Enabling DNN Acceleration with Data and Model Parallelization over Ubiquitous End Devices

Yakun Huang, Xiuquan Qiao, Wenhai Lai, Schahram Dustdar, *Fellow, IEEE*, Jianwei Zhang and Jiulin Li

***Abstract*—Deep neural network (DNN) shows great promise in providing more intelligence to ubiquitous end devices. However, the existing partition-offloading schemes adopt data-parallel or model-parallel collaboration between devices and the cloud, which does not make full use of the resources of end devices for deep-level parallel execution. This paper proposes eDDNN (i.e. enabling Distributed DNN), a collaborative inference scheme over heterogeneous end devices using cross-platform web technology, moving the computation close to ubiquitous end devices, improving resource utilization, and reducing the computing pressure of data centers. eDDNN implements D2D communication and collaborative inference among heterogeneous end devices with WebRTC protocol, divides the data and corresponding DNN model into pieces simultaneously, and then executes inference almost independently by establishing a layer dependency table. Besides, eDDNN provides a dynamic allocation algorithm based on deep reinforcement learning to minimize latency. We conduct experiments on various datasets and DNNs and further employ eDDNN into a mobile web AR application to illustrate the effectiveness. The results show that eDDNN can achieve the latency decrease by 2.98x, reduce mobile energy by 1.8x, and relieve the computing pressure of the edge server by 2.57x, against a typical partition-offloading approach.**

***Index Terms*—Deep learning, ubiquitous end devices, cross-platform, distributed DNN, collaborative inference.**

I. Introduction

DEEP learning (e.g., Deep neural networks, DNNs) is currently a representative way of achieving Artificial Intelligence (AI) in numerous applications [1], [2], [3]. With the maturity of AI technology and the reduction of AI hardware costs, more and more smart end devices such as smartphones, AR/VR glasses, smart cameras, etc., also including Internet of Things (IoT) devices, have emerged in daily life [4], [5], [6]. Smart end devices are increasing, scenarios and services that require AI capabilities are constantly enriching [7]. However, the limited computing capability of end devices is hard to support executing computationally intensive DNNs for AI services independently. Thus, the need for collaborative inference among multiple end devices is increasing. Also, it causes a demand for integrating these scattered ubiquitous end devices and finding the best combination to provide collaborative AI services in different scenarios [8], [9]. For instance, Huawei has developed the HiAI 3.0, an open platform for AI devices, which makes contributions on establishing the connection among end devices and enabling distributed AI capability [10].

However, the majority of existing attempts of DNNs on ubiquitous end devices leave an unsatisfactory experience with the following two execution schemes in Fig. 1. The first execution scheme is the non-collaborative execution, including mobile-only and cloud-only, which executes the entire DNN on the end device or transmits tasks to the remote cloud for offloading DNN computations. The mobile-only approach performs a high DNN execution latency as conventional end devices lack computing capability. With the cloud-only approach, large amounts of data (e.g., image, audio, and video) are sent to the cloud via the wireless network, which results in high transmission latency and mobile energy consumption. Moreover, offloading all computations to the remote cloud will significantly increase the computing pressure and cost of the remote cloud, which also raises new privacy concerns for users (e.g., home security cameras) [11]. The

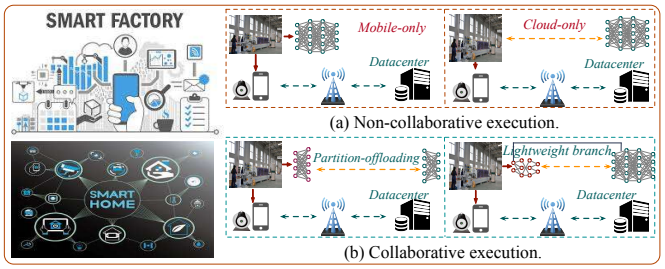


Fig. 1. Non-collaborative and collaborative execution schemes of the DNN on end devices.

second execution scheme is collaborative execution, including partition-offloading and adding a lightweight branch. Partition-offloading dynamically distributes the computations between the end device and the remote cloud [12], [13], [14], protecting data privacy and reducing the computing pressure of the remote cloud. The lightweight branch approach adds efficient branches to the initial DNN for executing inference on the end device independently. It also provides a collaborative mechanism for accuracy compensation [15], [16], [17], [18]. Obviously, these collaborative solutions mainly optimize DNN computations between the end device and the backend server. They ignore the use of idle computing resources of ubiquitous end devices for collaboration. Thus, how can we move computations further close to the edge by using ubiquitous end devices while providing an acceptable performance, reducing latency and the pressure of the backend server, and improving the resource utilization of end devices?

With the advent of the 5G era, mobile edge computing (MEC) and Device-to-Device (D2D) communication technol-

Y. Huang, X. Qiao and W. Lai are with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail:{ykhuang, qiaoxq, laiwenhai }@bupt.edu.cn. (X. Qiao is the corresponding author.)

S. Dustdar is with the Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria. E-mail:dustdar@dsg.tuwien.ac.at.

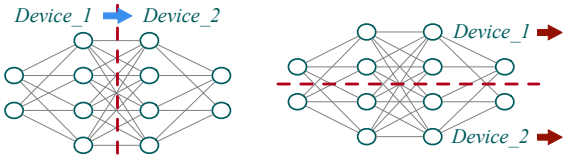
J. Zhang is with the Capinfo Company Limited, Beijing 100161, China. E-mail:zhangjw@capinfo.com.cn.

J. Li is with the Beijing National Speed Staking Oval Operation Company Limited, Beijing 100092, China. E-mail:lijiulin@bjucd.com.

ogy have gradually been deployed and applied in different scenarios. MEC has the benefit of low communication costs compared with offloading computations to the remote cloud and relieves the burdens of the core network. Also, it is expected to use D2D communication over ubiquitous end devices to achieve distributed DNN inference, making full use of computing resources of ubiquitous end devices. However, implementing distributed DNN inference over ubiquitous end devices still faces three major challenges:

- **How to execute distributed DNN over heterogeneous end devices with different computing architectures and inference frameworks?** Ubiquitous end devices have a large difference in the brand, the computing system (e.g., iOS, Android, and embedded OS), the inference framework, and the computing capability. This requires configuring and deploying different DNNs according to the characteristics of devices, thus providing distributed DNN inference across heterogeneous end devices. Besides, various communication protocols among heterogeneous end devices also hinder the implementation.
- **Layer dependency of DNN inference seriously hinders the implementation of distributed DNN inference on ubiquitous end devices in parallel.** DNN inference process inputs a task and acquires the different features layer by layer until obtaining the result. This shows a strong layer dependency (i.e. the input of the next layer is the output of the previous layer). Although partition-offloading approaches divide the DNN into multiple pieces by layers and distribute them to ubiquitous end devices for execution, it is hard to execute them in parallel due to such serial inference characteristic.
- **How to provide a dynamic allocation for end devices with different computing capabilities that can execute efficient DNN with optimal latency and resource utilization?** Note that end devices have different computing capabilities. The available computing resource of end devices is dynamic with the change of running applications. Thus, the key to achieving distributed DNN across ubiquitous end devices is to design an effective subtask allocation algorithm to match and execute appropriate computation. Additionally, allocating DNN tasks to end devices with a reliable and efficient inference is also significant for optimizing latency and resource utilization.

To address these concerns and enable distributed DNN inference, we implement D2D communication and collaboration over heterogeneous end devices with the help of cross-platform web technology and Web Real-Time Communication (WebRTC) protocol. We develop a distributed DNN inference scheme on a web platform, named eDDNN, that efficiently reduces layer dependency and accelerates the whole process. Unlike traditional approaches that partition DNN by layers (i.e. vertical partition), eDDNN divides the task and the corresponding DNN model into pieces in Fig. 2 (i.e. horizontal partition whose submodel has all DNN layers and the size of each layer becomes smaller). Therefore, we can acquire almost independently DNN inference on submodels and reduce the layer dependency of inputs at dividing edges. Besides, we establish an inference dependency table with the help of the edge server and broadcast it over end devices to load required dependency information from each other. Then, we propose a dynamic allocation algorithm named DecisionMaker, based on deep reinforcement learning for optimizing the overall processing latency. With these efforts, an end device can share other devices resources for acceleration and a better experience. The contributions can be summarized as follows:



(a) Traditional partitioning (b) eDDNN partitioning

Fig. 2. Traditional partitioning and eDDNN dividing.

- Developing a distributed DNN inference scheme over heterogeneous end devices leveraging the cross-platform web, which horizontally divides the task and DNN model into pieces and executes them on each end device almost independently by sharing a DNN dependency table.
- Proposing a dynamic allocation algorithm to reduce complexity, which adaptively matches and executes subtasks on end devices, reduce overall execution latency, and optimize resource utilization of end devices.
- Evaluating the proposed eDDNN on various DNNs and datasets, and implementing a collaborative recognition for mobile Web AR, showing the satisfactory performance against a typical partition-offloading approach.

II. Background and Motivation

Cooperative D2D communication technology is widely studied and applied for data transmission across various end devices, which uses the end device as the relay for communication [19], [20]. This work intends to explore data transmission and collaborative DNN inference based on cross-platform web technology over heterogeneous end devices. Therefore, to achieve D2D communication on the web and collaborate with each other, we establish a web-oriented D2D communication for multiple end devices based on the WebRTC [21] protocol in Fig. 3. Each end device establishes a device-to-device connection with others. Note that the MEC server plays a vital role in registering the connections and managing the connections of end devices that join or leave. Besides, data may be transmitted through the MEC server viewed as a relay rather than that only transmits data among devices. We can also view the MEC server as an application provider, such as providing DNN models for recognition and rendering 3D models in a mobile Web AR application.

Considering that if we put all intensive calculations on the MEC server, this may generate many data transmission and computing pressure. Meanwhile, most ubiquitous end devices are in idle states, which reminds us to use these idle computing resources to reduce the computing pressure, avoid large amounts of data transmission, and improve resource

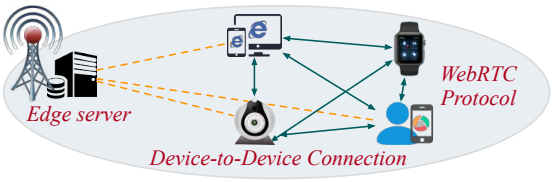


Fig. 3. Illustration of D2D communication technology for ubiquitous end devices based on WebRTC.

utilization. Thus, the task can be processed by the collaboration without transmitting tasks to the MEC server. Hence, moving computations close to the edge by using end devices, improving resource utilization, and reducing the MEC server's computing pressure is the primary motivation.

III. Proposed Distributed Deep Neural Network

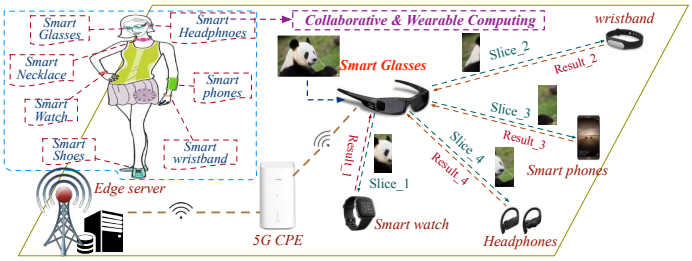


Fig. 4. A typical application scenario of eDDNN.

We present a typical application of eDDNN in wearable computing in Fig. 4. When a user is experiencing AR applications using smart glass, with eDDNN, we can offload intensive DNN recognition computations to other wearable devices, such as smartphones, smart wristbands, and smartwatches. eDDNN can greatly reduce the computing pressure of the glass, thereby enabling the glass to be thinner and lighter and the user experience to be better. Besides, eDDNN offloads the DNN computation to other end devices as much as possible. It is necessary that the wireless customer premise equipment (CPE) and the edge server can also perform part of the computation, thereby avoiding the data transmission between the end device and the edge server. Considering that the wearable end devices have different brands, different systems, and computing architectures, which make it difficult to perform communication and collaborative DNN inference. To this end, eDDNN implements distributed DNN inference over heterogeneous end devices based on cross-platform web technology and WebRTC protocol. The main process includes task pre-processing, distributed inference, and merging results.

A. Distributed and parallelizing inference of eDDNN

In Fig. 5, we describe how to perform distributed DNN inference among ubiquitous end devices in parallel, consisting of establishing the connection, dividing the task and DNN models, distributed inference, verifying and merging the results. The detailed eDDNN process is as follows:

STEP 1. Establishing D2D connections among end devices and updating connections with the help of the edge server. Generally, the edge server selects end devices that have available and idle resources as much as possible for participation. In the event of an end device failure or loss of connection, the edge server updates the connections among end devices.

STEP 2. The task requestor (i.e. Glass in Fig. 4) sends a request signal to the edge server, consisting of the task size, required DNN model, and available resource of the current end device that can provide.

STEP 3. Once the edge server receives the request from the requestor, and it divides the task and DNN models into Num pieces by DecisionMaker, which provides dynamic allocation according to the current status. Also, it establishes an inference dependency table based on the allocation, which mainly supports the inference of the convolutional layer at the split edge of pieces and then broadcasts the dependency table, subtasks, and submodels. Note that Num is generally determined according to the size of the input and DNN models.

STEP 4. End devices execute sub-inference according to the dependency table and send response results to the requestor immediately. There may occur some abnormalities during distributed execution such as the connection loss or insufficient computing resource, making the collaborative device offline and unavailable Besides, the end device with low computing capability consumes much time on DNN inference, triggering the failure. Thus, we set a response latency for each end device to monitor the status by the edge server. Once an end device is found to be offline, it immediately forwards its corresponding computations to other available end devices and updates the dependency table to keep consistency.

STEP 5. When the requestor collects the results returned by all collaborative devices, it immediately merges sub results to output the final results. The requestor verifies that all distributed calculation results have been received; otherwise it will execute or request the edge server for the rest inference.

We present distributed DNN inference on the convolutional layer and dense layer in detail. We first illustrate the dependency table of three end devices in Table I. We divide the input evenly into three pieces for a given input and generate a dependency table according to pieces. For example, Table I shows the dependency relations of piece_1, piece_2, and piece_3 of the sample image on device_1, device_3, and device_2, respectively. Contents of the dependency column in the table indicate that the current piece relies on some edge information on other pieces.

TABLE I: Dependency table for distributed inference

$Num=3$	*Device*	*Dependency piece*	*Dependency device*
device_1	piece_1	device_2	device_3
device_2	piece_3	device_1	device_2
device_3	piece_2	device_3	device_1

For instance, piece_1 and piece_2 have a specific dependency relationship at the dividing edge of the initial image. This is because computational calculation requires partial edge inputs of two pieces. Thus, with the dependency table, it is possible to execute partial DNN execution almost independently and reduce the amount of data dependency compared

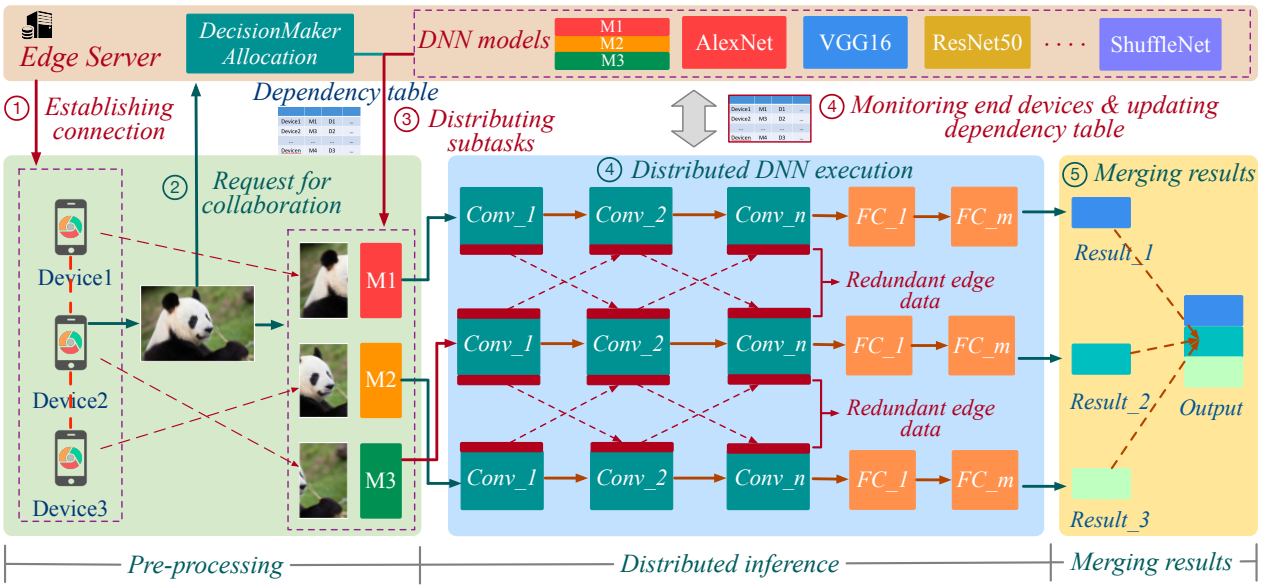
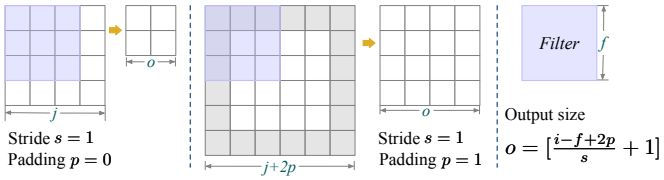
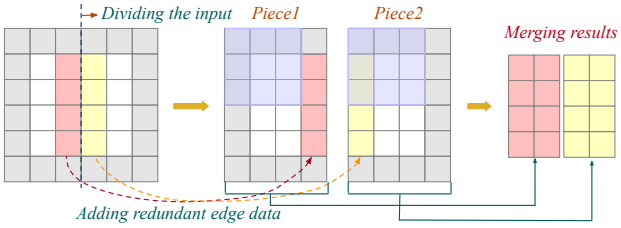


Fig. 5. eDDNN inference procedure.

with traditional DNN inference In also can directly obtain dependency data from other end devices. Besides, we use this dependency table to monitor and update offline end devices in time, and forward tasks to other available devices, increasing the robustness.



(a) General convolutional layer.

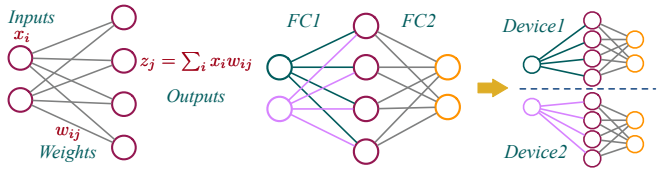


(b) Distributed convolutaional layer.

Fig. 6. Convolutional layer computation.

We discuss how to solve distributed independent DNN inference in the convolutional layer by adding redundant edge data based on the dependency table. Fig. 6(a) shows the input size and output size of general conversational layers. Moreover, Fig. 6(b) describes distributed inference when the image is divided into two pieces. To obtain the same inference results as in Fig. 6(a), we have to add a small portion of redundant edge data to each image piece for obtaining the correct result the filter. The redundant input data may result in the error and be transmitted to the final inference results. Besides, the dependent inference is only a small part of the input data involved at the edge of pieces. Thus, in practice, pieces without dependency data can be performed independently, accelerating the inference.

Fig. 7 describes how to perform distributed DNN layer inference in terms of a dense layer or fully connection (FC) layer. In Fig. 7(a), each activation is calculated from the sum of inputs and weights for two dense layers. These parameters are determined during the training phase and remaining constant in the inference process. We present a computing schematic diagram of a basic unit composed of 2-layer FC, which divides it into two parts according to the input size of the first FC layer and distributes them to different end devices. We only need to sum the results of two end devices during the merging phase, which has correct results and has been tested on classic DNNs and datasets. Additionally, we directly distribute the last FC layer on the requestor when the end device has sufficient resources. It can avoid redundant calculations caused by dividing the last layer. Hence, the requestor receives the result of the first FC layer calculated by others, merges these results, and executes the last FC layer.



(a) General fully connection. (b) Distributed fully connection.

Fig. 7. Fully connection layer computation.

B. DecisionMaker for online allocating in eDDNN

When a target end device requests the edge server for allocation, the edge server calculates an optimal allocation according to the task and available resource status of participating end devices. Note that the computing capabilities of

ubiquitous end devices are various, and the current computing status of end devices and available resources are dynamically changing. Therefore, for a DNN task request, the edge server should provide a comprehensive allocation by considering the resource status of all end devices and global status. To this end, we come up with a dynamic allocation algorithm based on deep reinforcement learning, named DecisionMaker, from the perspective of reducing complexity.

Let $\mathcal{D} = \{d_1, d_2, ..., d_M\}$ denotes the end device set of M end devices. We define the task processing capability of each end device as $\mathcal{C} = \{c_1, c_2, ..., c_M\}$. The current available computing resource of each end device at time t is $\mathcal{R}_t = \{r_t^1, r_t^2, ..., r_t^M\}$. Each end device can run multiple DNN subtasks, which can not be further divided at time t. End device generates a DNN task of $\mathcal{H} = \{h_1, h_2, ..., h_N\}$ with the probability of $\alpha_i \in (0, 1)$, which obeys poisson distribution. Let $\mathcal{T} = \{1, 2, .., N\}$ be all DNN subtasks set at a time slot. $h_j = <p_j, q_j>, j \in \mathcal{T}$ denotes j^{th} subtask of all DNN subtasks that need to be inferenced, where p_j and q_j are the amounts of computing resource and data communication required for the inference, respectively. Since any subtask can be distributed on any end device, we use $\beta_{i,j,k}, (i \leq k)$ define the probability that d_i offloads subtask $h_{i,j}$ to d_k for collaboration. Besides, we define the uplink transmission rate between d_i and d_k as:

$$r_{i,k} = B \log_2(1 + \frac{g_{i,k} P_i}{B N_0}), \tag{1}$$

where B denotes D2D communication bandwidth, P_i is the transmitting power of d_i, g_i is the information gain, and N_0 is white Gaussian noise variance. Thus, we can describe the inference latency of $h_{i,j}$ on d_i as $t_{i,j}^{inf} = \frac{p_{i,j}}{c_i}$. Note that we ignore the communication cost of the broadcasting dependency table over the edge server and end devices. When d_i requires collaboration from d_k for the subtask $h_{i,j}$, communication latency can be defined as $t_{i,j,k}^{com} = \frac{q_{i,j}}{r_{i,j}}$. Based on these definitions, we describe the task allocation as minimizing the overall latency of executing all DNN subtasks over distributed end devices as

$$min \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=1}^{M} \beta_{i,j,k} (t_{i,j,k}^{com} + t_{k,j}^{inf}) + (1 - \beta_{i,j,k}) t_{i,j}^{inf}. \tag{2}$$

It is hard to meet the real-time allocation for eDDNN by using optimization algorithms such as genetic algorithms for solving the above problem. For a given environment in Fig. 8, we assume that an agent is interacting with it. During each time interval t, the agent chooses an action a_t via observing the state s_t. By executing the selected action, the state is changing from s_t to s_{t+1}. The agent decides its actions according to the policy, which is a probability distribution $\pi(s, a)$ and is performed by a deep neural network. Also, we collect historical records from the online allocating phase to: (1) provide more training samples for the offline DRL model, and (2) more importantly, we use the historical records to train a reward prediction model based on another DNN model. This is because using the learning-based rewards can effectively address our inability to obtain sufficient training samples [22]. In general, the offline training phase of DecisionMaker includes the training of the reward model and the DRL-policy model. We formulate dynamic task allocation as a DRL-based question and describe DecisionMaker's design by emphasizing the state space, the action space, and the reward.

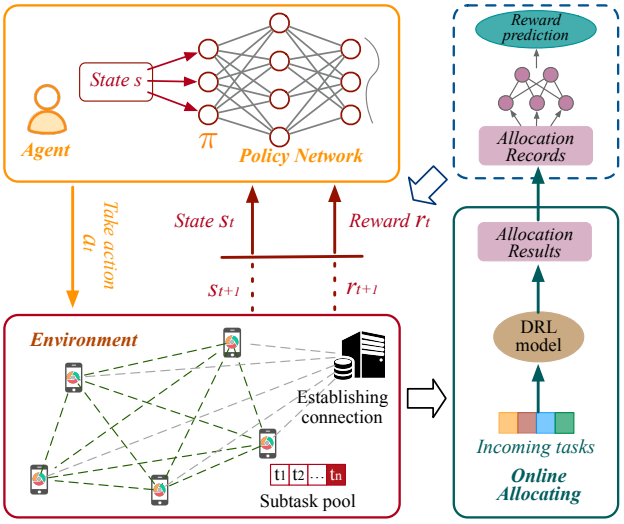


Fig. 8. DecisionMaker for online subtasks allocation.

State space. The input state $s_t = \{\boldsymbol{x}, \mathcal{R}_t, \boldsymbol{B}\}$ includes the DNN subtasks, available computing resources of devices. $B = M \times M$ represents the network conditions between M collaborative devices. Commonly, the number of collaborative devices and the number of subtasks are dynamically changing. It increases the obstacles to design and train the policy due to the fixed input layer. We define a large M for the collaborative devices. When the number of collaborative devices M' is less than M, we set $M - M'$ tasks as $\emptyset$. It is impossible to set a max value effectively for representing dynamic subtasks due to a large number of subtasks, which may cause training difficulties in convergence when the actual number of sub-tasks is small. To this end, we use a trick by defining a subtask pool to store incoming subtasks for each time interval. (i.e. at each time interval [23], [18]. L subtasks are scheduled from the subtask pool where L is the input length of the DNN subtasks in s_t. This means that we need to freeze the time and use multiple allocations to complete all the subtasks in the subtask pool at each time interval. Also, Fig. 9 gives a clear case of state space for better understanding by describing each end device's currently available computing resource and subtasks waiting to be allocated. Red blocks denote computing resource, and blue blocks are the communication cost. When an end device requests a DNN computing collaboration, it divides the DNN task into multiple subtasks and adds to the task queue to be allocated. Note that device2 cannot cooperate to complete any subtask in the current state; thus, these subtasks are allocated to other end devices.

Action space. As mentioned in the state space, since we use a subtask pool to fix the input of the policy network effectively, this also allows us to reduce the action space from M^N down to linear in L. Thus, the action space is given by $a_t = \{a_1, a_2, ..., a_L\}$. $a_i = 0$ means the i^{th} subtask will not be assigned, and $a_i = d_j, j \in [1, M]$ indicates allocating the

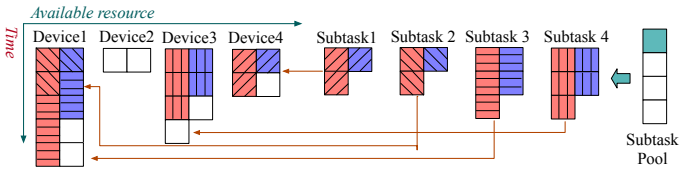


Fig. 9. An example of state, with four devices and subtasks.

i^{th} subtask to j^{th} device. At each time interval, time is frozen until completes the assignment of all subtasks in the subtask pool. By defining such subtask pool and multiple allocating mechanisms in each time interval, the agent accomplishes the dynamic assignment and maintains a linear action space in L.

Rewards. Our goal is to minimize the average subtask completion time by training the DNN policy network. Thus, the completion time of subtasks can be set as the reward, which is defined as $\sum_{l\in\mathcal{N}} \frac{-1}{T_l}$. $\mathcal{N}$ is all subtasks that are currently assigned and pending. DecisionMaker is to maximize the long-term reward; thus, the reward is the negative sum of all latency at each time step. The agent only receives the reward until the current subtasks in the pool are allocated entirely, which means the agent has no rewards of intermediate actions in a time interval. Thus, we can maximize the cumulative reward to mimic, minimizing overall complement time when setting the discount factor as $\gamma = 0.9$.

Design and Training of the DNN models. We first introduce the DNN model used for the policy network in DecisionMaker. N subtasks and M collaborative devices are connected to a fully connected layer, and the next two hidden layers have 196 neurons, respectively. Before the output layer, another hidden layer with 128 neurons are used. Then, since we use another DNN model to predict the reward given the incoming subtasks and collaborative device states, we can generate many samples for training the reward prediction model. We also use three hidden layers with 128 neurons in the reward prediction network before the output layer. Then, we introduce the training of the policy network of DecisionMaker. The policy network is trained in various episodes, which inputs N subtasks for each episode for allocating according to the policy network until all subtasks are allocated. In each training epoch for each task set that simulates E episodes, we compute the probabilistic space of actions with the policy and improve the policy for all subtasks 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. Generally, the optimized objective is to maximize the expected reward, and the gradient of the objective can be described as:

$$\Delta E_\theta[\sum \gamma^t r_t] = E_{\pi_\theta}[\Delta_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)], \tag{3}$$

where $E_\theta[\sum \gamma^t r_t]$ represents the expected cumulative reward. $\gamma \in [0,1]$ is to discount reward and we set it as 0.9 in the experiment. $Q^{\pi_\theta}(s,a)$ is the expected reward. Besides, we use the following gradient-descent method to update the parameters for the policy network.

$$\theta \Leftarrow \theta + \alpha \sum_t \Delta_\theta \log \pi_\theta(s_t,a_t) v_t, \tag{4}$$

where α denotes the adjustment size. Since our policy gradient of Eq. (3) 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. Once we acquire the trained DRL-based DecisionMaker, eDDNN chooses to top Num collaborators for executing distributed DNN inference following the detailed eDDNN process in Fig. 5. Also, we describe in detail how to train the reward prediction model. By collecting the history records, we use these samples to train the reward DNN model by supervised learning. We define the loss function as:

$$L(g,g') = \frac{1}{|G|} \sum\nolimits_{n\in[N]} \frac{|g'_n - g_n|}{g'_n}, \tag{5}$$

where g_n and g'_n represent predicted inference latency and the ground truth label [24].

IV. Evaluation

A. Experiments setup

1) Datasets and Benchmarks: We evaluate the correctness and effectiveness of eDDNN using typical deep neural networks such as AlexNet [25], ResNet-50 [26] and ShuffleNet [27] on CIFAR-10 [28] and ImageNet-150K [29]. ImageNet-150K is the subset of ILSVRC ImageNet and contains 183K training images and 7.5K testing images belonging to the top 150 most popular object categories. To illustrate the improvements of our DecisionMaker allocation regarding the latency, the mobile energy, and resource utilization against some baseline methods, including two typical methods, Random allocation, and Distance-prior respectively. Random allocation distributes DNN computations to the connected end devices randomly, and the Distance-prior method prefers to select nearby end devices for collaboration. We also compare the DecisionMaker with the two latest DRL-based allocation methods, including DRLoS[18] and DeepRM [23] to highlight the strengths and effectiveness. Last, we compare eDDNN with other non-collaborative DNN inference schemes, mobile-only and cloud-only, and existing collaborative inference methods such as Neurosurgeon [12], JointDNN [14], LCRS [16], and DeepAdapter [17] to demonstrate the improvements.

2) End devices and edge server setup: To establish a real scenario for mobile Web AR application with three smartphones, both installing Chrome browser and running Andriod 8.0, a HUAWEI Mate10 with 4 GB RAM, Samsung Galaxy S5 with 4 GB RAM, and iPhone X with 3 GB RAM, respectively. We use a common server with a six-core Interprocessor of 2.9 GHz and 16 GB RAM running Ubuntu 18.04 LTS, which is deployed near the base station. We present the core network topology with a max uplink bandwidth of 150 Mbps and a max downlink bandwidth of 600 Mbps. It is a commercial 5G network provided with communication latency between the mobile device and the edge server of 5-10ms by China Unicom in Fig. 10. As for the D2D communication links, mobile devices are interconnected via the D2D (Wi-Fi Direct) communication technique with a bandwidth of 85-300 Mbps and a latency of 5-20 ms. We use a HUAWEI 5G CPE to connect to the base station and use Wonder Shaper [30] to control the network on the edge server.

Fig. 10. The network topology in a real scenario.

3) Measurements: We introduce tools and methodologies to measure the latency, mobile energy, and resource utilization as follows: (a) **Latency measurement.** the entire latency can be calculated based on two timestamps before and after a complete DNN computation request. We repeat the same DNN task request multiple times and use the average latency as the final latency performance to reduce random errors. (b) **Mobile energy measurement.** We use a hardware power monitor with a model number of AAA10F [31]. 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. (c) **Resource utilization measurement.** We define the resource utilization as the computing resource variance of participated end devices. The resource usage variance of $d_t = \{d_t^1, d_t^2, ..., d_t^m\}$ at time slot t can be calculated as:

$$RU_t^2 = \frac{\sum_{i=1}^{m}(d_t^i - \bar{d}_t)}{m-1}, \tag{6}$$

where $\bar{d}_t$ denotes the mean value of the resource usage of end devices at time slot t.

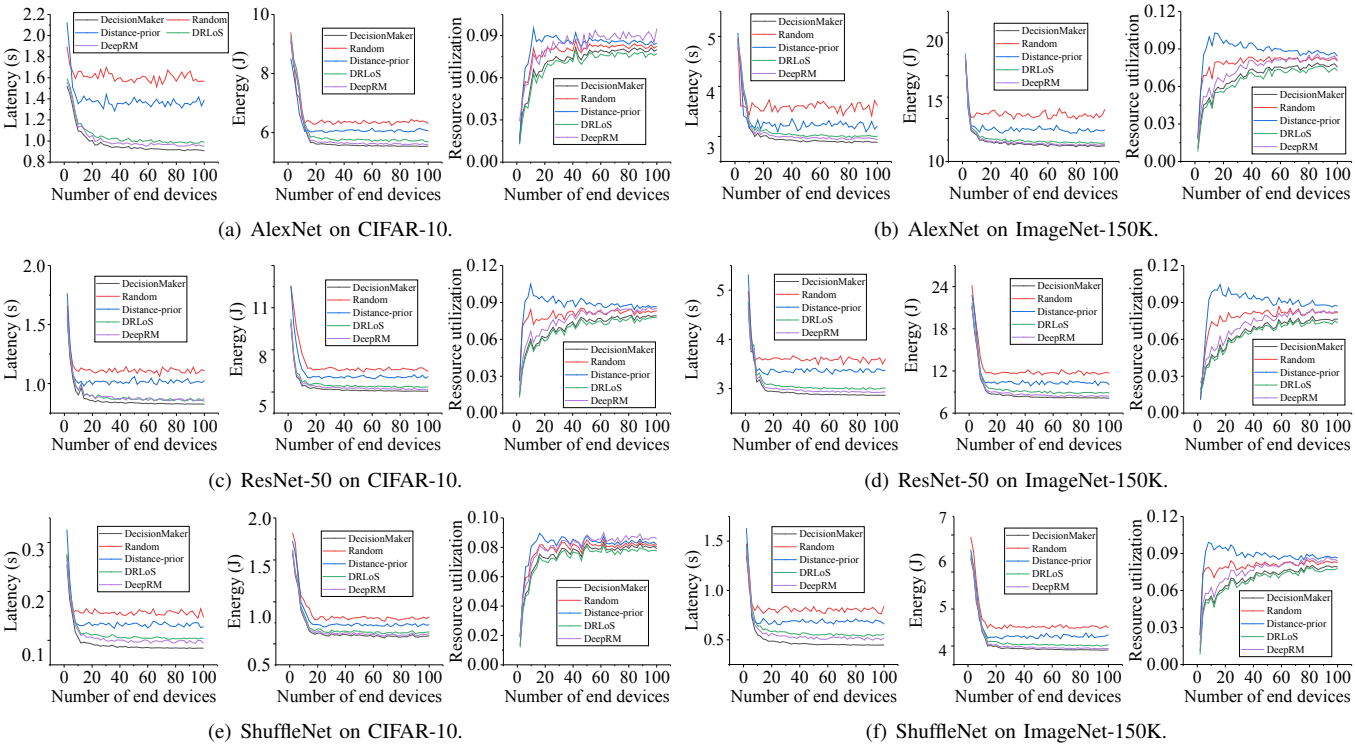
B. Performance of eDDNN

We simulate deploying a large number of end devices to verify the effectiveness of eDDNN in Fig. 11, and evaluate a small number of end devices for a real-world scenario. Mobile energy consumption is based on the benchmark of the actual mobile transmission energy and DNN inference energy, measured by the method in subsection *A*. The parameters involved in the experiment are as follows: the downlink bandwidth is 400 Mbps, and the uplink bandwidth is 100 Mbps. The amount of end devices is 100. Task sizes of ImageNet and CIFAR-10 are 127 KB and 3 KB, respectively. Moreover, DNN model sizes are 221 MB, 76.3 MB of AlexNet, 90.7 MB, 31.6 MB of ResNet-50, and 8.22 MB, 2.4 MB of ShuffleNet for ImageNet and CIFAR-10, respectively. We simulate a DNN task requestor that obeys the Bernoulli distribution with a request frequency range from 10% to 130%, and randomly set α and β. For three DRL-based methods, we both use Adam optimizer [32] with a learning rate of 0.0001 and a mini-batch of 64. The reward discount factor of DecisionMaker, DeepRM, and DRLoS is set as 0.9, 1.0, and 1.0, respectively. Besides, we set the task processing batch and the output size of the policy network L as 100. And the training iteration of all methods is 1500. DRLoS's reward differs from the other two methods in that it maximizes resource utilization.

We see that: (1) DecisionMaker performs better than other allocation methods in all indicators. This is because DecisionMaker considers the system status of all end devices and leverages them to acquire for the best allocation iteratively. However, the other two methods only consider allocation on the part of end devices. (2) When the number reaches 10, we observe that the latency drops significantly, the mobile energy also shows a large decrease, while the resource utilization variance of all end devices increases significantly. Concretely, when the number of collaborative end devices is set at 3, the latency decreases, and the probability of acquiring collaboration for the requestor increases. However, when the number of end devices increases to 20 or more, the latency tends to be stable without further improvements. As for mobile energy consumption, with the increase of collaborative end devices, mobile energy consumption shows a similar trend to latency. Meanwhile, the mobile energy of data transmission is greater than that of inference, resulting in that the whole mobile energy tends to decrease with the collaboration of increasing end devices. However, the resource utilization variance of all end devices shows a strong upward trend when the number of end devices reaches 10. This is because the indicator calculates the resource utilization variance of all end devices, and the number of collaborative end devices is definite. End devices without participating in collaboration have a large increase and perform a higher resource utilization variance. Besides, with the continuous increase of the number of end devices, resource utilization variance of all allocation methods gradually becomes consistent. (3) Judging from the performance on different datasets and DNN networks, DecisionMaker shows a similar curve trend. Whether it is an extensive network or a small network, in a distributed inference, DecisionMaker is more beneficial to search for the optimal collaborative end devices to the requestor.

When we compare DecisionMaker with other advanced DRL-based allocation methods, we see that (1) DecisionMaker has lower latency and mobile energy performance than others. This is because traditional DRL methods are challenging to learn all the possible samples and influence the convergence of the policy network with insufficient training samples. The advantage of our DecisionMaker is that it can automatically learn the reward from the historical records and avoid using manual features, thus improving the convergence. When applied to the eDDNN, it also shows better system latency and mobile energy performance. (2) Another noteworthy point is that DRLoS performs better than DecisionMaker and DeepRM in resource utilization. This is mainly because the optimization goal of DRLoS is resource utilization, which is directly introducing its poor performance in the other two indicators. Although DRLoS has better performance in resource utilization, our DecisionMaker is outstanding in all indicators than others, especially in latency and mobile energy.

In Fig. 12, we further discuss the influence of the average subtask slowdown of various allocation algorithms on the average load of collaborative devices. Since we can use the Android debug bridge (ADB) [33] to control the CPU load of Android devices, we use three Android devices as collaborative devices in the experiment in Fig. 12. The data of each point is the average value of 100 experiments, which are not used to train the DRL model. We see that (1) the increase of the average load of collaborative devices directly increases the average slowdown. It also shows that the reward prediction method has the improvement when the training samples are not enough. In addition, we notice that when the average load



(a) AlexNet on CIFAR-10. (b) AlexNet on ImageNet-150K.

(c) ResNet-50 on CIFAR-10. (d) ResNet-50 on ImageNet-150K.

(e) ShuffleNet on CIFAR-10. (f) ShuffleNet on ImageNet-150K.

Fig. 11. Performance of eDDNN on various datasets and typical DNNs. We increase the number of end devices, and Y-axis represents performance of various indicators, such as the latency, mobile energy, and resource utilization variance.

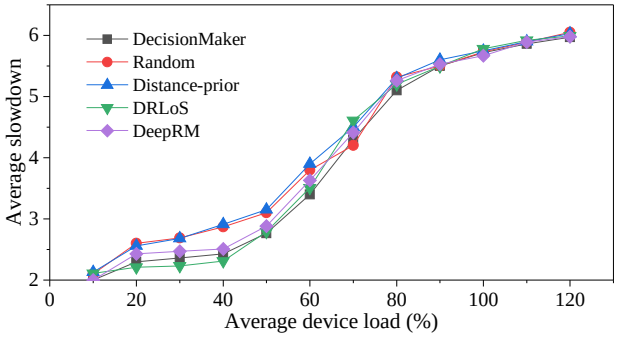
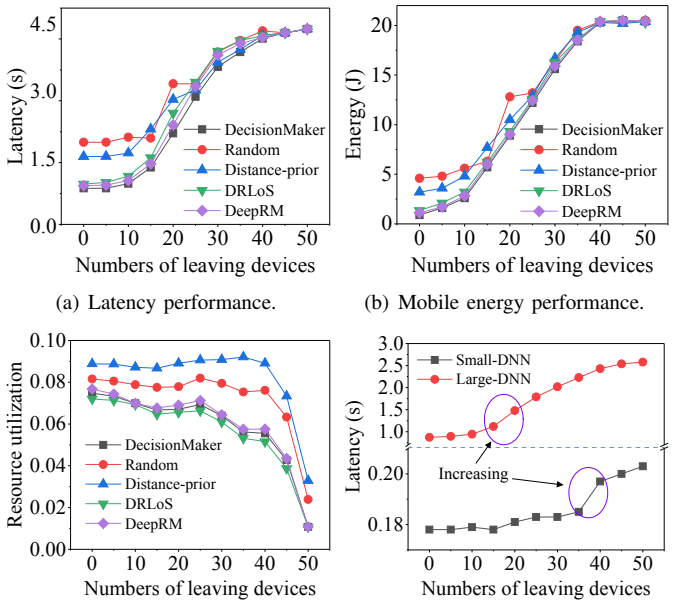


Fig. 12. Average slowdown at different device loads.

of collaborative devices is more than 90%, the performance difference of these DRL-based methods is not apparent. When the available resources of collaborative devices are insufficient, the direct factor affecting the performance will be the shortage of computing resources, while the subtask allocation has a small impact.

C. Robustness analysis of eDDNN

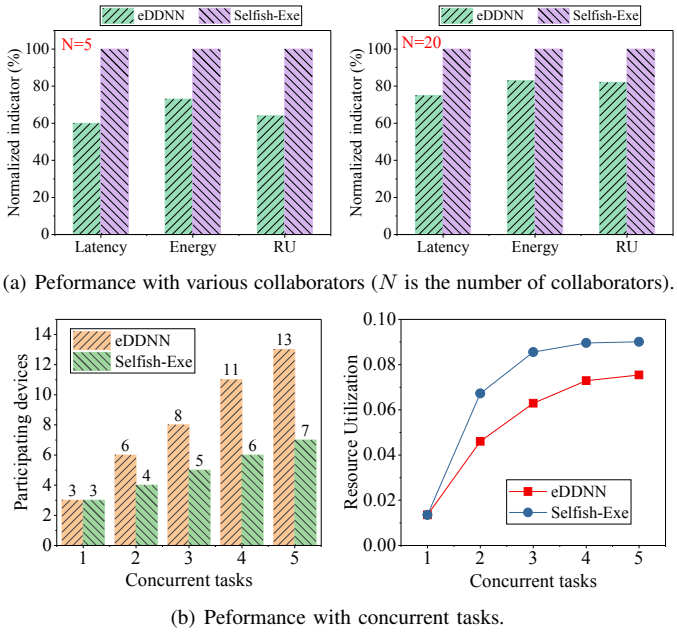
eDDNN requires the collaborative computing of multiple end devices, thus depending on a reliable communication condition and sufficient computing resources. Although we give priority to those end devices with sufficient computing resources and good communication conditions during the DecisionMaker allocation phase, end devices may still be in trouble, such as disconnecting and cancelling the current collaboration. To cope with unstable conditions and ensure reliable inference results, we use the edge server to obtain the response of end devices in real-time, monitor each end device's status, and then set it as 200 ms. We simulate experiments with collaborative end devices $P = 10$, which assumes that the task and DNN model can be divided into P pieces. We show the latency, mobile energy, and resource utilization performance when reducing the number of available collaborative end devices in Fig. 13. The simulation results show that: (1) with the continuous decrease of available end devices, the whole processing latency and mobile energy cost of eDDNN are constantly increasing. This indicates that failed end devices will cause repeat task distribution and inference, whose dependency area also affects the whole inference. Besides, resource utilization of eDDNN is lower than the other two methods, which show that DecisionMaker is more conducive to optimizing task allocation and improving resource utilization. (2) We also observe that when the available end devices decrease, the system's resource utilization continues to decrease, and eDDNN performs better than others. This decrease phenomenon is because all participated end devices calculate resource utilization variance, and eDDNN enables other available end devices in collaboration, thus improving the resource utilization. This also indicates that although lost end devices increase latency and mobile energy consumption, other end devices are enabled to participate in collaborative computing. (3) In Fig. 13(d), we see that different DNN networks and tasks are affected by the number of abnormal devices. In a small DNN network, since there is no need for many end devices for collaboration, few subtasks need to be forwarded and repeated. Hence, it avoids redundant calculations and mobile energy consumption caused by the failure of



(a) Latency performance. (b) Mobile energy performance.

(c) Resource utilization. (d) Performance on various DNNs.

Fig. 13. Robustness peformance of eDDNN



(a) Peformance with various collaborators (N is the number of collaborators).

(b) Peformance with concurrent tasks.

Fig. 14. Comparisons between eDDNN with Selfish-Exe.

end devices. For a large DNN network, the whole inference needs more end devices to participate in than small DNN, and then it increases additional calculations and data transmission. This indicates that it is necessary to ensure a reliable communication condition, and the number of collaborative end devices should be appropriate. Otherwise, with an unstable condition, the latency and mobile energy consumption performance will be challenging to accept in practical applications. When an end device participates in another end device's computation, it may also have the computing requirement at the same time. Generally, the end device inclines to stop current collaboration and turn to a task of its own, denoted as Selfish-Exe.

We describe the performance of latency, mobile energy, and resource utilization of eDDNN against the Selfish-Exe method in Fig. 14(a). The parameters are the same as the above experiments. We see that: (1) eDDNN can use the computing resources of ubiquitous end devices, whose DecisionMaker allocation algorithm guarantees the payoff of each end device. This also says that end devices that share the computing resource can also get others resources for their tasks. Thus, eDDNN performs a higher resource utilization than that of Selfish-Exe. This also indicates that only with more participation in collaborative computing can collaboration with other end devices be obtained. In Fig. 14(b), we study the latency, mobile energy consumption, and resource utilization as the increase of collaborative end devices. The results show that as the number of concurrent tasks participating in the collaboration device increases, the number of participating devices continues to increase. Especially, when Num equals 3, the number of participating end devices reaches 8, which indicates that at least two end devices participating in the task calculation of other end devices at the same time It shows the improvement of eDDNN compared with Selfish-Exe.

D. Application of eDDNN in mobile web AR

We employ eDDNN into a mobile web AR application, consisting of scanning and recognizing images and then rendering 3D models to the mobile web browser, interacting with users, and improving the user's AR experience. We use the ShuffleNet training on ImageNet as recognition DNN, and the entire AR application service is deployed on the edge server, and experimental settings can be found in the above description. In Fig. 15, we define the Huawei Mate10 as the requestor and the other two smartphones as the collaborators. The DNN execution follows the description in Section III.

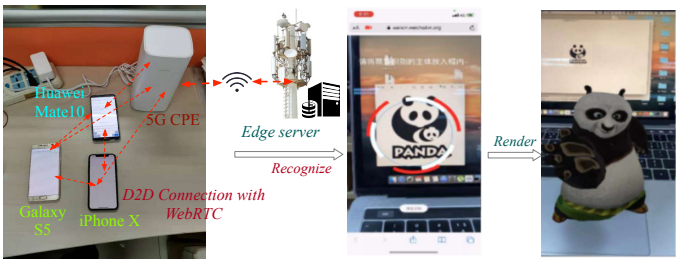


Fig. 15. A mobile web AR application with three end devices

We present the performance of eDDNN against other approaches such as mobile-only and edge-only on the latency, mobile energy, and resource cost of the edge server in Fig. 16. Also, we compare and analyze eDDNN against the traditional vertical partition-offloading and typical collaborative schemes, such as Edgent, Neurosurgeon, JointDNN, LCRS, and DeepAdapter from the latency, mobile energy, and resource utilization. The partition-offloading methods include Edgent, Neurosurgeon, JointDNN, whose main optimization objective is the latency. The lightweight branch used in LCRS and DeepAdapter is trained in advance. The network and other experimental settings are consistent with the above. Note that

the resource cost mainly refers to the CPU consumption of the edge server during the AR recognization. We observe that: (1) eDDNN performs better than that of Neurosurgeon in terms of latency, mobile energy cost, and resource cost. This is because eDDNN transmits a small amount of data among smartphones, such as dependency data. However, Neurosurgeon requires transmitting a large number of intermediate results between the smartphone and the edge server. Similarly, Edgent and JointDNN are similar to the partition-offloading scheme adopted by Neurosurgeon, so they have a similar performance to Neurosurgeon in performance and are not as good as eDDNN.

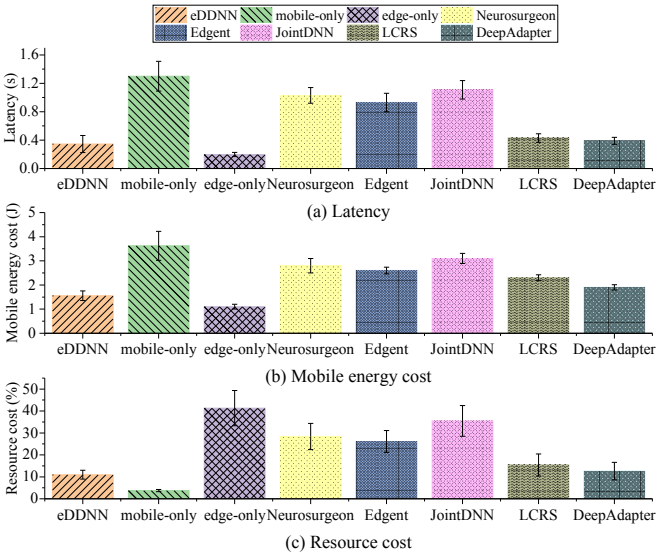


Fig. 16. Comparing eDDNN with other inference schemes.

(2) Mobile-only has the worst performance in terms of mobile energy cost and inference latency. Thus, although this approach has little resource consumption of the edge server, it is still difficult to widely use due to poor experience on the latency and mobile energy cost. Compared with the edge-only approach, eDDNN does not perform edge-only in terms of latency and mobile energy consumption, while it dramatically reduces the edge server's resource cost and computing pressure. In summary, eDDNN can achieve the goal of using ubiquitous end devices to reduce 2.98x execution latency and mobile energy cost by 1.8x, and relieve the computing pressure of the edge server by 2.57x, against Neurosurgeon.

V. Related Work

Deep learning (DL) with cross-platform web. Implementing DL inference with cross-platform web reduces the deployment and service cost on ubiquitous end devices. Caffe.js [34] and Karas.js [35] represent the typical libraries for implementing DNNs using JavaScript on the web. Recently, WebAssembly [36] becomes a new standard language of the web, dramatically accelerates the possibility of performing efficient DNNs on the web. For instance, TensorFlow.js [37] is a popular DL framework that provides training and inference on the web using various accelerating technology (e.g., WebAssembly, WebGPU). Also, WebDNN [38] and ONNX.js [39] provide a tool that can optimize, compile, and deploy any DNN models trained with other backend DNN frameworks onto a web-executable WebAssembly file. Our work takes full advantage of the cross-platform web to provide the possibility of implementing more efficient distributed DNN on heterogeneous end devices.

Distributed deep learning. The distributed DL inference scheme is mainly categorized into the collaboration between a single device and the cloud and execution among multiple devices and the cloud. For the first category, Neurosurgeon [12] and Edgent [13] use a single partition point to distribute the inference between the device and the cloud optimally. JointDNN [14] partitions the DNN into multiple small modules and dynamically assigns these modules to be executed cooperatively on the device or the edge. Besides, LCRS [16] and LcDNN [18] propose a lightweight collaborative scheme based on binary neural networks, which also offloading the computation between the device and the cloud. Similarly, DeepAdapter [17] provides a dynamic and adaptive compression model based on the device's computing capability. For the second category, DNNs are sliced and distributed to different devices or clouds (including edge cloud and remote cloud) using vertical or horizontal partitioning. DDNN [15] further extends BranchyNet [40] and distributes DL inference hierarchy over the cloud, the edge, and devices. Although Musical Chair [41], [42] and [43] distribute the DL inference over multiple IoT devices using data and model parallelism methods, these efforts are still in urgent need of better solutions for cross-platform collaboration on heterogeneous devices and scheduling of collaboration. Besides, there is some work focusing on distributed DL, mainly in discussing how to provide parallel training. BPT-CNN [44] proposes a two-layer parallel training architecture for the large-scale CNNs and addresses the key issues, such as data communication, synchronization, and workload balancing. Poseidon [45] is an efficient communication architecture for distributed DL on GPUs, reducing bursty network communication. In this paper, our work focuses more on how to provide cross-platform distributed DL inference services on ubiquitous end devices and considers how to maximize the resources of devices to obtain optimal inference latency and mobile energy.

Deep reinforcement learning for scheduling. Online task and resource scheduling is another important issue of distributed DL. Deep reinforcement learning has achieved promising results for online resource scheduling and task assignment in recent years. Mao et al. [23] view the task-packing problem as a learning problem that learns from experience to manage resources. MORL-BD [46] is a multi-objective reinforcement learning, which focuses on solving the problem of scheduling at optimal paths and applying it to the multi-route bicycle scheduling problem. More similar to our work are Harmony [24] and DRLoS [18]. However, Harmony primarily addresses machine learning cluster scheduling, aiming to place training jobs that minimize interference and maximize performance. DRLoS [18] is more concerned with how to schedule DL request tasks across multiple edge centers rather than the more fine-grained distributed collaborative inference

scheduling faced in this work. Hence, our work addresses how to maximize resource utilization for optimal inference experience over ubiquitous end device, rather than average task completion time, and more fully utilize the computing resource of end devices.

VI. Conclusion

This work proposes the eDDNN aiming at implementing distributed DNN over heterogeneous end devices by dividing the task and DNN model and executing them on end devices almost independently with a shared dependency table. We also provide a dynamic task allocation algorithm (DecisionMaker) to make full use of these collaborative end devices, which show advantages in reducing overall latency and reducing the computing pressure of the edge server. Experimental results and actual application have indicated the effectiveness of eDDNN. We only evaluate eDDNN on image recognition over heterogeneous end devices, and it can also be generalized to support other networks and applications. Besides, we plan to explore more robust and more stable execution to improve fault tolerance and availability in complex environments.

Acknowledgment

This research was supported in part by the National Key R&D Program of China under Grant 2019YFF0301500, in part by the Funds for International Cooperation and Exchange of NSFC under Grant 61720106007, in part by the 111 Project under Grant B18008.

References

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

[2] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[3] Y. Huang, H. Zhao, X. Qiao, J. Tang, and L. Liu, "Towards video streaming analysis and sharing for multi-device interaction with lightweight dnns," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

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

[5] X. Qiao, P. Ren, and e. Dustdar, "Web ar: A promising future for mobile augmented reality—state of the art, challenges, and insights," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 651–666, 2019.

[6] Y. Huang, X. Qiao, P. Ren, S. Dustdar, and J. Chen, "Edgebooster: Edge-assisted real-time image segmentation for the mobile web in wot," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7288–7302, 2020.

[7] H. Flores, P. Nurmi, and P. Hui, "Ai on the move: From on-device to on-multi-device," in *Pervasive Computing and Communications Workshops*. IEEE, 2019, pp. 310–315.

[8] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Collaborative execution of deep neural networks on internet of things devices," *arXiv preprint arXiv:1901.02537*, 2019.

[9] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Real-time image recognition using collaborative iot devices," in *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*. ACM, 2018, doi:10.1145/3229762. 3229765.

[10] "Huawei develops hiai 3.0 to create an innovative space for ai development." 2019, https://www.techgenyz.com/2019/11/21/huawei-develops-hiai-3-0-ai-development/.

[11] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[12] Y. Kang, J. Hauswald, and etl., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 615–629.

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

[14] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing.*, early access, Oct. 16, 2019, doi: 10.1109/TMC.2019.2947893.

[15] S. Teerapittayanon, B. McDanel, and etl., "Distributed deep neural networks over the cloud, the edge and end devices," in *International Conference on Distributed Computing Systems*, 2017, pp. 328–339.

[16] 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 *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1497–1506.

[17] Y. Huang, X. Qiao, J. Tang, P. Ren, L. Liu, C. Pu, and J. Chen, "Deepadapter: A collaborative deep learning framework for the mobile web using context-aware network pruning," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2020, pp. 834–843.

[18] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, S. Dustdar, and J. Chen, "A lightweight collaborative deep neural network for the mobile web in edge cloud," *IEEE Transactions on Mobile Computing*, early access, Dec. 08, 2020, doi: 10.1109/TMC.2020.3043051.

[19] Y. Jin, F. Liu, X. Yi, and M. Chen, "Reducing cellular signaling traffic for heartbeat messages via energy-efficient d2d forwarding," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1301–1311.

[20] F. S. Shaikh and R. Wismüller, "Routing in multi-hop cellular device-to-device (d2d) networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2622–2657, 2018.

[21] B. Sredojev, D. Samardzija, and D. Posarac, "Webrtc technology overview and signaling solution design and implementation," in *International Convention on Information and Communication Technology, Electronics and Microelectronics*. IEEE, 2015, pp. 1006–1009.

[22] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 505–513.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[24] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 505–513.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] X. Zhang, X. Zhou, M. Lin, and etl., "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[28] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," 2014, http://www.cs.toronto.edu/kriz/cifar.html.

[29] H. Liu, R. Wang, S. Shan, and X. Chen, "Learning multifunctional binary codes for both category and attribute oriented retrieval tasks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3901–3910.

[30] "Wonder shaper," 2017, https://github.com/magnific0/wondershaper.

[31] "Monsoon solutions," 2020, https://www.msoon.com.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] "Android debug bridge," 2018, https://developer.android.com/studio/command-line/adb.

[34] "Caffe.js framework," https://chaosmail.github.io/caffejs.

[35] "Keras.js," 2016, https://github.com/transcranial/keras-js.

[36] "World wide web consortium (w3c) brings a new language to the web as webassembly becomes a w3c recommendation," 2019, https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en.

[37] D. Smilkov, N. Thorat, Y. Assogba, and etl., "Tensorflow. js: Machine learning for the web and beyond," *arXiv:1901.05350*, 2019.

[38] M. Hidaka, Y. Kikura, Y. Ushiku, and etl, "Webdnn: Fastest dnn execution framework on web browser," in *ACM international conference on Multimedia*. ACM, 2017, pp. 1213–1216.

[39] "Onnx.js," 2018, https://github.com/Microsoft/onnxjs.

[40] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.
[41] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Musical chair: Efficient real-time recognition using collaborative iot devices," *arXiv:1802.02138*, 2018.
[42] Y. Huang, X. Qiao, S. Dustdar, J. Zhang, and J. Li, "Toward decentralized and collaborative deep learning inference for intelligent iot devices," *IEEE Network*, early access, 2021, doi:10.1109/MNET.011.2000639.
[43] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.
[44] J. Chen, K. Li, K. Bilal, K. Li, S. Y. Philip *et al.*, "A bi-layered parallel training architecture for large-scale convolutional neural networks," *IEEE transactions on parallel and distributed systems*, vol. 30, no. 5, pp. 965–976, 2018.
[45] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters," in *2017 USENIX Annual Technical Conference*, 2017, pp. 181–193.
[46] J. Chen, K. Li, K. Li, P. S. Yu, and Z. Zeng, "Dynamic bicycle dispatching of dockless public bicycle-sharing systems using multi-objective reinforcement learning," *ACM Transactions on Cyber-Physical Systems*, vol. 9, pp. 1–26, 2021.

Yakun Huang is currently working toward the Ph.D. degree at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. He has authored or co-authored over 10 technical papers in international journals and at conferences, including the IEEE Transactions On Mobile Computing, the IEEE Transactions on Service Computing, the IEEE Network, INFOCOM, ICDCS, MM. His current research interests include mobile computing, edge computing, distributed deep learning.

Xiuquan Qiao is currently a Full Professor with 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 co-authored over 60 technical papers in international journals and at conferences, including the IEEE Communications Magazine, Proceedings of IEEE, Computer Networks, IEEE Internet Computing, the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and the 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. Dr. Qiao 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.

Wenhai Lai is currently working toward the B.E. degree at the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include mobile computing, machine learning, edge computing.

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 at the University of California at Berkeley, Berkeley, CA, USA. He is currently a Professor of Computer Science with the Distributed Systems Group, Technische Universität Wien, Vienna, Austria. Dr. Dustdar 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 Computing (Springer). He is also an Associate Editor of the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the ACM Transactions on the Web, and the ACM Transactions on Internet Technology. He serves on the Editorial Board of IEEE INTERNET COMPUTING and the IEEE Computer Magazine.

Jianwei Zhang is currently the director of Institute of Capinfo Company Limited. He received his PhD from BeiHang University in 2015, and completed his postdoctoral research in 2018. He obtained EMBA of Cheung Kong Graduate School of Business in 2019. His research interests include Cloud Computing, big data, and AI. He is dedicating himself to the research of smart stadium of the national speed skating oval which is funded by National Key R&D Program of China under grant 2019YFF0301500.

Jiulin Li was born in HeBei, China. He received the master's degree from China University of Geosciences, China, in 1991 and 1992. He is currently the deputy chief engineer of Beijing Urban Construction Group Company Limited. His research interests include modern construction technology of Olympic venues and green construction and smart construction technology. He is currently participating the research of smart stadium of the national speed skating oval which is funded by National Key R&D Program of China under grant 2019YFF0301500 as a senior consultant.