EdgeBooster: Edge-Assisted Real-time Image Segmentation for the Mobile Web in WoT

Yakun Huang, Xiuquan Qiao, Pei Ren, Schahram Dustdar, *Fellow, IEEE,* and Junliang Chen

***Abstract*—Combining image segmentation with web technology lays a good foundation for lightweight, cross-platform and pervasive Web AI applications, and further improves the capability of Web of Things (WoT) applications. However, no matter whether we use a WebRTC media server for advanced processing that views camera inputs as a video stream, or transfer continuous camera frames to the remote cloud for processing, we are unable to obtain a satisfactory real-time experience due to high resource consumption and unacceptable latency. In this paper, we present EdgeBooster, a computational-efficient architecture that leverages a common edge server to minimize the communication costs, accelerates the camera frame segmentation, and guarantees an acceptable segmentation accuracy with the prior knowledge. EdgeBooster provides real-time segmentation by developing parallel technology that enables segmentation on slices of a camera frame and using pre-segmentation based on superpixels to accelerate the graph-based segmentation. It also introduces recent DNN-based segmentation results as the prior knowledge to improve the performance of the graph-based segmentation, especially in non-ideal scenes such as dark light and weak contrast. Finally, it creates a pure front-end segmentation that can provide continuous and stable services for mobile users in unstable networks such as a weak network or with an unstable edge server. The experimental results show that EdgeBooster is able to achieve a considerable accuracy for the mobile web, running at no less than 30 FPS in real scenes.**

***Index Terms*—Mobile web, edge computing, image segmentation, WoT applications**

I. Introduction

THE Web of Things (WoT) is a refinement of the Internet of Things (IoT) to enable interoperability and usability across heterogeneous IoT platforms and application domains, which has been a W3C international standard proposal recently [1], [2], [3]. Meanwhile, cross-platform web artificial intelligence (AI) improves the capability for WoT applications, expands application fields of WoT [4], [5], and is becoming a promising research topic [6], [7]. Thus, it is necessary and significant to develop AI-enabled WoT applications and provide "Write Once, Run Anywhere" portability for various IoT devices, rather than developing different applications for each IoT platform. This also indicates that IoT applications can be further enhanced by integrating smart things not only into the network, but into the Web architecture. However, limited by the weak computing capability of the Web and the constrained resources of IoT devices, it is difficult to complete the computationally intensive AI-enabled WoT applications. Fortunately, mobile edge computing (MEC), as a basic infrastructure in the 5G era, can provide low-latency and powerful services for computationally intensive WoT applications. As shown in Fig. 1, we describe a typical scenario of employing cross-platform web technology and MEC into IoT, which lays a good foundation for lightweight, cross-platform and pervasive Web AI applications. For example, traditional use cases such as augmented reality [8], [9], medical image analysis [10], unmanned vehicles [11], security monitoring [12], can be further enhanced by offloading heavy computations to the edge server. In this paper, we take real-time image segmentation, which provides indispensable sensing capability and is the most core technology involved in scene analysis [13], object detection [14], 3D reconstruction [15] etc., as a typical illustration. We explore an edge-assisted real-time system and expand the WoT application fields and influence for ubiquitous end devices, also including IoT devices.

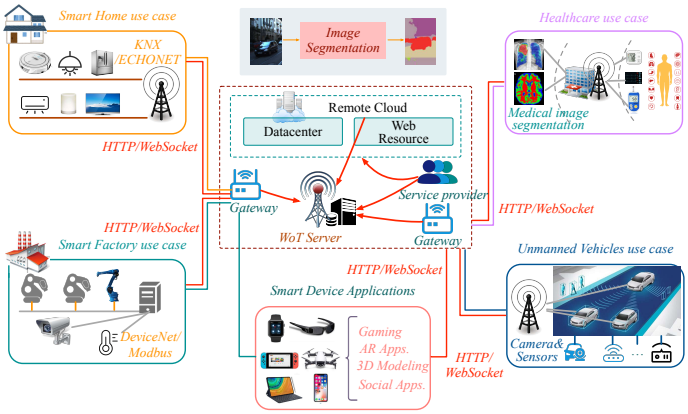


Fig. 1. Typical scenario of image segmentation & the advantage of employing cross-platform web technology in WoT applications (i.e., using HTTP/WebSocket protocols to unify and manage ubiquitous end devices).

To fluently execute real-time image segmentation on the mobile web, which means that average frame processing rate cannot be less than 30 FPS (frames per second). However, most of existing approaches to image segmentation on the mobile web take one of the following two approaches that give an unsatisfactory experience. The first approach is to obtain the video stream by leveraging the Web Real-Time Communication, (WebRTC) API [16], and transfer it to a remote media server for decoding, the video frame segmentation, fusing segmentation information, re-encoding and returning the processed video stream back to the mobile web

Y. Huang, X. Qiao, P. Ren and J. Chen are with 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.

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

for rendering. A typical scenario is employing Kurento [17], which is a WebRTC media server and a set of client APIs for making the development of advanced video applications for WWW simple, and providing advanced media processing capabilities such as computer vision, or augmented reality. However, this approach is limited by the network bandwidth and the computing capability of the media server. Intensive video stream computing increases the resource consumption of the media server dramatically in a short period of time, which results in the inability of the media server to handle large scale requests from multiple users and even makes the media server unavailable. In addition, although the mobile web achieves a fluent video stream that fuses the segmentation results from the media server, high network delay results in a drift phenomenon between the received video stream and the real scene during the rendering. The second approach directly obtains continuous camera frames from the mobile web and executes image segmentation either on the mobile web or transfers the camera frames to the remote cloud for image segmentation, named pure front-end method and cloud-assisted method respectively. Limited by weak computing capability of the mobile web, the pure front-end method can execute lightweight image segmentation such as the threshold-based algorithm [18] at no less than 30 FPS on the mobile web. However, since this method only considers the grayscale feature of images and ignores the spatial features, it is sensitive to noise and not robust enough to achieve an acceptable accuracy in the real scene. The cloud-assisted method can achieve acceptable accuracy by transferring the camera frames to the remote cloud for executing image segmentation algorithms such as traditional computer vision algorithms [19], [20], DNN-based algorithms [21] and semantic segmentation algorithms [22], and then returning the segmentation result to the mobile web for rendering. However, high transmission delay with a Round-Trip Time (RTT) about 120 ms indicates that it is far from meeting the real-time experience of no less than 30 FPS. In addition, intensive computing of the graph-based algorithm requires about 230 ms for image segmentation on a common server with a six-core Intel processor of 2.9 GHz and 16 GB RAM, let alone DNN-based algorithms that require dedicated devices such as powerful GPUs.

With the rapid development of 5G network to address these shortcomings, it is promising to consider the use of an edge-assisted approach that has the benefit of low communication costs compared to offloading computations to the remote cloud, and relieves the burdens of the core network [23], [24]. Compared to those app-based applications that can offload partial computations to the edge server to enable continuous vision analytics on mobile devices and dedicated devices, the mobile web application has to offload the entire computations to the edge server due to the limited computing capability of the mobile web [25], [26]. In addition, both app-based and web applications require high performance edge server with special hardware, such as GPUs for accelerating DNN algorithms. Although we can deploy expensive GPU servers in a special scene (e.g., an indoor stadium for holding activities), it is not realistic for network service operators, such as China Mobile and AT&T, to deploy high performance edge servers for widely use when MEC is used as the basic infrastructure in 5G. Thus, employing this kind of edge-assisted approach to offload real-time image segmentation tasks from the mobile web to common edge servers is still challenging for a number of reasons, including the following:

- **No edge-assisted framework is computational-efficient for image segmentation between the mobile web and common edge servers.** JavaScript is the main way for implementing mobile web applications, which provide weaker computing capacity than app-based applications that are directly executed on the mobile device. Thus, intensive segmentation computation needs to be entirely offloaded from the mobile web to the edge server. However, to the best of our knowledge, there is no computational-efficient framework and implementation for executing at least 30 FPS that is assisted with a common edge server.
- **A common and economic edge server fails to provide real-time segmentation of no less than 30 FPS for continuous camera frames with existing approaches.** Since traditional graph-based segmentation requires about 230 ms on a common edge server, it is far from meeting the real-time segmentation requirement of the mobile web. And a common edge server without GPUs cannot support the DNN-based segmentation, requiring about 400 ms for a camera frame. Additionally, the edge server executes segmentation task on a complete image that consumes much time waiting for the computing resource. This indicates that it is necessary to design an efficient segmentation algorithm and advanced technology to accelerate the segmentation and meet the real-time requirement of no less than 30 FPS on a common edge server.
- **Traditional image segmentation of the graph-based algorithm performs poorly in non-ideal scenes such as dark light and weak contrast.** Even if we can design efficient algorithms and advanced technologies for the graph-based segmentation, it generally performs worse than the DNN-based algorithm, especially in non-ideal environments [21], [27]. In addition, there is no existing and available method to dynamically adjust appropriate parameters for the graph-based algorithm in real time. Although we are unable to leverage DNN-based algorithms directly on a common edge server for real-time segmentation, it is a good choice to leverage recent DNN-based segmentation results as the prior knowledge to provide appropriate parameters and improve the performance of the graph-based segmentation on a common edge server.
- **How to provide continuous and stable services in unstable environments such as a weak network or an unavailable edge server.** Offloading computing tasks from the mobile web to the edge server completely relies on the high performance and reliability of the network and the edge server. An unstable edge server will introduce a short service failure when performing a service migration or switching between edge servers, resulting in an uncontinuous service and unsatisfactory experience. In addition, this approach is also affected by the network status. This indicates that existing edge-assisted offloading requires

highly reliable edge server and the network bandwidth. Thus, it is significant to enhance the robustness and QoS with a low bandwidth requirement approach for real-time segmentation.

To address these concerns, we present EdgeBooster, an edge-assisted architecture that leverages a common edge server to minimize the communication costs, accelerates the frame processing rate, and guarantees an acceptable segmentation with prior knowledge of DNN-based segmentation. Comparing with existing real-time image segmentation algorithms that mainly directly utilize the underlying hardware such as GPUs on end devices, and require a stable network environment, this paper is the first to implement computationally heavy image segmentation for resource-constrained and cross-platform mobile web applications. Also, it aims to provide better frame processing rate than traditional graph-based methods with a real-time experience of no less than 30 FPS. Toward this goal, we propose an acceleration technology by dividing the frame into slices and developing parallel technology to execute segmentation on these slices. To further accelerate the segmentation of frame slices, we propose the pre-segmentation based on superpixels to significantly reduce the complexity of the graph-based segmentation. Second, we propose the use of the most recent results of DNN-based segmentation to provide appropriate parameters for improving the efficiency of graph-based segmentation. More importantly, this DNN-based prior knowledge can be used to correct the results at the frame slices edge for parallel graph-based segmentation. Thus, we can dynamically set reasonable parameters and improve the performance for the parallel graph-based segmentation, especially in a non-ideal environment. Last, we create a lightweight pure front-end segmentation based on a marker-based watershed algorithm, which uses the recent frame segmentation cache as the marker information to set appropriate parameters for improving the segmentation performance. This pure front-end segmentation also contributes to providing continuous and stable services for mobile users in a weak network or an unstable edge server. The contributions of this work can be summarized as follows:

- Proposing a computational-efficient framework and implementing a real-time segmentation for continuous camera frames between the mobile web and a common edge server for the first time to meet the requirement of no less than 30 FPS.
- Developing a parallel technology that enables segmentation on slices of a camera frame and designing an efficient graph-based segmentation algorithm using superpixel pre-segmentation to further accelerate the processing.
- Introducing recent DNN-based segmentation results as the prior knowledge to improve the performance of graph-based segmentation in a non-ideal environment.
- Creating a lightweight pure front-end segmentation to provide continuous and stable services in unstable environments such as a weak network or an unstable edge server.

II. Background and Motivation

In this section, we briefly introduce the background of image segmentation on the mobile web, present our observations from preliminary measurements, and discuss edge-assisted computing offloading technology that motivates us.

A. Image Segmentation on the Mobile Web

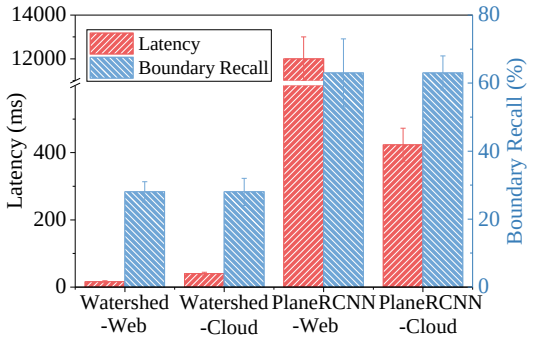


Fig. 2. Latency and boundary recall performance of watershed segmentation and DNN-based segmentation.

The cross-platform mobile web brings advantages to the user's perception of the environment and the real world. We conduct experiments for indoor scenes to measure the latency and boundary recall to illustrate the current problem of image segmentation on the mobile web with the same deployment in Section IV. Fig. 2 presents the latency and boundary recall performance of watershed segmentation and DNN-based segmentation, which are executed on the mobile web and remote cloud respectively. According to the results in the figure, we observe that the watershed algorithm [28] which is executed by OpenCV.js [29] on the mobile web has high efficiency, while the segmentation performance is poor, being seriously affected by the dynamic environment. Executing PlaneRCNN [21], which is a DNN-based algorithm, on the mobile web introduces high latency including model transmission and inference. Thus, it is natural to offload the computation to the cloud for efficient image segmentation due to the unacceptable latency. However, transmission latency and inference latency are still too high to be acceptable when faced with continuous frames.

B. Edge-assisted Computing Offloading

We present typical scenarios and comparisons between cloud computing and edge computing in Fig. 3. Our goal is to achieve real-time image segmentation for the mobile web while maintaining a satisfactory accuracy. Delay performance in the bottom left of Fig. 3 shows that it is better to offload the computation from the mobile web to the edge server due to low communication costs. In this paper, we mainly focus on providing a real-time image segmentation for the mobile web by means of edge computing. We believe that a carefully designed edge-assisted image segmentation will provide not only a high accuracy but also a better response time.

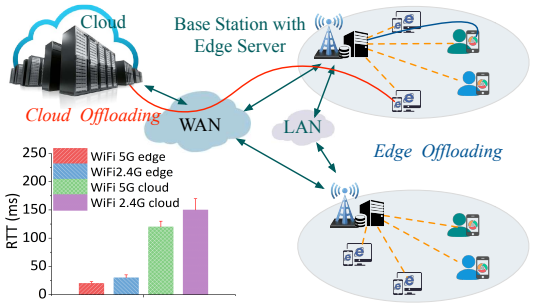


Fig. 3. Typical scenarios and comparisons between cloud computing and edge computing.

III. Design

EdgeBooster proposes a computational-efficient architecture that leverages a common edge server to minimize the communication costs, accelerate the segmentation, and maintain an acceptable accuracy. EdgeBooster consists of three major components as shown in Fig. 4.

(i) The booster is located at the edge server and uses a parallel technology for the graph-based algorithm which enables segmentation on slices of a frame, so that the camera frame streaming and segmentation can be effectively pipelined and computed in parallel.

(ii) The booster leverages the results of recent DNN-based segmentation results as the prior knowledge to guide real-time segmentation of a graph-based algorithm. This prior experience can not only improve the segmentation accuracy in non-ideal environments such as dark light and weak contrast, but also accelerate the efficiency of the graph-based segmentation by providing reasonable parameters.

(iii) The controller that executes at the mobile web runs a camera frame scheduling algorithm, deciding whether to request the edge server for boosting segmentation processing according to the network condition and the computing pressure of the edge server. A pure front-end segmentation based on marker-based watershed algorithm in JavaScript provides a real-time segmentation when confronted with a weak network or an unstable edge server.

To further improve the performance of segmentation for the mobile web, we perform pre-processing such as encoding, and image denoising on the camera frame obtained by the WebRTC API. Furthermore, the front-end Result Receiver module caches the recent segmentation record from the booster, thus providing a wealth of knowledge for corrections to achieve more accurate results from the pure front-end segmentation.

Design goals and scope. The primary goal of EdgeBooster is to minimize the processing latency of camera frames and provide real-time segmentation while maintaining acceptable accuracy under various complex environments between the mobile web and a common edge server. We also limit the scope of our design, noting the following non-goals:

1. Each image segmentation algorithm has a certain accuracy and performance on a specified device. This paper is devoted to providing an acceptable image segmentation on the mobile web in real time. Thus, improving the accuracy of segmentation to be better than the state-of-the-art segmentation is outside this work's scope.

2. EdgeBooster is mainly used for real-time computation using common edge servers deployed close to the mobile web users, so the deployment and resource allocation of mobile edge servers is beyond the scope of this paper.

3. In EdgeBooster deployment, we mainly consider Chrome, Safari and other mainstream browsers that support WebAssembly [30] and other common technologies. Due to compatibility issues, we do not consider the application of EdgeBooster in embedded mobile web browsers such as WeChat.

In the following, we describe EdgeBooster in detail, followed by a description of how the pure front-end controller makes decisions on processing a camera frame (Section A), parallel processing for accelerating (Section B), and a real-time graph-based segmentation algorithm that leverages the prior knowledge from DNN-based segmentation (Section C).

A. The Controller and Pure Front-end Segmentation

Generally, the network status and the state of the edge server which mobile web users are using are usually complex and unstable, resulting in unsatisfactory service with low QoS. We provide a controller and a pure front-end segmentation that execute on the mobile web to ensure a real-time image segmentation even in a weak network or with an unstable edge server.

1) The Controller: On the premise of meeting the minimum camera frame rate, the controller schedules a camera frame that is segmented by the pure front-end segmentation or the powerful backend segmentation, according to the current network status and the computation status of the edge server. We define $F = \{f_1, f_2, \ldots, f_n\}$ as the sequence of camera frames obtained via the WebRTC API. When the mobile web user requests the web server for services, the controller and the pure front-end segmentation that are implemented in JavaScript are loaded to the mobile web. The current network status $N = \{3G, 4G, WiFi\}$ and computing status of the backend server $p = max\{p_{cpu}, p_{IO}\}$ are also acquired in requests from the mobile web, where p_{cpu} and p_{IO} represent the current CPU consumption and the IO consumption of the edge server respectively.

Generally, the pure front-end should provide at least 30 FPS to ensure a fluent experience for mobile web users, thus as our controller threshold to make the decision for each camera frame. We describe the detailed scheduling process of the controller in Algorithm 1 and present four situations in Fig. 5 to further explain the function of the controller. Fig. 5 shows that when there are problems in the network conditions or services of the edge server, the edge server cannot be used to provide services to users. Especially in Fig. 5(c), although the edge server can provide available service, the network bandwidth is too low or the network failure occurs, the network conditions at this time are far from being able to support real-time frame transmission, which also causes a surge in latency.

2) The pure front-end segmentation: The computing capability of the mobile web browser is weaker than mobile devices or dedicated devices, because it mainly computes the task through JavaScript and is hard to use the underlying hardware such as GPU for acceleration. OpenCV.js [29] uses

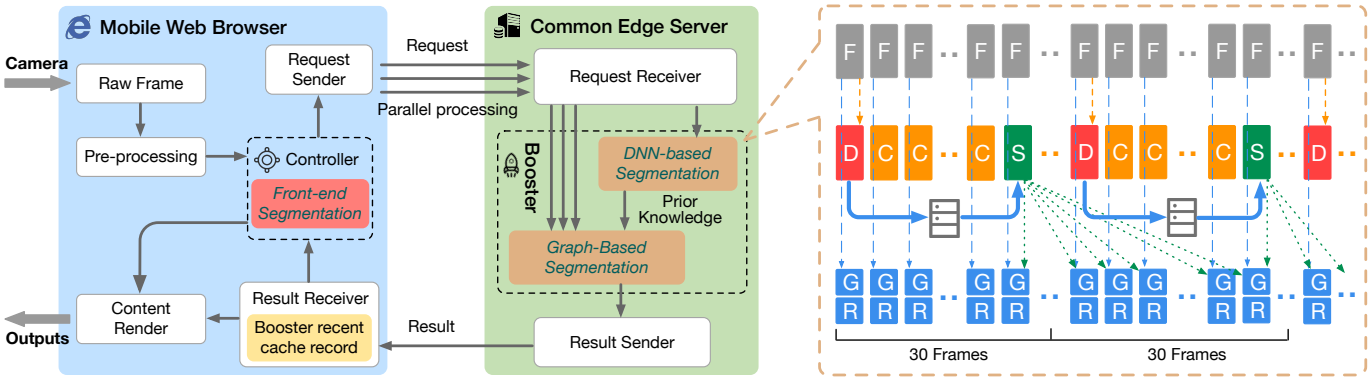


Fig. 4. Overall workflow of EdgeBooster. We also show the segmentation pipeline in the right of the figure. The sequence in gray scale represents the camera frames that are transferred to the edge server for segmentation. The sequence below in color denotes the processing of the DNN-based algorithm, where D is the processing frame, and S is the results, and C is the unprocessed frames. The bottom sequence denotes the processing of the graph-based algorithm, where G is the processing frame and R is a result frame.

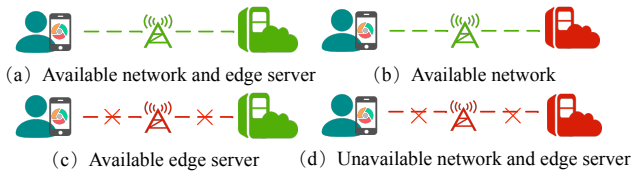


Fig. 5. Four situations of the controller. The green and red dotted lines indicate whether the network condition is available, respectively. Similarly, green and red edge servers indicate whether the current service is available, respectively. The controller sets the pure front-end segmentation by default and transfers camera frames to the edge server for segmentation in unstable environments.

Algorithm 1: The controller algorithm

Input: Network bandwidth B, parallel channels C, frame size T, minimum FPS threshold f_{thresh}, sever load pressure p_{cur}, maximum overload of the server IO p_{max_io}, maximum overload of the CPU p_{max_cpu}, processing and response latencies $t_{server}, t_{response}$.

Output: Optimal execution result.

$OptResult \Leftarrow$ *Front-end*; // Default setting
$p_{max} \Leftarrow max\{p_{max\_io}, p_{max\_cpu}\}$;
if $p_{cur} \leq p_{max}$ **then**
  $t_{send} \Leftarrow T/(C \cdot B)$;
  $t_e \Leftarrow t_{send} + t_{server} + t_{response}$;
  if $t_e \leq \frac{1}{f_{thresh}}$ **then**
    $OptResult \Leftarrow$ *Booster*;
return $OptResult$;

the WebAssembly [30] to optimize and compile the OpenCV library from the traditional C++ platform to run smoothly on the mobile web browser, which also provides a favorable basis for solving the pure front-end segmentation algorithm. Although traditional algorithms that are based on the region, the threshold or the edge have fast segmentation efficiency, they are not robust to ensure a satisfactory experience in a complex environment such as dark light and weak contrast. To achieve stable segmentation in a weak network or when the edge server is not available, we propose a prior marker-based watershed algorithm as the pure front-end segmentation for the mobile web browser.

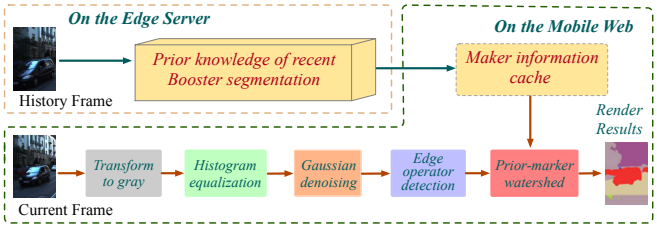


Fig. 6. The pure front-end segmentation.

We describe in detail the complete process of lightweight pure front-end segmentation, which mainly includes preprocessing, prior-marker generation, and real-time segmentation in Fig. 6. OpenCV.js provides tools and tutorials to compile our lightweight pure front-end algorithm from an efficient C++ script to JavaScript that executes on the mobile web browser in real time.

B. Parallel Technology for Segmentation

We leverage robust booster segmentation at the edge server and return the results to the mobile web. This process requires camera frames to be transmitted from the mobile web to the edge server frequently, such as at 30 frames per second. Our test results show that the implementation of the graph-based algorithm of the booster on a common edge server takes about 230 ms with pre-processing, encoding, transmission, decoding, and segmentation, and returns the results to the mobile web. However, the DNN-based algorithm, such as PlaneRCNN, takes about 400 ms to process a camera frame. Therefore, it is difficult to meet the latency requirement by serially calculating a camera frame.

To improve the processing speed of camera frames and meet the latency requirement, we consider the use of a parallel processing technology which enables graph-based segmentation on slices of a camera frame in Fig. 7. The principle of graph-based segmentation is based on the similarity between pixels,

which have no computational dependency when dividing the frame into slices and executing graph-based segmentation independently. This makes it possible to apply parallel processing to accelerate the entire segmentation. However, when we merge these slices, the lack of computation of pixels to slice edges may introduce the loss of the accuracy. We propose the use of the prior knowledge of DNN-based segmentation to correct these pixels at slice edges to maintain the accuracy. We do not adopt the parallel processing for the DNN-based segmentation for the following reasons: (1) The network structure of PlaneRCNN for segmentation is different from traditional CNNs, which is also difficult to be parallelized. (2) DNN-based segmentation mainly provides prior knowledge for graph-based segmentation instead of providing real-time processing.

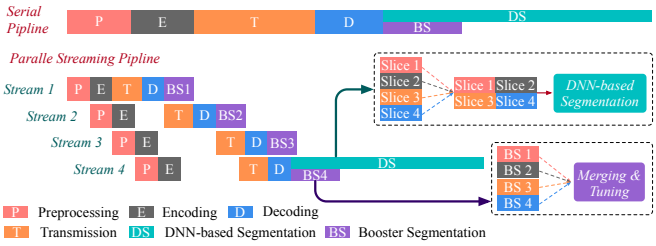


Fig. 7. Parallel processing technology for segmentation.

The booster segmentation deployed on the edge server mainly uses a graph-based algorithm that uses prior knowledge of the DNN-based segmentation. For the graph-based segmentation, we can easily accelerate the segmentation by a parallel processing in Fig. 7. We use multiple processes to process the task flow, such as the frame decoding thread, the graph-based segmentation thread, and the integration processing thread. These processes cooperate with the front-end parallel stream to form the completed parallel stream and accelerate the segmentation. For the DNN-based segmentation, it receives the task stream from the mobile web and combines the camera frames slices into a completed frame via the slice number as the input of the DNN-based segmentation. To better understand the details of the parallel processing, we first describe how to divide each frame into slices and number them. In each frame of the mobile web, we divide it evenly into N small pieces, where N is the number of parallel processing pipeline (e.g. $N = 4$ in Fig. 7). Therefore, each frame slice can be processed by an independent thread, which means the way of the frame dividing such as horizontal or vertical dividing has no effects on the independent processing. Then, once all of processing threads have been completed the segmentation of frame slice, the merging process mainly involves stitching the frames together to form a complete camera frame based on the frame number. However, the pixels at the slice edges of the camera frame results in a loss of accuracy due to the loss of computation with adjacent slices when compared with processing the whole camera frame. Although we can perform graph-based segmentation again for stitching edge, it obviously increases the delay of frame processing. To ensure the accuracy and avoid increasing the latency, we next describe the use of the prior knowledge of DNN-based segmentation to correct these pixels at slice edges.

C. Real-time Booster for Segmentation

A large number of existing DNN-based algorithms can achieve high precision, and real-time segmentation can be provided even in complex scenarios. However, these algorithms typically need to be deployed on dedicated devices for accelerating computations so that they can achieve accurate and real-time segmentation. We propose a real-time booster deployed on common servers that aims at faster segmentation and acceptable accuracy in a real environment. The booster combines the advantages of the graph-based algorithm such as high efficiency, easy deployment, and no requirement for special hardware computing resources, which also leverages the prior knowledge of the DNN-based algorithm to further improve the segmentation accuracy. We simply iterate through the pixels at the edge of the slices stitching, which will be corrected to the partitioned area that the pixels at the same location as the pixels of DNN-based segmentation belong to. The main reason is that the DNN-based segmentation of the previous frame considers the global pixel information, and the accuracy is better than graph-based segmentation.

Traditional graph-based algorithms treat each pixel in the camera frame as a vertex of the graph, and the relationship between pixels (e.g., grayscale and the distance) is regarded as the edge of the graph. The minimum spanning tree method in [19] is used to iteratively calculate the similarity between pixels and gradually merge them to form segmented regions. Because the algorithm iteratively calculates pixel points, it is hard to achieve real-time segmentation for a high-resolution camera frame. For example, it takes about 250 ms to calculate a frame of 480*680 on a common edge server. To further improve the efficiency and meet the real-time requirement, we also propose in Fig. 8 a booster algorithm based on superpixels for pre-segmentation consisting of superpixel pre-segmentation, graph-based segmentation and merging the results of the frame slices.

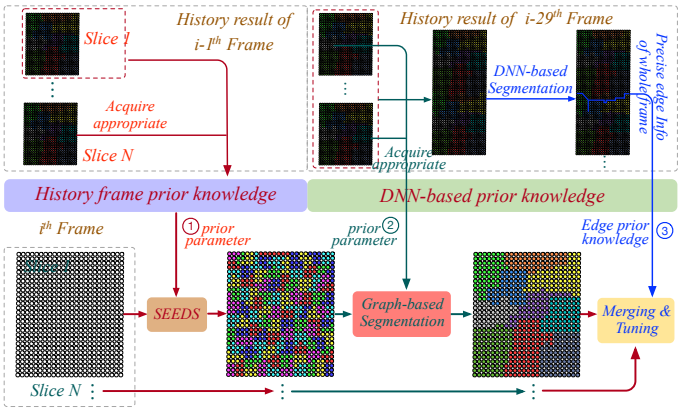


Fig. 8. Booster segmentation at the edge server.

First, we use the slice segmentation result of the latest historical frame as the prior knowledge to provide appropriate parameter for SEEDS [31], which is a fast superpixel segmentation algorithm. Then, the current superpixel edge is gradually moved to find the same uniform color feature of pixels inside the superpixel as possible so that the segmentation

can be executed quickly. Thus, we can perform the graph-based segmentation on these superpixels instead of the initial pixels so that the complexity of the segmentation algorithm is significantly reduced. SEEDS chooses the energy objective function $E(s)$ to measure the value of the superpixel:

$$E(s) = H(s) + \gamma G(s), \tag{1}$$

where $H(s)$ denotes the color of the superpixels and $G(s)$ is the shape of the superpixel boundaries. γ weighs the influence of each term, and it is determined by the prior knowledge of history frame segmentation. The detailed process of SEEDS can be found in [31]. Last, to acquire further robust segmentation, we leverage the DNN-based segmentation as the high-level prior knowledge to guide the graph-based segmentation once the camera frame has been segmented into superpixels (i.e., acquire appropriate parameter from DNN-based segmentation on (i–29^{th}) frame).

Let $G = (V, E)$ be an undirected graph with vertices $v_i \in V$ representing superpixels, and edges $(v_i, v_j) \in E$ are the pairs of neighboring vertices. Each edge has a weight $w_{(v_i,v_j)}$ that measures the dissimilarity between the two superpixels (e.g., the difference in intensity, color or other attributes). Based on the definition of the above superpixel map G and the simplified minimum spanning tree (MST) of G, we define the internal difference of the segmentation region (SR) as $Int(C)$, which means the weight of the largest edge in the SR as follows.

$$Int(C) = \max_{e \in MST(SR,E)} w(e). \tag{2}$$

For example, the difference between the maximum luminance mean of different superpixels in the SR can be used as the edge of the most dissimilar superpixel in the MST. We also define the difference between two segmentation regions $SR_1, SR_2 \in V$ as $Dif(SR_1, SR_2)$ to represent the minimum edge connecting the two superpixel regions as follows.

$$Dif(SR_1, SR_2) = \min_{v_i \in SR_1, v_j \in SR_2, (v_i,v_j) \in E} w((v_i, v_j)), \tag{3}$$

$$D(SR_1,SR_2) = \begin{cases} true & if\ Dif(SR_1, SR_2) > MInt(SR_1,SR_2) \\ false & otherwise \end{cases}. \tag{4}$$

Where $Dif(SR_1, SR_2) = \infty$ denotes that there is no edge between SR_1 and SR_2. To determine if there is a boundary between superpixel regions, we use a threshold function τ to control the difference between two superpixel regions. $MInt$ denotes the minimum internal difference. The threshold function is defined as $\tau(SR) = k/|SR|$, where $|SR|$ denotes the size of SR, and the parameter k is mainly based on the size of the frame and the prior knowledge of the DNN-based segmentation, which is a dynamic adjustment without any manual adjustment.

$$MInt(SR_1, SR_2) = \min(Int(SR_1) + \tau(SR_1), \\ Int(SR_2) + \tau(SR_2)). \tag{5}$$

The workflow of the complete graph-based segmentation method is as follows:

Step 1. Calculate the dissimilarity between each superpixel region and the adjacent region in order from left to right, and from top to bottom.

Step 2. Sort the edges connected by different superpixel regions in the order of dissimilarity to obtain $E_{order} = \{e_1, e_2, \ldots, e_N\}$ and select e_1 as the initial edge.

Step 3. Merge the currently selected edge $e_n \in E_{order}$ with e_1. If the connected vertex (v_i, v_j) satisfies that v_i and v_j do not belong to the same superpixel region, and that the dissimilarity is not greater than the internal dissimilarity, then turns to Step 4, otherwise turns to Step 5.

Step 4. Update the threshold and the number of superpixel regions.

Step 5. If $n \leq N$, the loop selects the next edge and goes to Step 3.

Algorithm 2: Booster segmentation algorithm

Input: Current frame slices $FS = \{s_1, s_2, ..., s_N\}$, recent segmentation result $MatrixMask_{c-1}$, recent prior knowledge of DNN-based segmentation pk.

Output: $MatirxMask_c$ of the segmentation result.

```
for i from 1 to N do
    /* executing seeds in parallel   */
    superpixels ⇐ parallelSEEDS(s_i, MatrixMask_{c-1});
    /* executing Graph-basd segmentation in parallel   */
    segSlic ⇐ parallelGraphSeg(superpixels, pk);
    segSlicArray ⇐ segSlicArray.add(segSlic);
MatirxMask_c ⇐ frameSlicMerge(segSlic, pk);
return MatirxMask_c;
```

We present the detailed booster segmentation in Algorithm 2, mainly including superpixels segmentation, graph-based segmentation and merging the segmentation results according to the DNN-based prior knowledge.

IV. Implementation

In this section, we describe our implementation of EdgeBooster, which is entirely based on the commom hardware and customizd software.

A. Hardware Setup

This section describes the hardware platform we used to deploy the EdgeBooster on the edge server. We use a regular IBM server with a six-core Intel processor of 2.9 GHz and 16 GB RAM running Ubuntu 16.04 LTS that is deployed near the base station. For the mobile device, we use a HUAWEI Mate 9 smart phone that runs Android 8.0 executing Firefox browser, which is equipped with an eight-core CPU (four cores at 2.4 GHz and four cores at 1.8 GHz) and 4 GB RAM. We also test EdgeBooster on other smart phones such as the Samsung Galaxy S5 and 360 N7 Pro, and web browsers, such as Chrome and Opera. Our system can work on any mobile device equipped with a web browser. To acquire the prior knowledge of the robust DNN-based segmentation, we train

the DNN network on the cloud server, which is equipped with 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 on each card.

B. Software Implementation

This section describes the software implementation of the EdgeBooster across the mobile web to the edge server. The entire software codebase consists of about 3000 lines of Python code that implements Booster processing for the edge server, along with about 1000 lines of JavaScript code for the mobile web.

Mobile web side. Our implementation follows the design in Fig. 4. We obtain camera frames by using *getUserMedia()* of WebRTC API running at 60 FPS, and pre-processing the frame using *resize()* and *GaussianBlur()* of OpenCV.js API. We realize a JavaScript controller on the mobile web with the parameters of the network state and the edge server state using Network Information API and a packaged API that monitors the available computing resources of the edge server. To implement the parallel processing, we realize a *frameSlice()* function to slice the camera frame and then we leverage WebSocket API to create stable communication pipelines between the mobile web browser and the edge server. The Request Receiver module of the edge server can receive the slices of camera frames and prepare the slices for the parallel processing. For the pure front-end segmentation thread, we realize our marker-based watershed algorithm API in C++ and compile a lightweight OpenCV.js version that has the same API of *cv2.watershed()*. For the Content Render module, we color different regions according to the segmentation edge data provided by the pure front-end segmentation or the Booster segmentation, which is also implemented in JavaScript.

Edge server side. The edge server side implementation contains three main modules: Request Receiver, Booster and Result Sender. In the Request Receiver thread, the system establishes the Socket connection with the mobile web, receives the frame slices, decodes the fame slices and then merges the slices into a complete frame. We use Flask as the web server framework that easily connects with the mobile web using Socket API. Booster is the core module on the edge server and provides a robust segmentation in real time. It contains a parallel graph-based segmentation and a DNN-based segmentation that provides prior knowledge for the booster. The training and inference phase of the DNN-based segmentation is implemented in PyTorch 0.4.0. For easy deployment and use, we realize the graph-based segmentation in Python with an opencv-python library. Note that our graph-based segmentation includes a superpixel segmentation named SEEDS and is guided by the prior knowledge of the segmentation of the history camera frame. The Result Sender module mainly returns segmentation results to the mobile web in a JSON formation which is implemented in Python.

V. Evaluation

In this section, we describe our experiments to serve two purposes. First, we want to evaluate the segmentation accuracy of EdgeBooster in complex environments compared with benchmark methods. To the best of our knowledge, we are the first to explore real-time image segmentation for the mobile web with common edge severs. We benchmark the EdgeBooster against the pure front-end segmentation that executes on the mobile web and PlaneRCNN, a state-of-the-art DNN-based segmentation algorithm, to indicate its advantages. Second, we seek to understand the behavior of our EdgeBooster as it affects the latency and resource consumption. Note that we did not make an evaluation on using edge compared to running the same application on the cloud or a remote media server. This is because we have compared the latency performance between the edge and the cloud in Section II, and the results indicate that is impossible to use the cloud to achieve the requirement of no less than 30 FPS for mobile web applications. Our evaluations show that EdgeBooster achieves a stable and acceptable performance in various environments.

A. Experiment Setup

We follow the setup and implementation described in Section 4 to conduct experiments. We use two types of datasets including the outdoor scene and the indoor scene for evaluating EdgeBooster. KITTI [32] is a dataset for computer vision algorithm evaluation in autopilot scenarios. In addition, we also evaluate the EdgeBooster in real indoor scenes with various conditions such as dark light, weak contrast, strong texture etc.

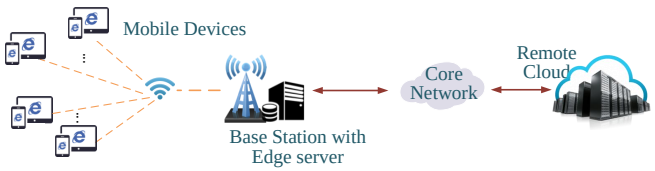


Fig. 9. Topology of the network deployment.

In Fig. 9, we present the core network topology with an average downlink bandwidth of 300 Mbps and an average uplink bandwidth of 80 Mbps, which is in a real-world 5G network at Beijing University of Posts and Telecommunications. A common server with a six-core Inter processor of 2.9 GHz and 16 GB RAM is deployed near base station. The remote cloud is responsible for providing training DNN models and the management of the edge servers. To acquire stable network conditions such as for 3G, 4G and WiFi, we use Wonder Shaper [33], which is a script that allows the user to limit the bandwidth of network adapters, to control the network conditions on the edge server. Besides, since EdgeBooster's performance is mainly affected by the downlink bandwidth, we set average downlink bandwidth of 3G, 4G and WiFi as 1.5Mbps, 12Mbps and 25Mbps respectively, aiming to simulate the network in various scenarios. We set the relevant parameters involved in EdgeBooster as following $f_{thresh} = 30$, $C = 4$ denotes the number of parallel channels, $p_{max\_io} = 60\%$ and $p_{max\_cpu} = 80\%$. We also extract raw camera frames at 480*320 resolution from the camera of the mobile phone for repeatable experiments. Then we can repeat the same camera frames for multiple evaluation to acquire an average value. All of the experiments strictly follow the same

workflow without any pre-processing and profiling on each frame to ensure that they run in real-time as shown in Fig. 4.

B. Qualitative evaluations

In this section, we perform qualitative evaluations on the pure front-end segmentation, our booster segmentation and the state-of-the-art segmentation algorithm PlaneRCNN in outdoor and indoor scenes. We sample four camera frames from KITTI to evaluate the performance of the outdoor scene shown in Fig. 10. For the evaluation of the indoor scene, we also use the same sampling rule to collect four camera frames via WebRTC from the real scene, as shown in Fig. 11.

Fig. 10. Segmentation of the outdoor scene.

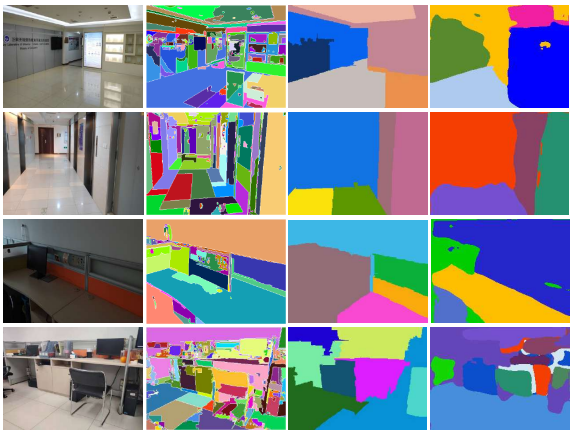
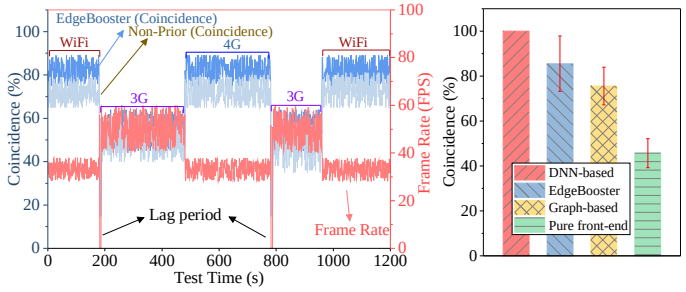


Fig. 11. Segmentation of the indoor scene.

We can make three key observations based on these results in Fig. 10. First, Booster's segmentation is better than that of the pure front-end algorithm, especially in the outdoor scene with lots of small surfaces such as leaves of the tree. Although booster is less effective than PlaneRCNN, the entire processing latency cannot meet the real-time requirements of the mobile web application. Second, we also analyze the segmentation performance of the booster under various conditions in indoor scenes, shown in Fig. 11. Compared to the outdoor scene, the light and contrast of the indoor scene is generally weaker. The results show that booster enables a considerable segmentation while the pure front-end algorithm cannot provide an acceptable segmentation when dealing with lots of small surfaces. In addition, PlaneRCNN has the best performance so that we use its recent segmentation results as the prior knowledge to improve the performance of booster. We also use this prior knowledge to correct the results of the graph-based segmentation of booster. In summary, our booster segmentation not only provides better performance than the pure front-end segmentation, but also realizes real-time segmentation on the mobile web.

C. Image Segmentation Accuracy

To further evaluate the accuracy performance of the EdgeBooster's segmentation in a real scene, we set the segmentation results of PlaneRCNN of a DNN-based method as the benchmark, the segmentation accuracy can be measured by the degree of coincidence of the segmentation results among PlaneRCNN and EdgeBooster. We conduct experiments on the performance of the accuracy of various methods, also including graph-based segmentation and pure front-end segmentation. As shown in Fig. 12(a), we continuously test EdgeBooster for 20 minutes in the indoor scene, and we measure the accuracy of the current time by taking the average camera frames segmentation accuracy in one second. We also illustrate the effect of prior knowledge to improve segmentation by comparing the EdgeBooster with the method that does not use the prior knowledge (i.e., Non-Prior in Fig. 12(a)). Note that we randomly change the network status every 3–5 minutes to simulate the unstable network status.



(a) Accuracy in vairous networks (b) Comaprisons with various methods

Fig. 12. Performance of image segmentation in accuracy.

As shown in Fig. 12(a), the left Y-axis represents EdgeBooster's segmentation accuracy, as expressed by the percentage of the degree of coincidence between EdgeBooster and PlaneRCNN. The right Y-axis represents the real-time frame rate of EdgeBooster's segmentation. We observe that EdgeBooster significantly achieves real-time performance when the average frame rate is not less than 30 FPS, and it shows an accuracy similar to that of PlaneRCNN. For the poor network condition, the pure front-end algorithm can still provide an average segmentation accuracy for the mobile web user with no less than 33%. Although it is difficult to achieve a similar segmentation accuracy to that of PlaneRCNN and Booster, the segmentation result is still acceptable, as shown in subsection

A of Section V. Note that the frame rate of such the pure front-end segmentation has reached 50 FPS or higher. This indicates that EdgeBooster for the first time provides mobile web users with a real-time and stable image segmentation, even in an unstable condition. Note that pure front-end segmentation is to provide smooth and continuous services for a short period of time under unstable conditions. Therefore, although the accuracy is not considerable, it is significant to provide continuous segmentation rather than immediately stopping segmentation when it encounters a weak network.

In addition, we also compare the EdgeBooster, Graph-based and pure front-end method with benchmark method of DNN-based in the segmentation accuracy by randomly collecting 1000 camera frames of indoor and outdoor scenarios in Fig. 12(b). We can see that when neglecting the network condition and the complexity of segmentation algorithm, EdgeBooster's performance is closer to the DNN-based method, and it also performs better than the Graph-based and pure front-end methods. Although the average segmentation accuracy of the EdgeBooster on 1000 random frames is not better than the DNN-based method, we can see that EdgeBooster's segmentation is close to the DNN-based method and has been effective from the previous qualitative evaluations. Even the segmentation of EdgeBooster on some frames is better than the DNN-based method, thereby being able to support the applications in reality. Besides, since our EdgeBooster's motivation is to provide the real-time segmentation for the resource-constrained and cross-platform mobile web applications, it is more focused on how to efficiently and stably provide continuous services for the mobile web on the basis of ensuring the availability of segmentation accuracy.

D. Latency Performance

1) Latency performance in various networks: We describe the latency performance from capturing the camera frame to completing the entire process of segmentation in Fig. 13. We can make two key observations based on these results. First, the results show that EdgeBooster provides a real-time image segmentation for the mobile web with an average latency of 35 ms. Concretely, under the network condition of 3G in Fig. 13(b), EdgeBooster achieves an average processing latency of 16 ms using the pure front-end segmentation, thereby achieving the same frame processing speed as the mobile web frame sampling frequency, and the points are more densely displayed. However, for 4G and WiFi in Fig. 13 (c) and Fig. 13(d), EdgeBooster leverages the edge-assisted Booster to process the camera frame flow with an average latency including transmission latency of 32 ms. Thus, it can handle 33 frames per second which is less than the sampling frequency of the mobile web. We recommend that it is better to set the sampling frequency of the mobile web as about 30 frames, which guarantees that EdgeBooster can provide a stable real-time segmentation algorithm of not less than 30 FPS. This also indicates that our Booster algorithm, using edge-assisted and parallel stream processing technology, can effectively improve the performance of offloading computation from the mobile web to a common edge server. Secondly, when the network changes, EdgeBooster's controller can adjust the computation offloading mode according to the state of the network. Since the frame processing rate of the pure front-end segmentation in 3G is higher than that in 4G and WiFi, the frame processing hysteresis is not felt back when the network is changed from 3G to 4G/WiFi. Hence, we only show the result that changes the network from WiFi/4G to 3G in Fig. 13(e) and Fig. 13(f). We find that the controller fails to sense the network and complete the computation offloading when the network changes to the low frame processing rate from a high frame processing rate. This is because the controller has a lag period to change the segmentation from the robust booster to the pure front-end segmentation.

Since the network bandwidth varies in reality, EdgeBooster's controller, which perceives dynamic changes of the network in real time and dynamically adjusts the currently offloading strategy based on the algorithm 1, requires providing efficient perception and decision-making computing to reduce the lag period as much as possible. We use the same testing platform and network settings in subsection A and develop a controlled script to automatically change the network bandwidth for Wonder Shaper to simulate a dynamic network environment. When we use the controller script to change the network bandwidth, it does not take effect immediately. Thus, according to the testing results, we set the minimum network change frequency, which means how often the network bandwidth changes effectively, as 50 seconds ranging from 1.5Mbps to 25Mbps. We conduct experiments to explore the latency, accuracy and offloading rate of EdgeBooster under dynamic network bandwidths in Fig. 14. Specifically, Fig. 14(a) and Fig. 14(b) show the performance of EdgeBooster in different change frequencies when respectively increasing or decreasing the network bandwidth between

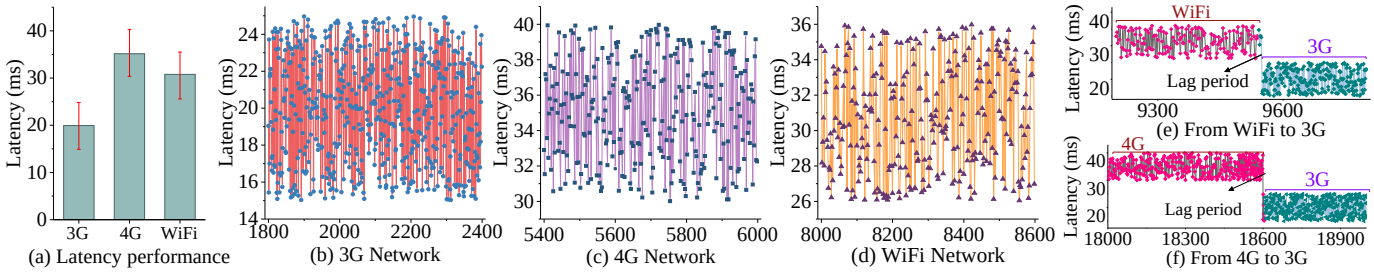
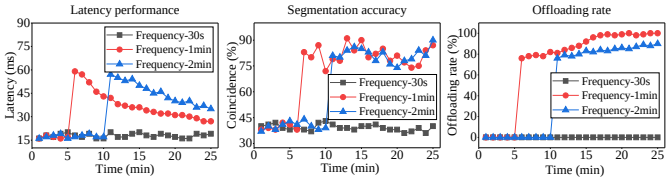
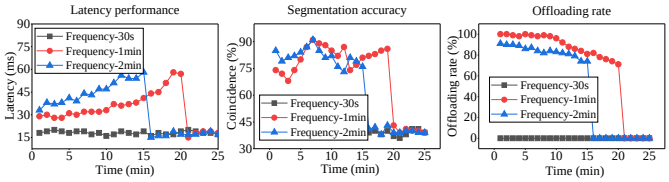


Fig. 13. Latency performance. The X-axis is the number of the frame, and the Y-axis is the entire latency required to process the current frame. We sample the camera frames of 5 minutes in chronological order, where sampling frequency of the mobile web is 60 frames per second. We also randomly change the network condition to simulate an unstable environment.

1.5Mbps and 25Mbps with a variation range of 1Mbps. We do not explore the performance under randomly increasing or decreasing the bandwidth, because other situations can be obtained by combining situations in Fig. 14(a) and Fig. 14(b).



(a) Increase the network bandwidth from 1.5Mbps to 25Mbps



(b) Decrease the network bandwidth from 25Mbps to 1.5Mbps

Fig. 14. Behavior of EdgeBooster's controller with different change frequencies of network bandwidth.

We observe that: (1) When the network change frequency is setting as 30 seconds, which is less than the minimum network change frequency, we find that EdgeBooster's controller always perceives the initial bandwidth and does not change the offloading strategy. This is because although the network is constantly changing, Wonder Shaper has not yet taken effect and has changed again, controlled script of the network bandwidth has fallen into a loop, resulting in no changes of EdgeBooster's controller. (2) Once the network change frequency exceeds the minimum change frequency, we can see that EdgeBooster's controller can perceive network changes as the bandwidth increases and make offloading decisions in time. Especially when the bandwidth increases to 5.5Mbps, the controller starts to offload calculations to the edge server to obtain more accurate service. However, when we increase the bandwidth to about 20Mbps, EdgeBooster performs a small climbing and in a stable state. A similar conclusion can also be obtained in Fig. 14(b) when we gradually reduce the network bandwidth from 25Mbps to 1.5Mbps. Although we do not further verify more complex scenarios, such as simultaneously increasing or decreasing the network bandwidth, the change boundary in these scenarios is similar to a certain stage in Fig. 14(a) or Fig. 14(b). Therefore, experimental results in Fig. 14 illustrate that EdgeBooster's controller can provide stable services through continuous testing under different network changing frequencies.

2) Latency comparisons: EdgeBooster provides a pure front-end segmentation locally for a weak network condition or with an unavailable edge server without offloading computations to the edge server. Thus, we present the latency comparisons between parallel Booster and non-parallel Booster under 4G and WiFi in Fig. 15(a) and Fig. 15(b). We observe that our parallel Booster can improve the frame processing by 42% and 36% in 4G and WiFi respectively compared with non-parallel frame processing. This acceleration of the parallel Booster can effectively improve the frame processing for the mobile web on a common edge server, which also makes it possible to meet a real-time requirement on the mobile web. In summary, performing a parallel technology on the edge server is significant in making it possible to process continuous vision tasks for the mobile web. It is a good choice to design parallel technology for accelerating image segmentation on a common edge server based on the above comparisons.

Then, we conduct experiments on the parallel Booster in different network bandwidths range from 5Mbps to 25Mbps in Fig. 15(c). We use the same experimental settings and methods as described in the above experiments. When the bandwidth is lower than 3Mbps, which is in the range of 3G network or in a weak network condition, EdgeBooster's controller will automatically choose the pure front-end segmentation rather than using the parallel Booster for real-time services. Hence, we are not necessary to explore the latency performance of parallel Booster in a weak network in Fig. 15(c). The results show that the average frame processing latency of parallel Booster has gradually decreased with the increase of network bandwidth, performing a significant improvement. This illustrates that a stable and high bandwidth can effectively improve the latency performance when using parallel technology of Booster. Moreover, we can see that when the bandwidth increases to 20Mbps, the parallel Booster has a slight increase as the bandwidth continues to increase. This is because the network bandwidth is no longer the bottleneck, and parallel technology of Booster will provide more improvements as the network bandwidth increases. Once the network bandwidth increases to 20 Mbps, the edge server receives enough frames to give full play to the parallel processing of Booster, improving throughput and reducing average processing latency.

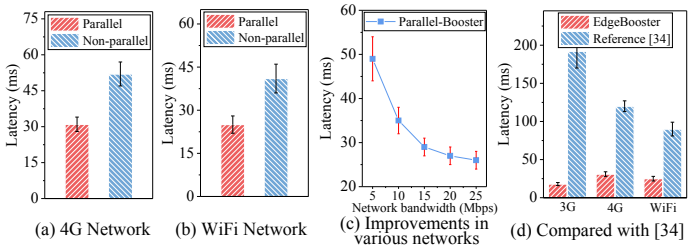


Fig. 15. Latency comparisons between parallel Booster and non-parallel Booster in (a) and (b), the latency improvement of parallel Booster with various networks in (c), and EdgeBooster's performance compared with reference [34] in (d).

To further illustrate the novelty and advantage compared with edge-asssited framework for object detection in reference [34], we conduct the experiments in various network conditions. We deploy PlaneRCNN to the same edge server and perform the same segmentation task according to the parallel inference method in [34]. Note that PlaneRCNN has different network structure with typical CNNs which used in [34], thus we only parallel the part of PlaneRCNN that can be parallelized. Y-axis of Fig. 15(d) represents the average processing latency to complete 100 camera frames. We can see that EdgeBooster has lower and better latency performance than that of reference [34], which means that

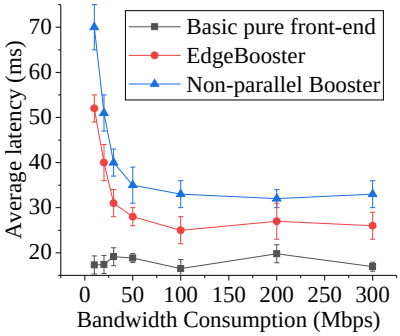


Fig. 16. Bandwidth consumption.

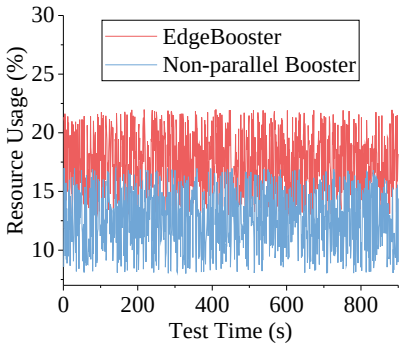


Fig. 17. Resource consumption.

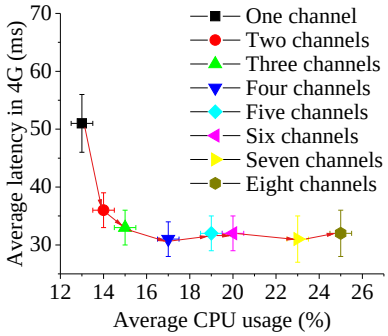


Fig. 18. Analysis of parallel channels.

on a common edge server without GPU, the DNN parallel inference technology in [34] cannot provide real-time frame segmentation. In particular, we can only acquire an average frame processing rate of about 11 FPS in WiFi by the method in [34]. For EdgeBooster, it achieves a better performance by applying the parallel technology to the graph-based segmentation, and guarantees the accuracy of graph-based segmentation by leveraging the DNN-based prior knowledge which has been illustrated in the above analysis. Besides, we observe that the method of [34] is significantly influenced by the network bandwidth, for instance, it performs an unacceptable frame processing rate in 3G. EdgeBooster exhibits better latency in 3G than that in 4G and WiFi. This is mainly because we provide a pure front-end segmentation on the mobile web to improve the continuity and stability of the system in unstable networks, which better illustrates the novelty and superiority of EdgeBooster compared with reference [34].

E. Bandwidth Consumption

We conduct experiments to measure the bandwidth consumption of three approaches (the pure front-end, parallel Booster, non-parallel booster) for image segmentation. We analyze the average segmentation latency and the bandwidth consumption of three approaches. To show how the average latency changes with the bandwidth consumption, Fig. 16 compares three approaches. For the same bandwidth consumption, our parallel Booster approach can achieve a lower segmentation latency than the non-parallel approach. Though the pure front-end approach can achieve the lowest latency without any bandwidth consumption, its accuracy is worse than other methods. Compared with the non-parallel method, our system reduces the bandwidth consumption by 53% while keeping the accuracy unchanged.

F. Resource Consumption

Since the pure front-end segmentation is a lightweight algorithm that requires a small amount of computation, our system only consumes few of the computation resources of the mobile device when the network status is not good or the server load is too high. Thus, we mainly consider the resource consumption of EdgeBooster when it offloads the entire computations from the mobile web to the edge server in good conditions. To monitor the continuous resource consumption such as the CPU usage, we use the mpstat [35], which is command line software used in Linux to collect CPU statistics (e.g., usage, user time, and idle time). Fig. 17 shows the raw resource usage traces for 15 minutes. The results show that our system requires 17% of the CPU resource, which is higher than the non-parallel processing value of about 13% with four parallel channels. Thus, it is acceptable for EdgeBooster to consume few resources for parallel processing to acquire a real-time continuous frame process for the mobile web. We also explore the impact of the number of parallel channels (the number of slices of the camera frame) on the resource consumption of the edge server in Fig. 18. The results show that although we increase the parallel channels, there is no linear improvement for the image segmentation of the mobile web due to the system overhead. Our experimental results show that three to five parallel channels can satisfy the real-time image segmentation for the mobile web.

VI. Related Works

Image segmentation. Traditional image segmentation approaches can be classified into those based on the threshold [18], the edge detection [36], the region [37], etc. They all use the low-level semantic information including the color, the texture and the shape of the image pixel, which are not good enough when encountering complex scenes. In addition, graph-based methods view the image segmentation as a vertex partition problem [19]. The spectral clustering methods construct a Laplacian matrix of the original image to solve the pre-background separation of the image[38], [39]. Although these approaches achieve a better segmentation than traditional methods, they have a high cost in time complexity which cannot be real-time for the mobile web.

Deep learning. In recent years, with the great success of deep learning technology in image classification, its ability to extract high-level semantic information largely solves the problem of missing semantic information in traditional image segmentation methods. FCN (Fully Convolutional Network) [40] is the first work that leverages deep learning into image segmentation and designs an end-to-end fullly convolutional network for pixel-by-pixel classification. DeepLab[41] adds fully connected CRFs at the end of the FCN frame to make segmentation more accurate. Besides, U-Net [42], SegNet [43], SSD [44], Faster R-CNN [45], [46], and Mask R-CNN [47] provide the image segmentation with abundant semantic information and a faster segmentation speed. In this paper, we use PlaneRCNN as our edge-assisted DNN-based

segmentation method, which provides prior knowledge for our Booster segmentation.

Mobile computing offloading. Offloading computation from the mobile web to the cloud or the edge server is a considerable way to enable continuous vision tasks [34], [25], [26], [48], [49], [50]. [34] designs a system that employs low-latency offloading techniques and uses a fast object tracking method to maintain detection accuracy for an AR/MR system running at 60 FPS. DeepDecision [25] provides a dynamic offloading based on the network conditions and decides whether to offload the task to the edge. Lavea [26] designs an intelligent framework that provides low-latency video stream analytics leveraging the edge computing platform. Most of these works aim at the mobile device that is more powerful than the mobile web, which focus more on improving the latency and accuracy performance between the mobile web and common edge server.

VII. Discussion

With the maturity of the W3C's WoT standard, the Web will become an important cross-platform application providing platform for the Internet of Things. To this end, we discuss the impacts of EdgeBooster on the mobile web, the generalizability, and some limitations. First, to the best of our knowledge, EdgeBooster is the first framework that leverages the edge server to provide real-time image segmentation with no less than 30 FPS on the mobile web in complex scenes. This has the advantage of low communication costs compared to offloading computations to the remote cloud and providing parallel technology for Booster to accelerate segmentation. Second, in this work, we mainly implement real-time image segmentation in leveraging our edge-assisted framework. Since the core contribution of the edge-assisted framework is to provide the mobile web with a real-time processing capacity for dealing with continuous camera frames, this real-time framework can be applied into other continuous vision tasks, such as tracking or object detection for the mobile web. Third, EdgeBooster mainly performs traditional image segmentation based on the pixel information of a single camera frame, without considering the semantic information and other useful sensor data such as IMU data. Thus, our segmentation results lack an understanding of the semantic information, especially for the outdoor scenario, which needs to be considered. In future research, we plan to find ways to consider the use of the semantic information of the camera frame and IMU data to achieve a real-time semantic segmentation based on our edge-assisted framework.

VIII. Conclusion

In this work, we proposed EdgeBooster, a practical framework that leverages common edge servers to minimize the communication costs, accelerates the camera frame segmentation, and guarantees an acceptable segmentation accuracy. For the purpose of obtaining fluently real-time segmentation on the mobile web, EdgeBooster provides a real-time booster for segmentation developing a parallel technology that enables booster segmentation on slices of a camera frame and using a superpixel pre-segmentation to further accelerate the processing. It also introduces robust DNN-based segmentation results as the prior knowledge to improve the performance of the graph-based algorithm, especially in non-ideal scenes such as dark light and weak contrast. It also creates a pure front-end segmentation algorithm to provide continuous and stable services for mobile users in unstable environments. Our evaluation indicates that EdgeBooster is able to achieve acceptable segmentation accuracy for the mobile web, running at no less than 30 FPS in various scenes.

Acknowledgment

This research was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61671081, in part by the National Key R&D Program of China under Grant 2018YFE0205503, 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.

References

[1] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *2010 Internet of Things (IOT)*. IEEE, 2010, pp. 1–8.

[2] "Web of things (wot) architecture," 2019, https://www.w3.org/TR/wot-architecture/.

[3] F. Antoniazzi and F. Viola, "Building the semantic web of things through a dynamic ontology," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 560–10 579, 2019.

[4] "Web of things (wot)," 2019, https://www.w3.org/WoT/.

[5] M. Noura, A. Gyrard, S. Heil, and M. Gaedke, "Automatic knowledge extraction to build semantic web of things applications," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8447–8454, 2019.

[6] 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 *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1497–1506.

[7] X. Qiao, P. Ren, S. Dustdar, and J. Chen, "A new era for web ar with mobile edge computing," *IEEE Internet Computing*, vol. 22, no. 4, pp. 46–55, 2018.

[8] 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," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 651–666, 2019.

[9] X. Qiao, P. Ren, G. Nan, L. Liu, S. Dustdar, and J. Chen, "Mobile web augmented reality in 5g and beyond: Challenges, opportunities, and future directions," *China Communications*, vol. 16, no. 9, pp. 141–154.

[10] A. Zhao, G. Balakrishnan, F. Durand, J. V. Guttag, and A. V. Dalca, "Data augmentation using learned transformations for one-shot medical image segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8543–8553.

[11] Y. Li, B. Peng, L. He, K. Fan, and L. Tong, "Road segmentation of unmanned aerial vehicle remote sensing images using adversarial network with multiscale context aggregation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.

[12] J. Han, L. Yang, D. Zhang, X. Chang, and X. Liang, "Reinforcement cutting-agent learning for video object segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9080–9089.

[13] G. Tsai, C. Xu, J. Liu, and B. Kuipers, "Real-time indoor scene understanding using bayesian filtering with motion cues." in *ICCV*, 2011, pp. 121–128.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014, pp. 580–587.

[15] A.-L. Chauve, P. Labatut, and J.-P. Pons, "Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 1261–1268.

[16] B. Sredojev, D. Samardzija, and D. Posarac, "Webrtc technology overview and signaling solution design and implementation," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics*. IEEE, 2015, pp. 1006–1009.

[17] L. López, M. París, S. Carot, B. García, M. Gallego, F. Gortázar, R. Benítez, J. A. Santos, D. Fernández, R. T. Vlad *et al.*, "Kurento: the webrtc modular media server," in *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 2016, pp. 1187–1191.

[18] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[19] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[20] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient nd image segmentation," *International journal of computer vision*, vol. 70, no. 2, pp. 109–131, 2006.

[21] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "Planercnn: 3d plane detection and reconstruction from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4450–4459.

[22] Y. Xian, S. Choudhury, Y. He, B. Schiele, and Z. Akata, "Semantic projection network for zero-and few-label semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8256–8265.

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

[24] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[25] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[26] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[27] C. Liu, J. Yang, D. Ceylan, E. Yumer, and Y. Furukawa, "Planenet: Piece-wise planar reconstruction from a single rgb image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2579–2588.

[28] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 583–598, 1991.

[29] S. Taheri, A. Vedienbaum, A. Nicolau, N. Hu, and M. R. Haghighat, "Opencv. js: Computer vision processing for the open web platform," in *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 2018, pp. 478–483.

[30] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with webassembly," in *ACM SIGPLAN Notices*, vol. 52, no. 6. ACM, 2017, pp. 185–200.

[31] M. Van den Bergh, X. Boix, G. Roig, and L. Van Gool, "Seeds: Superpixels extracted via energy-driven sampling," *International Journal of Computer Vision*, vol. 111, no. 3, pp. 298–314, 2015.

[32] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 1693–1700.

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

[34] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[35] X. Song, H. Chen, B. Zang, X. SONG, H. CHEN, and B. ZANG, "Characterizing the performance and scalability of many-core applications on virtualized platforms," *Parallel Processing Institute Technical Report Number: FDUPPITR-2010*, vol. 2, 2010.

[36] S. Lakshmi, D. V. Sankaranarayanan *et al.*, "A study of edge detection techniques for segmentation computing approaches," *IJCA Special Issue on "Computer Aided Soft Computing Techniques for Imaging and Biomedical Applications" CASCT*, pp. 35–40, 2010.

[37] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 6, pp. 641–647, 1994.

[38] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 11, pp. 1101–1113, 1993.

[39] Z. Li and J. Chen, "Superpixel segmentation using linear spectral clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1356–1363.

[40] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[41] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[42] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," pp. 234–241, 2015.

[43] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.

[44] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *european conference on computer vision*, pp. 21–37, 2016.

[45] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[46] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[47] K. He, G. Gkioxari, P. Dollar, and R. B. Girshick, "Mask r-cnn," *arXiv: Computer Vision and Pattern Recognition*, 2017.

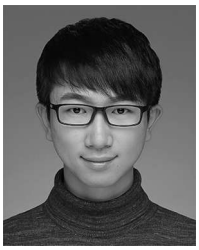
[48] P. Ren, X. Qiao, Y. Huang, L. Liu, S. Dustdar, and J. Chen, "Edge-assisted distributed dnn collaborative computing approach for mobile web augmented reality in 5g networks," *IEEE Network*, vol. 34, no. 2, pp. 254–261, 2020.

[49] 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*. IEEE, 2020, pp. 834–843.

[50] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 253–266.

Yakun Huang is currently working toward the Ph.D. degree at the Network Service Foundation Research Center, 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 with the Beijing University of Posts and Telecommunications, Beijing, China, where he is also the Deputy Director of the Network Service Foundation Research Center, State Key Laboratory of Networking and Switching Technology. 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. He served as the associate editor for the Computing (Springer) and the editor board of China Communications Magazine.



Pei Ren is currently working toward the Ph.D. degree at the Network Service Foundation Research Center, 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.

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.

Junliang Chen received the B.S. degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 1955, and the Ph.D. 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 with the Research Institute of Networking and Switching Technology. His current research interests include communication networks and next-generation service creation technology. Dr. Chen 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 Progess Award in 1988, 2004 and 1999 respectively.