A Lightweight Collaborative Deep Neural Network for the Mobile Web in Edge Cloud

Yakun Huang, Xiuquan Qiao, Pei Ren, Ling Liu, *Fellow, IEEE*, Calton Pu, *Fellow, IEEE*, Schahram Dustdar, *Fellow, IEEE*, and Junliang Chen

Abstract—Enabling deep learning technology on the mobile web can improve the user's experience for achieving web artificial intelligence in various fields. However, heavy DNN models and limited computing resources of the mobile web are now unable to support executing computationally intensive DNNs when deploying in a cloud computing platform. With the help of promising edge computing, we propose a lightweight collaborative deep neural network for the mobile web, named LcDNN, which contributes to three aspects: (1) We design a composite collaborative DNN that reduces the model size, accelerates inference, and reduces mobile energy cost by executing a lightweight binary neural network (BNN) branch on the mobile web. (2) We provide a jointly training method for LcDNN and implement an energy-efficient inference library for executing the BNN branch on the mobile web. (3) To further promote the resource utilization of the edge cloud, we develop a DRL-based online scheduling scheme to obtain an optimal allocation for LcDNN. The experimental results show that LcDNN outperforms existing approaches for reducing the model size by about 16x to 29x. It also reduces the end-to-end latency and mobile energy cost with acceptable accuracy and improves the throughput and resource utilization of the edge cloud.

Index Terms—Collaborative DNNs, mobile web, binary neural network, dynamic allcoation, edge computing

1 Introduction

With the help of some major technologies such as Artificial Intelligence (AI), Virtual Reality (VR) and Augmented Reality (AR), lightweight and cross-platform web applications are growing at a rapid rate with a focus on enhancing user experience [2], [3]. Deep learning (e.g., Deep Neural Networks, DNNs) is currently a representative way of achieving Web-based Artificial Intelligence (Web AI), which has set new records in accuracy for many important problems, such as image recognition [4], speech recognition [5], recommender systems [6], natural language processing [7] etc. However, the web is now unable to support advanced DNNs in a performant manner due to limited computing resources of the web and lack of optimized low-level APIs [8]. Additionally, heavy DNN models and intensive computations also introduce a high inference latency, including loading models and inefficient inference. Thus, the web community group of World Wide Web Consortium (W3C) is actively promoting innovation, and research of deep learning web technologies to enable the creation of deep learning web experiences about the intersection of augmented reality, deep learning, and the web, or more simply the augmented web [9], [10].

To achieve the full promise of Web AI for the mobile web, the majority of existing attempts under the cloud computing platform take one of the following three approaches to improve user experience (Fig. 1). The first approach is remote execution, which takes the advantage of resource-rich infrastructure by offloading the whole DNN computation to the remote cloud. With this approach, large amounts of data (e.g., images, audio, and videos) are sent to the cloud via the wireless network, resulting in high transmission latency and mobile energy consumption. Moreover, offloading all computations to the remote cloud may greatly increase the computational pressure and cost of the remote cloud.

The second approach is mobile-only, which performs all computations on the mobile web via JavaScript and WebAssembly [11]. However, heavy DNN models lead to high transmission latency and mobile energy consumption (e.g., Tensorflow.js's ResNet50[12] is a deep learning model, whose size can be up to 97.8 MB) [13]. Besides, executing DNNs on the mobile web performs worse than that of app-based applications due to weak computing capability (i.e., the mobile device performs poor compatibility to use WebGPU compared with the personal computer for web applications currently). Hence, it takes more time and mobile energy to execute computationally intensive DNN inference on the mobile web. The third approach is partition-offloading, which dynamically distributes the computation between the mobile web and the remote cloud, that is to partition and perform the computations that can be done within the mobile web, reducing the computing pressure of the remote cloud [14], [15], [16]. Although a small portion of DNN computations can be offloaded to the mobile web, the cloud server still has to undertake the most of DNN

- *Yakun Huang, Xiuquan Qiao, Pei Ren, and Junliang Chen are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China E-mail: {ykhuang, qiaoxq, renpei, chjl}@bupt.edu.cn.*
- *Ling Liu and Calton Pu are with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332 USA E-mail: {ling.liu, calton.pu}@cc.gatech.edu.*
- *Schahram Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at.*

Manuscript received 1 Feb. 2020; revised 5 Sept. 2020; accepted 2 Dec. 2020. Date of publication 8 Dec. 2020; date of current version 3 June 2022. (Corresponding authors: Xiuquan Qiao.) Digital Object Identifier no. 10.1109/TMC.2020.3043051

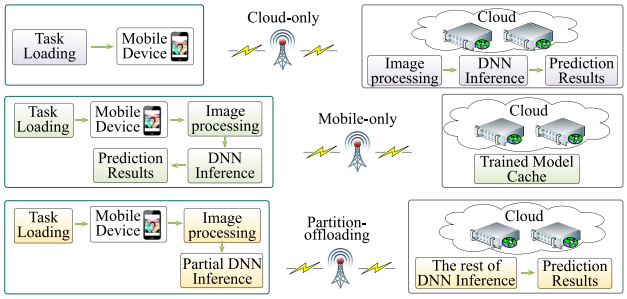


Fig. 1. Cloud-only, mobile-only and partition-offloading.

computations, which means that additional costs are still needed to improve the throughput of the remote cloud, especially in high concurrent requests. In other words, this approach improves the utilization of mobile web computing resources, but it is still challenging to improve the throughput of the remote cloud effectively.

As mobile edge computing (MEC) is becoming an important computing infrastructure in 5G era, it is promising to consider the use of the edge cloud that has the benefit of low communication costs compared to offloading computations to the remote cloud and relieves the burdens of the core network [17], [18]. However, high DNN execution latency and low throughput have still existed when leveraging edge clouds instead of remote clouds for providing Web AI services. In addition, employing edge computing infrastructure for Web AI service introduces some new issues such as limited computing resources and weak resource utilization of edge clouds. Since the edge cloud is deployed near the base station with common servers and uses virtualization technology to provide web services, computing resources (CPU, memory, hard disk) of the edge cloud is much limited than that of the remote cloud facing with cluster servers. Also, the edge cloud's service scalability is much worse than that of the remote cloud, making it difficult to provide resilient service with high concurrent requests. Thus, employing this kind of edge computing infrastructure to support executing DNNs for Web AI services is still challenging for a number of reasons. These include the following:

- *No lightweight and collaborative DNNs can be executed on the mobile web efficiently.* Since most well-performing DNNs have deeper layers and large amounts of parameters, it is impractical to execute such heavy DNNs on the mobile web with limited computing resources. Although some lightweight compressed DNNs have been successfully operated in the mobile device, they require extensive expert experience to design ingenious and reasonable DNN structures. Besides, existing compressed DNNs lack good performance in inference efficiency, especially for mobile web platforms. Additionally, there is currently no heterogeneous DNN inference framework that can perform collaborative DNN between the mobile web and the edge server.
- *Existing partition-offloading approaches are failed to perform high throughput for the mobile web.* One reason is that the limited computing resource of the edge server performs worse on dealing with high concurrent requests, resulting in low throughput even appearing an unavailable service. Another reason is that the partition-offloading mechanism requires the mobile web and the edge server to participate in every task request, which is not much different from the cloud-only in the way of providing web service. This indicates that existing partition-offloading approaches cannot improve the throughput effectively, but only reduces partial computing pressure of edge clouds.
- *No online scheduling scheme for integrating multiple edge servers and improving computing resources can provide resilient web services in edge clouds.* Instant request characteristics of mobile web applications introduce occasional high concurrent requests, which can cause edge servers deployed in a base station to be overwhelmed. While deploying a lot of spare edge server resources may make them idle in most cases, which is also an expensive cost, and unacceptable for network operators in practice. Therefore, the isolated edge server provisioning mechanism is difficult to cope with high concurrency processing, thus needs to establish a resilient scheduling scheme that fully utilizes the computing resources of various bases station in edge clouds.

To address these concerns, we design a composite deep neural network, named LcDNN by introducing a binary neural network as the side branch to acquire an independent lightweight DNN. First, when compared with existing DNN compressing methods such as depthwise separable convolution [19], [20], knowledge distilling [21], [22] and network prunning [23], [24], LcDNN achieves better performance in both compressing and inference efficiency, requiring less computing resources and mobile energy. Second, to solve the contradiction between the model size and accuracy in compressing DNNs, LcDNN employs the full-precision network as the backbone to guide the design of the binary neural network (BNN) branch which quantifies and compresses the DNN for executing on the mobile web without rich expert experience and knowledge. More importantly, the full-precision backbone network is to provide accuracy compensation when the BNN branch has accuracy loss facing with complex tasks. In other words, this collaborative compensation mechanism allows the BNN branch to focus more on the simple tasks, thereby implementing a lightweight, efficient, and low-energy inference on the mobile web. This also illustrates that adding an efficient lightweight branch is more suitable for the mobile web than directly compressing DNNs. Last, to further improve the inference efficiency of LcDNN on the mobile web, we have implemented an efficient inference library based on C++ and then converted them to JavaScript and WASM scripts to execute on the mobile web with optimizations on latency and mobile energy. The biggest difference between our inference library and existing web-oriented inference libraries is that it supports the BNN branch inference and distributed collaborative inference for LcDNN, reducing the accuracy loss of complex tasks, optimizing and accelerating the BNN branch inference on the mobile web. In addition, to deploy LcDNN in real scenarios, schedule dynamic task requests, and provide maximum resource utilization of the edge cloud, we develop a DRL-based scheduling scheme for LcDNN to serve for multiple edge

servers. Specifically, we create a reinforcement learning framework using DNN as a function appropriator to solve dynamic task requests in edge clouds, called DRLoS. In this way, DRLoS can provide global scheduling with self-learning capability under dynamic user requests via adding a buffer queue on each server. For evaluation, we conduct comprehensive experiments on different DNNs and datasets, the results show that LcDNN improves latency performance, reduces average mobile energy, and improves the throughput of edge clouds. Especially in a scenario with three edge servers, DRLoS's online scheduling can bring LcDNN more than 2.17x resource utilization improvement. The contributions of this work can be summarized as follows:

- Designing a composite lightweight DNN with a BNN branch to execute DNN inference on the mobile web, which can reduce latency and mobile energy, and improve the throughput of edge clouds.
- Proposing a jointly training method for LcDNN and implementing an energy-efficient inference library for the BNN branch on the mobile web, which also provides collaboration with the edge server for accuracy compensation.
- Developing a DRL-based online scheduling scheme to maximize the resource utilization of the edge cloud for LcDNN, which also provides resilient web AI services for mobile web applications in real scenarios.

2 Background and Motivation

In this section, we briefly introduce the background of edge computing for the mobile web, describe key properties of executing DNNs on the mobile web compared with the mobile device, and discuss the collaborative DNN technology that motivates us.

2.1 Edge Computing for the Mobile Web

We present typical scenarios and comparisons when applying mobile web applications between cloud computing and edge computing in Fig. 2. Cloud computing is limited by high transmission delay, which also introduces computational pressure and cost of cloud servers, and discourages service providers to invest large computing resource for developing mobile web applications. With the rapid development of 5G networks, edge computing, as the basic infrastructure, has the benefits of high network bandwidth and low communication costs.

As shown in Fig. 2, the transmission delay of cloud computing is nearly 10 times higher than that of edge computing, whether accessing with 2.4G or 5G. Besides, edge servers are deployed at the base station near mobile users, which can distribute computations to various regions instead of being centralized in one region. Hence, with the help of edge computing, we are promising to introduce AI technology, which is computationally intensive and time-consuming on transmission, into mobile web applications with a satisfactory experience. However, the computational cost is still expensive for service providers, and computing pressure is still high for edge computing platforms. Thus, it is natural to take full advantage of the computing capability of the mobile device, then reducing the computing pressure and cost of the server.

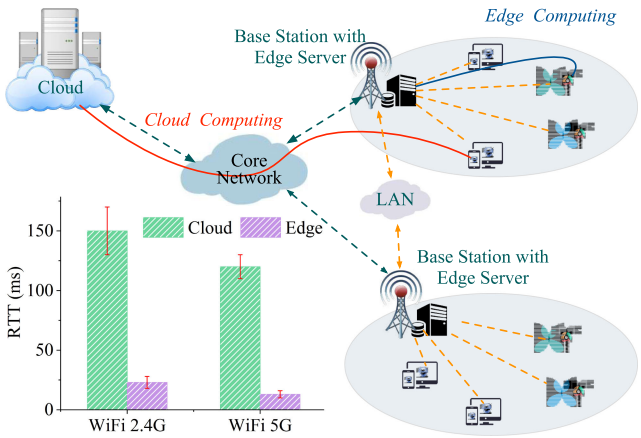


Fig. 2. Typical scenarios and comparisons between cloud computing and edge computing.

2.2 Executing DNNs on the Mobile Web

To efficiently execute DNNs on the mobile web for providing Web AI services, we point out the following key properties that mobile web required when comparing with executing DNNs on the mobile device directly.

- *Installation-free and deployment-free.* Generally, DNN models can be embedded in app-based applications and be installed in a mobile device, which performs DNN inference locally without communication with edge servers. However, it is unrealistic to use the same deployment for mobile web applications, which provides service through instant loading.
- *Weak computing capability.* Mobile web applications usually run in various browsers, whose computing capability is currently performed by JavaScript and accelerated by WebAssembly. However, it performs poor compatibility to use the WebGPU currently, which is now supported for computers or app-based applications better than that of the mobile web. In other words, the computing capability of the mobile web is much weaker than that of running on the mobile device.
- *Frequent communication.* Instant loading characteristic of the mobile web introduces frequent communication in existing approaches, including mobile-only, cloud-only, and partition-offloading. Specifically, mobile-only requires loading the whole DNN models onto the mobile web, cloud-only requires transmitting tasks from the mobile web to the remote cloud, and partition-offloading has to load partial DNN models and transmit intermediate results between the mobile web and the remote cloud.
- *Cross-platform.* To better play out the cross-platform feature of the mobile web, mainstream browsers (e.g., Chrome, Firefox, IE, and Safari) and embedded APP browsers (e.g., web browser in WeChat) require supporting DNN execution and show effective compatibility. This indicates that we can only use technologies supported by mainstream browsers such as JavaScript and WebAssembly when executing DNNs on the mobile web.

2.3 DNNs Acceleration and Motivation

Compressing DNNs and adding early exit branches are two representative methods for accelerating DNN inference [19], [20], [25]. Although the compressing method makes DNNs smaller and faster and has been successfully applied to app-based applications on the mobile device, it requires rich experience in DNNs design and efficient training to acquire lightweight DNNs. DNN compression methods such as the use of deep separable convolution, usually require expert experience and reasonable structural design to obtain a lightweight DNN model with a low loss in accuracy. For executing DNNs on the mobile web, it requires a smaller and faster DNN model than that of the mobile device, which causes a large loss in accuracy. Thus, a simple compressing method is sometimes difficult to meet the needs of mobile web applications. BranchyNet [25], as a typical DNNs with early exit branches, accelerates DNN inference via adding branches into the original DNN. The extra branches allow the majority of tasks to exit early, thus reducing the needless inference. However, since BranchyNet adds multi branches into the main branch, which causes the inference to be more time-consuming facing complex tasks.

Inspired by the advantages of the mentioned methods, in this paper, we design a lightweight collaborative DNN for the mobile web, which uses the feature of an early exit and the structure of lightweight and fast DNN. Besides, it can be efficiently executed on the mobile web, easily collaborated with the edge server, and enhance the user's experience.

3 LcDNN for the Mobile Web in Edge Clouds

3.1 Design of LcDNN

In this section, we design a composite DNN with a BNN branch, which can handle DNN tasks independently on the mobile web. We also provide a jointly training method consisting of training the main branch, binarizing inputs and weight filters, and training the BNN branch. Since there is no existing inference library for the BNN branch on the mobile web, we are first to implement an inference library by transforming and optimizing the original BNN into the JavaScript library for the mobile web. This inference library provides the capability to run the BNN branch on the mobile web, which also has a collaborative mechanism with the edge server.

To better understand the difference and novelty of LcDNN against existing methods, we introduce the design and an efficient inference library of LcDNN in detail. Although binary neural networks (BNNs) have been widely investigated with the advantage of efficient inference, they perform worse in accuracy compared with other compressing methods. Instead of directly applying BNNs to the mobile web, LcDNN puts the BNN as an independent branch and proposes a composite network structure with the following advantages and novelties: (1) From the perspective of the network structure, LcDNN differs from traditional BNNs in that it integrates the quantized and compressed network as a collaborative branch into the full-precision network for joint training, rather than performing quantizing and compressing on the full-precision network directly. (2) For the inference accuracy, LcDNN uses a full-precision backbone network as accuracy compensation facing with complex tasks, and thus performing better than traditional compressed DNNs including BNNs. This also shows that although it has excellent inference efficiency by directly employing the BNN on the mobile web, it is hard to be accepted due to excessive loss in accuracy. (3) Besides, LcDNN's collaborative inference library is optimized and compiled for the mobile web platform, while the traditional inference library for the BNN is mainly for embedded platforms. Therefore, inference efficiency and mobile energy consumption of LcDNN are better than that of directly using the BNN on the mobile web. In summary, LcDNN effectively combines the advantages of the BNN, while providing a compound collaborative mechanism to compensate for the performance of the independent BNN branch when the accuracy is insufficient.

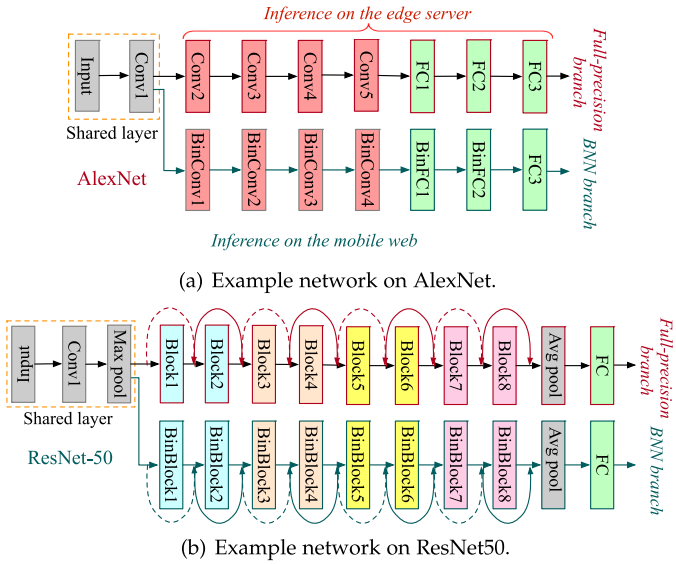


Fig. 3. The architecture of LcDNN.

3.1.1 The Architecture of LcDNN

LcDNN adds one BNN branch into the original full-precision network, which is the benefit of collaboration between the mobile web and the edge server. This architecture can also promote DNN inference efficiency, reduce mobile energy cost, and improve the system throughput.

We take AlexNet [4] and ResNet50 as examples that both the full-precision branch and the BNN branch share the first convolutional layer in Fig. 3. For one thing, if we binarize all layers in the BNN branch, which may introduce a dramatic loss of classification accuracy with few reductions of model size. For another, when the classification accuracy of the BNN branch cannot satisfy the tasks, we only need to transmit intermediate outputs of the shared layer to the edge server rather initial tasks, which also protects the privacy of mobile users. Taking the advantage of such a shared structure, the edge server can provide accuracy compensation for the BNN branch. Note that we have to store intermediate outputs of the first convolutional layer for the collaboration. We can free the memory of intermediate outputs when the BNN branch has the confidence for tasks. Otherwise, the mobile web frees them after sending them to the edge server.

3.1.2 Jointly Training Method for LcDNN

We use AlexNet as an example of image classification and the softmax cross-entropy loss function is used as the optimization objective. To train LcDNN, we aim to minimize the loss function of all branches that forms a joint optimization problem as a sum of the loss functions of the full-precision branch and the BNN branch.

$$\mathcal{L}(\hat{y}, y; \theta) = \mathcal{L}_{full}(\hat{y}_{full}, y; \theta) + \mathcal{L}_{BNN}(\hat{y}_{BNN}, y; \theta). \tag{1}$$

Where y is a one-hot ground-truth label vector, $\hat{y}$ is the predicted label vector. For loss function of full-precision branch, $\mathcal{L}_{full}(\hat{y}_{full}, y; \theta)$ can be represented as

$$\mathcal{L}_{full}(\hat{y}_{full}, y; \theta) = -\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} y_c \log \hat{y}^c_{full}. \tag{2}$$

$$\hat{y}_{full} = \text{softmax}(z) = \frac{e^z}{\sum_{c\in\mathcal{C}} e^z}. \tag{3}$$

Where $z = f_{full}(x; \theta)$ denotes the outputs of full-precision branch with input sample x. $\mathcal{C}$ represents the set of all possible labels. Hence, we are easy to execute feedforward pass and backward propagation with gradient descent.

For the BNN branch, we first binarize weights of the neural network and estimate binary weights. During the updating of parameters, we use high precision weights to prevent gradient disappears of binarization. The forward propagation of the training phase is similar to that of standard forward propagation except for convolutional layers, which is computed by

$$I \times W \approx (\text{sign}(I) \otimes \text{sign}(W)) \odot K \cdot \alpha. \tag{4}$$

The following equation exhibits the core convolutional operation, which approximates the convolution between input I and weight filter W using binary operations. Here, α is the scaling factor for the weight and K denotes all subtensors.

During the backward propagation, we follow the same approach as [25] to compute the gradient for $\text{sign}(x)$.

$$\frac{\partial \text{sign}}{\partial x} = x_{1_{|x|\leq 1}}. \tag{5}$$

Thus, we calculate the gradient in backward propagation after the scaled sign function is

$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial \widetilde{W}_i}\left(\frac{1}{n} + \frac{\partial \text{sign}}{\partial W_i}\alpha\right), \tag{6}$$

where gradients are computed with respect to the estimated weight filters $\widetilde{W}_i$.

To train LcDNN, we calculate training datasets through the whole network including the full-precision branch and the BNN branch. LcDNN binarizes both inputs and weights across the BNN branch, which introduces the factor α to approximate with traditional feedforward of the typical convolutional layer. Besides, we record the outputs from exit points of all branches for calculating the error of the joint network. As for the backward propagation, we update the weights by the error passed back through the full-precision branch. Due to tiny changes of parameters in gradient descent, we only binarize the weights during the forward pass and backward propagation, which uses high precision weights to update parameters that are also employed in [26], [27] and [28]. We present the whole training process of a N-Layers LcDNN in Algorithm 1, which consists of training the full-precision branch, binarizing weight filters, and training the BNN branch. Note that we only perform forward propagation with binarized weights during the BNN branch inference without keeping the full precision weights.

Algorithm 1. Training a N-Layers LcDNN

Input: Inputs and labels X, Y, loss function $\mathcal{L}(\hat{Y}, Y)$, layer weights of full-precision branch W^l_{full}, learning rate of full-precision branch η^l_{full}, layer weights of the BNN branch W^l_{BNN}, learning rate of the BNN branch η^l_{BNN}.

Output: Updated weights $W^{l+1}_{full}, W^{l+1}_{BNN}$ and learning rate $\eta^{l+1}_{full}, \eta^{l+1}_{BNN}$.

/* Training full-precision branch */

$\hat{Y}_{full} \Leftarrow StandardForward(X, W_{full})$;

$\frac{\partial \mathcal{L}}{W^l_{full}} \Leftarrow StandardBackward(\frac{\partial \mathcal{L}}{\hat{Y}_{full}}, W^l_{full})$;

$W^{l+1}_{full} \Leftarrow Update(W^l_{full}, \frac{\partial \mathcal{L}}{W^l_{full}}, \eta^l_{full})$;

$\eta^{l+1}_{full} \Leftarrow Update(\eta^l_{full}, l)$;

/* Training the BNN branch */

//Binarizing weights from the 2th convolutionl layer;

for n from 2 to N **do**

$\widetilde{W}^l_n \Leftarrow \frac{1}{n}\left\|W^l_n\right\|_{\ell 1} \cdot sign(W^l_n)$;

$\hat{Y}_{BNN} \Leftarrow BinaryForward(X, sign(W^l), \frac{1}{n}\left\|W^l_n\right\|_{\ell 1})$;

$\frac{\partial \mathcal{L}}{\widetilde{W}^l} \Leftarrow BinaryBackward(\frac{\partial \mathcal{L}}{\hat{Y}_{BNN}}, \widetilde{W}^l)$;

$W^{l+1}_{binary} \Leftarrow Update(W^l_{BNN}, \frac{\partial \mathcal{L}}{\widetilde{W}^l}, \eta^l_{BNN})$;

$\eta^{l+1}_{BNN} \Leftarrow Update(\eta^l_{BNN}, l)$;

return $W^{l+1}_{full}, \eta^{l+1}_{full}, W^{l+1}_{BNN}, \eta^{l+1}_{BNN}$;

3.1.3 Collaborative DNN Inference Between the Mobile Web and the Edge Server

Once the joint network is trained, LcDNN can load and execute inference efficiently because of tiny BNN branch. For a given sample x, if the BNN branch is confident to predict the task and satisfy users, the sample can exit from the tiny BNN branch directly. Otherwise, we have to transfer the output of the first convolutional layer to the edge server for a precise result. Thus, to measure the classification accuracy of the BNN branch, we introduce normalized entropy, $S(x)\in[0, 1]$ for the exit point of the binary branch which is also employed in [28], where $\mathcal{C}$ is the set of labels.

$$S(\mathbf{x}) = -\sum_{i=1}^{|\mathcal{C}|}\frac{x_i \log x_i}{\log |\mathcal{C}|}. \tag{7}$$

Then, we compared $S(\mathbf{x})$ against a collaborative threshold to determine whether or not to exit from the BNN branch. In practice, the optimal value of the exit threshold for the BNN branch depends on networks and datasets. To obtain optimal τ for the highest overall accuracy, a simple way is to search the ranges of τ and pick the value that can acquire a maximum accuracy in LcDNN. Generally, we choose multiple validation sets randomly, set training

iteration as 600, and define 0.5 as the initial τ. Then we start to search in the range of [0,1] at different searching rates of 0.1, 0.01, 0.001, 0.0001, and 0.00001 until optimal τ value is obtained and the searching tends to be convergent. We describe the whole process of searching τ in Algorithm 2.

Algorithm 2. Seaching for Optimal τ Value

Input: Validation sets V, initial $\tau = 0.5$, searching rate $\Delta S = \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$.
Output: Optimal result of τ.
$\tau \Leftarrow 0.5$;
for each sr in ΔS **do**
/* Search rate from big to small */
  for each *iteration* **do**
    for each v *in* V **do**
      $ns \Leftarrow S(x)$; // By Eq. (7)
      $ba \Leftarrow BinaryFowrwad(v)$;
      if $(ns < \tau \;\&\&\; ba = 1) \| (ns > \tau \&\& ba = 0)$ **then**
        $d \Leftarrow -1$;
      $\tau \Leftarrow \tau + (-1) \cdot d \cdot \Delta sr$;
return τ;

During the deployment and execution of LcDNN, mobile web users first request the edge server for the BNN branch model to execute the inference on the mobile web. Then, we determine whether to exit from the BNN branch or send the intermediate results to the edge server against the threshold τ. We also present collaborative inference between the mobile web and the edge server in Algorithm 3 when the BNN branch can not satisfy the user.

Algorithm 3. Collaborative Inference of LcDNN

Input: Input sample x, threshold τ for determining wether to exit.
Output: Predicted result $\hat{y}$.
/* Output of the first Conv layer */
$t \Leftarrow f_{full}^{Conv1}(x)$;
$z_{BNN} \Leftarrow f_{BNN}(t)$;
$\hat{y}_{BNN} \Leftarrow softmax(z_{BNN})$;
$e \Leftarrow S(\hat{y}_{BNN})$;
if $e < \tau$ **then**
  return $\arg\max \hat{y}_{BNN}$;
$z_{full}^{rest} \Leftarrow f_{full}^{rest}(t)$;
$\hat{y}_{full}^{rest} \Leftarrow softmax(z_{full}^{rest})$;
return $\arg\max \hat{y}_{full}^{rest}$;

3.2 Efficient Inference of the BNN Branch on the Mobile Web

Note that existing deep learning frameworks are generally implemented in Python or C++, which does not support the mobile web browser. JavaScript and WebAssembly are dominant methods for executing DNNs on the mobile web. Thus, we implement the LcDNN inference library in C++ and convert them to JavaScript and WASM files to allow executing the BNNs on the mobile web browser directly. We present the whole process for executing the BNN branch of LcDNN on the mobile web in Fig. 4.

To provide an energy-efficient DNN inference with low latency on the mobile web, we optimize the inference of the BNN branch, including convolutional layers and fully connection layers, which effectively improves the inference efficiency and reduces the energy consumption. Once LcDNN is trained in Python (e.g., Pytorch), inference scripts that implemented in C++ convert the first convolutional layer and binary convolutional layers into JavaScript and WASM by Emscripten [29]. We also validate the correctness of our implementation by comparing the outputs with the inference of other frameworks such as Pytorch and TensorFlow.

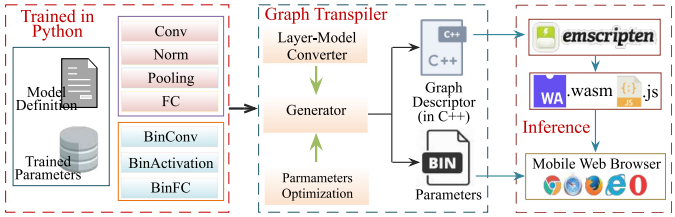


Fig. 4. Fast inference library for the mobile web.

3.3 Resilient Scheduling Scheme for LcDNN

In this section, we formulate web requests scheduling across multiple edge servers and represent it as a Reinforcement Learning (RL) task. We then introduce our DRL-based online scheduling algorithm (DRLoS) facing with high concurrent service of LcDNN in real scenarios.

3.3.1 Online Web Requests Scheduling for LcDNN With DRL

We first outline a typical reinforcement learning framework in Fig. 5, which uses the DNN as a function approximator that has a manageable number of parameters and has been used successfully for large-scale RL tasks [30], [31]. For a given environment in Fig. 5, we assume that there is an agent interacting with it. During each time stamp t, the agent chooses an action a_t through observing the state s_t. By executing the selected action, the state is changed from s_t to s_{t+1}. Also, transitions and rewards of each state are assumed to obey the Markov property [32]. The agent decides actions according to the policy, which is a probability distribution $\pi(s, a)$ and is performed by a deep neural network. Since the optimized objective is to maximize the expected reward, we show the gradient of objective as the follows [32], [33]:

$$\Delta E_\theta[\sum \gamma^t r_t] = E_{\pi_\theta}[\Delta_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]. \tag{8}$$

Where, $E_\theta[\sum \gamma^t r_t]$ is the expected cumulative reward and $\gamma \in [0, 1]$ is to discount reward. $Q^{\pi_\theta}(s, a)$ denotes the expected reward when picks action a in state s. To updates the parameters of deep neural networks for policy distribution, gradient-descent is a popular way [32], [34], which can be described as

$$\theta \Leftarrow \theta + \alpha \sum_t \Delta_\theta \log \pi_\theta(s_t, a_t) v_t, \tag{9}$$

where α is the adjustment size. We then present the design and details for online web requests scheduling for LcDNN with DRL.

(A) DRLoS Model. Assuming that there evenly distributes m edge servers in a service area for mobile web users, which

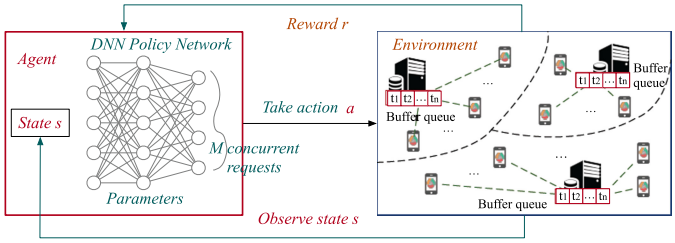


Fig. 5. A typical reinforcement learning framework with policy represented via DNN.

is represented as a vector $e=\{e_1, e_2, \ldots, e_m\}$. We define the task processing capability (i.e., how many web requests can be processed) as $c=\{c_1, c_2, \ldots, c_m\}$. The current processing state of edge servers (i.e., how many tasks are currently processed) at time t can be described as $g_t = \{g_t^1, g_t^2, \ldots, g_t^m\}$. Thus, available computing capability of edge servers at time t can be expressed as $ac_t = \{c_1 - g_t^1, c_2 - g_t^2, \ldots, c_m - g_t^m\}$. Generally, users' requests arrive at nearby edge servers in discrete time steps. Traditional proximity service may result in uneven distribution of computing pressure across edge servers due to the mobility of users. With the help of edge computing infrastructure in 5G, it is promising to offload tasks to idle edge servers via Remote Procedure Call (RPC) with the benefits of improving resource utilization, and increasing the throughput of edge cloud. To this end, DRLoS requires distributing requests to edge servers in balance according to the real-time status of all edge servers and the request status of users. We use the resource usage variance of edge servers as the system objective. Commonly, the resource usage variance of $g_t = \{g_t^1, g_t^2, \ldots, g_t^m\}$ at time t can be calculated as

$$S_t^2 = \frac{\sum_{i=1}^{m}(g_t^i - \bar{g}_t)}{m-1}, \tag{10}$$

where $\bar{g}_t$ denotes mean value of the resource usage of edge servers at time t. A high variance indicates that computing tasks are allocated unevenly, which means that some edge servers are facing high computing pressure while others may be idle.

(B) DRLoS Formulation. To formulate online web requests scheduling as a DRL-based question, we emphasize the state space in Fig. 6.

State Space. We present the state space by describing current resource usage of edge servers and tasks waiting to be scheduled, which is represented by distinct colors.

Practically, each edge server may face a different number of tasks waiting for service at time t. However, a DRL-based system requires a fixed state representation to apply it as the input into a neural network for function approximation. Otherwise, we have to re-train DRLoS for different numbers of task requests. To this end, we add a buffer queue at the edge server before providing service, which selects M tasks as scheduling input of DRLoS according to the timestamp order of requests. Generally, adding a request buffer queue not only makes DRLoS possible to be used in practice but also improves the robustness, especially facing high concurrent requests or malicious requests. Note that when the number of request tasks M_p is less than M, we set $M - M_p$ tasks as $\emptyset$, indicating that these tasks do not need to be processed.

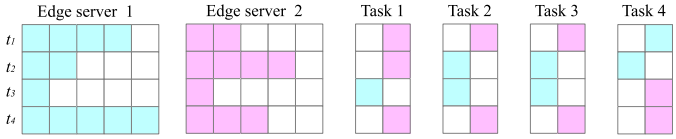


Fig. 6. An example of system state, with two edge servers and four tasks.

Action Space. Since DRLoS requires scheduling M tasks on N edge servers at each point in time, thus this would introduce a large amount of action space of N^M, which is hard to train the neural network of DRLoS. We provide a small action space using a trick by simplifying the actions of edge servers that decide whether to handle a large number of tasks or to handle a small number of tasks based on the current state. For a DRLoS system with three edge servers, the action space can be described as $a = \{< es_1^h, es_2^h, es_3^h >, < es_1^l, es_2^h, es_3^h >, < es_1^h, es_2^l, es_3^h >, < es_1^h, es_2^h, es_3^l >, < es_1^l, es_2^l, es_3^h >, < es_1^l, es_2^h, es_3^l >, < es_1^h, es_2^l, es_3^l >, < es_1^l, es_2^l, es_3^l >\}$. Where $es_i^h = (1-\eta)\cdot(c_i - g_i)$ and $es_i^l = \eta \cdot (c_i - g_i)$ denotes edge servers provide a large or a small portion of available computing resource, respectively. $\eta \in (0, 0.1)$ is a regulatory factor to prevent edge servers from getting into maximum computing pressure. Supposed that the number of edge servers deployed for an area is less than five, thus the action space of DRLoS is kept within 2^5.

Rewards. To guide the agent of DRLoS towards a good direction for minimizing resource usage variance, we define the reward as $r_t = \frac{\sum_{i=1}^{m}(g_t^i - \bar{g}_t)}{1-m}$ at each time step. Note that the agent has no rewards of intermediate actions in a time step. Thus, we can maximize the cumulative reward to mimic that minimizing the resource usage variance when setting the discount factor as $\gamma = 1$.

3.3.2 Training Algorithm for DRLoS

We use the neural network representing the policy to obtain the probability distribution of actions corresponding to the task input in Fig. 5. The policy network is trained in various episodes, which inputting M tasks for each episode for scheduling according to the policy network until all tasks are scheduled. In each training epoch for each task set that simulates N episodes, we compute the probabilistic space of actions with the policy and improve the policy for all tasks by the inference results. Note that the state, the action, and the reward of each episode are used to compute the cumulative reward of v_t. Since our policy gradient of Eq. (8) has a high variance on gradient estimation, we use the average value of the return results of the same time step across all episodes with the same task set, which is also employed in [32], [35].

3.4 Analysis of LcDNN

In this section, we discuss the detailed design of the added BNN branch from the number, location, and architecture of the BNN branch. Specially, we analyze the reasons for such design by some examples and experiments.

3.4.1 The Number of Binary Neural Network Branches

In the design of LcDNN, one of the most important problems is that whether it is also necessary to add multiple

BNN branches to the full-precision network, which is similar to the BranchyNet [25] that loading and executing more BNN branches on the mobile web when the accuracy of the first branch does not meet the demand. To this end, we analyze whether it is reasonable to add more BNN branches to the mobile web from the perspective of the latency. First, we give the following definitions. Assuming that we adding n BNN branches into full-precision branch, representing by $\mathcal{N} = \{b_1, b_2, \ldots, b_n\}$. Where ith branch is represented by b_m, whose structure is corresponding to the rest layers after the adding layer of the full-precision branch. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ be the memory usage of each BNN branch. Next, we define the possibility of exiting from a BNN branch as

$$\mathcal{P} = \left\{ \frac{p_1}{p_m}, \frac{p_2 - p_1}{p_m}, \ldots, \frac{p_n - p_{n-1}}{p_m} \right\}, \tag{11}$$

where p_i, $i \in [0, n]$ and p_m denote the accuracies of b_i and the full-precision branch, respectively. For an input sample x, whose size is w_x, we can use $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ to denote the input of each BNN branch. Considering the full-precision branch of LcDNN with l-layers, we use $\mathcal{T}_M = \{t_m^1, t_m^2, \ldots, t_m^l\}$ and $\mathcal{T}_W = \{t_w^1, t_w^2, \ldots, t_w^l\}$ denote inference latencies of each layer for the full-precision branch executing on the edge server and the mobile web, respectively. Thus, if we add one BNN branch into the full-precision branch, we can formulate inference latency of the BNN branch as

$$T_b^1 = \frac{m_1}{B_d} + t_w^1 + \sum_{i=2}^{l} t_b^i, \tag{12}$$

where $\mathcal{T}_B = \{t_b^1, t_b^2, \ldots, t_b^l\}$ denotes inference latency of each layer in the BNN branch, and B_u and B_d are the current wireless network uplink and downlink bandwidths, respectively. Then, we can obtain expectation inference latency of LcDNN as

$$E_1 = \frac{p_1}{p_m} T_b^1 + \left(1 - \frac{p_1}{p_m}\right) \left(\sum_{i=2}^{l} t_m^i + \frac{w_1}{B_u} + T_b^1 \right). \tag{13}$$

Besides, if we add $n \in [2, l-1]$ BNN branches to the full-precision branch, and use $ep = \{e_1, e_2, \ldots, e_n\}$ to denote entry points of branches where $0 < e_1 < e_2 < \ldots < e_n < l-1$. Then, the expectation inference latency can be described as

$$\begin{aligned} E_n = & \frac{p_1}{p_m} T_b^1 + \left(1 - \frac{p_1}{p_m}\right) \frac{p_2 - p_1}{p_m} T_b^2 + \ldots + \\ & \left(\prod_{i=1}^{n-1} \left(1 - \frac{p_i - p_{i-1}}{p_m}\right) \right) \frac{p_n - p_{n-1}}{p_m} T_b^n + \\ & \left(\prod_{i=1}^{n} \left(1 - \frac{p_i - p_{i-1}}{p_m}\right) \right) \left(\sum_{i=n}^{l} t_m^i + \frac{w_n}{B_u} + \sum_{i=1}^{n} T_b^i \right), \end{aligned} \tag{14}$$

where $p_0 = 0$,

$$T_b^2 = T_b^1 + \sum_{j=e_1}^{e_2} t_w^j + \sum_{k=e_2}^{l} t_b^k + \frac{m_2}{B_d}, \tag{15}$$

and

$$T_b^n = \sum_{i=1}^{n} T_b^i + \sum_{j=e_{n-1}}^{e_n} t_w^j + \sum_{k=e_n}^{l} t_b^k + \frac{m_n}{B_d}. \tag{16}$$

We expect to find the minimum expectation inference latency from E_1 to E_n. When n equals 2, which means adding two BNN branches into the full-precision branch, thus we have

$$\begin{aligned} E_2 - E_1 = & \left(1 - \frac{p_1}{p_m}\right) \left[\frac{p_2 - p_1}{p_m} T_b^2 + \left(1 - \frac{p_2 - p_1}{p_m}\right) \right. \\ & \left(\sum_{i=2}^{l} t_m^i + \frac{w_2}{B_u} + T_b^1 + T_b^2 \right) \\ & \left. - \left(\sum_{i=2}^{l} t_m^i + \frac{w_1}{B_u} + T_b^1 \right) \right]. \end{aligned} \tag{17}$$

Next, we reveal the above expectation inference latency has a little lifting because the layer distance is close between two BNN branches (e.g., adding the BNN branch after the first convolutional layer obtains the largest profit in AlexNet). This indicates adjacent location of e_2 and e_1 has little promotions between p_2 and p_1, resulting in a large expectation latency due to more communication costs for the second BNN branch. If we increase the layer distance between the first BNN branch and the second BNN branch, although we can obtain the increase of accuracy, the memory usage of the second BNN branch introduces high communication costs which losts the advantage of adding the binary convolutional networks. Especially, in practice, the network bandwidth is unstable, resulting in high communication costs during the interactions between the mobile web and the edge server. Hence, considering the communication costs, memory usage, and accuracy lifting of the BNN branch, we suggest adding one BNN branch into the full-precision branch for LcDNN.

3.4.2 Location of the BNN Branch

The optimal location of the BNN branch is discussed in the aforementioned definitions. Let $e_1 = 1$ be adding the BNN branch after the first convolutional layer. Thus, e_h, $(2 < e_h < l)$ is the BNN branch of the e_h^{th} layer. E_n represents the expectation inference latency. We can obtain $E_{e_h} - E_{e_1} > 0$ due to communication costs and a small amount of accuracy lifting. The is because that if e_h is close to l, the whole network degenerates into a traditional deep neural network with high weights and computations. When e_h is close to e_1, the accuracy lifting is slight, while increasing the communication costs. Thus, if we consider adding only one BNN branch, it is preferable to add it after the first convolutional layer.

3.4.3 Structure of the BNN Branch

In this paper, the BNN branch mainly consists of the binarized convolutional layer and binarized fully connected layers. Based on the above discussions, we suggest adding one BNN branch after the first convolutional layer of the full-precision network. Using the AlexNet network as an example, Fig. 7 shows the performance of accuracy and

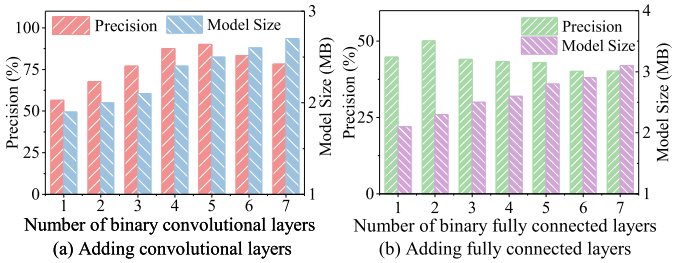


Fig. 7. The structure of the BNN branch.

model size of various structures of BNN branches. Concretely, we explore the role and influence of binarized convolutional layer in Fig. 7a with n, $(n < l)$ binarized convolutional layers and a binaried fully connected layer. In Fig. 7b, we use only one binarized convolutional layer and n, $(n < l)$ binarized fully connected layers to reveal the effectiveness of the BNN branch. The last layer of all structures is a common fully connected layer.

Experimental results show that it is not a better choice to add more binary convolutional layers into the full-precision branch due to the accuracy loss. Similarly, one or two binary fully connected layers may introduce higher accuracy. Hence, we suggest consulting the structure of the full-precision branch when designing the BNN branch for a satisfactory experience.

4 Evaluation

In this section, we evaluate LcDNN's performance from the latency, mobile energy cost, and system throughput. We introduce experimental settings in Section 4.1 and the training performance of LcDNN in Section 4.2. We describe improvements of LcDNN in Section 4.3. Then, we aim to explore the impact of the entropy threshold of LcDNN in Section 4.4. We also discuss the performance of our resilient scheduling mechanism, DRLoS of LcDNN in Section 4.5.

4.1 Experiments Setup

4.1.1 Benchmarks and Datasets

We demonstrate that LcDNN achieves improvements in terms of average latency, average mobile energy cost, and system throughput against the state-of-the-art partition-offloading approaches, Neurosurgeon [14] and Edgent [15]. We also compare LcDNN against mobile-only and edge-only to illustrate the advantages. We evaluate LcDNN using typical deep neural networks such as LeNet, AlexNet, ResNet18[12], VGG16 and compressed DNNs such as MobileNet [19], to show the practicality. For datasets, we use a series of benchmark datasets such as MNIST [36], Fashion-MNIST [37], CIFAR-10/CIFAR-100[38], and ImageNet-150K [39] to present the effectiveness of LcDNN. ImageNet-150K is the subset of ILSVRC ImageNet, which contains 183K training images and 7.5K testing images belonging to the top 150 most popular categories based on the popularity ranking provided by the official ImageNet website.

4.1.2 Mobile Device and Edge Server Setup

We use Pytorch as the deep learning framework to train LcDNN for various networks and datasets on a GPU server,

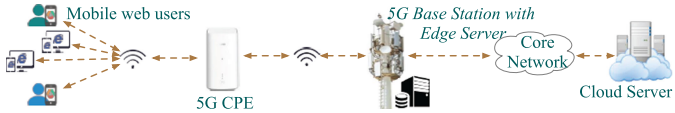


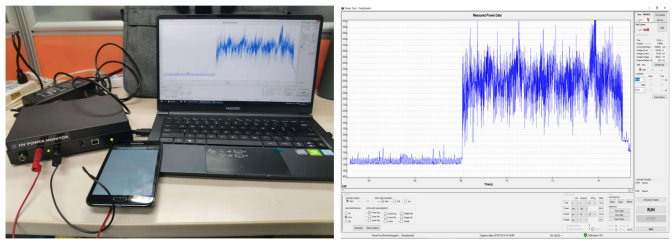
Fig. 8. The network topology in a real scenario.

which is a fourteen-core Intel Xeon processor running at 2.0 GHz with 128 GB of RAM and dual GTX TITAN Xp GPU cards with 12 GB of RAM of each card. For the edge server, which is deployed near the base station, we use a common server with a six-core Inter processor of 2.9 GHz and 16 GB RAM running Ubuntu 18.04 LTS. We present the core network topology with a max uplink bandwidth of 150 Mbps and a max downlink bandwidth of 600 Mbps, which is a real-world 5G network at Beijing University of Posts and Telecommunications in Fig. 8. We use a HUAWEI Mate 9 smartphone running Andriod 8.0 with 4 GB RAM. To simulate stable communication conditions such as 3G, 4G, and WiFi, we use a HUAWEI 5G CPE to connect to the base station, and use Wonder Shaper [40], which is a script that allows the user to limit the bandwidth of network adapters, to control the network on the edge server.

4.1.3 Measurements

To acquire precise numerical performance of LcDNN, we introduce tools and methodologies to measure the latency, mobile energy and system throughput as follows.

- *Latency measurement*. Since latency components of a request in LcDNN mainly include downloading data and DNN inference, it is necessary to measure the whole latency from requesting a task until getting the final result to evaluate the real performance. We first record the timestamp through a JavaScript function when the web page triggers the request of DNN computation. Then, once the web page receives the final results from the edge server or the mobile web, we record the current timestamp again. Thus, the entire latency can be calculated based on two timestamps before and after a complete DNN computation request. Moreover, we can also acquire the latency of downloading data from the edge server in this way if necessary. To reduce random errors, we repeat the same DNN task request multiple times and use the average latency as the final latency performance. Note that we clear the data cache of the mobile web browser to keep each request independent.
- *Mobile energy measurement*. To precisely measure the mobile energy consumption of mobile devices when performing LcDNN via the mobile web browser, we use a hardware power monitor with a model number of AAA10F in Fig. 9. We also use it to provide a stable voltage of 3.7 V for mobile devices and obtain the system energy cost such as the screen brightness cost in the standby state. And the high value of the curve in Fig. 9b denotes the average energy consumption of running LcDNN. Then we can easily acquire real mobile energy consumption based on these two curves.
- *System throughput measurement*. We define the edge server throughput as how many units of requests



(a) AAA10F power monitor. (b) Record tool for energy cost.

Fig. 9. The power monitor used for energy measurements.

from mobile web users can be processed at a given time. For multiple edge servers in DRLoS, we consider the average processed requests as the system throughput. Especially, to better describe the throughput improvements of LcDNN against state quo approaches, we normalize the throughput of state quo approaches to that of LcDNN.

4.2 Training Results of LcDNN

We present the main performance of LcDNN on top-1 accuracy, exit threshold, exit possibility, and model size respectively in Table 1. F_Acc and B_Acc represent top-1 accuracies of the full-precision branch and the BNN branch, respectively. Exit threshold is to determine whether exiting from the BNN branch or transferring to the full-precision branch for precise inference. Exit possibility denotes the ratio of exiting from the BNN branch in 100 random samples. We also compare the model size of branches to highlight the advantage of adding the BNN branch for the mobile web.

We observe that the BNN branch can reduce the model size about 16x to 30x when compared with the full-precision branch. Top-1 accuracy of DNN networks with shallow binary layers has fewer reductions compared with the full-precision branch. However, a deep neural network such as VGG16 performs a sharp decrease due to more binary layers. We also evaluate the exit probability from the BNN branch with a baseline of 100 random samples. The results show that the simpler the network is, the more likely it is to exit from the BNN branch. For example, LeNet's BNN branch contributes to about 90 percentages of samples exiting without the collaboration of the full-precision branch. Meanwhile, the τ value of shallow networks is stricter than that of deep networks, which means the BNN branch has similar accuracy to the full-precision branch. However, the τ value of deep networks such as AlexNet and VGG16 is larger than LeNet, which indicates more BNN branches may result in a large loss. In summary, although the accuracy of the BNN branch has gaps with the full-precision branch, LcDNN leverages the collaboration of the full-precision branch located at the edge server to supply the shortage of the BNN branch. Simultaneously, a lightweight BNN branch provides a crucial foundation for executing deep neural networks on the mobile web in real-time.

4.3 Improvements

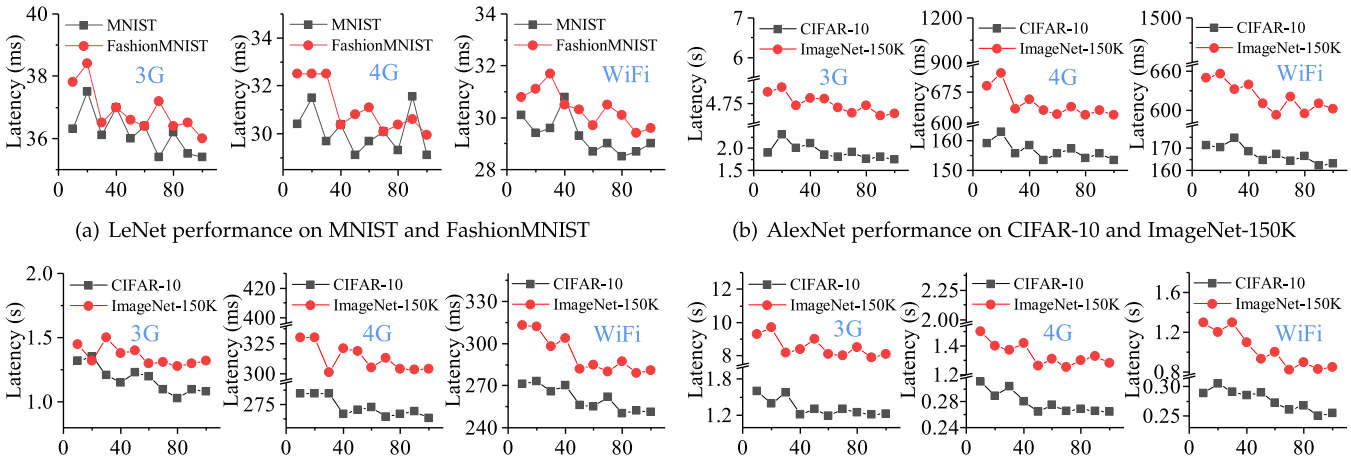
4.3.1 Latency Performance

We present the average latency of LcDNN over different DNN networks, datasets, and network conditions in Fig. 10. We follow the same communication settings described in Section 4.1, and threshold settings are described in Table 1. To reduce random error, we choose the average latency of executing any sample 10 times as the final latency performance. We observe that (1) With the increase of testing samples, DNN networks with high exit probability perform stable in latency, and show a small downward trend, which indicates that most of the samples can be processed by the BNN branch directly. We note that there is a large fluctuation when samples increase to 90 in a 4G network of LeNet. Through the analysis of test samples, it is found that there are more complex tasks in this set of sampling data that require the assistance of the edge server. However, for VGG16 on ImageNet-150K, average latency has larger fluctuations than shallow networks, which is mainly because a large number of examples rely on edge-assisted inference. (2) Communication condition has also caused fluctuations in average latency performance, which has impacts on loading models, transmitting intermediate results, and other data. Hence, as the number of samples requiring the assistance of the edge server increases, communication conditions have a large impact on average latency, which shows obvious fluctuations under different states. In summary, although LcDNN's BNN branch can not process all samples, the collaborative mechanism of LcDNN effectively provides accurate compensation for the BNN branch. Especially in a long run, the benefits of the BNN branch are considerable.

We discuss the latency performance of LcDNN against Neurosurgeon, Edgent, mobile-only, and edge-only using an average latency of 100 random samples under 4G network in Table 2. We observe that the BNN branch accelerates the execution and decreases the whole inference latency. Especially, for deeper neural networks (e.g., ResNet18 and VGG16), because the computing capability of the mobile web is limited, the full-precision branch is unable to be executed directly on the mobile web. Although Neurosurgeon and Edgent load and execute partial DNN layers pursuing fast inference, they also become unavailable facing with deeper networks. This indicates that the model size is still too large to load and execute efficiently. We note that edge-only has the lowest latency, but this method has heavy computing pressure without using the computing resource of the mobile device, which is essentially different from distributed and collaborative DNNs proposed in this paper.

TABLE 1
Training Results on Various Networks and Datasets

Network/Dataset	F_Acc. (%)	B_Acc. (%)	Threshold (τ)	Exit (%)	F_size (MB)	B_size (MB)
LeNet-MNIST	99.50	98.81	0.0001	94	1.7	0.103
LeNet-FashionMNIST	99.41	98.67	0.0001	93	1.695	0.102
AlexNet-CIFAR-10	76.85	73.99	0.0251	79	90.911	3.3
AlexNet-CIFAR-100	57.31	54.73	0.0251	76	92.351	3.5
AlexNet-ImageNet-150K	59.60	46.2	0.3031	61	242.3	8
ResNet18-CIFAR-10	93.02	88.89	0.0453	73	43.705	1.6
ResNet18-CIFAR-100	78.32	73.96	0.0453	60	43.885	1.7
ResNet18-ImageNet-150K	69.37	51.2	0.2521	65	44.1	1.9
VGG16-CIFAR-10	92.29	87.76	0.0523	78	59.0	2.0
VGG16-CIFAR-100	70.48	65.32	0.0523	76	59.759	2.1
VGG16-ImageNet-150K	73.44	55.7	0.2036	55	523.8	14.9



(a) LeNet performance on MNIST and FashionMNIST

(b) AlexNet performance on CIFAR-10 and ImageNet-150K

(c) ResNet18 performance on CIFAR-10 and ImageNet-150K

(d) VGG16 performance on CIFAR-10 and ImageNet-150K

Fig. 10. Average latency performance. The X-axis represents the number of testing samples, and the Y-axis is the corresponding average latency. The input sample can be processed, and exit from the BNN branch directly or collaborate with the edge server for more precise inference.

4.3.2 Mobile Energy Performance

We use the same measurement in Section 4.1.3 to accurately measure mobile energy consumption of executing LcDNN in different networks, datasets (e.g., LeNet-MNIST, AlexNet-ImageNet, ResNet18-ImageNet, VGG16-ImageNet) and communication conditions in Fig. 11.

We randomly sample 10 groups of testing samples, and each group consists of 10 samples to reduce the random error. The results show that LcDNN performs better than existing partition-offloading approaches in mobile energy consumption, which is a benefit from the lightweight BNN branch that reduces a large amount of communication cost on loading the DNN model. The phenomenon is more pronounced for deep neural networks such as AlexNet and VGG16, which also shows that loading heavy models onto the mobile web is not applicable, and lightweight BNN branches perform the advantage. Besides, it is noted that the mobile energy cost of the edge-only is lower than the others, which have a similar performance on latency. This is because the edge-only approach only consumes mobile energy on task transmission without loading DNN models. Although it performs well, it is not the optimal solution, especially from the perspective of system throughput, computing pressure, and economy.

4.3.3 System Throughput Performance

In Fig. 12, we describe comparisons and analysis of system throughput in different networks, datasets, and communication conditions.

TABLE 2
Comparisons on Latency Performance

Networks	LcDNN	Neurosurgeon	Edgent	Mobile-only	Edge-only
LeNet	37	110	204	109	15
AlexNet	153	5,256	4,617	9,313	21
ResNet18	261	2,820	2,613	5,882	19
VGG16	264	3,421	3,231	8,205	25

[a] *The unit of measurement is ms.*
[b] *The parameters of LcDNN are same with those mentioned above.*

The results show that LcDNN has the best system throughput in all approaches, it improves the throughput about 2.9x to 13.1x against the mobile-only approach. This is because the BNN branch can independently handle the majority of tasks without the assistance of the edge server for powerful computing resources, especially in the LeNet network. However, Neurosurgeon and Edgent require the collaboration of the edge server to complete a whole task. Although the computing distribution varies in different network conditions, the edge server always has to participate in inference for each task. Thus, such a partition-offloading approach has large gaps with LcDNN in terms of system throughput. Besides, for the edge-only, since all computations are arranged on the edge server, thereby it performs the lowest system throughput, whether on LeNet or VGG16. It also shows that centralized service provision has heavy computing pressure, and resource consumption is more serious. Hence, it is not economical when considering the edge serves as the foundation to process such heavy computations. Finally, we do not compare LcDNN with the mobile-only approach, because it only consumes the edge server's

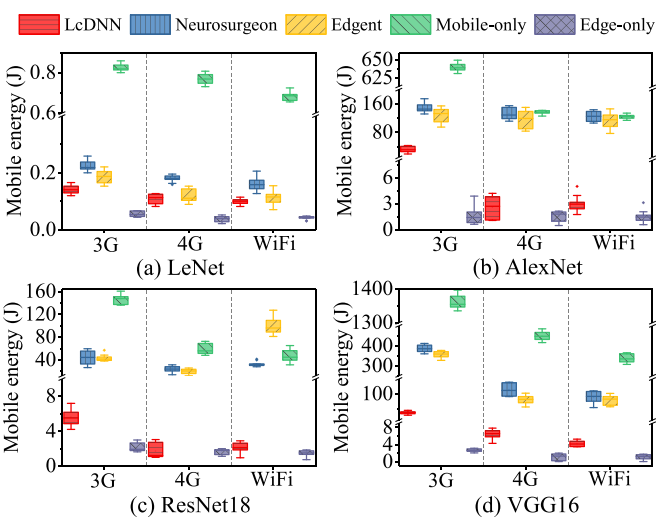


Fig. 11. Mobile energy improvements.

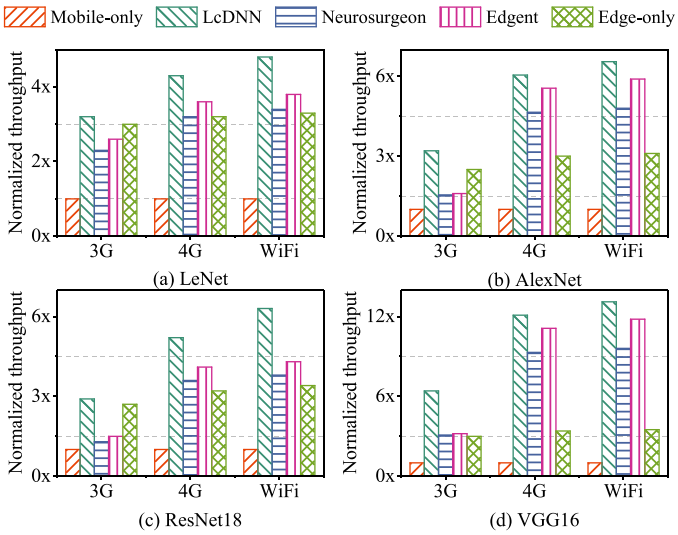


Fig. 12. System throughput improvements. We normalize LcDNN and other status quo approaches to the mobile-only approach, which has the lowest throughput than others.

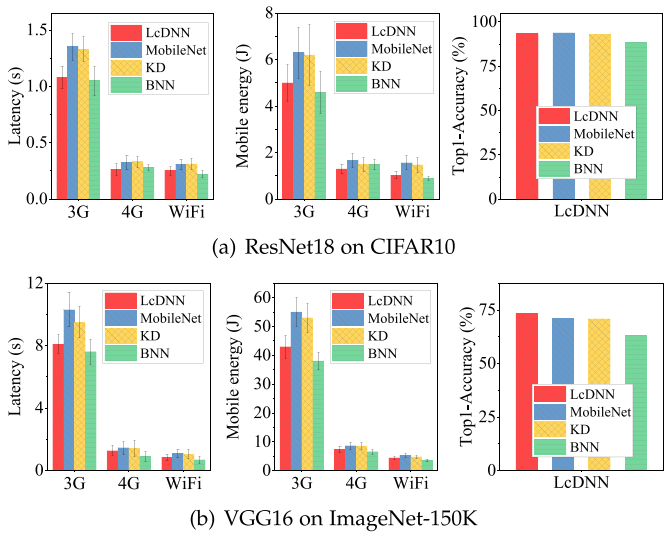


Fig. 13. Performance of LcDNN compared with other compressed methods.

request link resources without extra consumption on the computing resource of the edge server. Moreover, it is not a practical approach from the perspective of latency and mobile energy consumption.

4.3.4 Comparisons of LcDNN With Compressed DNNs

We compare the performance among LcDNN and typical compressing DNN methods such as MobileNet, knowledge distillation, and network quantization in terms of the latency, mobile energy, and inference accuracy in Fig. 13. MobileNet is a well-designed network with high accuracy, a lightweight model, and fast inference, representing a new convolution method. The knowledge distillation (KD) method uses the full-precision network to train a smaller student network.

Besides, we also compare LcDNN with binarized neural network which is similar to the BNN branch of LcDNN. We use ResNet18 and VGG16 on CIFAR10 and ImageNet-150K as examples to reveal comparisons among LcDNN and other methods. Experimental settings and measurements are same to the above subsections.

We see that (1) LcDNN acquires higher accuracy than other compressed methods on CIFAR10 of ResNet18 in Fig. 13a, which also performs better in the latency and mobile energy cost. Although MobileNet and KD methods can reduce the model size and parameters, their inference efficiency still has some gaps with LcDNN and BNN of using quantitative technology, especially running on the mobile web. This is because the 2-bit quantization technology used in LcDNN and BNN has the advantage in accelerating inference and reducing the mobile energy. Note that BNN gains faster inference by sacrificing accuracy, while LcDNN not only uses the BNN branch to accelerate inference but also improves accuracy through a collaborative mechanism. Besides, the inference library used by LcDNN is optimized according to the characteristics of the quantitative network, whose inference performs faster than existing libraries such as ONNX.js and TensorFlow.js used by MobileNet and KD. (2) As for VGG16 on ImageNet-150K, although the full-precision network of VGG16 has superior accuracy, parameters of the full-precision branch are too large to be directly applied to the mobile web. By adding a BNN branch into the full-precision branch, the model size of the BNN branch can be greatly reduced, and inference can be accelerated, which performs advantages on the latency and mobile energy when compared with MobileNet and KD methods. More importantly, the accuracy of the BNN branch is lower than MobileNet and KD, we can still obtain high accuracy with the help of collaboration of the full-precision branch in LcDNN. We can conclude that MobileNet and KD methods are more focused on reducing the model size and parameters, and lack consideration and optimization in inference efficiency, especially for the mobile web platform. And the BNN network of quantization method lacks an effective balance between inference efficiency and accuracy loss. Thus, this indicates LcDNN can achieve a good balance in accuracy, latency, and mobile energy consumption through a collaboration mechanism. (3) We can also see that the structure of LcDNN is easy to popularize and can be directly applied to existing neural networks without the need for expert knowledge, or a large number of calculations for automatic neural network structure search. Hence, if we use ResNet152 as a full-precision branch and apply it to LcDNN, it will get better performance than the majority of compressed DNNs, no matter in latency, mobile energy, and accuracy. In summary, LcDNN is easier to operate than well-designed DNNs in practical applications and can be flexibly applied to different datasets and networks.

4.4 Entropy Threshold of LcDNN

In LcDNN, the exit judgment of the BNN branch directly affects overall accuracy due to the wrong decision of the BNN branch. We have given the method of selecting thresholds based on the validation dataset in Algorithm 2 of Section 3.1. Meanwhile, we further analyze the impact of different thresholds on accuracy of various networks and datasets in Fig. 14.

We observe that the optimal threshold is various and directly related to the structure of DNN networks and datasets. The threshold of a high-precision small network such

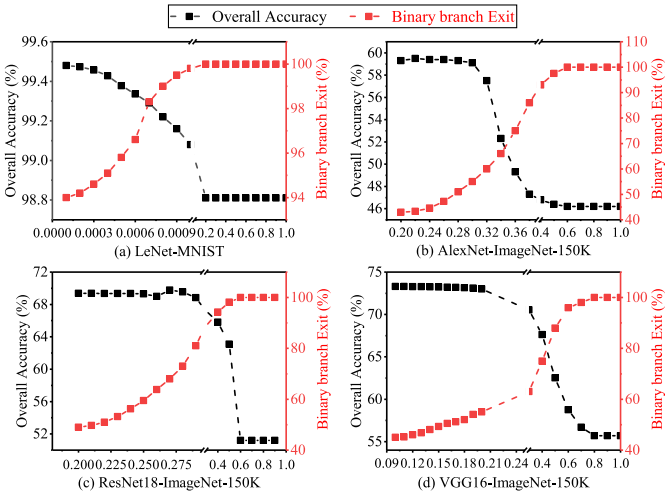
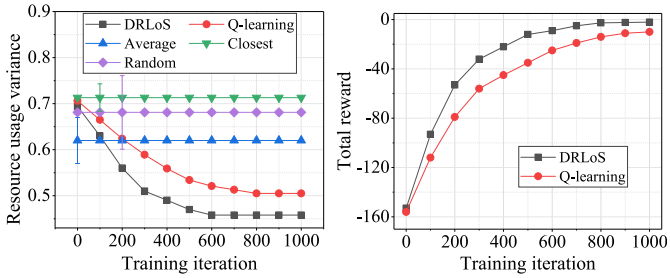


Fig. 14. Discussion of different entropy thresholds.

as LeNet is much stricter than others. This is because the BNN branch has similar accuracy to the full-precision branch, which requires a strict threshold to ensure the overall accuracy. However, the threshold is relatively large such as AlexNet, ResNet, and VGG16, whose accuracy of the full-precision branch in these networks is low. Thus, excessively strict thresholds may cause a large number of examples to be unable to exit from the BNN branch, which requires the collaboration of the edge server, increases the overall latency, and cannot reduce the computing pressure of the edge server. In addition, we compare the threshold between the optimal value found by our Algorithm 2 and the actual optimal threshold. The thresholds obtained using the validation dataset are close to the actual optimal thresholds. Therefore, we can use Algorithm 2 to quickly obtain an accurate and reasonable threshold in the actual application. Then, we can update the threshold value to meet the actual sample data through feedback from the samples of the actual application.

4.5 Performance of DRLoS

We first introduce experimental settings for evaluating DRLoS, which leverages the same communication and hardware described in Section 4.1. We build a DRLoS testbed consisting of three edge servers with the same computing capability and simulate a task requestor that obeys the Bernoulli distribution with a request frequency range from 10 to 130 percent. For the policy network, we design a deep neural network consisting of a fully connection layer with 20 neurons, whose learning rate for training is set as $lr = 0.001$. We set the output size of the policy network as $M = 500$, the number of tasks scheduled at the same time, which also means the total number of concurrent processing tasks. In each iterative training, each request dataset is processed in parallel with $N = 20$, and the number of training iterations is 1500. Then, we analyze the effectiveness and novelty of DRLoS from convergence behavior, scheduling efficiency, and stability based on the above experimental scenario. Specifically, we compare DRLoS with some commonly used scheduling methods such as Q-learning [41], Average, Closest, and Random on indicators of the latency, mobile energy, and resource usage to illustrate the effectiveness and novelty.



(a) Convergence of the training (b) Total reward of the training

Fig. 15. Convergence behavior of DRLoS.

The Q-learning method refers to the value-based RL solution which uses the same definition to [41]. The Average method refers to the average distribution of task requests to all edge servers for processing. And the Closest method is to distribute task requests to the available edge servers according to the geographical location. The Random method assigns task requests to any edge server for processing randomly.

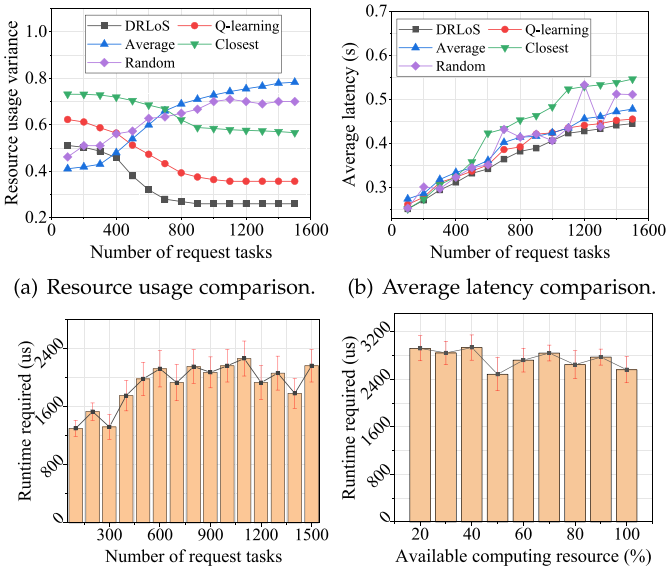
4.5.1 Convergence Behavior

To explore the convergence performance of DRLoS's policy network with the increase of training iterations, we show the convergence behaviors on normalized resource utilization variance over various methods and compare the convergence of DRLoS and Q-learning method on the total reward. We set the task request load to 80 percent and delve into the system's resource utilization and total reward. Besides, non-RL methods have no changes with iterations due to unnecessary training, thus using an average result of 20 times scheduling to evaluate the performance. We can see that (1) DRLoS improves with the increase of iterations, and at the beginning, DRLoS even performs worse than other methods in resource usage variance in Fig. 15a. When the training iteration reaches 500, DRLoS has considerable performance. Especially, DRLoS's online scheduling can bring LcDNN more than 2.17x resource utilization improvement against the Closest approach. DRLoS's performance is closer to the Random method at the beginning. With the increase of training iterations, DRLoS gradually converges and obtains better performance than all non-RL methods. Moreover, DRLoS has better convergence speed and convergence performance than the Q-learning method, also illustrating the effectiveness and advancement. (2) Based on the convergence performance of DRLoS and the Q-learning method on total reward in Fig. 15b, we observe the convergence speed of DRLoS is better than the Q-learning method, and total reward gradually converges at 700 and 900, respectively.

This is because the Q-learning method usually obtains a deterministic policy while DRLoS's policy is a probability distribution over possible actions, thus the action-value of Q-learning method eventually converges to the corresponding true values and DRLoS tends to generate optimal random strategies.

4.5.2 Comparing the Efficiency of DRLoS Scheduling

In Fig. 16a, we compare the performance of different approaches on the normalized resource usage variance with the increase of request tasks. As we expect, we see that (1)



(a) Resource usage comparison. (b) Average latency comparison.

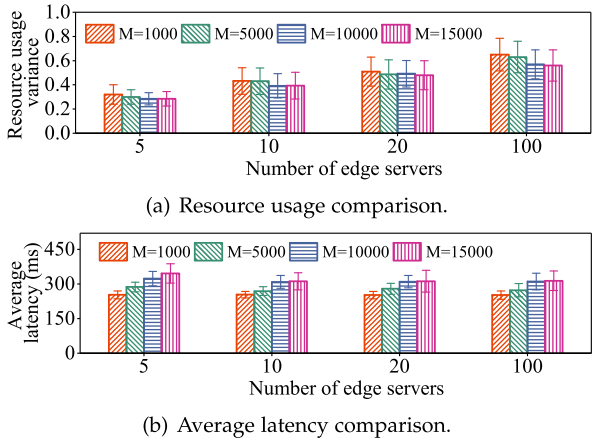
(c) Runtime analysis of increasing tasks. (d) Runtime analysis of various computing resources.

Fig. 16. Scheduling efficiency of DRLoS.

the system resource usage variance continues to increase as the amount of requests increases until it reaches maximum processing capacity. (2) DRLoS shows the best resource utilization because it combines the status of all edge servers and task request status to maximize and optimize scheduling. However, while the method based on the Closest allocation may only be processed on nearby edge servers, which will introduce to backlog a large number of tasks while other servers may be idle, causing the waste of resources. (3) The Average method also does not take into account the global status of edge servers and request tasks, resulting in imbalanced resource utilization, which cannot maximize the use of computing resources of edge servers and improve the processing capacity of the system. In Fig. 16b, we exhibit the average processing latency of DRLoS and other methods as the task requests increase. Assuming that all request tasks require edge assistance in LcDNN, and set the maximum concurrency of three edge servers to 500. We know that the RL-based methods perform better in terms of stability and average latency, while non-RL methods, especially the Random method, have high fluctuation. This is due to the fact that random allocation may cause excessive load on the part of edge servers, introducing the waiting of tasks and increasing the average processing latency. In addition, the Closest method performs a certain increase when the number of task requests reaches 500 and 1,000. This is because the adjacent edge server is fully loaded, and all tasks need to be forwarded to other edge servers, resulting in an increase of average latency. Thus, DRLoS defeats other methods, which can be summarized that the auto-learning policy network can dynamically perform global allocation based on the system's real-time status and context information (available resource status, request status).

Besides, DRLoS can automatically learn and adjust scheduling policy to dynamically adapt to different environments according to historical experience.

We also analyze the runtime performance of DRLoS with changes in task requests and available computing resources



(a) Resource usage comparison.

(b) Average latency comparison.

Fig. 17. Availability performance of DRLoS.

in Figs. 16c and 16d, respectively. To measure the runtime of DRLoS on common edge servers as accurately as possible, we use the method in the Section 4.1.3 to record logs and control the available computing resource of edge servers via CPU Usage Limiter [42]. Besides, we follow the same settings to the above experiments. The results show that DRLoS can execute real-time task scheduling within milliseconds with the increase of the number of tasks and the reduction of available computing resources. On the one hand, the DRL-based method generally spends time on training policy network offline. Once the convergent policy model is obtained, it only needs to be deployed on the edge server to execute the inference of the policy network in real-time. On the other hand, the policy network designed and used by DRLoS is much smaller in terms of neurons, parameters, and model size. Therefore, running DRLoS in real-time does not require high hardware configuration and computing resources, and it can still provide real-time scheduling even on resource-constrained embedded devices.

4.5.3 *Availability Analysis*

Since the size of action space of DRLoS is affected by the scale of action states and the number of available edge servers, we simplify the action state of the edge server to process each task as process a small batch of tasks or a large number of tasks (i.e., small capacity and high capacity). It is important to explain that for a batch of task requests at any time slot, high capacity actually includes a general number of task requests between small capacity and high capacity. Hence, to explore the availability of the simplification method used by DRLoS, we simulate different scales edge servers N and various task M and analyze the influence on the resource usage and average processing latency in Fig. 17. We define the maximum concurrent processing of any edge server as 1,000, and the task request also follows the same Bernoulli distribution and settings as described at the beginning of this section.

We observe that (1) Normalized resource usage variance of DRLoS does not change significantly as the task request scale increases, which illustrates the effectiveness of our simplified strategy of action space to execute scheduling under various task scales in Fig. 17a. Besides, when there is deployed with five edge servers and the task scale is greater

than 5,000, it has exceeded the limited processing capacity of the whole system, that is, all edge servers are in a saturated state. Thus, if we continue to increase the task scale, the resource usage variance will not change. (2) We also analyze the effect of the simplified strategy on the average processing latency in Fig. 17b. The results show that there is a slight climbing in the average latency with the increase of task scale, which is caused by the increase of forwarding task requests among edge servers. From the perspective of the increase in latency, DRLoS still performs well in terms of average latency facing various task scales, which demonstrates that the simplified strategy of DRLoS is effective. Also, we observe that the average processing latency still has an increase when expanding the scale of edge servers, which can indicate that adding more edge servers for collaborations in actual deployments does not always improve performance. Besides, since edge servers in experiments are supported by China Unicom and usually cover several tens of kilometers, there is no need for a large number of edge servers to cooperate, and only a small number of edge servers within a certain range can meet the demand. For further considerations, if the scale of the task request is extremely large under special circumstances, that is, it cannot be processed at the edge server, the more conventional approach is to forward the task to a remote cluster cloud for collaboration, which is more economical than using more edge servers for collaboration.

5 Related Work

5.1 Distributed Inference of Deep Neural Networks

Partition-offloading is the most fundamental methodology to execute distributed inference working on traditional deep neural networks [43]. Recently, Neurosurgeon [14] automatically chooses the partition points pursuing the optimal latency and energy consumption to offload DNN computations. Edgent [15] searches the adaptive partitions of DNN computation and accelerates DNN inference through an early exit at a proper intermediate DNN layer. Similarly, there are prior explorations focusing on intelligent collaboration between the mobile device and the cloud without combining the granularity of neural network layers. McDNN [44] investigates the intelligent collaboration to execute either in the cloud or on the mobile device by generating alternative DNN models for performance and energy costs. MoDNN [45] proposes two partition schemes to minimize non-parallel data delivery latency and accelerate DNN inference by alleviating device-level computing cost and memory usage. Moreover, JointDNN [16] further proposes efficient DNN inference giving the training process simultaneously, which also provides various optimizations in DNNs tackling with resource constraints. However, these anterior researches only benefit the lightweight DNN models, and they do not support deeper neural networks due to massive weights and computations. Thus, it is worthy of considering lightweight neural networks like binary convolutional neural network for reducing the communication and computation costs.

5.2 Binary Deep Neural Network

As one of the effective compression methods, binary convolutional neural network attempts to address efficient training and inference by binarizing weights and activations compared to typical deep neural networks. The directions are of two kinds: Expectation BackPropagation (EBP) and BinaryConnect. EBP in [46] achieves a neural network with binary weights and binary activations by variational Bayesian approach. Esser [47] treats spikes and discrete synapses as continuous probabilities, which allows to train the network using the standard backpropagation and shows the advantages in energy efficiency. BinaryConnect [26] trains a DNN with binary weights during the forward and backward propagation while retaining precision of the stored weight. It shows high performance on small datasets while large-scale datasets are not suitable. XNOR-Network [27] is more extreme to binarize filters and inputs in convolutional layers which results in faster and more memory-saving inference. Undoubtedly, network binarization makes the trade-off between model size and precision. Nevertheless, it is hard to acquire satisfactory precision with an efficient compression of networks in terms of complex datasets (e.g., ImageNet).

5.3 Web-Based DNN Inference Framework

To allow the web to execute DNNs, JavaScript and Webassembly are the representative technologies currently. CaffeJS [48] loads pre-trained deep neural networks entirely in JavaScript, which aims to execute forward and backward propagation. All of this runs on mobile devices without installing any software. Similarly, Keras.js [49] supports GPU and CPU mode of Keras models in the browser on personal computers, which can be trained in any backend, including TensorFlow, CNTK, etc. TensorFlow.js [13] is a JavaScript library for training and deploying deep learning models in the browser and on Node.js. WebDNN [50] also provides an installation-free DNN execution environment by optimizing the trained DNN model to compress model data and accelerating the execution. Although the aforementioned technologies provide the chance to execute the entire DNN inference on the mobile web browser, they get into trouble with higher latency, computation constraints, and loss of accuracy. Our approach in this paper provides lightweight collaborative DNNs to alleviate the conflict between the model size and accuracy.

6 Conclusion

In this work, we proposed LcDNN, a lightweight collaborative deep neural network for the mobile web in the edge cloud. Towards low-latency, energy-saving, high throughput, and efficient resource utilization of edge cloud, LcDNN is the first to introduce binary convolutional neural network into a typical deep neural network for reducing the model size and accelerating inference. We also provide a joint training method and implement an inference library for running LcDNN on the mobile web. Moreover, we leverage the collaboration of the full-precision branch located at the edge server to supply the compensation for the BNN branch. Last, we develop a DRL-based online scheduling scheme to obtain an optimal allocation for LcDNN to promote the resource utilization of the edge cloud. Experimental results on several well-known networks and datasets give us insightful motivation to expend LcDNN on more complex networks and applications. In future research, we may do more simulations in different system environments for more insightful knowledge.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China under Grant 2018YFE0205503, in part by the National Natural Science Foundation of China (NSFC) under Grant 61671081, in part by the Funds for International Cooperation and Exchange of NSFC under Grant 61720106007, in part by the 111 Project under Grant B18008, in part by the Fundamental Research Funds for the Central Universities under Grant 2018XKJC01, and in part by the BUPT Excellent Ph.D. Students Foundation under Grant CX2019135. A preliminary version of this paper appears as the proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS 2019) [1].

REFERENCES

[1] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, and J. Chen, "A lightweight collaborative recognition system with binary convolutional neural network for mobile web augmented reality," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1497–1506.

[2] X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen, "Web AR: A promising future for mobile augmented reality—State of the Art, challenges, and insights," *Proc. IEEE*, vol. 107, no. 4, pp. 651–666, Apr. 2019.

[3] X. Qiao, P. Ren, S. Dustdar, and J. Chen, "A new era for web AR with mobile edge computing," *IEEE Internet Comput.*, vol. 22, no. 4, pp. 46–55, Jul./Aug. 2018.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[5] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Processing*, 2016, pp. 4945–4949.

[6] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, 2019, Art. no. 5.

[7] A. Kumar *et al.*, "Ask me anything: Dynamic memory networks for natural language processing," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1378–1387.

[8] Machine learning for the web community group charter. 2018. [Online]. Available: https://webmachinelearning.github.io/charter/

[9] Immersive web working group. 2018. [Online]. Available: https://www.w3.org/ immersive-web/

[10] W3C strategic highlights October 2018. 2018. [Online]. Available: https://www.w3. org/2018/10/w3c-highlights/

[11] A. Haas *et al.*, "Bringing the web up to speed with webassembly," *ACM SIGPLAN Notices*, vol. 52, no. 6, pp. 185–200, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[13] D. Smilkov *et al.*, "TensorFlow.js: Machine learning for the web and beyond," 2019, *arXiv: 1901.05350*.

[14] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[15] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 31–36.

[16] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, early access, Oct. 16, 2019, doi: 10.1109/TMC.2019.2947893.

[17] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 329–343, Secondquarter 2019.

[18] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

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

[20] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[22] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," 2018, *arXiv:1804.03235*.

[23] T.-J. Yang *et al.*, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 285–300.

[24] Y. Huang *et al.*, "DeepAdapter: A collaborative deep learning framework for the mobile web using context-aware network pruning," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 834–843.

[25] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.

[26] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[28] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[29] A. Zakai, "Emscripten: An LLVM-to-JavaScript compiler," in *Proc. ACM Int. Conf. Companion Object Oriented Program. Syst. Lang. Appl. Companion*, 2011, pp. 301–312.

[30] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015, Art. no. 529.

[31] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016, Art. no. 484.

[32] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[34] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[35] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 31st Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[37] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv: 1708.07747*.

[38] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Master's thesis, Dept. Comput. Sci., Univ. Toronto, Citeseer, 2009.

[39] H. Liu, R. Wang, S. Shan, and X. Chen, "Learning multifunctional binary codes for both category and attribute oriented retrieval tasks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3901–3910.

[40] Wonder shaper. 2017. [Online]. Available: https://github.com/magnific0/wondershaper

[41] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 158–11 168, Nov. 2019.

[42] A. Marletta, "CPU usage limiter," 2014. [Online]. Available: https://github.com/opsengine/cpulimit

[43] P. Ren, X. Qiao, Y. Huang, L. Liu, S. Dustdar, and J. Chen, "Edge-assisted distributed DNN collaborative computing approach for mobile web augmented reality in 5G networks," *IEEE Netw.*, vol. 34, no. 2, pp. 254–261, Mar./Apr. 2020.

[44] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2016, pp. 123–136.

[45] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2017, pp. 1396–1401.

[46] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 963–971.
[47] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1117–1125.
[48] Caffe.js framework. 2017. [Online]. Available: https://chaosmail.github.io/caffejs
[49] Keras.js. 2016. [Online]. Available: https://github.com/transcranial/keras-js
[50] M. Hidaka, Y. Kikura, Y. Ushiku, and T. Harada, "WebDNN: Fastest DNN execution framework on web browser," in *Proc. 25th ACM Int. Conf. Multimedia*, 2017, pp. 1213–1216.

Yakun Huang is currently working toward the PhD degree from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include mobile computing, distributed systems, machine learning, augmented reality, edge computing, and 5G networks.

Xiuquan Qiao is currently a full professor at the Beijing University of Posts and Telecommunications, Beijing, China, where he is also the deputy director of the Key Laboratory of Networking and Switching Technology, Network Service Foundation Research Center of State. He has authored or coauthored more than 60 technical papers in international journals and at conferences, including the *IEEE Communications Magazine*, *Proceedings of IEEE*, *Computer Networks*, *IEEE Internet Computing*, *IEEE Transactions on Automation Science and Engineering*, and *ACM SIGCOMM Computer Communication Review*. His current research interests include the future Internet, services computing, computer vision, distributed deep learning, augmented reality, virtual reality, and 5G networks. He was a recipient of the Beijing Nova Program, in 2008 and the First Prize of the 13th Beijing Youth Outstanding Science and Technology Paper Award, in 2016. He served as the associate editor for the the *Computing (Springer)* and the editor board of the *China Communications Magazine*.

Pei Ren is currently working toward the PhD degree from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include the future Internet architecture, services computing, computer vision, distributed deep learning, machine learning, augmented reality, edge computing, and 5G networks.

Ling Liu (Fellow, IEEE) is currently a professor at the School of Computer Science, Georgia Institute of Technology, Atlanta, Georgia. She directs the research programs at the Distributed Data Intensive Systems Lab, examining various aspects of large-scale big data systems and analytics, including performance, availability, security, privacy, and trust. Her current research is sponsored primarily by the National Science Foundation and IBM. She has published more than 300 international journal and conference articles. She was a recipient of the IEEE Computer Society Technical Achievement Award, in 2012 and the Best Paper Award from numerous top venues, including ICDCS, WWW, IEEE Cloud, IEEE ICWS, and ACM/IEEE CCGrid. She served as the general chair and the PC chair for numerous IEEE and ACM conferences in big data, distributed computing, cloud computing, data engineering, very large databases, and the World Wide Web fields. She served as the editor-in-chief for the *IEEE Transactions on Service Computing* from 2013 to 2016. She is the editor-in-chief of the *ACM Transactions on Internet Technology*.

Calton Pu (Fellow, IEEE) is received the PhD degree from the University of Washington, Seattle, Washington, in 1986 and served on the faculty of Columbia University, New York and Oregon Graduate Institute, Beaverton, Oregon. Currently, he is holding the position of professor and John P. Imlay, Jr. chair in software at the College of Computing, Georgia Institute of Technology, Atlanta, Georgia. He has worked on several projects in systems and database research. He has published more than 70 journal papers and book chapters, 200 conference and refereed workshop papers. He served on more than 120 program committees. His recent research interest include Big Data in Internet of Things, automated Ntier application deployment, and denial of information. He is also a member of the ACM.

Schahram Dustdar (Fellow, IEEE) was an honorary professor of information systems at the Department of Computing Science, University of Groningen, Groningen, The Netherlands, from 2004 to 2010. From 2016 to 2017, he was a visiting professor at the University of Sevilla, Sevilla, Spain. In 2017, he was a visiting professor with the University of California at Berkeley, Berkeley, California. He is currently a professor of computer science at the Distributed Systems Group, Technische Universität Wien, Vienna, Austria. He was an elected member of the Academy of Europe, where he is the chairman of the Informatics Section. He was a recipient of the ACM Distinguished Scientist Award, in 2009, the IBM Faculty Award, in 2012, and the IEEE TCSVC Outstanding Leadership Award for outstanding leadership in services computing, in 2018. He is the co-editor-in-chief of the *ACM Transactions on Internet of Things* and the editor-in-chief of the *Computing (Springer)*. He is also an associate editor of the *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*. He serves on the editorial board of the *IEEE Internet Computing* and *IEEE Computer Magazine*.

Junliang Chen received the BS degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 1955, and the PhD degree in electrical engineering from the Moscow Institute of Radio Engineering, Moscow, Russia, in 1961. He has been with the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, since 1955, where he is currently the chairman and a professor at the Research Institute of Networking and Switching Technology. His current research interests include communication networks and next-generation service creation technology. He was elected as a member of the Chinese Academy of Sciences, in 1991, and a member of the Chinese Academy of Engineering, in 1994 for his contributions to fault diagnosis in stored program control exchange. He received the First, Second, and Third Prizes of the National Scientific and Technological Progress Award, in 1988, 2004, and 1999, respectively.