consumption, the issue of transmission latency inherent in cloud computing has not been adequately resolved.

2.2. Edge computing and model partitioning in DNNs

In edge computing, Alnemari et al. [16] adopt filter pruning and tensor decomposition methods to trim networks, enabling them to run on resource-constrained edge devices while achieving satisfactory accuracy. Mills et al. [17] propose a Multi-Task Federated Learning (MTFL) algorithm that introduces non-collaborative Batch Normalization (BN) layers into joint DNNs. This approach allows users to train personalized models based on their own data, improving training convergence speed and efficiency. With the continuous development of edge computing, the optimization of inference latency has become significant, and can meet most real-world production detection. As a result, its application in smart factories is widespread [18–20]. Although recent research in edge computing has achieved desirable accuracy, and leveraged the proximity of edge computing to terminals so as to significantly reduce transmission time, once neural networks are deployed at edge sides, they are often difficult to update parameters. This can result in suboptimal performance when handling different types of tasks.

Model partitioning, as a method to accelerate training and inference, is also frequently used in cloud–edge collaborative architectures [21,22]. To address the issue of limited resources at the edge, Liang et al. [5] adopt a layer partition method that can split sequential structures or graph topology networks layer by layer, deploying them separately on cloud and edge sides. This method improves the inference speed by optimizing both transmission time and inference time. Zeng et al. [6] employ an input partition method, which, in cases of large training datasets, splits the input image data and deploys it across different edge nodes. The convolution operations are completed through communication between the nodes, and finally, the results are aggregated at a central node for classification output.

2.3. Knowledge distillation

Knowledge Distillation (KD) has emerged as a pivotal technology for balancing model accuracy and inference efficiency within cloud–edge collaborative architectures. Tao et al. [23] propose a Neuron Manifold Distillation (NMD) method, which guides student models to mimic the output distribution of teacher networks and capture underlying feature geometries, significantly reducing the computational and storage overhead of deploying large-scale models on resource-constrained devices. Addressing the high sensitivity of Multi-Object Tracking (MOT) tasks to foreground information, Liang et al. [24] pioneer the use of KD for MOT model compression. Their approach utilizes attention-guided feature distillation and foreground masking to maintain superior tracking performance while substantially accelerating inference speed.

In the specialized field of model compression, KD is frequently synergized with other techniques to achieve extreme light-weighting. Pei et al. [25] explore the integration of distillation with contrastive learning during the quantization process, employing a hyperbolic tangent function to smooth gradient estimation and effectively compensate for feature-level information loss in low-bit models. To refine the granularity of feature alignment, Zhang et al. [26] introduce DenseKD, which captures diverse channel-wise features through a learnable dense architecture, and incorporates region and sample importance mechanisms to enable the student network to focus adaptively on critical knowledge. Furthermore, Yang et al. [27] develop Cross-Image Relational Knowledge Distillation (CIRKD), which focuses on transferring structured pixel-to-pixel and pixel-to-region relationships across images, thereby enhancing performance in semantic segmentation tasks.

Regarding the dynamic environments and security requirements at the edge, significant breakthroughs have been achieved. Mishra et al. [28] present the EarlyLight scheme, which integrates an early halting technique to monitor processing time and energy consumption during distillation, facilitating the efficient training of lightweight neural networks directly on edge devices. To bolster model defense capabilities, Zhao et al. [29] introduce a Balanced Multi-Teacher Adversarial Robustness Distillation (B-MTARD) framework. By employing an entropy-based balance algorithm to coordinate the knowledge contribution from both "clean" and "robust" teachers, they effectively mitigate the trade-off between accuracy and robustness. Moreover, the temperature hyperparameter in KD has become a focal point of recent research. Li and Wei et al. [30,31] propose variable-temperature distillation to provide specialized temperatures for diverse student models, while Zhang et al. [32] further introduce reinforcement learning to enable agents to learn optimal and dynamic temperature adjustment strategies. Collectively, KD has been extensively applied in cloud–edge orchestration [33,34], as its potent model compression capabilities align seamlessly with the design philosophy of cloud–edge collaboration, providing reliable inference support for edge devices. However, most of the aforementioned studies focus on model compression within static environments, leaving significant challenges in cloud–edge collaborative incremental learning scenarios.

2.4. Cloud–edge collaborative intelligent computing

Recently, this technology has obtained extensive industrial adoption. Cloud–edge collaborative computing has emerged as a critical paradigm for enabling resource-constrained devices to handle computation-intensive tasks by integrating the robust processing power of the cloud with the low-latency response of the edge. Laili et al. [35] propose a practical task-scheduling model by utilizing a parallel grouped fusion evolutionary algorithm to optimize task allocation within extremely short timeframes, significantly reducing both computational latency and energy consumption. Ding et al. [7] develop an updatable cloud–edge framework that employs a parameter-freezing strategy, allowing the cloud to share shallow layers to assist edge-side updates, thereby enhancing cross-task generalization. Furthermore, Chen et al. [33] integrate Federated Learning with Wasserstein distance metrics to identify optimal network layers for personalized adaptation. In the era of Large Language Models (LLMs), cloud–edge collaboration has become a vital strategy for addressing the prohibitive computational demands of fine-tuning and deploying these models at the edge [36–38].

In the domain of task offloading and resource optimization, Deep Reinforcement Learning (DRL) has demonstrated superior performance in navigating dynamic network environments. Fan et al. [39] introduce a hybrid offloading scheme based on the SD3 algorithm to jointly optimize D2D communication and multi-tier server resource allocation. Song et al. [40] develop an asynchronous collaborative algorithm (CEC-DRL) tailored for consumer electronics, ensuring high resilience while minimizing system costs. For complex manufacturing scenarios, Guo et al. [41] utilize a Transformer-based DRL framework to achieve near-real-time production line optimization. Additionally, Jiang et al. [42] propose the DRKC strategy, which enhances load balancing within Kubernetes clusters in cloud–edge environments through multi-dimensional resource awareness.

To further refine collaborative efficiency, researchers have explored hierarchical architectures and fine-grained scheduling. Cai et al. [43] investigate a cloud–edge-end three-tier architecture, employing the LST model and LSTM networks to facilitate precise multi-granularity task processing. Zheng et al. [44] present the EdgeNetLLM Transformer framework, which adapts LLMs to mobile networks at minimal cost via cloud-based fine-tuning and edge-side one-time pruning. In remote sensing, Wu et al. [45] propose a hyperspectral image fusion method that alleviates transmission delays by deploying detection algorithms close to the data source at the edge.

Regarding collaborative training and data processing, reducing bandwidth consumption and accelerating convergence remain central challenges. Zeng et al. [46] design an "Offline-Transfer-Online" framework using algorithm-independent knowledge distillation, enabling edge agents to achieve a manifold increase in convergence speed. Chai et al. [47] approach this from a data perspective, proposing a diffusion-model-based data distillation method that generates synthetic datasets to relieve edge storage pressure and communication burdens.

Despite these advancements in resource optimization and task adaptation, existing methods often struggle with complex incremental tasks where a restricted number of trainable layers prevents the network from learning sufficient feature representations, thereby compromising model accuracy. Currently, there remains a notable lack of methodologies for automatically determining the optimal number of frozen layers to balance training efficiency with performance. To address these limitations, this paper proposes an Adaptive Frozen-Layer Knowledge Distillation (AFLKD) method. By dynamically freezing specific layers of the deep neural network, AFLKD effectively mitigates the computational pressure on edge devices during incremental training, enabling the automated and efficient optimization of model accuracy.

3. Cloud–edge collaboration architecture

In this section, the cloud–edge collaborative architecture is constructed for training edge network, as shown in Fig. 1. It is the architecture of initial training phase, where the edge (student) network undergoes initial training process supported by the pre-trained cloud (teacher) network and knowledge distillation, resulting in a lightweight and accurate network model.

During the initial training phase, we firstly input a large amount of labeled data $\{x_i, y_i\}_{i=1}^{M}, x_i \in R^d, y_i \in R^c$ to train a reliable cloud-based network model N_c with high accuracy. Subsequently, the cloud network model N_c is employed to assist in training the edge network model N_e. Where M represents the amount of data, d denotes dimension of data, and c is the number of classes.

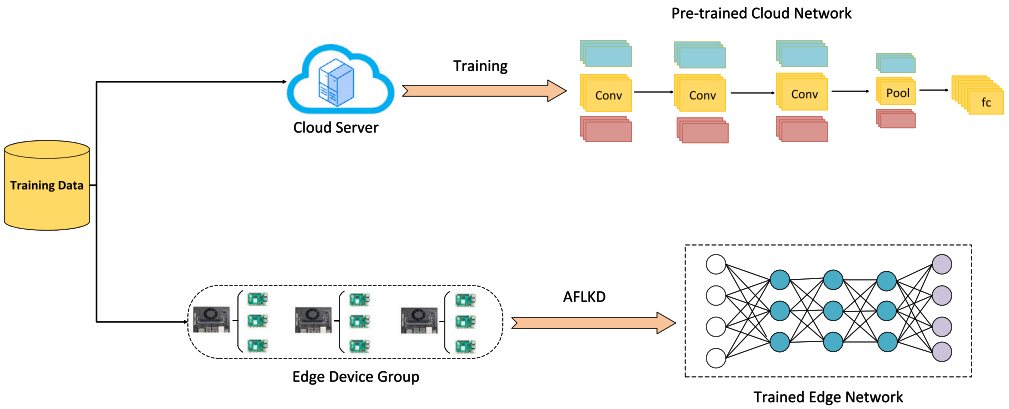


Fig. 1. Cloud–edge collaboration architecture for initial training edge network.

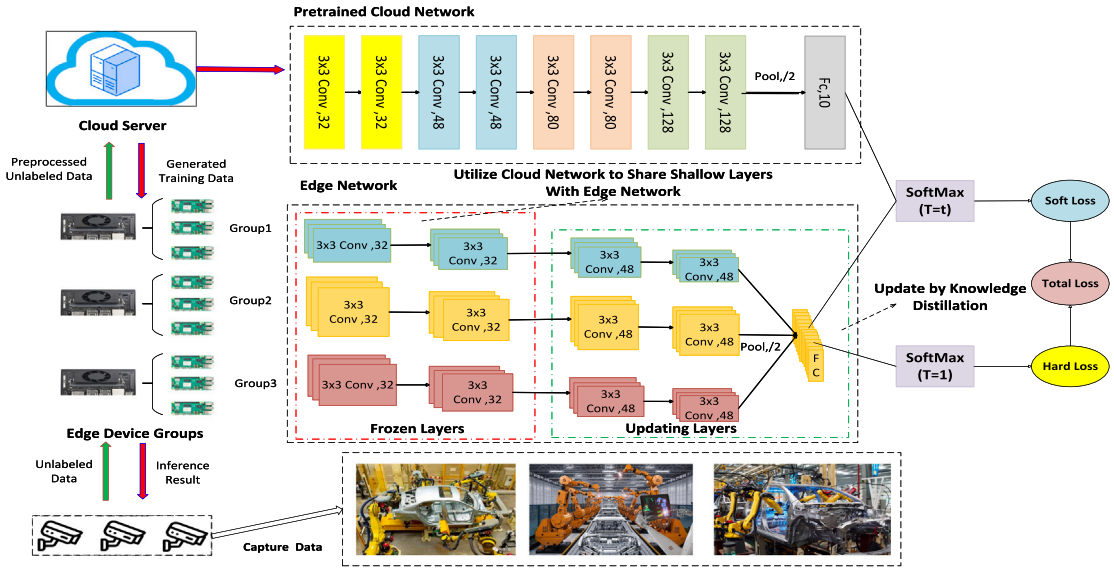


Fig. 2. Incremental updating the edge network with captured data.

For the edge network model, the response-based distillation loss function is usually expressed by Eq. (1):

$$L(N_e) = H_{ce}(N_e(x), y) + \lambda D_{KL}(N_e(x), N_c(x)) \tag{1}$$

where λ denotes weight parameters which control the ratio between hard-loss and soft-loss to optimize the loss function of N_e.

Distillation loss function typically consists of two parts. One part represents the difference between the student network and the true labels, often referred to as hard-loss H_{ce}. In this paper, we use cross-entropy loss to calculate it in Eq. (2).

$$H_{ce}(N_e(x), y) = -\sum_{i=1}^{c} y_i log N_e(x_i) \tag{2}$$

The other part indicates the difference between the teacher (cloud) network and the student (edge) network, often referred to as soft-loss D_{KL}. In this paper, we adopt KL divergence for calculation. Typically after both the student and teacher networks' outputs are softmaxed by $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{c} e^{z_j}}$. Knowledge distillation introduces the concept of temperature T to the softmax function, which smoothens the logits output by $\sigma_{KD}(z_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{c} e^{z_j/T}}$. This allows the student network to learn more information from the teacher network effectively. Further, the soft-loss can be calculated as Eq. (3).

$$L_{soft}(N_c(x), N_e(x), T) = \sum_{i=1}^{N} T^2 D_{KL}(\sigma_{KD}(N_c(x_i), \sigma_{KD}(N_e(x_i)))) \tag{3}$$

It is worth noting that during the training of the edge network, it is not necessary to update parameters for all layers. Since it is a pre-trained model, and current research indicates that the initial layers of neural networks often have useful general features across different image recognition tasks [48]. Therefore, the edge network model can share the shallow layers of the cloud network model by parameter freezing. Assuming the network has a total of m+n layers, the parameters of the first n layers can be frozen during updating, and only the parameters of the remaining m layers need to be updated.

We assume that W_c and W_e respectively denote the weight parameters of the cloud network model and edge network model, and W_{ne} represents the frozen parameters of the first n layers of the edge network model shared with the cloud network. During the training process, we only need to update the parameters W_{me} of the unfrozen m layers. This method updates the parameters using the aforementioned loss function in Eqs. (1) to (3).

4. Incremental updates and adaptive frozen layer knowledge distillation

On basis of the initial training cloud–edge collaborative architecture in Fig. 1, this section focuses on the incremental update in the edge network model assisted by the cloud network model, as depicted in Fig. 2. After the overall architecture has been running for a certain period, a sufficient amount of unlabeled data is obtained from the user terminals. The architecture then utilizes the cloud network model to transform the unlabeled data into a labeled training dataset, which is subsequently transmitted to the edge model for further incremental updates with adaptive frozen layer knowledge distillation (AFLKD), thereby improving the network's accuracy and generalization.

4.1. Incremental updates using knowledge distillation

In the process of knowledge distillation, besides distillation loss, temperature with a specific meaning is also a crucial factor. Different temperature choices can have varying impacts on the distillation results. Therefore, updating the temperature during the training process can further enhance the effectiveness of the model.

Since the target of knowledge distillation is general $\arg\min_{N_e} L(N_e)$, we can update the temperature T during the training, then the target of distillation is now changed into $\arg\min_{N_e,T} L(N_e, T)$.

The parameters N_e and T can be updated by Eqs. (4) and (5):

$$N_e \leftarrow N_e - \frac{\partial L}{\partial N_e} \tag{4}$$

$$T \leftarrow T - \frac{\partial L}{\partial T} \tag{5}$$

After taking the learnable parameter T into consideration, the loss function of the edge network is shown in Eq. (6).

$$\begin{aligned} L(N_e(x), T) = &-\textstyle\sum_{i=1}^{c} y_i log N_e(x_i) + \\ \lambda \sum_{i=1}^{N} &T^2 D_{KL}(\sigma_{KD}(N_c(x_i)), \sigma_{KD}(N_e(x_i)))) \end{aligned} \tag{6}$$

In this architecture, we utilize the cloud network model to train the edge network model by freezing layers and combining it with knowledge distillation. This ensures reliable performance while maintaining the sufficiently compact model size. Therefore, besides the distillation method and the control of parameters during distilling, it is also necessary to consider how to select the number of layers to freeze in the cloud network model during edge training.

4.2. The AFLKD mechanism and its proof

In knowledge distillation, the temperature parameter T controls the smoothness of the probability distribution generated by the teacher model. When the temperature is T, the teacher softmax output is defined as

$$p_i^{(t)}(T) = \frac{\exp(z_i^{(t)}/T)}{\sum_j \exp(z_j^{(t)}/T)} \tag{7}$$

where $z_i^{(t)}$ denotes the teacher model logits. Applying the first-order Taylor expansion to the exponential function gives

$$\exp(z_i/T) \approx 1 + \frac{z_i}{T} \tag{8}$$

Thus the probability distribution can be approximated as

$$p_i^{(t)}(T) \approx \frac{1 + \frac{z_i}{T}}{\sum_j \left(1 + \frac{z_j}{T}\right)} \tag{9}$$

From this approximation, it can be observed that the temperature T reduces the differences between class probabilities. As T increases, the distribution becomes smoother and the discriminative information decreases. The distillation loss function is usually defined as the Kullback–Leibler divergence

$$L_{KD}(T) = T^2 \sum_i p_i^{(t)}(T) \log \frac{p_i^{(t)}(T)}{p_i^{(s)}(T)} \tag{10}$$

where $p_i^{(s)}(T)$ denotes the student probability distribution.

Researches have shown that the choice of temperature in knowledge distillation is crucial to the distillation performance [30]. Therefore, adopting a fixed-temperature distillation strategy is not optimal for progressive learning tasks. It has been suggested that the distillation process is similar to curriculum teaching, such as starting with basic and simple concepts, and gradually increasing the difficulty leads to better distillation results. Models at different training stages require training difficulty that matches their learning capability.

In this paper, the temperature is treated as a learnable parameter, and is adjusted dynamically during training to ensure it aligns with the model's current learning ability. When the temperature is well-matched to the model's capacity, a higher adjusted temperature indicates a more difficult distillation task, which also implies that the current model has stronger learning capability. For such models, we can freeze fewer layers, allowing them to learn more parameters directly from the edge-end data. Conversely, when the adjusted temperature is lower, it indicates that the model's learning ability is still insufficient, and thus more layers should be frozen to ensure that the model acquires more information from the teacher model. This approach employs the high-precision cloud model to assist the edge model through obtaining reliable intermediate feature maps after the shallow convolution process. Therefore, the higher the temperature, the fewer layers need to be frozen, showing an overall negative correlation.

To prove the conjecture about the proposed AFLKD mechanism and derive the relationship of temperature T with frozen layers, let the student network contain m layers, and suppose n layers are frozen during training. Taking the derivative of the distillation loss function (Eq. (10)) with respect to temperature T gives

$$\frac{\partial L_{KD}}{\partial T} \tag{11}$$

Using the derivative property of the softmax function with respect to temperature, the dominant term of the gradient can be approximated as

$$\frac{\partial L_{KD}}{\partial T} \propto \frac{\mathrm{Var}(z^{(t)}) - \mathrm{Cov}(z^{(s)}, z^{(t)})}{T^3} \tag{12}$$

When the gradient approaches zero at equilibrium, we obtain

$$\mathrm{Var}(z^{(t)}) \approx \mathrm{Cov}(z^{(s)}, z^{(t)}) \tag{13}$$

When the representation capability of the student model is insufficient, the covariance between logits of the student model $z^{(s)}$ and the teacher model $z^{(t)}$ becomes smaller, leading to a smaller value on the right-hand side of the equilibrium condition. To maintain the balance, the temperature T should be reduced. This reduction decreases the entropy of the probability distribution, and increases the inter-class discrimination, making the supervision signal more informative and discriminative. Conversely, when the student model has stronger representation capability, a larger temperature can be employed. This condition indicates that the optimal temperature depends on the representation capability of the student model.

According to statistical learning theory, the capacity of a neural network function space is positively correlated with the number of trainable parameters. Let the student model contain m layers in total and n frozen layers, then the number of trainable layers is $m - n$. Thus, the effective representation capacity of the model can be expressed as

$$C \propto (m - n) \tag{14}$$

Since the optimal temperature increases with the model capacity, we obtain

$$T \propto C \tag{15}$$

Based on the transmission relationship of Eqs. (14) and (15), we obtain

$$T \propto (m - n) \tag{16}$$

When the total number of layers m is fixed, the temperature T becomes negatively related to the frozen layer number n.

Therefore, the adaptive distillation temperature is negatively related to the number n of frozen layers. To simplify the derivation and calculation, the number n of frozen layers is adjusted according to Eq. (17).

$$n = \left\lfloor m \cdot (1 - \frac{T - T_{min}}{T_{max} - T_{min}}) + 0.5 \right\rfloor \tag{17}$$

where the number of total layers and frozen layers is respectively m and n, the range of temperature T belongs to $[T_{min}, T_{max}]$, the temperature is normalized to obtain the negative correlation ratio of the total layer number, and 0.5 is the guarantee that when T equals T_{max}, the value of n reaches its minimum of 1. The real frozen layer number n can finally be calculated by Eq. (17). Based on empirical observations, we set the overall temperature range between 1 and 10, and dynamically adjust the number of frozen layers according to the temperature changes during each training iteration.

Algorithm 1 Model parameter initialization and *AFLKD* execution process

Input: Training data $\{x_i, y_i\}_{i=1}^{M}$, pre-trained cloud network model N_c, the number of layers m that can be frozen, the number of layers N in the edge network model N_e
Output: The updated N_e
1: Randomly initialize the temperature T in [1,10] and calculate the frozen-layer number n according to Equation (7)
2: N_c sends its first n layers' frozen parameters to N_e as W_{ne}
3: Randomly initialize unfrozen parameters of the rest $N - n$ layers in N_e
4: **for** $i = 1$ to $epoch$ **do**
5: Calculate the loss $L(N_e(x), T)$ according to Equation (6)
6: Update $W_{(N-n)e}$ and T according to Equation (4) and (5)
7: Update the number n of frozen layers according to Equation (17)
8: Freeze the first n layers in N_e
9: **end for**
10: **return** N_e with the updated model parameters $W_{ne} \cup W_{(N-n)e}$

4.3. Incremental training and update algorithms

The whole cloud–edge collaboration initial training process includes initializing model parameters and updating the edge network model N_e by using the adaptive frozen layer knowledge distillation (AFLKD) mechanism, as shown in Algorithm 1. Lines 1 to 3 initialize the model parameters that need to be updated in the network and, send the shared layer parameters W_{ne} from the cloud to the edge network. Lines 4 to 9 represent the training process of the edge network, L represents the value of the loss function, while $L > \eta$ the network will undergo training and updating the model parameters. Let us take an example of one training epoch. When the network starts training, it first calculates the current value of the loss function. Then, based on the loss function, it updates the model parameters of the unfrozen layers $W_{(N-n)e}$ and the temperature T. After updating the temperature T, we calculate the current optimal number of frozen layers n according to Eq. (7). Finally, the corresponding layers are frozen in preparation for the next training epoch. The time complexity of Algorithm 1 is $O(n)$, n represents the epoch of training.

After the initial training phase in Algorithm 1, the edge network can achieve reliable accuracy by addressing the resource limitations of the edge side and utilizing knowledge distillation. However, in real-world tasks, a large amount of data is continuously received from user terminals during the operation of the architecture. If the model for inference is trained only by initial training data, the accuracy of the results may be questionable when faced with data that differs from the training data distribution. Therefore, if the model could be continuously updated during inference, the network would become more reliable. In practice, the data received is often large in volume and usually unlabeled, and it is used to provide to the edge network for inference. So, we need to find a method to utilize this unlabeled data for further training of the edge network model. Since the cloud network model is highly reliable and usually has high accuracy, we can leverage the cloud network to use this unlabeled data for assisting in training the edge network, thereby further improving its performance.

In this paper, we assume that the inference results from the cloud network are the true labels for the data. After the edge network receives the data, when it stores a certain amount of unlabeled data x_{new} and the devices are free, it will transfer the collected data to the cloud. Upon receiving the data, the cloud model initiates the cloud network to perform inference, and results in the inference output $N_c(x_{new})$. This output is then combined with the data x_{new} to form the training dataset $\{x_{new}, N_c(x_{new})\}$. Given the high accuracy of the cloud network, we treat the cloud network's inference result $N_c(x_{new})$ as the true label y_{new}, thus creating a new training dataset $\{x_{new}, y_{new}\}$. This dataset is then transferred to the edge device for incremental updates to the edge network. The loss function used for updating the edge network is followed by Eq. (6).

Algorithm 2 Incrementally updating the edge network

Input: New data x_{new} from user terminals, N_c, N_e
Output: The incrementally updated N_e
1: Transfer new data x_{new} to cloud when devices are free
2: Run the cloud network N_c to get the predict label $N_c(x_{new})$
3: Transfer the results to the edge network and get the new training data $\{x_{new}, N_c(x_{new})\}$
4: **for** $i = 1$ to $epoch$ **do**
5: Calculate the loss $L(N_e(x), T)$ according to Equation (6)
6: Update $W_{(N-n)e}$ and T according to Equation (4) and (5)
7: Update the frozen layer number n according to Equation (17)
8: Freeze the N_e's first n layers
9: **end for**
10: **return** N_e with $W_{ne} \cup W_{(N-n)e}$

The incrementally updating process in the edge network is depicted in Algorithm 2. Lines 1 to 3 describe the process of generating new training data from the collected unlabeled data using the cloud network. First, cameras or other IoT devices capture some production data and transmit it to the edge network. Then, when the edge devices are free, they will transfer this data to the cloud network and utilize the cloud network's high accuracy, the obtained inference results are treated as the true labels of the data, forming a new training dataset. After acquiring the new training data, it is then transmitted back to the edge devices for further updates to the edge network. Lines 4 to 9 outline the process of updating the edge network, which follows the same method as the cloud–edge collaboration initial training phase. It also utilizes knowledge distillation and adaptive freezing layer for the training updates. The time complexity of Algorithm 2 is $O(n)$, n is the epoch of training.

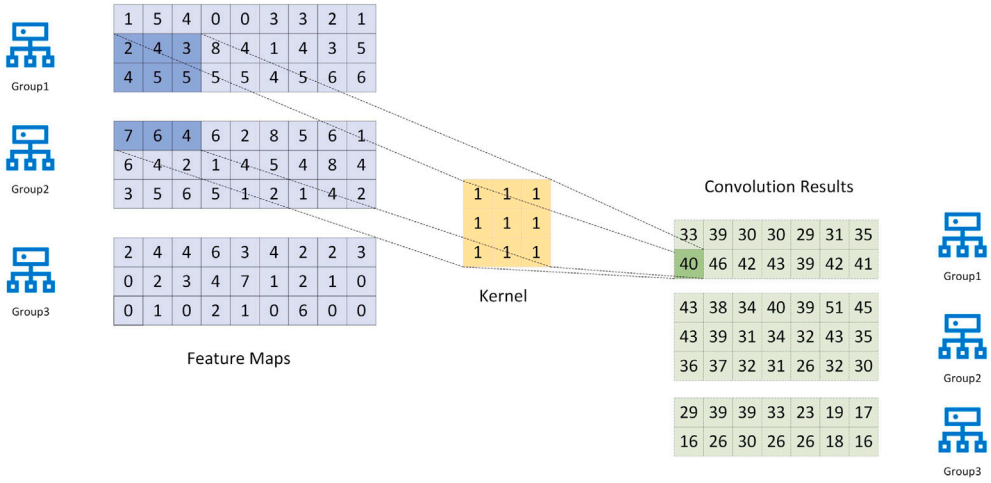


Fig. 3. Convolution computations cross different block groups.

5. Parallel inference in the edge network

During training the edge network, the limited computing power of edge devices often leads to excessive computational loads and delays, making it difficult to meet service requirements. Therefore, addressing resource constraints during both training and inference is a crucial consideration in the cloud–edge collaboration framework.

In this paper, we adopt a splitting method to accelerate training and reduce the resource load on edge nodes. We use the feature map splitting to distribute the computational workloads across different clusters. Within each cluster, the network model layers are also split and allocated to different edge devices based on the resource capabilities of each device. Each cluster consists of a Jetson inference device and several Raspberry Pis. Each of Jetson inference devices, with its relatively strong processing power, acts as the Master node, responsible for task scheduling and a portion of the inference tasks. All of the Raspberry Pis serve as worker nodes, executing tasks received from the Master node. Once the computations are completed, the Jetson inference devices, acting as the Master node, collect the intermediate feature maps from different clusters and concatenate for fully connected layer computation.

In the process of data splitting, we allocate intermediate feature maps to different groups for convolutions to reduce the computational loads on the edge devices. However, performing convolutions after splitting inevitably requires communication between edge devices to exchange feature information. Algorithm 3 illustrates how cross-device convolution computations are handled. To compute a 3×3 convolution kernel on a 3×9 splitting block, Group 1 needs to communicate with Group 2 to obtain feature information from the 1×9 region of Group 2. Generally, if the convolution kernel size is k, each splitting block needs to obtain $\lfloor k/2 \rfloor$ rows of feature information from neighboring blocks. If the kernel size is too large, it may involve computations across multiple devices, leading to excessive communication overhead. In this paper, the convolution kernel size is uniformly 3×3, so each group only needs to fetch 1 row of feature information from adjacent groups, minimizing communication overhead. The convolution computations cross different block groups are shown in Fig. 3.

Algorithm 3 Feature map splitting inference

Input: Inference feature map data x_{train}, number n of clusters, the edge network N_e
Output: Inference result y

1: Split the input feature map data x_{train} into n blocks along the height dimension
2: For each of n splitting blocks x_i in the edge network N_e, performs the following parallel inference:
3: **for** each layer in N_e's layers **do**
4: **if** the layer is convolutional layer **then**
5: **if** $i = 1$ **then**
6: Retrieve the first row of intermediate features from x_{i+1}
7: Concatenate the retrieved data and perform the convolution computation
8: **else if** $i = n$-1 **then**
9: Retrieve the last row of intermediate features from x_{i-1}
10: Concatenate the retrieved data and perform the convolution computation
11: **else**
12: Retrieve the first row of intermediate features from x_{i+1} and the last row of intermediate features from x_{i-1}
13: Concatenate all the retrieved data and perform the convolution computation
14: **end if**
15: **else if** layer is FC layer **then**
16: x_1 retrieves the computed results from all other clusters and performs the fully connected computation to get inference result y
17: **else**
18: Perform the computation for the current layer
19: **end if**
20: **end for**
21: **return** y

The overall computing process of feature map splitting inference is shown in Algorithm 3. Firstly, the input feature map is split into n blocks based on the number of clusters to enable parallel computing. Lines 3 to 17 specifically describe the computation process of the input data on the edge network within the cluster. Since additional communication consumption is only required during convolution computations, the process first determines whether the current layer is a convolutional layer. If the current layer is not a convolutional layer, the network follows the standard computation flow. If the current layer is a convolutional layer, intermediate feature maps need to be exchanged between clusters. If the current cluster is the first cluster, it only needs to retrieve the first row of intermediate features from the next cluster. If the current cluster is the last cluster, it only needs to retrieve the last row of intermediate features from the previous cluster. Otherwise, the

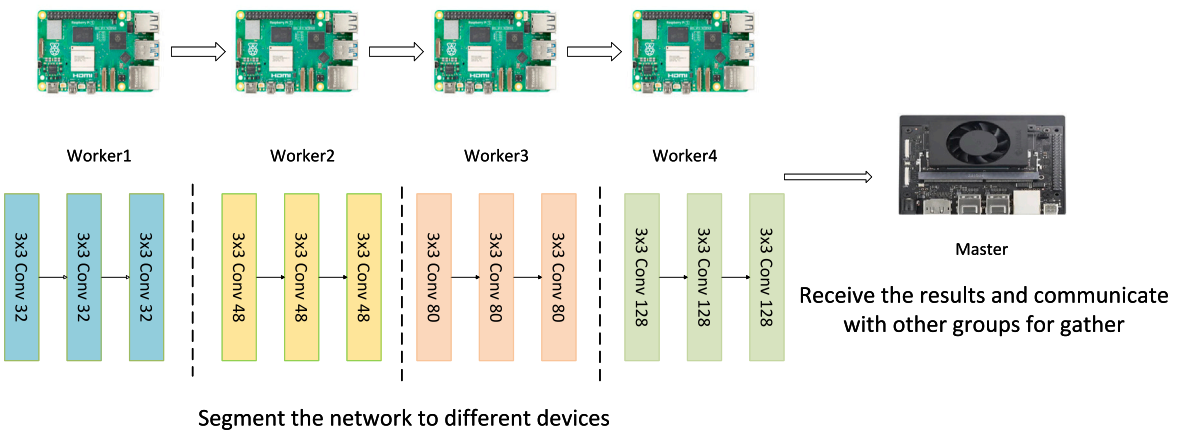


Fig. 4. Layer splitting in edge device groups.

cluster must retrieve both the last row of intermediate features from the previous cluster and the first row from the next cluster. Once the necessary feature data is retrieved, it is concatenated with the local feature map and the results of performing the convolution computation. When reaching the final fully connected layer, the first cluster collects the computed results from all other clusters, concatenates them, and performs the final fully connected computation to generate the inference output y. The time complexity of Algorithm 3 is $O(n)$, n denotes the number of layers.

As exhibited in Fig. 4, after splitting the feature maps across different clusters through semantic data partitioning, we further employs layer splitting within each cluster. Unlike semantic data partitioning, layer splitting divides the edge network by layers and distributes them across different edge devices. Since layer splitting does not alter the computing process, it does not affect the accuracy of the network and help each device release the computational pressure. However, because devices assigned to the later layers in the network must wait for the preceding devices to finish computing and pass their feature maps before continuing, there is no improvement in overall computing time.

6. Performance evaluation

In this section, we firstly validate the performance of three different edge network models using our adaptive frozen layer knowledge distillation (AFLKD) method on three commonly used datasets, and compared the results with the conventional (baseline) and the shared shallow layer frozen (SSLF) [7] training methods. Next, we verify the inference speed improvement of using feature map splitting for parallel computing in convolutional layers during the inference process. The time consumed by each layer in the network was compared during inference with and without feature map splitting. All the experimental code can be accessed on GitHub.[1]

6.1. Datasets

Our experiments adopt the three commonly used open source datasets, CIFAR-10, Fashion-MNIST and CIFAR-100, which are briefly described as follows.

CIFAR-10 is a widely used image classification dataset in the field of computer vision, collected and organized by Alex Krizhevsky, Geoffrey Hinton and others at the University of Toronto. CIFAR-10 is primarily used for image classification tasks in supervised learning. It consists of 32×32 RGB images across ten categories, with a total of 60,000 images. These images are divided into two sets, a training set containing 50,000 images and a test set containing 10,000 images.

Fashion-MNIST was released by the Zalando Research team in Germany, intended as a replacement for the MNIST dataset (which contains handwritten digits) for image classification tasks. Compared to MNIST, Fashion-MNIST is more challenging as it contains more complex images of fashion items, making it suitable for testing deep learning models in real-world scenarios. The dataset consists of 70,000 grayscale images of size 28×28, divided into two sets, a training set of 60,000 images and a test set of 10,000 images.

CIFAR-100 is another widely used image classification dataset in the field of computer vision, also collected and organized by Alex Krizhevsky, Geoffrey Hinton and others at the University of Toronto. CIFAR-100 is mainly used for image classification tasks in supervised learning and is considered more challenging than CIFAR-10 due to its larger number of categories. It consists of 32×32 RGB images across 100 categories, with a total of 60,000 images. These images are divided into two sets, a training set containing 50,000 images and a test set containing 10,000 images. Compared with CIFAR-10, each class in CIFAR-100 contains fewer training samples, which makes the classification task more difficult and requires stronger feature representation capability from the model.

For the cloud network training, we leveraged all the data from both the training and test sets. For the edge network training, to simulate real-world scenarios where sufficient training data may not be available, we selected a subset of the training data to evaluate the model's performance under small sample training conditions.

6.2. Experimental setup

6.2.1. Experimental environment

The simulation experimental environment is primarily composed of cloud devices, edge devices and a local area network connecting with the devices through a switch. The cloud devices are used to train the cloud network and require strong computational capabilities to perform comprehensive training. In real-world production, powerful cloud servers are typically used. For the sake of simplicity, a PC with high performance of CPU and GPU is used as a substitute in our experiments. The edge devices are used for small-sample training and task inference of the edge network models. These devices generally do not need to have strong computing power. Since distributed inference is also required, the edge devices will form a cluster composed of multiple devices. In the experiments, multiple Jetson orin nanos and Raspberry Pis are used to simulate edge devices. The specific device parameters and performance details are shown in Table 1. The experimental device environment in the edge network is depicted in Fig. 5.

[1] https://github.com/blindgo1/AFLKD-feature-map-splitting.

Table 1
Experimental device specifications.

Hardware	Jetson Orin Nano	Raspberry Pi 5	PC
CPU	6-core ARM Cortex-A78AE @ 1.5 GHz	Broadcom BCM2712 quad-core Arm Cortex A76 processor @ 2.4 GHz	12th Gen Intel(R) Core(TM) i7-12650H 2.30 GHz
GPU	NVIDIA Ampere architecture 1024 × NVIDIA CUDA Cores 32 × 3rd Gen Tensor Cores	/	NVIDIA GeForce RTX 4060 Laptop GPU
Memory	8 Gb 128-bit LPDDR5 68 GB/s	8 GB RAM	16 GB RAM
Number	3	12	1

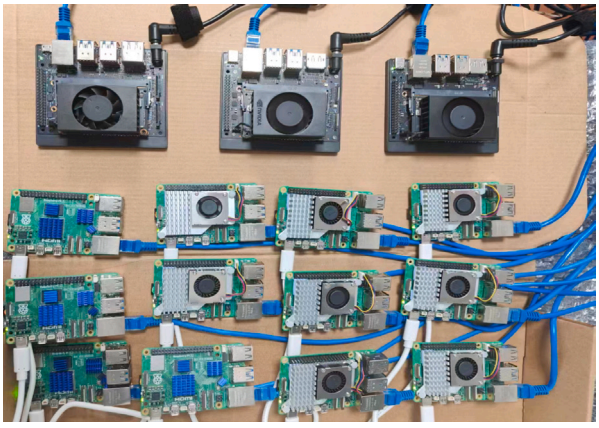


Fig. 5. The experimental environment in the edge network.

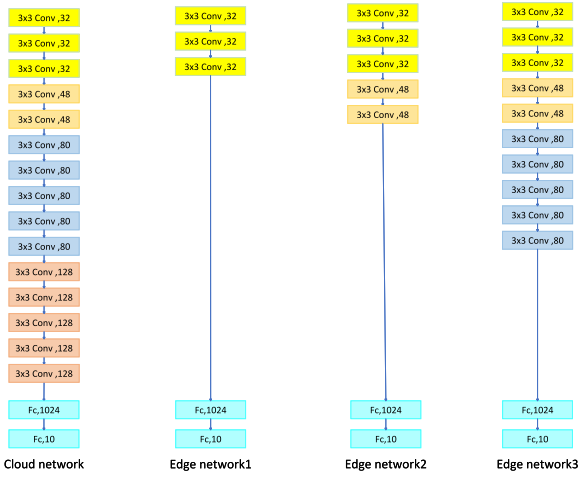


Fig. 6. The VGG model structures in cloud and edge networks.

6.2.2. Network model setup

The network model used in this experiment is shown in Fig. 6. The cloud network is similar to the VGG network, consisting of multiple convolutional blocks. The kernel size is uniformly set to 3 × 3. In the cloud network, the first part contains three convolutional layers, each with 32 kernels, the second part consists of two convolutional layers, each with 48 kernels, the third part comprises five convolutional layers, each with 80 kernels, the fourth part is composed of five convolutional layers, each with 128 kernels, and the fully connected layers have 1024 and 10 neurons. After each convolutional layer, we add a ReLU layer, and after each convolutional block, we add a MaxPooling layer.

To enable inference on the resource-constrained edge devices, different types of edge networks share a different number of convolutional blocks from the cloud network. Each convolutional block has the same structure as the one in the cloud network, but the overall network structure for each of edge networks is simpler compared to the cloud network.

6.3. Accuracy comparison of related methods

In this section, we compared the training accuracy between the three methods, the proposed adaptive frozen layer knowledge distillation (AFLKD), conventional training which means train without freeze layers (baseline) and the shared shallow layer frozen (SSLF) [7]. The performance of different methods was tested on the CIFAR-10, Fashion-MNIST and CIFAR-100 datasets. For these three datasets, the VGG models trained on the cloud network commonly achieve accuracies of 86.62%, 92.87%, and 68.36%, respectively.

Firstly, the accuracy of training the edge network was verified on the CIFAR-10 dataset at different numbers of training data. We consider that each type of edge networks has 500 initial training samples during its initialization phase. The subsequent increase in training data simulates the network performing incremental updates upon receiving new data. Although in real-world scenarios the baseline may not have access to new labeled data for incremental updates, for fairness in comparison, we also provide the baseline with new training data in this experiment.

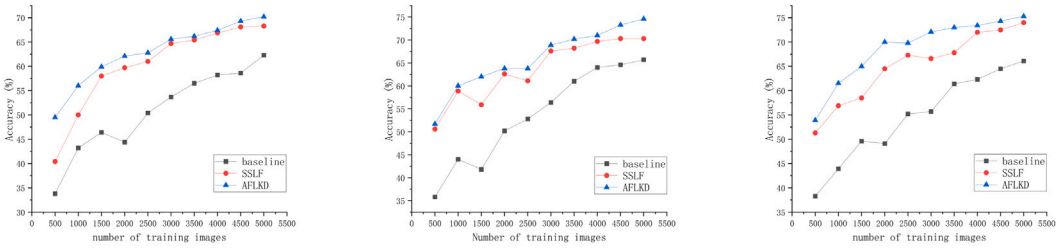
Fig. 7 shows that as the amount of data increases, the overall accuracy of all the three training methods in three edge networks improves, demonstrating that incremental updates can indeed enhance network performance. Among them, the training accuracy of the proposed AFLKD exceeds that of the other two methods. These results also highlight the importance of having sufficient data in practical production environments, making it highly beneficial to acquire new labeled data through the cloud network.

AFLKD significantly improves the accuracy of the network model when training with a small sample size and simple network structure. From Table 2, we can find that when the training data size is 500, the conventional training method (baseline) results in an accuracy of less than 40% for any types of edge network structures. However, by using our AFLKD, even the simplest model, Edge network 1, can achieve an accuracy of 49.5% with only 500 training images, improving by 15.7% compared to the baseline. This is due to the fact that a well-trained cloud network model has strong generalization ability in the shallow layers of the network, helping the edge network quickly improve its initial accuracy and training efficiency.

Furthermore, compared to SSLF, which also shares shallow layers of the network, AFLKD outperforms SSLF by 9.1% in terms of accuracy. This is because AFLKD utilizes knowledge distillation, which allows the student network to learn not only from the training data images but also from the teacher model (Cloud network). This is particularly beneficial for edge networks which are in the face of limited training data and need to train on small samples. By leveraging the knowledge from the cloud network, which is trained on a large dataset, AFLKD reduces the reliance of the network's accuracy on the amount of training data.

In the incremental update phase, as the training data increases, our AFLKD consistently outperforms the other two methods in terms of accuracy. This demonstrates that AFLKD enhances network model performance at every stage of the training process.

We also tested the three edge networks on the Fashion-MNIST dataset. The results are shown in Fig. 8. Since the Fashion-MNIST

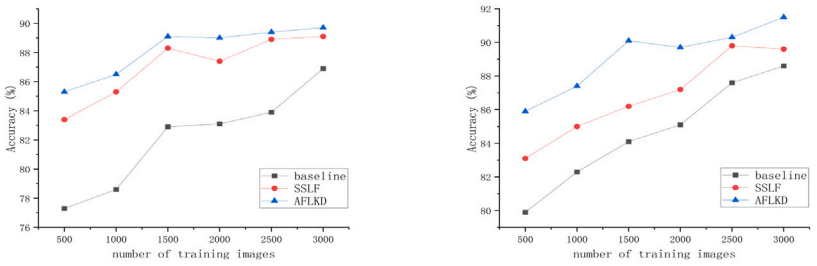


(a) Edge network1 (b) Edge network2 (c) Edge network3

Fig. 7. Accuracy comparison of different edge networks on CIFAR-10.

Table 2
Detailed result data for accuracy comparison under different sizes of CIFAR-10 dataset in different edge networks.

Number of training images	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
AFLKD(Edge network1)	49.5	56	59.9	62.1	62.8	65.6	66.2	67.4	69.3	70.2
SSLF(Edge network1)	40.4	50	58	59.7	61	64.7	65.4	66.9	68.1	68.3
Baseline(Edge network1)	33.8	43.2	46.4	44.4	50.4	53.7	56.5	58.2	58.6	62.3
AFLKD(Edge network2)	51.7	60	62	63.8	63.8	68.9	70.2	71.0	73.3	74.6
SSLF(Edge network2)	50.6	58.9	55.9	62.6	61.1	67.6	68.2	69.7	70.3	70.3
Baseline(Edge network2)	35.8	44	41.8	50.2	52.8	56.4	61.0	64.0	64.6	65.7
AFLKD(Edge network3)	53.9	61.5	65	70	69.8	72.1	73	73.4	74.3	75.3
SSLF(Edge network3)	51.3	56.9	58.5	64.5	67.3	66.6	67.8	72	72.5	74
Baseline(Edge network3)	38.3	43.9	49.6	49.1	55.2	55.7	61.4	62.3	64.5	66.1



(a) Edge network1 (b) Edge network2

Fig. 8. Accuracy comparison of different edge networks on Fashion-MNIST.

Table 3
Detailed result data for accuracy comparison under different sizes of Fashion-MNIST dataset in different edge networks.

Number of training images	500	1000	1500	2000	2500	3000	Cloud network accuracy
AFLKD(Edge network1)	85.3	86.5	89.1	89	89.4	89.7	92.87
SSLF(Edge network1)	83.4	85.3	88.3	87.4	88.9	89.1	
Baseline(Edge network1)	77.3	78.6	82.9	83.1	83.9	86.9	
AFLKD(Edge network2)	85.9	87.4	90.1	89.7	90.3	91.5	
SSLF(Edge network2)	83.1	85	86.2	87.2	89.8	89.6	
Baseline(Edge network2)	79.9	82.3	84.1	85.1	87.6	88.6	

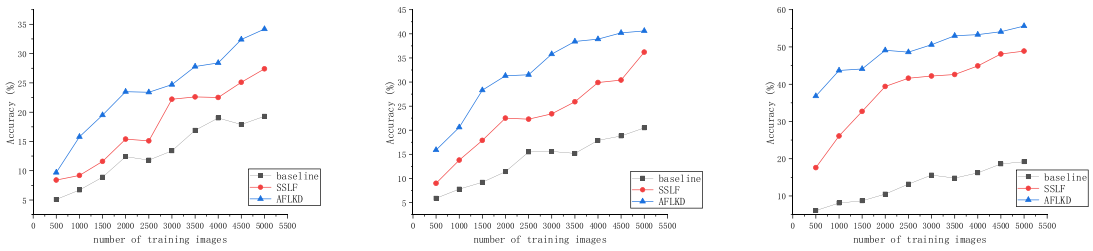
dataset is relatively easy to train and the network converges quickly, most models reach an optimal state once the training data exceeds 3000. Therefore, we compare only the simpler Edge network 1 and Edge network 2 for cases where the training data is fewer than 3000.

From the results, we can observe the conclusions similar to those from the CIFAR-10 dataset. Using our AFLKD on Edge network 1 and Edge network 2 leads to better accuracy than other methods when training with a small dataset. As illustrated in Table 3, when the training data size is 500, which denotes AFLKD improves the accuracy of Edge network 1 and Edge network 2 by 8% and 6%, respectively, compared to the baseline. Further compared to SSLF, AFLKD achieves an additional improvement of 1.9% and 2.8%.

In addition to the CIFAR-10 and Fashion-MNIST datasets, we further evaluate the proposed method on the CIFAR-100 dataset, which is a more challenging image classification benchmark. Compared with CIFAR-10, CIFAR-100 contains 100 categories with fewer samples per class, resulting in higher inter-class similarity and greater intra-class diversity. Consequently, models trained on CIFAR-100 require stronger feature representation capability and better generalization ability. Therefore, experiments on CIFAR-100 provide a more rigorous evaluation of the effectiveness of the proposed AFLKD method.

Fig. 9 illustrates the accuracy trends of the three training methods as the number of training samples increases. As shown in the figure, the classification accuracy of all methods gradually improves with the growth of the training dataset. These results indicate that incremental updates with newly collected data can effectively enhance the performance of edge networks. Moreover, the curves show that AFLKD consistently maintains higher accuracy than both SSLF and the baseline method across all training stages and network structures.

A more detailed quantitative comparison is presented in Table 4. When the number of training images is only 500, the baseline method performs poorly due to the complexity of the CIFAR-100 dataset. Specifically, the baseline accuracies of Edge network 1, Edge network 2 and Edge network 3 are only 5.1%, 5.9% and 6.1%, respectively. In contrast, the proposed AFLKD achieves 9.7%, 15.9% and 36.8%



(a) Edge network1 (b) Edge network2 (c) Edge network3

Fig. 9. Accuracy comparison of different edge networks on CIFAR-100.

Table 4
Detailed result data for accuracy comparison under different sizes of CIFAR-100 dataset in different edge networks.

Number of training images	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
AFLKD(Edge network1)	9.7	15.8	19.5	23.5	23.4	24.7	27.8	28.4	32.4	34.2
SSLF(Edge network1)	8.4	9.2	11.6	15.4	15.1	22.2	22.6	22.5	25.1	27.4
Baseline(Edge network1)	5.1	6.7	8.9	12.4	11.8	13.4	16.9	19	17.9	19.3
AFLKD(Edge network2)	15.9	20.6	28.3	31.3	31.5	35.8	38.4	38.9	40.2	40.6
SSLF(Edge network2)	9	13.8	17.9	22.5	22.3	23.4	25.9	29.9	30.4	36.2
Baseline(Edge network2)	5.9	7.8	9.2	11.4	15.5	15.6	15.2	17.9	18.8	20.5
AFLKD(Edge network3)	36.8	43.7	44.1	49.1	48.6	50.6	53	53.3	54.1	55.6
SSLF(Edge network3)	17.6	26.1	32.7	39.4	41.6	42.2	42.6	44.9	48.1	48.9
Baseline(Edge network3)	6.1	8.1	8.7	10.5	13.2	15.6	14.8	16.2	18.6	19.2

accuracy on the three edge networks. Compared with the baseline, AFLKD improves the accuracy by 4.6, 10.0 and 30.7 percentage points, respectively. Particularly for Edge network 3, the accuracy increases by more than six times, demonstrating the significant advantage of AFLKD in extremely small-sample scenarios.

Compared with SSLF, which only shares shallow layers of the network, AFLKD also demonstrates clear improvements. When the number of training samples is 500, AFLKD outperforms SSLF by 1.3%, 6.9% and 19.2% on Edge network 1, Edge network 2 and Edge network 3, respectively. This improvement mainly comes from the knowledge distillation mechanism, which enables the student network to learn not only from the labeled training samples but also from the soft targets generated by the cloud network. Such additional supervisory information allows the edge network to learn richer semantic representations even when the available training data is limited.

From the trend shown in Fig. 9, the advantage of AFLKD remains stable throughout the incremental training process. As the training data increases to 5000 images, the accuracy of AFLKD reaches 34.2%, 40.6% and 55.6% for Edge network 1, Edge network 2 and Edge network 3, respectively. In comparison, SSLF achieves 27.4%, 36.2% and 48.9%, while the baseline method only reaches 19.3%, 20.5% and 19.2%. Although the accuracy gap between different methods gradually narrows as the dataset becomes larger, AFLKD consistently maintains the highest accuracy.

Overall, the experimental results on CIFAR-100 further validate the effectiveness and robustness of the proposed AFLKD method. By leveraging knowledge transferred from the cloud network, AFLKD significantly improves the initial training performance of edge networks, and maintains superior accuracy throughout the incremental learning process. The consistent improvements observed across different network architectures also demonstrate the strong generalization capability of the proposed method on complex datasets.

With an increase of training data, the accuracy gap between the different methods gradually decreases, but our AFLKD can always maintains an advantage. This indicates that the AFLKD method improves network performance regardless of whether the training data is limited or sufficient, and it exhibits strong generalizability across different datasets.

6.4. Inference time comparison of splitting and no splitting feature maps

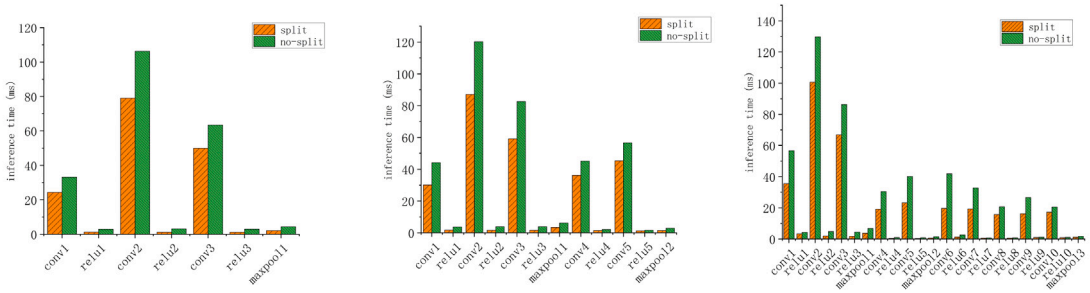
As described in Section 5, to accelerate inference speed, we applied the feature map splitting method by adopting the specific partitioning strategy. We conducted experiments to compare the inference time of feature map splitting with non-split across the three different edge networks.

To ensure fairness in the experiments, the same 224×224 input image is used for inference across all network models. Since computations in the fully connected layers are not split, we excluded them from the comparison.

Fig. 10 shows the inference time per layer for three different edge networks. After applying feature map splitting, the inference time of each layer displays a noticeable improvement compared to the non-split approach. In our experiments, an input feature map is split into two parts and processed in parallel across two clusters. By splitting the feature map, the computational complexity of the convolutional layers is reduced, as each convolutional layer only needs to process half the size of the intermediate feature maps compared to the non-split case. Since the partitioned feature maps are processed in parallel by different clusters, theoretically, the inference time should be reduced by half. However, due to the need for neighboring devices to exchange feature maps, additional communication overhead is generated. Even so, our experimental results show that feature map splitting still improves inference speed, with the total inference time being approximately 70% of the non-split approach on convolutional layers. For model layers that do not require data exchange, such as ReLU and MaxPooling, the inference time behaves as expected, requiring only 50% of the time compared to the non-segmented approach. It demonstrates that feature map splitting enables significant speed improvements in parallel inference. The more complex the network, the longer the inference time, making feature map split even more beneficial for reducing computing time in real-world applications.

6.5. Summary

From the above analysis of experimental results, we can conclude that the proposed AFLKD demonstrates excellent performance across multiple datasets. Compared to conventional training, AFLKD



(a) Edge network1 (b) Edge network2 (c) Edge network3

Fig. 10. Inference time comparison of feature map splitting and no splitting.

achieves significant accuracy improvements when training with small-scale datasets. Compared to the SSLF method, AFLKD consistently outperforms in both the initialization and the incremental update phase. On the Fashion-MNIST dataset, which has a relatively lower training difficulty, AFLKD even achieves performance close to that of the cloud network model. This indicates the feasibility of deploying DNN directly on edge devices for task inference. By leveraging the cloud network to assist in training and updating the edge Network, i.e., cloud–edge collaborative incremental learning approach, AFLKD effectively enhances both the performance of the initial deployment and after incremental updates through inputting newly collected data.

The inference time comparison experiments further show that the feature map splitting method effectively accelerates inference speed in edge networks. The more complex the network and the greater the inference workload, the more time is saved. Applying this method in real-world scenarios can significantly reduce inference time, prevent SLA violations, and improve user service quality. Moreover, the feature map splitting is generally applicable to various convolutional neural networks. Although our research is based on a VGG-like network, this method is also equally applicable to other CNN structures, such as AlexNet, Inception and RegNet.

7. Conclusions and future work

In order to deal with the problem that the inference performance of edge sides decreases under the cloud–edge collaborative inference mode, this paper proposes the adaptive frozen layer knowledge distillation (AFLKD), a knowledge distillation-based training method for edge networks.

7.1. Advantages and adaptability

The proposed method employs the cloud network to assist in training edge networks, and allows for incremental learning using newly collected data. Experimental results demonstrate that the proposed approach effectively improves the initial accuracy of edge networks during deployment, particularly in cases where the network models are lightweight and training data is insufficient. This also aligns well with real-world edge computing scenarios, where computational resources are constrained and labeled data is scarce. Additionally, as training data increases, the accuracy of the presented AFLKD continues to outperform current relevant methods during the incremental update phase. For inference acceleration, a feature map splitting method is introduced, which reduces inference time through parallel computing.

7.2. Limitations and discussion

Although the proposed AFLKD cloud–edge collaborative incremental learning framework demonstrates promising improvements in inference latency and accuracy under resource constraints, it still has several limitations that deserve further investigation.

Firstly, the proposed parallel inference and feature-map splitting strategy is, in practice, more naturally aligned with convolutional neural networks (CNNs). CNNs exhibit a hierarchical feature extraction pipeline with relatively clear spatial structures and channel semantics, which makes the workflow of "splitting feature maps to parallel computing and then to fusion" easier to implement and optimize. However, when extending the framework to broader model families (e.g., Transformers, hybrid CNN-Transformer architectures, graph neural networks, and other networks with complex branches and dynamic routing), representations may no longer preserve the same degree of spatial locality, and attention-style global dependencies introduce stronger coupling across tokens or features. As a result, the choices of splitting granularity and splitting points become more challenging, and the required cross-node communication and synchronization overhead may increase significantly.

Secondly, the framework exhibits a non-negligible dependency on the quality and stability of the cloud-side teacher model. In real-world complex scenarios, cloud data may suffer from noisy labels, domain shifts, long-tailed distributions or continuously evolving tasks, which can degrade the teacher's accuracy, calibration and generalization. When the teacher model produces unreliable soft targets or distorted inter-class relations, knowledge distillation may propagate such biases to the edge student model, resulting in degraded accuracy, shifted decision boundaries or even amplified forgetting in continual settings. In addition, communication latency and asynchronous updates in cloud–edge collaboration may cause the edge side to distill from stale teacher knowledge under distribution drift, further magnify distillation errors.

7.3. Future work

To address the above limitations, future work will focus on improving the generality, robustness and deployability of the framework from the following directions.

(1) Architecture-agnostic partitioning and freezing strategies.

We plan to extend the current splitting and parallel inference mechanism beyond CNNs to Transformers and hybrid architectures by exploring more general partitioning schemes (e.g., module-wise partitioning, tensor-dimension splitting, pipeline parallelism, early-exit and multi-exit collaboration). Meanwhile, we will develop a unified cost model that characterizes the trade-offs among splitting points, communication volume, end-to-end latency and accuracy, enabling automated strategy selection under heterogeneous constraints. In addition, we will investigate finer-grained and learnable freezing mechanisms (e.g., gated freezing and parameter-efficient tuning such as adapters/LoRA jointly with freezing), so that the freezing policy can adapt to task difficulty and resource budgets rather than relying on hand-crafted rules.

(2) Reducing sensitivity to imperfect teachers.

To mitigate the risk that an inaccurate or poorly calibrated cloud teacher model harms edge performance in complex settings, we will incorporate more robust teacher knowledge construction and distillation

objectives, such as uncertainty- or confidence-aware sample filtering and loss re-weighting, calibrated soft targets, adaptive temperature scheduling, teacher ensembles, and auxiliary self-supervised or contrastive representation learning to reduce the reliance on high-quality labels. Furthermore, we will explore "degradation-aware" safeguards on down-weight distillation when the teacher model's reliability is low, preventing erroneous knowledge from being irreversibly transferred to the edge model.

(3) System-level evaluation in realistic deployments.

We will further validate the framework on more realistic tasks and datasets (e.g., industrial defect detection, cross-domain surveillance, long-tailed recognition, and continual learning scenarios), and under diverse network conditions (bandwidth constraints, packet loss, and asynchronous updates) as well as heterogeneous edge device settings. Beyond accuracy and latency, we will report end-to-end latency breakdowns, communication overhead, energy consumption and other robustness metrics to comprehensively assess the scalability and practicality of cloud–edge collaborative incremental learning in real deployments.

CRediT authorship contribution statement

Jiaqi Fei: Writing – original draft, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Mengdie Qin:** Validation, Software. **Xiaogang Wang:** Writing – review & editing, Resources, Project administration, Methodology, Funding acquisition, Conceptualization. **Hui Guo:** Validation, Software. **Ziqi Zhu:** Visualization, Validation. **Jian Cao:** Resources, Project administration, Funding acquisition. **Rajkumar Buyya:** Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China (Granted Number 62072301), in part by Natural Science Foundation of Shanghai Science and Technology Innovation Action Plan (Granted Number 22ZR1425300), in part by Melbourne-Chindia Cloud Computing (MC3) Research Network, and in part by Program of Technology Innovation of the Science and Technology Commission of Shanghai Municipality (Grant ed Number 21511104700).

Data availability

The research code for this paper is illustrated in the experimental section of the paper, linked to the share base of the author's github account.

References

[1] E. Dritsas, M. Trigka, A survey on the applications of cloud computing in the industrial internet of things, Big Data Cogn. Comput. 9 (2) (2025) 44.

[2] Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, L. Cheng, Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review, 2025, arXiv preprint arXiv:2501.01007.

[3] C.R. Panigrahi, J.L. Sarkar, B. Pati, R. Buyya, P. Mohapatra, A. Majumder, Mobile cloud computing and wireless sensor networks: A review, integration architecture, and future directions, Iet Netw. 10 (4) (2021) 141–161.

[4] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al., A survey of large language models, 2023, arXiv preprint arXiv:2303.18223.

[5] H. Liang, Q. Sang, C. Hu, D. Cheng, X. Zhou, D. Wang, W. Bao, Y. Wang, DNN surgery: Accelerating DNN inference on the edge through layer partitioning, IEEE Trans. Cloud Comput. 11 (3) (2023) 3111–3125.

[6] L. Zeng, X. Chen, Z. Zhou, L. Yang, J. Zhang, Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices, IEEE/ACM Trans. Netw. 29 (2) (2020) 595–608.

[7] C. Ding, A. Zhou, Y. Liu, R.N. Chang, C.-H. Hsu, S. Wang, A cloud-edge collaboration framework for cognitive service, IEEE Trans. Cloud Comput. 10 (3) (2022) 1489–1499.

[8] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: A survey, Int. J. Comput. Vis. 129 (6) (2021) 1789–1819.

[9] S.D. Alizadeh Javaheri, R. Ghaemi, H. Monshizadeh Naeen, An autonomous architecture based on reinforcement deep neural network for resource allocation in cloud computing, Computing 106 (2) (2024) 371–403.

[10] M. Xu, C. Song, H. Wu, S.S. Gill, K. Ye, C. Xu, esDNN: deep neural network based multivariate workload prediction in cloud computing environments, ACM Trans. Internet Technol. (TOIT) 22 (3) (2022) 1–24.

[11] H. Li, X. Li, Q. Fan, Q. Xiong, X. Wang, V.C. Leung, Transfer learning for real-time surface defect detection with multi-access edge-cloud computing networks, IEEE Trans. Netw. Serv. Manag. 21 (1) (2023) 310–323.

[12] W. Tang, Q. Yang, X. Hu, W. Yan, Deep learning-based linear defects detection system for large-scale photovoltaic plants based on an edge-cloud computing infrastructure, Sol. Energy 231 (2022) 527–535.

[13] Y. Wu, J. Wang, Y.-Q. Chen, An efficient defect detection system for printed circuit boards with edge-cloud fusion computing, in: 2021 3rd International Conference on Industrial Artificial Intelligence, IAI, IEEE, 2021, pp. 1–6.

[14] D. Yang, K. Zheng, S. Qian, Q. Hua, K. Zhang, J. Cao, G. Xue, Mitigating interference of microservices with a scoring mechanism in large-scale clusters, J. Supercomput. 81 (1) (2025) 1–31.

[15] Q. Hua, D. Yang, S. Qian, J. Cao, G. Xue, M. Li, Humas: A heterogeneity-and upgrade-aware microservice auto-scaling framework in large-scale data centers, IEEE Trans. Comput. 74 (3) (2025) 968–982.

[16] M. Alnemari, N. Bagherzadeh, Efficient deep neural networks for edge computing, in: 2019 IEEE International Conference on Edge Computing, EDGE, IEEE, 2019, pp. 1–7.

[17] J. Mills, J. Hu, G. Min, Multi-task federated learning for personalised deep neural networks in edge computing, IEEE Trans. Parallel Distrib. Syst. 33 (3) (2021) 630–641.

[18] S. Song, J. Jing, Y. Huang, M. Shi, EfficientDet for fabric defect detection based on edge computing, J. Eng. Fibers Fabr. 16 (2021) 15589250211008346.

[19] Z. Zhu, G. Han, G. Jia, L. Shu, Modified densenet for automatic fabric defect detection with edge computing for minimizing latency, IEEE Internet Things J. 7 (10) (2020) 9623–9636.

[20] X. Li, W. Li, Q. Yang, W. Yan, A.Y. Zomaya, Edge-computing-enabled unmanned module defect detection and diagnosis system for large-scale photovoltaic plants, IEEE Internet Things J. 7 (10) (2020) 9651–9663.

[21] W. Fan, P. Chen, X. Chun, Y. Liu, MADRL-based model partitioning, aggregation control, and resource allocation for cloud-edge-device collaborative split federated learning, IEEE Trans. Mob. Comput. 24 (6) (2025) 5324–5341.

[22] C. Zhang, J. Chen, W. Li, H. Sun, Y. Geng, T. Zhang, M. Ji, T. Fu, A cloud-edge collaborative task scheduling method based on model segmentation, J. Cloud Comput. 13 (1) (2024) 81.

[23] Z. Tao, Q. Xia, S. Cheng, Q. Li, An efficient and robust cloud-based deep learning with knowledge distillation, IEEE Trans. Cloud Comput. 11 (2) (2022) 1733–1745.

[24] T. Liang, M. Wang, J. Chen, D. Chen, Z. Luo, V.C. Leung, Compressing the multiobject tracking model via knowledge distillation, IEEE Trans. Comput. Soc. Syst. 11 (2) (2023) 2713–2723.

[25] Z. Pei, X. Yao, W. Zhao, B. Yu, Quantization via distillation and contrastive learning, IEEE Trans. Neural Netw. Learn. Syst. 35 (12) (2023) 17164–17176.

[26] H. Zhang, L. Liu, Y. Zhang, X. Lei, F. Hui, B. Wen, DenseKD: dense knowledge distillation by exploiting region and sample importance, IEEE Trans. Neural Netw. Learn. Syst. 36 (6) (2025) 11243–11257.

[27] C. Yang, H. Zhou, Z. An, X. Jiang, Y. Xu, Q. Zhang, Cross-image relational knowledge distillation for semantic segmentation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12319–12328.

[28] R. Mishra, H.P. Gupta, Designing and training of lightweight neural networks on edge devices using early halting in knowledge distillation, IEEE Trans. Mob. Comput. 23 (5) (2023) 4665–4677.

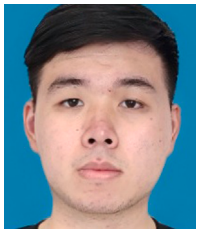
[29] S. Zhao, X. Wang, X. Wei, Mitigating accuracy-robustness trade-off via balanced multi-teacher adversarial distillation, IEEE Trans. Pattern Anal. Mach. Intell. 46 (12) (2024) 9338–9352.

[30] Z. Li, X. Li, L. Yang, B. Zhao, R. Song, L. Luo, J. Li, J. Yang, Curriculum temperature for knowledge distillation, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 1504–1512, 2.

[31] Y. Wei, Y. Bai, Dynamic temperature knowledge distillation, 2024, arXiv preprint arXiv:2404.12711.

[32] Z. Zhang, Y. Zhou, J. Gong, J. Liu, Z. Tu, Instance temperature knowledge distillation, 2024, arXiv preprint arXiv:2407.00115.

[33] X. Chen, T. Du, M. Wang, T. Gu, Y. Zhao, G. Kou, C. Xu, D.O. Wu, Towards optimal customized architecture for heterogeneous federated learning with contrastive cloud-edge model decoupling, IEEE Trans. Comput. 74 (4) (2025) 1123–1137.
[34] Y. Wang, Z. Yu, J. Wu, C. Wang, Q. Zhou, J. Hu, Adaptive knowledge distillation-based lightweight intelligent fault diagnosis framework in IoT edge computing, IEEE Internet Things J. 11 (13) (2024) 23156–23169.
[35] Y. Laili, F. Guo, L. Ren, X. Li, Y. Li, L. Zhang, Parallel scheduling of large-scale tasks for industrial cloud–edge collaboration, IEEE Internet Things J. 10 (4) (2021) 3231–3242.
[36] Y. Tian, Z. Zhang, Y. Yang, Z. Chen, Z. Yang, R. Jin, T.Q. Quek, K.-K. Wong, An edge-cloud collaboration framework for generative ai service provision with synergetic big cloud model and small edge models, IEEE Netw. 38 (5) (2024) 37–46.
[37] H. Jin, Y. Wu, Ce-collm: Efficient and adaptive large language models through cloud-edge collaboration, in: 2025 IEEE International Conference on Web Services, ICWS, IEEE, 2025, pp. 316–323.
[38] S. Wang, Z. Zheng, X. Wang, Q. Zhang, Z. Liu, A cloud-edge collaboration framework for cancer survival prediction to develop medical consumer electronic devices, IEEE Trans. Consum. Electron. 70 (3) (2024) 5251–5258.
[39] W. Fan, Hybrid deep reinforcement learning-based task offloading for D2D-assisted cloud-edge-device collaborative networks, IEEE Trans. Mob. Comput. 23 (12) (2024) 13455–13471.
[40] Z. Song, W. Chen, T. Gong, S. Rani, W. Wei, G. Feng, Cloud-edge collaborative computing for consumer electronics via deep reinforcement learning, IEEE Trans. Consum. Electron. 71 (2) (2025) 4120–4129.
[41] P. Guo, J. Xiong, Y. Wang, X. Meng, L. Qian, Intelligent scheduling for group distributed manufacturing systems: Harnessing deep reinforcement learning in cloud-edge cooperation, IEEE Trans. Emerg. Top. Comput. Intell. 8 (2) (2024) 1687–1698.
[42] J. Jiang, Q. Li, P. Wang, Y. Liu, DRKC: Deep reinforcement learning enhanced microservice scheduling on kubernetes clusters in cloud-edge environment, IEEE Trans. Cloud Comput. 13 (4) (2025) 1472–1486.
[43] J. Cai, W. Liu, Z. Huang, F.R. Yu, Task decomposition and hierarchical scheduling for collaborative cloud-edge-end computing, IEEE Trans. Serv. Comput. 17 (6) (2024) 4368–4382.
[44] X. Zheng, Y. Li, B. Zheng, C. Zhang, L. Zhu, EdgeNetLLM: Cloud-edge collaborative adaptation of large language models for mobile networking, IEEE Trans. Netw. Sci. Eng. 13 (2026) 3928–3943.
[45] Z. Wu, C. Liu, J. Sun, J. Hu, Z. Wei, J. Plaza, A. Plaza, Hyperspectral and multispectral image fusion for remotely sensed target detection: A new cloud-edge collaborative approach, IEEE Trans. Geosci. Remote Sens. 63 (2025) 5509613.
[46] T. Zeng, X. Zhang, J. Duan, C. Yu, C. Wu, X. Chen, An offline-transfer-online framework for cloud-edge collaborative distributed reinforcement learning, IEEE Trans. Parallel Distrib. Syst. 35 (5) (2024) 720–731.
[47] Z. Chai, Y. Lin, Z. Gao, X. Yu, Z. Xie, Diffusion model empowered efficient data distillation method for cloud-edge collaboration, IEEE Trans. Cogn. Commun. Netw. 11 (2) (2025) 902–913.
[48] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks? Adv. Neural Inf. Process. Syst. 27 (2014) 3320–3328.



Jiaqi Fei was born in Suzhou, Jiangsu, P.R. China. He received the master's degree with the School of Electronics and Information, Shanghai Dianji University, China, in 2026. His research interests include distributed machine learning, neural networks and deep learning, edge computing and scheduling algorithms.

Mengdie Qin was born in Shangqiu, Henan, P.R. China. She is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. Her research interests include distributed machine learning, neural networks and deep learning, edge computing and scheduling algorithms.

Xiaogang Wang (IEEE Member) was born in Nanchang, Jiangxi, P.R. China. He received the Ph.D. degree in computer science and technology from Shanghai Jiao Tong University, China, in 2018. He is currently a Professor with the School of Electronics and Information, Shanghai Dianji University, China. He was also the Visiting Research Scholar with the CLOUDS Laboratory, University of Melbourne, Australia, from 2019 to 2020. He has published more than 50 papers in some journals and conferences such as TC, TSC, JSS, KBS, CN, FGCS, CC, APIN, WI-IAT, APSCC. His research interests include distributed machine learning, neural networks and deep learning, cloud and edge computing, scheduling algorithms. He is a member of the China Computer Federation.

Hui Guo was born in Zhumadian, Henan, P.R. China. She received the master's degree with the School of Electronics and Information, Shanghai Dianji University, China, in 2026. Her research interests include distributed machine learning, neural networks and deep learning, scheduling algorithms and edge computing.

Ziqi Zhu was born in Shuyang, Jiangsu, P.R. China. She is currently pursuing the master's degree with the School of Electronics and Information, Shanghai Dianji University, China. Her research interests include distributed machine learning, neural networks and deep learning, edge computing.

Jian Cao (IEEE Senior Member) was born in Yixing, Jiangsu, P.R. China. He received the Ph.D. degree from the Nanjing University of Science and Technology, in 2000. He is currently a professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His main research interests include service computing, cloud computing, network computing and scheduling algorithms, cooperative information systems and software engineering. He has published more than 200 papers in prestigious journals. He is a distinguished member of the China Computer Federation.

Rajkumar Buyya (IEEE Fellow, ACM Fellow) received the Ph.D. degree in computer science and software engineering from Monash University, Melbourne, Australia, in 2002. He is a Redmond Barry distinguished professor and director of the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, the University of Melbourne, Australia. He served as a Future fellow of the Australian Research Council during 2012–2016. He has authored more than 850 publications and seven textbooks. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=181, g-index=394, 171,000+ citations). He served as the founding editor-in-chief of the IEEE Transactions on Cloud Computing. He is currently serving as co-editor-in-chief of Journal of Software: Practice and Experience.