Edge Assisted Real-time Instance Segmentation on Mobile Devices

Jialin Zhang[1*], Xiang Huang[1*], Jingao Xu[1], Yue Wu[1], Qiang Ma[1], Xin Miao[1],
Li Zhang[2], Pengpeng Chen[3], Zheng Yang[1†]
[1]School of Software and BNRist, Tsinghua University
[2]HeFei University of Technology
[3]China University of Mining and Technology
[†]Corresponding author [*]Co-primary author

***Abstract*—Accurate and real-time instance segmentation on mobile devices enables a wide spectrum of applications such as augmented reality, context-aware inspection and environmental cognition. However, the computation resource demanded by instance segmentation impedes its deployment on resource-constrained commercial mobile devices. Prior studies enable smartphones to conduct computational-intensive tasks in real-time with the assistance of an edge server. However, simply applying an edge-assisted framework hardly achieves delightful segmentation performance due to the movements of devices and targets, pixel-level precision requirements, and huge computational overhead even for edge nodes. This work proposes *edgeIS*, an edge-assisted system that enables real-time and accurate instance segmentation on mobile devices. *edgeIS* embraces the mobile device sensing ability of surroundings and its own motion, and redesigns an innovative mobile-edge collaboration paradigm suitable for segmentation tasks. We implement *edgeIS* on a lightweight edge node and different mobile devices. Extensive experiments are conducted under four datasets. The results show that *edgeIS* can run on mobile devices in real-time and achieve a 0.92 segmentation IoU, outperforming existing state-of-the-art solutions. We further embed *edgeIS* in an AR-based inspection system deployed in an oil field and the performance of *edgeIS* meets the demand of the industrial scenario.**

I. Introduction

Instance segmentation aims at detecting and segmenting object instances in an image [1] and lies in the heart of numerous applications in areas such as robotics and image/video retrieval. Nowadays, the ability to perform real-time instance segmentation on mobile devices (*e.g.*, smartphones, smart glasses, light-weight robots or drones) also gradually becomes a vital part of Augmented Reality (AR) [2], autonomous driving [3], context-aware localization [4], *etc.*. For instance, in safety-critical applications in complex environments, robots or drones need to perceive objects and humans instantly, especially their contours; and in AR applications, understanding the spatial layout of objects through semantic segmentation will enhance the realism of virtual effects rendering and provide immersive experiences for users [5]. However, Deep Learning (DL) models of instance segmentation tasks usually contain tons of parameters and thus require powerful GPUs (*e.g.*, Titan V, Titan RTX) to achieve both high accuracy and real-time performance [6]. While there has been significant progress in DL acceleration [7], [8], thus far, no prior work could perform instance segmentation inference at real-time speeds (*e.g.*, 20-30 fps camera rate) on commercial mobile devices.

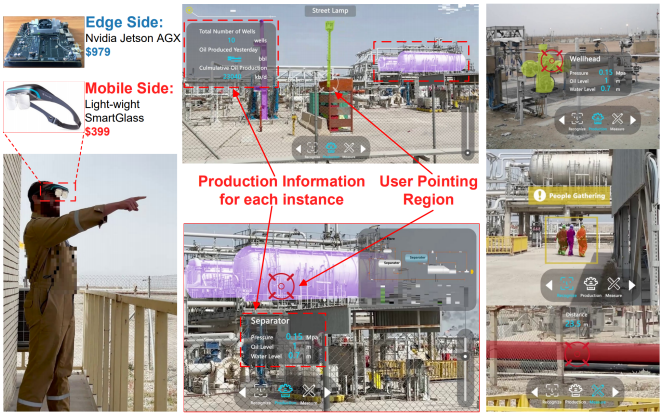
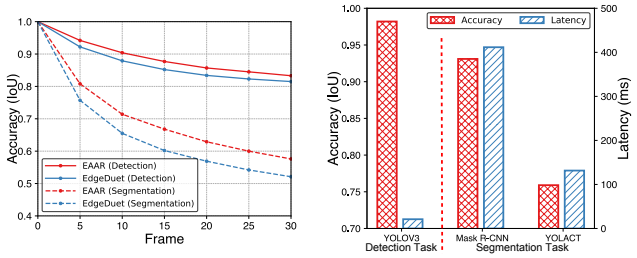


Fig. 1. The user interface of *edgeIS*. We deploy *edgeIS* in one of the world's largest oil-field and provide an AR-based solution for efficient and intelligent industrial inspection. *edgeIS* achieves **real-time** and **high-precision** segmentation on light-weight mobile devices with the assistance of a commercial edge node.

To date, edge computing has been gaining increasing research interests and makes it feasible to run some tasks with high computation overhead on mobile devices. With the assistance of edge nodes or servers, computational-intensive yet delay-tolerant computation could be offloaded to the edge, while the mobile devices only focus on those light-weight yet time-sensitive tasks [9]. Pioneer studies [10]–[14] enable mobile devices to perform accurate object detection tasks in real-time with edge assistance. Specifically, the most recent works, EdgeDuet [10] and EAAR [11], adopt a prevailing "track+detect" framework [12] where a mobile device uses cached results to *track* some objects in the current frame and an edge node *detects* those objects merely in selected keyframes to further update the mobile cache (Section II-A).

Inspired by current practice, an intuitive solution is to exploit edge offloading to realize real-time segmentation. We attempt to embed such an edge-assisted "track+detect" framework into an instance segmentation system to provide an AR-based system for intelligent industrial inspection. As shown in Fig.1, when a user points to a region (*i.e.*, a set of pixels), an ideal system would provide semantics (*e.g.*, oil separator, tube) about it and renders associated industrial information in his view. In our case, both *real-time* and *high-precision* segmentation performance on a light-weight mobile device (*e.g.*, AR glasses) are desirable: a semantic misunderstanding



(a) Mobile Tracking Performance (b) Edge Inference Performance

Fig. 2. An evaluation of existing solutions used for segmentation tasks. The frame rate is set at 30 fps and detailed setups are provided in Section VI-B. (a) We leverage the accurate detection bounding box or segmentation mask at frame #0 as the cached result and use the trackers proposed in EAAR and EdgeDuet to track it among adjacent frames respectively. As seen, the segmentation mask tracking performance degrades significantly as the inter-frame interval increases compared with the detection task, and drops >30% with a merely 10 frames' interval (0.33s). (b) Unlike YOLO v3 which achieves both an accurate ($>$ 0.98 IoU) and real-time ($<$ 30ms delay) detection performance, the two segmentation models, Mask RCNN (0.92 IoU with 400ms delay) and YOLACT (0.75 IoU with 120ms delay), make a trade-off between inference accuracy and latency.

of merely a few pixels will result in rendering the wrong information; and the user's view changes rapidly due to the motion of objects and himself, thus requiring a lower end-to-end latency. Unfortunately, we eventually find existing solutions significantly fall short in performing the more complicated instance segmentation than the object detection task. The grand challenges are twofold:

• **Cached results are hard to reuse on mobile.** Unlike detection tasks, object segmentation requires a compact pixel-level image mask rather than a rough region-level bounding box for each object. Due to the movement of the object being tracked and the camera itself, the pixel association varies greatly among frames. Existing object trackers [10], [11], [15] thus cannot transfer the complicated masks among adjacent frames in fine granularity. To validate our analysis, we measure the existing solutions' mobile side tracking performance under different tasks. As shown in Fig.2a, the performance drops drastically when handling the segmentation task.

• **Heavy DL models are difficult to accelerate on edge.** Unlike well-studied fast object detection DL models (*e.g.*, YOLOv3 [16] with 65 GFLOPs[1]), it's still difficult to achieve real-time yet high-precision instance segmentation even on an edge server due to the huge computational resource overhead (*e.g.*, Mask R-CNN [17] with ResNet-50 FPN has 294 GFLOPs and a relative light-weight YOLACT [8] still has 118 GFLOPs). As demonstrated in Fig.2b, compared with YOLOv3, it's still challenging for Mask RCNN or YOLACT to achieve an accurate and real-time segmentation performance. It's worth noting that commercial yet resource-limited edge nodes (*e.g.*, Nvidia Jetson TX2 [18], AGX Xavier [19]) are more preferred due to the limitations such as space and electric power constraints in actual industrial deployments, which further boosts the inference delay.

In summary, there still lacks an edge-assisted solution applying to real-time instance segmentation tasks.

[1]Giga Floating-point Operations (GFLOPs) is a golden metric indicating the computational overhead of a DL model

Nowadays, commercial mobile devices are granted with capabilities of self-tracking and sensing the ambient environment. For instance, existing AR frameworks such as ARCore and ARKit leverage visual odometry (VO) to track the pose of the smartphone [20]. We found an opportunity to **integrate VO into the edge-assisted framework to realize real-time segmentation on mobile devices.** Our key insight is that the results (poses of a mobile camera and 3D feature points of observed objects) acquired from VO could be leveraged to (i) *transfer* the segmentation masks between adjacent frames by exploiting the spatial projection relationship between them; and (ii) accelerate the mask *inference* process. As the prediction result (*i.e.*, transferred mask) covers the object, a DL model on the edge could thus attend to a smaller region around it, reducing computation operations.

To this end, we propose *edgeIS*, the first **edge**-assisted framework that enable real-time **I**nstance **S**egmentation on mobile devices. *edgeIS* re-designs the well-studied "track+detect" paradigm and provides a "**transfer+infer**" mobile-edge collaboration framework, where mobile devices leverage visual sensing ability to transfer cached segmentation results while the prediction results are also leveraged to accelerate the edge side inference. Our design of *edgeIS* excels in three unique aspects as follows.

First, we introduce a *motion aware mobile mask transfer* module to transfer the segmentation result from a previous keyframe to the current frame (Section III). Implementing this basic idea is non-trivial since existing VO algorithms merely maintain a 3D map of feature points that are much sparser than compact pixels in the image plane, and it's typically unknown whether a feature point belongs to the object [21]. As a consequence, only a few pixels on the target object can be tracked, which is inadequate for the prediction of an accurate mask. In *edgeIS*, we enrich the representations of concerned objects and keep track of their motions as well as the device's to assure the smooth mask transferring.

Second, we design a *contour instructed edge inference acceleration* scheme on the edge (Section IV). With motion-aided predictions (*i.e.*, transferred masks), *edgeIS* monitors and discards unnecessary operations in the DL model, thus reducing inference latency without compromising accuracy.

Moreover, we present a *content-based fine-grained RoI selection* strategy to effectively compress the transmitted data (Section V). *edgeIS* divides each frame into tiles of different compression levels based on its content and upgrades a tile-level frame encoding scheme [10] to reduce transmission latency without harming the performance on both sides.

We fully implement *edgeIS's* server side on an Nvidia Jetson TX2 and mobile side on three different types of mobile devices. We conduct extensive experiments under different network conditions and examine *edgeIS* on three public datasets and a self-labeled dataset with more than $19k$ frames. We also compare *edgeIS* with two state-of-the-art related works, EdgeDuet [10] and EAAR [11]. The results show that *edgeIS* achieves an average frame rate of 30fps with 0.92 segmentation IoU (intersection over union), improving the accuracy by 10-20% and reducing the false rate by at least 70% compared with previous studies.

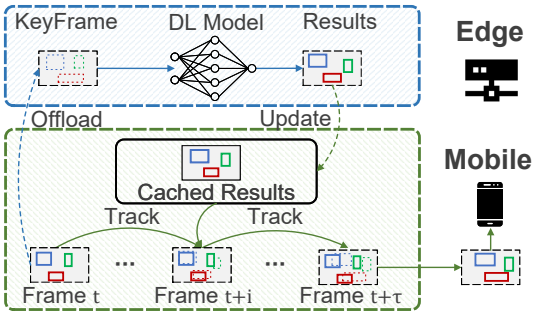


Fig. 3. Overview of existing "track+detect" framework

Real-world deployment. We have developed a real-time mobile AR inspection application based on *edgeIS* and deployed it in one of the world's largest oil fields for industrial equipment inspection (Fig.1). Our field study consists of 8 mobile devices connected with an Nvidia Jetson AGX Xavier [19] as the edge node through WiFi and LTE connections. A one-week pilot study shows that *edgeIS* is capable of providing real-time segmentation and AR visual effects with an average IoU accuracy of 0.87, and the computation overhead records indicate that *edgeIS* can run stably for hours within the resource constraints.

In a nutshell, our core contributions are three-fold.

- We propose *edgeIS*, as far as we are aware of, the first system that enables mobile devices to perform instance segmentation, an extremely resource-intensive task, in real-time by re-designing the edge-assisted architecture.
- We integrate VO into the mobile-edge collaboration system and design a "transfer+infer" framework that achieves an innovative two-way facilitation. Not only the mobile side can transfer segmentation results between adjacent frames, but the inference of the heavy DL model on the resource-constrained edge node is dramatically accelerated.
- We extensively evaluate the performance of *edgeIS* and compare it with two related works. The results show that *edgeIS* achieves remarkable performance in all scenarios. We also embed *edgeIS* in an AR-based inspection system in an oil-field, and the segmentation accuracy and real-time performance of *edgeIS* meet the demand of the industrial scenario.

II. System Overview

A. Existing Edge-assisted "Track+Detect" Paradigm

We first briefly introduce the prevailing "track+detect" framework (Fig.3). On the mobile side, the detection results of the current frame are obtained by adapting cached detection results of prior frames using lightweight trackers (*e.g.*, motion vector [11], KCF [22]). Meanwhile, the cached results are routinely updated by offloading some keyframes for relatively expensive yet accurate object detection on the edge side (*e.g.*, YOLOv3 [16], ViT [23]). Additionally, current efforts [10], [11] also design region-of-interest (ROI) encoding or content-prioritized image clips offloading schemes to reduce the data transferred in-between, improving the real-time performance.

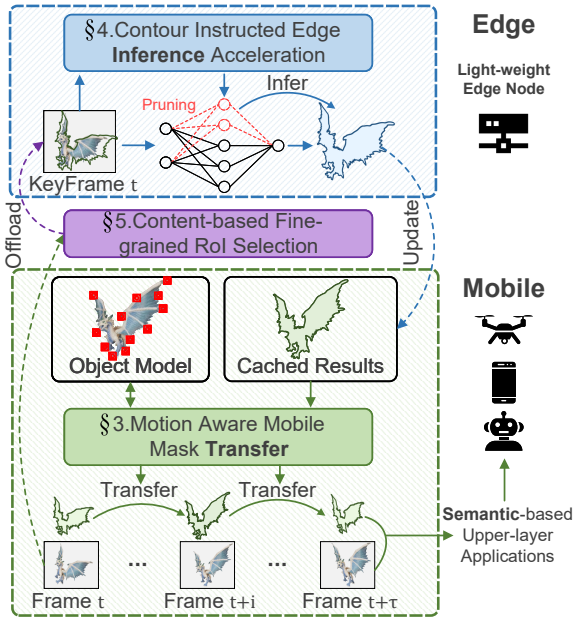


Fig. 4. System architecture of *edgeIS*

B. edgeIS's "Transfer+Infer" System Architecture

edgeIS aims to re-design the "track+detect" paradigm, making it capable of addressing the heavy tasks of real-time instance segmentation. Fig.4 sketches the system architecture of *edgeIS*. From the top perspective, *edgeIS* proposes a "transfer+infer" framework, which achieves a two-way promotion of both mobile and edge sides. Specifically, on the mobile side, we modify the primary functions of VO and integrate them into the *motion aware mobile mask transfer* module. *edgeIS* continuously tracks the motion of the device and models the observed objects, and on the basis of these, computes the masks for the current frame in real-time.

On the server side, a *contour instructed edge inference acceleration* module is leveraged to speed up the inference of the heavy DL model. Specifically, *edgeIS* deeply couples the previous prediction results from the mobile device into the DL model by *dynamic anchor placement* and *RoI pruning*.

Furthermore, we design a *content-based fine-grained RoI selection* scheme to decide i) whether a frame will be sent to the edge; and ii) where should we focus in the frame and which irrelevant areas can be compressed. Benefiting from the design, *edgeIS* significantly reduces the network bandwidth dependency of the system without harming the performance.

III. Motion Aware Mobile Mask Transfer

In *edgeIS*, we design a *motion aware mobile mask transfer* scheme to enable mobile devices to accurately predict the segmentation results of the current frame in real-time (*i.e.*, with the 30 fps camera rate). As aforementioned, due to the complexity of the instance segmentation task, previous updating schemes (*e.g.*, local tracker based on motion vector, KCF or feature matching) can hardly achieve delightful mask prediction performance.

In *edgeIS*, we leverage VO and embrace the awareness of objects' and the device's motion to achieve accurate pixel-level mask transfer between adjacent frames. The workflow

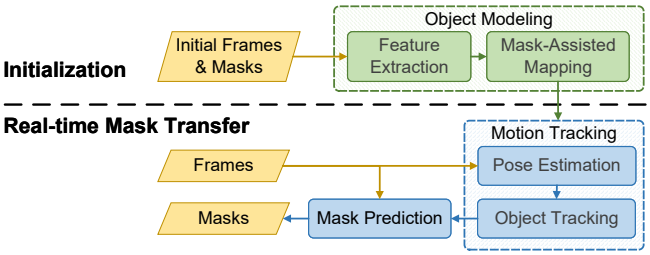


Fig. 5. The workflow of the *Motion Aware Mobile Mask Transfer* scheme

of *mobile mask inference* module is shown in Fig.5. In the beginning, with consecutive frames and their corresponding accurate masks from the edge server, *edgeIS* initializes the pose of the mobile device and models the observed objects in 3-D space. Once the initial map is constructed, the poses of the device and observed objects are continuously updated with a scheme of motion tracking. Then for each frame, *edgeIS* leverages spatial information and previous masks to predict the mask at the current pose. Details of each function are described below.

A. Initial Object Modeling

To construct the 3-D model of the environment and objects, *edgeIS* performs feature extraction and matching between consecutive frames to choose a set of feature pixels representing the same 3-D points in the two frames. In our implementation, we use ORB [24] feature for its efficiency in computing and robustness against the change of viewpoints.

During initialization, a pair of video frames are selected to estimate the starting pose and model the observed objects in 3-D space. To select the two initial frames, *edgeIS* continuously tries consecutive frames for motion estimation and chooses a pair of them with enough parallax and matched feature points to construct an initial map. The two determined frames will then be sent to the edge server for masks with accurate semantic annotations generated by deep learning models.

According to accurate masks from the edge server, the features are divided into two groups. As illustrated in Fig.6a, denote the two frames as Frame 0 and Frame 1, any pair of features belonging to the background mask as $\mathbf{p}_0$ and $\mathbf{p}_1$, and other matched features as $\mathbf{q}_0$ and $\mathbf{q}_1$. To better model the objects of interest while avoiding heavy computation, a selection process is executed on all features. For background features, *edgeIS* will check whether they are too blurred or too close to neighboring ones and filter out features that fail the check. For features within masks, *edgeIS* first preserves all features near the edge of the mask since pixels on the contour are more representative for the object's shape, and then performs blurriness check on features inside the mask. Once the selection of features is done, the relative pose (consisting of a rotation matrix $\mathbf{R}_{10}$ and a translation vector $\mathbf{t}_{10}$) between the two frames is computed by first using epipolar geometry to solve fundamental matrix $\mathbf{F}_{10}$ [25].

$$\begin{aligned} \mathbf{p}_1^{\mathrm{T}}\mathbf{F}_{10}\mathbf{p}_0 &= 0, \\ \mathbf{q}_1^{\mathrm{T}}\mathbf{F}_{10}\mathbf{q}_0 &= 0, \end{aligned} \tag{1}$$

Solving $\mathbf{F}_{10}$ requires no less than 8 pairs of features and *edgeIS* will first uses all pairs of $\mathbf{p}_0$ and $\mathbf{p}_1$ since the pixels of background are more likely to be static. Then the relative pose between the two frames is solved by:

$$\mathbf{t}_{10}^{\wedge}\mathbf{R}_{10} = \mathbf{K}^{\mathrm{T}}\mathbf{F}_{10}\mathbf{K}, \tag{2}$$

where $\mathbf{K}$ is the intrinsic matrix of the camera and $(\cdot)^{\wedge}$ calculates the skew-symmetric matrix from a vector. With the recovered pose, VO triangulates 3-D points from selected features and construct the initial map of the captured scene (taking $\mathbf{p}_0$ and $\mathbf{p}_1$ as example):

$$s_1\mathbf{R}_{10}(\mathbf{K}^{-1}\mathbf{p}_0)^{\wedge}(\mathbf{K}^{-1}\mathbf{p}_1) + \mathbf{t}_{10}^{\wedge}(\mathbf{K}^{-1}\mathbf{p}_1) = 0, \tag{3}$$

where s_1 is the depth of features in Frame 1 and s_0 can also be computed using the same method. Once a 3-D point is created, *edgeIS* annotates it according to its corresponding features. If the two features belong to masks with the same label (*e.g.* people or car), the 3-D point will be labeled as the same class and otherwise as background. After initialization, *edgeIS* obtains a starting pose for continuous motion tracking and an annotated map that models its observed objects.

B. Motion Tracking

As the mobile device moves around, VO continuously tracks its pose as well as updates the 3-D map. For the i-th coming frame, VO first detects feature pixels in it and then matches them to 3-D points in the map. With this correspondence from 2-D pixels to 3-D points, VO can solve the device pose of i-th frame in the world coordinate $\mathbf{T}_{\mathtt{CW}}^i = [\mathbf{R}_{\mathtt{CW}}^i|\mathbf{t}_{\mathtt{CW}}^i]$ by minimizing the reprojection error between the map points projected to the frame given the pose of the frame and the features in it. The minimization can be solved as an optimization problem using bundle adjustment [26]:

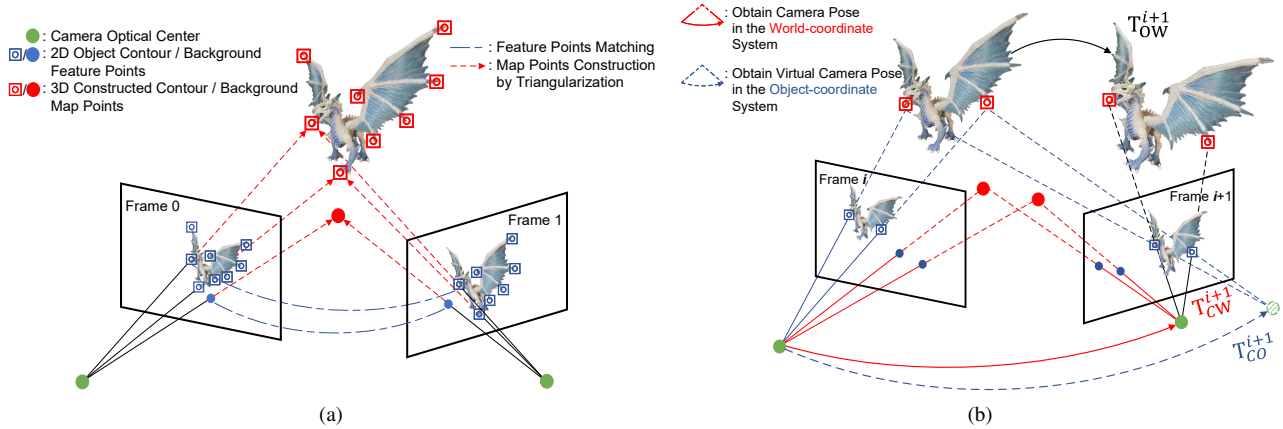
$$\mathbf{T}_{\mathtt{CW}}^i = \arg\min_{\mathbf{T}_{\mathtt{CW}}^i} \sum_k \left\| \boldsymbol{\pi}(\mathbf{T}_{\mathtt{CW}}^i, \mathbf{P}_{\mathtt{W}}(k)) - \mathbf{p}^i(k) \right\|_2^2, \tag{4}$$

where $\mathbf{P}_{\mathtt{W}}(k)$ denotes the k-th 3-D point in the map, $\mathbf{p}^i(k)$ denotes its corresponding feature in the i-th frame and $\boldsymbol{\pi}(\cdot)$ stands for the function of projecting a 3-D point to its pixel location on the frame given the pose $\mathbf{T}_{\mathtt{CW}}^i$:

$$\boldsymbol{\pi}(\mathbf{T}_{\mathtt{CW}}^i, \mathbf{P}_{\mathtt{W}}(k)) = \mathbf{K}\mathbf{R}_{\mathtt{CW}}^i\mathbf{P}_{\mathtt{W}}(k) + \mathbf{t}_{\mathtt{CW}}^i, \tag{5}$$

Once the pose of the frame is determined, VO triangulates 3-D points in the newly observed areas and the map gets updated in the same frequency as input.

Especially, in our implementation, *edgeIS* also identifies the 3-D points belonging to moving objects and computes their poses individually. The process of pose estimation as well as tracking observed objects is shown in Fig.6b. To compute the pose the mobile device itself, *edgeIS* mainly selects 3-D points which are labeled as background and their matched features to perform the bundle adjustment and get the current pose $\mathbf{T}_{\mathtt{CW}}^i$. Then for each object, *edgeIS* uses all 3-D points belonging to it and the corresponding features to estimate the device's pose relative to the object, denoted as $\mathbf{T}_{\mathtt{CO}}^i$. If the object is moving, the estimated poses of the mobile device with two different reference systems will be different. With this difference, the object's motion relative to the background can be computed as:



(a) (b)

Fig. 6. Motion aware mask transfer module. (a) Initial object modeling. (b) Pose estimation and object tracking.

$$\mathbf{T}_{\mathtt{CO}}^{i} = \arg\min_{\mathbf{T}_{\mathtt{CO}}^{i}} \sum_{k} \left\| \boldsymbol{\pi}(\mathbf{T}_{\mathtt{CO}}^{i}, \mathbf{Q}_{\mathtt{O}}(k)) - \mathbf{q}^{i}(k) \right\|_2^2. \tag{6}$$

To associate the dynamic object's pose in real-world reference, the following transformation is needed:

$$\begin{aligned} \mathbf{T}_{\mathtt{OW}} &= \mathbf{T}_{\mathtt{OC}}\mathbf{T}_{\mathtt{CW}} = \mathbf{T}_{\mathtt{CO}}^{-1}\mathbf{T}_{\mathtt{CW}} \\ &= \begin{bmatrix} \mathbf{R}_{\mathtt{CO}}^{\mathrm{T}}\mathbf{R}_{\mathtt{CW}} & \mathbf{R}_{\mathtt{CO}}^{\mathrm{T}}(\mathbf{t}_{\mathtt{CW}} - \mathbf{t}_{\mathtt{CO}}) \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}, \end{aligned} \tag{7}$$

where the $\mathbf{T}_{\mathtt{OW}}$ is the required pose of the dynamic object in the world coordinate.

Leveraging the annotation and grouping of the 3-D points, the pose of each object gets to be updated individually and thus yields better performance in dynamic scenarios. Besides, it's worth mentioning that performing BA requires at least 3 pairs of 3-D points and matched features, which is a small number. Therefore, if the number of points belonging to an object is less than 3, *edgeIS* takes it as either it is too small or too far away for accurate estimation.

C. Mask Prediction

The motion information and the 3-D map of observed scenes that VO provides lay the foundation of predicting the mask for the current frame without deep learning models. When a new frame arrives, *edgeIS* first estimates its pose and updates the map with it. However, to render the mask of each instance on the frame, there are still several problems that need addressing.

The first one is to determine a set of previous frames which *edgeIS* uses to predict the mask for the current frame. By matching feature pixels with existing 3-D points with annotations, *edgeIS* gets the initial information about what objects appear on the current frame. For each object, *edgeIS* searches the frame which meets the requirements of both observing the object clearly (*i.e.* the object is fully captured without occlusion) and sharing similar viewpoints with the current one (the angle between the frames is not too large). This process can be well integrated into VO structure since each 3-D point stores the information about frames observing it and the angle between poses can be easily computed.

After the source frames are selected, the next question is how to use their object masks to compute one for the current frame. Compared with the image resolution, the density of features used for matching and map construction is pretty sparse, which indicates that the mask can not be directly reprojected onto the new one. To address this problem, a basic observation is that the shape of a mask is determined by its contour. Therefore, if the pixels on the edge of an instance mask can be located on the new frame, the mask itself is determined accordingly.

In *edgeIS*, we first extract the contour of the object mask on the source frame using the `findContours` function in OpenCV. The extracted contour, denoted as S, is represented by a list of connected pixels. Then for all $s_i \in S$, if not a feature pixel itself, *edgeIS* performs a search for k closest features within the mask. k is an empirical value based on our observation that the actual positions in 3-D space corresponding to a small neighborhood of the object mask are not likely to experience shape changes in depth (in our implementation, we set k to 5). With the average depth as its own depth, s_i can be projected to the current frame with the relative pose and the projection function $\boldsymbol{\pi}(\cdot)$. Applying the procedure to all observed objects and labeling other pixels as background, the mask for the current frame is generated by *edgeIS* for future use.

IV. Contour Instructed Edge Inference Acceleration

As aforementioned, unlike previous object detection tasks, the instance segmentation is still challenging to perform in real-time with high accuracy even on an edge node or server. In general, the latency of predicting instance masks takes a large portion of total latency and thus significantly influence the performance with a delayed image [12].

In *edgeIS*, we design the *contour instructed edge inference acceleration* solution. We dig deeper into each functional module of RoI-based segmentation DL models and eventually find an opportunity to accelerate the model inference by leveraging the instant prediction results for object contours to discard redundant calculations. In what follows, we use Mask R-CNN [17], one of the most famous and accurate instance segmentation solutions, as a representative to illustrate the modification we made on the network structure. We first describe the scheme of *dynamic anchor placement* to explain

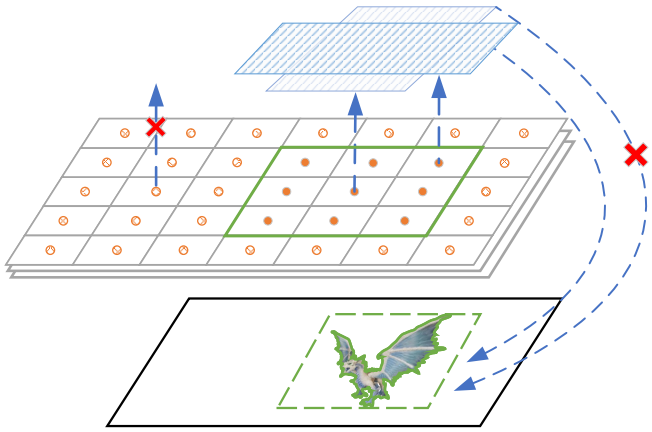


Fig. 7. Illustration of model acceleration. The computation of deep learning model is reduced by two functions performing on different stages namely dynamic anchor placement and RoI pruning.

how we use estimated mask to prune unnecessary operations and then the design of *RoI pruning* which improves the efficiency of selecting active areas.

A. Dynamic Anchor Placement

The architecture of Mask R-CNN consists of two stages. The first stage is Region Proposal Network (RPN) which proposes candidate object boxes for the following tasks. And in the second stage, tasks including classification, object detection and mask prediction are executed in parallel. When taking an image as input without extra information, RPN needs to slide a small network across the whole convolutional feature map generated by the backbone network to propose initial bounding boxes. Since the object of interest generally occupies only a small portion of the image while the background takes the majority, calculating at every sliding window location in the feature map inevitably producing a large number of redundant calculations.

With the transferred mask of the object from the mobile device, the model can be "instructed" to a perform more targeted region proposal, inspired by which we design the *dynamic anchor placement*. First, a surrounding box is calculated from the mask of each object. Then all convolutional layers in the backbone of RPN are registered with the size of feature maps they produced since the FPN (Feature Pyramid Backbone) can provide features from different levels. Apart from areas containing objects, new areas captured by the mobile device as mentioned in section Section V are also annotated with an initial box. Once the feature for region proposal is determined, each initial bounding box in the image will be downsampled to form a rectangular area in the feature map according to the scale from image size to feature size with an empirical padding. As illustrated in Fig.7, to propose regions likely to contain objects, the range of the feature map that RPN needs to traverse and place anchors is limited within these areas. Restricting areas based on the mobile side information, *dynamic anchor placement* rid the RPN of a large portion of unnecessary calculations.

B. RoI Pruning

At each sliding window location, several anchor boxes of different shapes are used to generate possible regions with box coordinates and class confidence. Though some selection processes are executed to reduce the number of RoI used in the second stage, with the prior knowledge of the approximate location and the class of the object, this pruning process can be optimized for better efficiency and accuracy.

First, *edgeIS* groups all RoIs by the area annotated in the *dynamic anchor placement*. Then RoIs coming from the same area of a known object with class label c are sorted according to their confidence score on this class. In the sorted queue, an IoU score is computed between each RoI and the initial box of the object. (The computing metrics of IoU is described in Section VI-B.) As shown in Fig.7, an RoI will be pruned if there exists an RoI with both higher confidence score on class c and IoU score with the initial box of the object. As for RoIs from the area corresponding to the unknown contents in the image, the Fast NMS introduced by [8] is applied to perform the selection.

The intuition behind the pruning process is that for each discarded RoI, there is a better candidate for accurate mask prediction. By utilizing the prior information of the class and the estimated location of the object, *RoI pruning* saves the second stage where a mask is generated within each RoI from a large portion of unnecessary computations. Besides, the pruning of different classes can run in parallel while IoU computing and batch sorting are already available in current frameworks. Therefore, the operations can be efficiently implemented and accelerated by standard GPU.

V. Content-based Fine-grained ROI Selection

The performance of the mobile device and the edge server is also highly influenced by the transmission delay [27]. As discussed in our previous work [10], the transmission scheme should meet the requirements of both providing images (or some critical areas in images) of high resolution for the inference on the edge server and reducing transmission latency for real-time performance on the mobile side. In *edgeIS*, we propose a *content-based fine-grained RoI selection* scheme that first decides the frame to be transmitted and then efficiently compresses frames and masks on fine-grained RoI level based on the awareness of image content.

To determine the timing of transmitting a frame to the edge, *edgeIS* leverages the motion information obtained from Section III-B. As the device moves, the content observed at the current location is continuously updated with the device's pose and modeled objects. As illustrated in Fig.8b, if the proportion of the features matched with unlabeled points (the yellow ones in the frame) is larger than a threshold t, *edgeIS* will take it as that a large area of the frame is new and thus needs accurate pixel-level annotation (in practice, we set t to 0.25). Apart from the unseen areas, *edgeIS* checks the motion of observed objects, especially dynamic ones. If an object's pose changes significantly over a period, the transmission is also triggered for mask correction.

The frame to be transmitted is first divided into several different types of rectangular areas. As shown in Fig.8c,

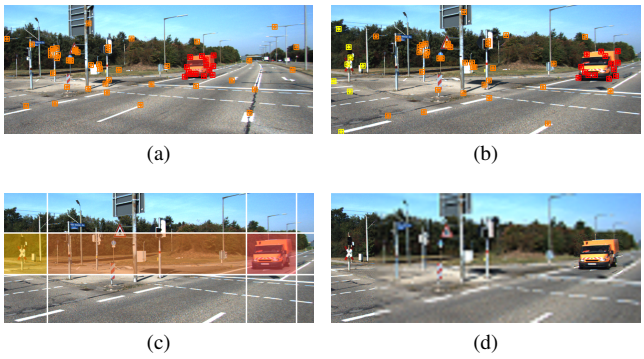


Fig. 8. An example of the proposed RoI selection and image compression module. (a) The original frame, the segmentation mask of a truck, and feature points about static background (marked in orange) and dynamic objects (marked in red). (b) A video frame to be transmitted after both the camera and the object move. The yellow points indicate the feature points extracted from newly emerging scenes. (c) Frame area partition according to different contents. (d) Encoded frame with different compression levels for each region in it.

for dynamic objects marked with red features and unknown areas with yellow ones in Fig.8b, the corresponding areas are assigned with low compression levels to preserve the high quality of the image. The remaining blocks, containing either background or static pixels (marked with orange features), are compressed to low quality without harming the performance of the segmentation model on the edge server. Put them together, the eventually image with different compression levels for each region is shown in Fig.8d.

Based on the definition of tile defined in HEVC [28], we implement the encoding method by modifying Kvazaar [29] which supports parallel tile encoding and therefore is suitable for acceleration. For each tile of different compression level, we encode the types and contour coordinates of objects it contains along with its picture parameter set into the bit-stream of the frame. Correspondingly, we modify the OpenHEVC [30] to decode the blocks and provide the position information of target objects to the inference acceleration module.

As for the mask generated on the edge, instead of transmitting the whole mask back to the mobile device, *edgeIS* only extracts and sends the contour and class label of each instance back to the mobile device, which are enough to reproduce and render the whole mask. By compressing individual blocks of the frame differently according to their contents, our transmission module achieves the goal of reducing the amount of transmitted data while maintaining high accuracy. By using pixel coordinates rather than image representation, the transmission cost of sending back masks is also considerably reduced.

VI. Experiments and Evaluation

A. Implementation

Client. We implement the mobile part of *edgeIS* on mobile devices, including an Apple iPhone 11 and a Samsung Galaxy 10. The functions of the mobile module are mainly implemented in C++ for ease of cross-platform deployment. For input and output, we use OpenCV [31] for feeding video frames at fixed 30 fps and rendering masks and visual effects on the screen. The core parts of the mask transfer module follow the workflow in Fig.5, we modify the VO from ORB-SLAM [32] to realize motion tracking and object modeling, based on which the functions of object tracking and mask prediction are implemented. Specifically, we modify its feature extraction and matching function to apply information from the instance mask. In the tracking thread, we separate the original motion estimation function to maintain both the poses of objects and the mobile device in parallel. The triangulation function is also rewritten to view the points belonging to target objects as a whole rather than only operate on element level (*i.e.* map points and frames), which exceedingly improves its efficiency. The mask prediction function is added to output the segmentation result once the tracking for the current frame is done. Besides, an additional thread is used for communicating with the transmission module to send frames according to the mobile device and receive masks from the server.

Server. We deploy our edge-side module on an Nvidia Jetson TX2 edge server. For the deep learning model, we choose Mask R-CNN [17] with a ResNet-101-FPN as backbone feature extractor and the parameters are pretrained on COCO image dataset [33]. Our *contour instructed edge inference acceleration* as well as the Mask R-CNN is implemented in python-3.6.5 with PyTorch-0.4.0 [34].

Remote Data Interaction. As introduced in Section V, we implement our video encoder on the mobile side based on Kvazaar [29] in C++ and the decoder on the server side with the OpenHEVC [30] library. For information such as vertices of the contour, we use C++ Boost [35] library for the serialization and transmission between the two sides.

B. Experiment Setup

We perform an extensive experimental validation of *edgeIS* on three public video datasets namely DAVIS [36], KITTI [37] and Xiph [38] as well as a handcrafted dataset with typical indoor and outdoor AR scenarios. The videos we select for processing from the three public datasets contain a total number of 6,834 frames and the videos in our dataset have a total lasting time of 7.3 minutes with 13,276 frames. In our experiments, all videos are set to an input rate of 30fps with uniform resolution.

Similar to previous works, the most critical performance indicators we care about an edge-assisted system are latency and accuracy, since other factors all lead to a delayed rendering and finally a decreased accuracy [12], [27]. In practice, we use the average IoU representing the similarity between predictions and ground truths to measure the accuracy of *edgeIS*. For an object, denote the set of pixels in its ground truth mask as $\mathcal{S}_{groundtruth}$ and the set of pixels in the predicted mask as $\mathcal{S}_{prediction}$, IoU is given by:

$$\mathbf{IoU} = \frac{|\,\mathcal{S}_{groundtruth} \cap \mathcal{S}_{prediction}\,|}{|\,\mathcal{S}_{groundtruth} \cup \mathcal{S}_{prediction}\,|}. \tag{8}$$

A high IoU score indicates that the predicted mask covers the ground truth without taking in too many unrelated pixels.

To evaluate the segmentation performance, we implement two baseline approaches: running entirely on the mobile device and on the edge node. The former one uses TensorFlow Lite [39] to fully deploy the DL model on the mobile devices. The

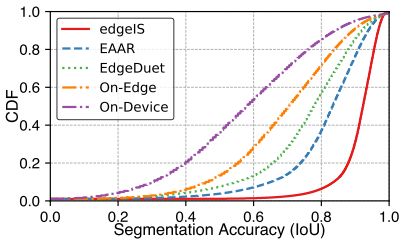


Fig. 9. Overall Performance Comparison

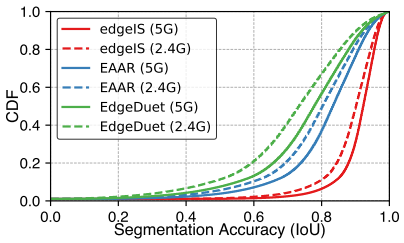


Fig. 10. Performance with Network Conditions

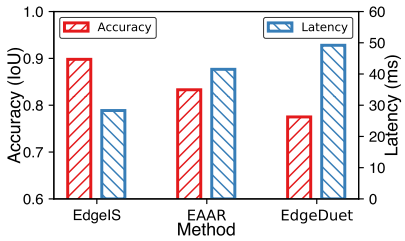


Fig. 11. Overall Latency Comparison

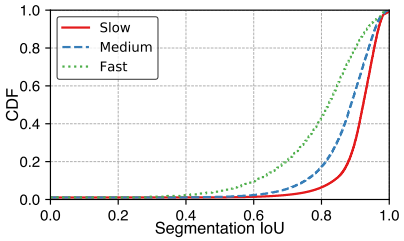


Fig. 12. Impact of Movement Speeds

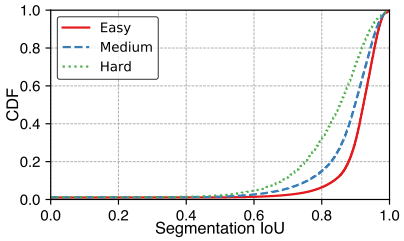


Fig. 13. Impact of Scene Complexities

latter one sends all frames to the edge to perform segmentation and returns results to the mobile device with a best effort strategy. Besides, we apply a motion vector-based scheme on the mobile side to approximately track the mask locally.

Additionally, we modify two previous systems, EdgeDuet [10] and EAAR [11], designed for real-time object detection with edge-assisted architecture to perform the instance segmentation task. We adopt their transmission strategy and replace the object detection model on the edge server with the same Mask R-CNN as *edgeIS* uses. On the mobile side, we adopt the local tracker proposed by them based on either KCF or motion-vector to update the contour of observed objects rather than detection boxes.

C. Performance Comparison

We first evaluate the instance segmentation accuracy and latency of *edgeIS* and compared systems, which are most relevant to user experience. In our evaluation, we adopt a loose threshold of 0.5 and a strict threshold of 0.75 which are commonly used in computer vision community to determine an acceptable accuracy. IoU smaller than the threshold is called a false result. We mainly focus on the strict threshold since high accuracy is required for target application scenarios. For each video clip *edgeIS* runs 3 times and calculate the mean IOU value for each frame.

1) Overall Performance: Fig.9 illustrates the CDF of instance segmentation accuracy. The false segmentation rate of pure mobile computing and best-effort edge-assistance is 78.3% and 60.1%, respectively. With local tracker and transmission schemes, *EAAR* and *EdgeDuet* reduce the false rate to 21% and 39% respectively. outperforming these straightforward methods. However, the trackers designed for objection detection is still too coarse for segmentation task, and the segmentation model fail to provide timely results without acceleration. With motion information obtained for more accurate mask prediction, the false rate of *edgeIS* is only 3.9%, reducing that of *EAAR* and *EdgeDuet* by 82% and 91% correspondingly. Besides, the average accuracy of *edgeIS* is 0.92, improving that of the two systems by 10% and 20%.

2) Performance under different networks: Since the network conditions vary in real scenarios and have an important influence on edge-assisted systems, we further evaluate the performance under different network settings including WiFi 2.4GHz and WiFi 5GHz. We first evaluate the segmentation accuracy of *edgeIS*. As shown in Fig.10, for segmentation tasks, the false rate of *edgeIS* under WiFi 2.4GHz and WiFi 5GHz is 6.1% and 4.1% respectively while the false rate of *EAAR* and *EdgeDuet* under the fastest WiFi 5GHz is 21% and 41%, which will be higher when switched to WiFi 2.4GHz. Under either network condition, *edgeIS* reduce the false rate by at least 78% compared with *EAAR* and 83% compared with *EdgeDuet*, proving effective in various circumstances.

3) Latency Comparison: Fig.11 shows the average latency and accuracy on the mobile side of *edgeIS* and compared systems under WiFi 5GHz. The average IOU of *edgeIS* is 0.89 while that of EAAR and EdgeDuet is 0.83 and 0.78 respectively. As for the time of processing each frame, the average latency of *edgeIS*, EAAR and EdgeDuet is 28ms, 41ms and 49ms correspondingly. With input video of 30fps, latency longer than 33ms accumulates and eventually results in a delayed mask rendering on a later frame, which explains strong relevance between the accuracy and latency. Apart from the impacts of model acceleration which is discussed in the following section, our content-based encoding scheme also helps with compressing the frame with little influence on segmenting the objects. On the contrary, EdgeDuet only preserves small objects in high resolution which leads to a harmed accuracy of large objects and EAAR predicts RoI using motion vector which is more coarse and leaves room for further compression.

D. Robustness Evaluation

1) Camera Motion: To evaluate *edgeIS's* robustness against different moving status, we record videos of the same route with people walking, striding and jogging. Fig.12 shows that the false rate of *edgeIS* is 4.7%, 9.8% and 29.9% respectively in slow, medium and fast circumstances. In the worst case, *edgeIS* can still achieve an average IoU of 0.82. The results

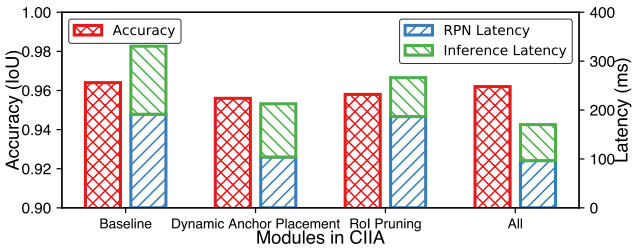


Fig. 14. Effectiveness of Model Acceleration

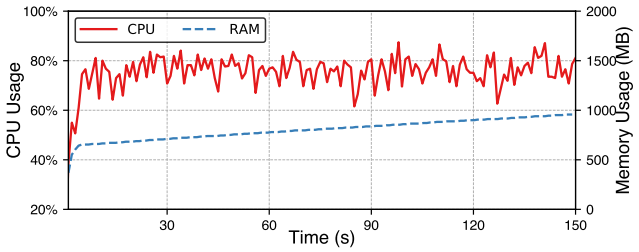


Fig. 15. CPU and Memory Usages on Device

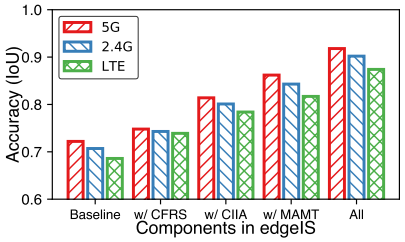


Fig. 16. Effectiveness of each Component

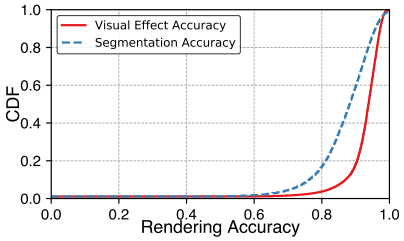


Fig. 17. Rendering Accuracy

demonstrate that with motion information used for predicting masks and transmitting data, *edgeIS* proves robust against various motions and competent for most application scenarios.

2) Scene Complexity: To evaluate *edgeIS's* performance in various scenarios, we manually arrange scenes of different complexity. The number of objects in simple scenes is no more than three and rises to ten in medium scenes. In scenes of hard level, objects will move during the experiment. As shown in Fig.13, the average accuracy in terms of IoU in easy, medium and complex scenes is 0.91, 0.88 and 0.83, respectively. In scenes where objects move during the process, the false rate of *edgeIS* is 19.7%, indicating that most of the observed objects can be well segmented. The results prove that tracking objects' 3-D poses effectively improves the robustness of *edgeIS* in dynamic environments, making it applicable in real scenarios.

E. Benefits of Individual Components

We further analyze the three core components of *edgeIS*, particularly the performance gains that each brings to the overall system. In the following, we abbreviate the three modules, ***m**otion **a**ware mobile **m**ask **t**ransfer*, ***c**ontour **i**nstructed edge **i**nference **a**cceleration*, and ***c**ontent-based **f**ine-grained **R**oI **s**election* as MAMT, CIIA, and CFRS, respectively.

1) Benefits of Individual Modules: In this part, we analyze the improvements obtained with the three core components in *edgeIS* individually. The baseline method we use is the best-effort edge-assistance with motion vector tracking mentioned in Section VI-B. As shown in Fig.16, with CFRS to reduce transmitted data, *edgeIS* achieves a more even performance under different network conditions and the accuracy is improved by 3-7% due to reduced transmission latency. With CIIA, the inference latency on the edge is significantly reduced (the evaluation is shown in Fig.14) and thus provides more timely results. The accuracy is improved by 12-14% under different network connections. The improvement gained with MAMT is more than 19% since motion information directly benefits mask prediction on the mobile side. Compared with baseline methods, *edgeIS* achieves an overall improvement of 27% on accuracy under all network conditions.

2) Benefits of Model Acceleration: We further examine the latency improvement that model acceleration brings to Mask R-CNN. The overall latency is divided into RPN latency and inference latency. As in Fig.14, with dynamic anchor placement, RPN latency is reduced by 46% benefiting from discarding unnecessary anchor computation and inference latency is reduced by 21% with less RoI produced. With RoI pruning, the inference latency is reduced by 43% for redundant RoIs are not used for further inference. Overall, the model acceleration module reduces the latency by 48% while maintaining an accuracy higher than 0.92.

F. Resource Overhead

1) Mobile Resource Usage: As depicted in Fig.15, we measure the CPU and memory usage of *edgeIS* for a period of time after initialization on an iPhone 11. The CPU usage of the system is around 75% and the memory usage is increased by nearly 2MB/s. The increase in memory usage comes from the recording of new data in picture frames and local maps. Through the additional clearing algorithm, the system can periodically clear the data of low utilization to control the global memory usage within 1GB, allowing *edgeIS* to run smoothly on the latest mobile phones.

2) Power Consumption: We record the power consumption of an iPhone 11 and a Samsung Galaxy 10 starting with full power for 10 minutes through the management application The experiment is repeated dozens of times. On average, the power consumption of *edgeIS* in 10 minutes is 4.2% and 5.4% respectively on iPhone11 and Galaxy 10, which is comparable to running ARKit and ARCore demo applications continuously for the same time duration.

G. Case Study

Based on *edgeIS*, we develop a real-time AR inspection application and test it in one of the world's largest oil fields in the Middle East. Our system uses an Nvidia Jetson AGX Xavier as the edge node. For indoor scenarios and places near the campus where WiFi connection can be established, we provide 5 pairs of Dream Glasses as mobile devices. For other

circumstances such as a user checking equipment in locations without WiFi signal covered, we use 3 iPhone 11 with LTE connection instead. We conduct the study for a week and from each device collect video clips of an hour long and summarize our findings regarding system performance.

We measure the system accuracy in two perspectives. In terms of segmentation accuracy, we use a same Mask R-CNN model as the one used on the datasets to segment every frame in the video clips offline as the ground truth and compare the system output with it. Besides, we also evaluate the accuracy of visual information such as equipment details rendered on the frame. We select a consecutive video clip of 5 minutes from each mobile device and randomly sample 1 frame out of every 30 frames (the corresponding time length is 1 second). For each frame, the users are asked to record the number of objects they are interested in and the number of visual effects they are satisfied with, based on which the average accuracy of rendered information is calculated.

As shown in Fig.17, the average segmentation accuracy is 87%, which is lower than that on the datasets but still satisfying considering the more complicated scenarios and longer latency in such outdoor field. The average accuracy for the rendered information is 92% since the users tend to focus more on objects that are either large or in the central position in the frame which are well segmented and rendered by *edgeIS* and ignore the small ones. Besides, the false segmentation and rendering rate is 8% and 2% respectively, proving that *edgeIS* is well capable of providing helpful AR assists.

VII. Related Work

Instance Segmentation. Deep learning (DL) models have been well studied in recent years to perform accurate instance segmentation tasks. Typical methods usually follow the routine of generating pixel-level segmentation results within a proposed region [40], [41]. Mask R-CNN [17], one of the most representative models in this category, adds a branch based on Faster R-CNN [42] to perform segmentation in parallel with object detection. Pioneer studies such as PANet [43] and FCIS [41] make efforts to exploit information inside the detection box to improve segmentation accuracy. Apart from these phased solutions, there emerge end-to-end segmentation models that are free of region proposal schemes using techniques such as pixel clustering and coefficient regression [8], [44]. However, these models are usually less accurate than region-based ones and hard to decomposite, leaving little room for improvement when deployed for mobile applications. In *edgeIS*, we significantly reduce region-based models' latency with mobile side information while preserving their accuracy.

Edge Offloading. Offloading computational intensive tasks to powerful edge servers to enable complex mobile applications has been a popular research topic. The most important concern about various edge-assisted applications is the balance between the two sides and latency incurred by offloading [15], [27], [45]. In recent years, the "detect+track" framework that requires key information from the edge server and performs tracking locally on mobile devices has proved effective. Some methods [12], [46], [47] design mechanisms to decide whether to upload the content to the server or update the result locally. Chameleon [14] and VideoStorm [48] apply video configurations to achieve higher accuracy with the same amount of resources on the server. Besides, some previous systems focus on reducing transmission latency. The recent work of Liu *et al.* [11] introduces an edge-assisted system achieving high accuracy object detection and human keypoint detection task on existing AR/MR systems running at 60fps for both the object detection. EdgeDuet [10] further decomposite the offloading pipeline to tile-level for more efficient video transmission. While existing techniques mainly use the edge server to assist mobile devices, *edgeIS* leverages information unique on the mobile side to build a two-way instructional system to achieve better performance in our "transfer+infer" paradigm.

Visual Odometry. Visual odometry takes in consecutive frames to estimate device trajectory and construct a 3-D map of environment [49], [50]. Pioneer studies apply filtering-based approaches [51], which are replaced by optimization-based ones [52] for better accuracy and efficiency. ORB-SLAM [53], one of the most representative work, uses ORB features for tracking and mapping and achieves excellent performance. Using visual clues, visual odometry has the potential of combining with other vision-based methods to realize various applications [3]. *edgeIS* exploits the sensing ability of visual odometry and integrates it with accurate instance annotations to enable robust object tracking and segmentation on mobile devices.

VIII. Conclusion

In this work, we propose *edgeIS*, an instance segmentation solution for mobile devices, achieving accuracy and real-time simultaneously with the assistance of edge computation resources. Distinguished from existing "track+detect" edge-assisted architecture, we re-design and propose a fresh "transfer+infer" mobile-edge collaboration paradigm which achieves a two-way promotion of both sides by fully exploiting the motion and environmental sensing capabilities of mobile devices. We fully implement *edgeIS* on a resource-limited edge node and different types of mobile devices and extensively evaluate the performance on four datasets under different network conditions. The experimental results, as well as a field study in an industrial scenario, show that *edgeIS* achieves satisfying results in all scenarios.

Acknowledgment

We sincerely thank the anonymous reviewers for their helpful comments and advices. This work is supported in part by the NSFC under grant No. 61832010 and No. 61972131.

References

[1] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *International journal of multimedia information retrieval*, pp. 1–19, 2020.

[2] "Hololens," https://www.microsoft.com/en-us/hololens.

[3] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "Carmap: Fast 3d feature map updates for automobiles," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 1063–1081.

[4] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2013, pp. 221–234.

[5] J. Xu, G. Chi, Z. Yang, D. Li, Q. Zhang, Q. Ma, and X. Miao, "Followupar: Enabling follow-up effects in mobile ar applications," in *Proceedings of the ACM MobiSys*, June 24-July 2 2021.

[6] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[7] M. Gamal, M. Siam, and M. Abdel-Razek, "Shuffleseg: Real-time semantic segmentation network," *arXiv preprint arXiv:1803.03816*, 2018.

[8] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9157–9166.

[9] 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.

[10] X. Wang, Z. Yang, J. Wu, Y. Zhao, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," in *Proceedings of the IEEE INFOCOM*, 2021.

[11] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proceedings of the ACM Mobicom*, 2019.

[12] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the ACM Sensys*, 2015.

[13] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the ACM SIGCOMM*, 2020, pp. 557–570.

[14] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the ACM SIGCOMM*, 2018.

[15] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1828–1837.

[16] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[18] "Nvidia jetson tx2," https://developer.nvidia.com/embedded/jetson-tx2.

[19] "Nvidia jetson agx xavier," https://developer.nvidia.com/embedded/jetson-agx-xavier.

[20] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1. Ieee, 2004, pp. I–I.

[21] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[22] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 3, pp. 583–596, 2014.

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[25] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[26] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[27] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the ACM/IEEE Symposium on Edge Computing*, 2017.

[28] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in hevc," *IEEE journal of selected topics in signal processing*, vol. 7, no. 6, pp. 969–977, 2013.

[29] "Kvazaar," https://github.com/ultravideo/kvazaar.

[30] "Openhevc," https://github.com/OpenHEVC/openHEVC.

[31] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[32] "Orb-slam2," https://github.com/raulmur/ORB_SLAM2.

[33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014.

[34] "Pytorch mask r-cnn," https://github.com/multimodallearning/pytorch-mask-rcnn.

[35] S. Koranne, "Boost c++ libraries," in *Handbook of open source tools*. Springer, 2011, pp. 127–143.

[36] S. Caelles, J. Pont-Tuset, F. Perazzi, A. Montes, K.-K. Maninis, and L. Van Gool, "The 2019 davis challenge on vos: Unsupervised multi-object segmentation," *arXiv:1905.00737*, 2019.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[38] C. Montgomery and H. Lars, "Xiph. org video test media (derf's collection)," *URL: https://media. xiph. org/video/derf*, 1994.

[39] "Tensorflow lite," https://tensorflow.org/lite.

[40] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3150–3158.

[41] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2359–2367.

[42] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[43] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768.

[44] D. Neven, B. D. Brabandere, M. Proesmans, and L. V. Gool, "Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8837–8845.

[45] J. Xu, H. Cao, Z. Yang, L. Shangguan, J. Zhang, X. He, and Y. Liu, "Swarmmap: Scaling up real-time collaborative visual slam at the edge," in *Proceedings of the USENIX NSDI*, 2022.

[46] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "Marvel: Enabling mobile augmented reality with low energy and low latency," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 292–304.

[47] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proceedings of the IEEE INFOCOM*, 2018.

[48] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proceedings of the USENIX NSDI*, 2017.

[49] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proceedings of the IEEE ICCV*, 2003.

[50] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[51] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[52] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the IEEE ISMAR*, 2007.

[53] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.