Edge Assisted Real-time Object Detection for Mobile Augmented Reality

Luyang Liu
WINLAB, Rutgers University
North Brunswick, NJ, USA
luyang@winlab.rutgers.edu

Hongyu Li
WINLAB, Rutgers University
North Brunswick, NJ, USA
hongyuli@winlab.rutgers.edu

Marco Gruteser
WINLAB, Rutgers University
North Brunswick, NJ, USA
gruteser@winlab.rutgers.edu

ABSTRACT

Most existing Augmented Reality (AR) and Mixed Reality (MR) systems are able to understand the 3D geometry of the surroundings but lack the ability to detect and classify complex objects in the real world. Such capabilities can be enabled with deep Convolutional Neural Networks (CNN), but it remains difficult to execute large networks on mobile devices. Offloading object detection to the edge or cloud is also very challenging due to the stringent requirements on high detection accuracy and low end-to-end latency. The long latency of existing offloading techniques can significantly reduce the detection accuracy due to changes in the user's view. To address the problem, we design a system that enables high accuracy object detection for commodity AR/MR system running at 60fps. The system employs low latency offloading techniques, decouples the rendering pipeline from the offloading pipeline, and uses a fast object tracking method to maintain detection accuracy. The result shows that the system can improve the detection accuracy by 20.2%-34.8% for the object detection and human keypoint detection tasks, and only requires 2.24ms latency for object tracking on the AR device. Thus, the system leaves more time and computational resources to render virtual elements for the next frame and enables higher quality AR/MR experiences.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computer systems organization** → *Real-time system architecture*;

MobiCom '19, October 21–25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6169-9/19/10…$15.00
https://doi.org/10.1145/3300061.3300116

KEYWORDS

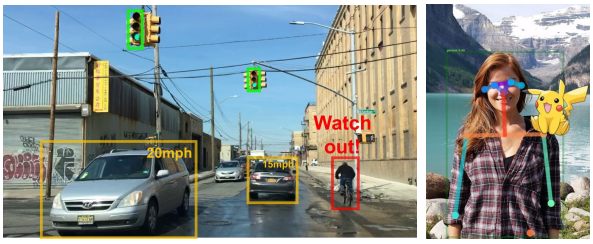
Edge Computing, Mobile Augmented Reality, Real-time Object Detection, Convolutional Neural Network, Adaptive Video Streaming

ACM Reference format:
Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *Proceedings of The 25th Annual International Conference on Mobile Computing and Networking, Los Cabos, Mexico, October 21–25, 2019 (MobiCom '19),* 16 pages.
https://doi.org/10.1145/3300061.3300116

1 INTRODUCTION

Augmented Reality, and in particular Mixed Reality systems, promise to provide unprecedented immersive experiences in the fields of entertainment, education, and healthcare. These systems enhance the real world by rendering virtual overlays on the user's field of view based on their understanding of the surroundings through the camera. Existing AR headsets further promise to support an unprecedented immersive experience called Mix Reality. Compared to tradition AR system, MR requires the system to have a comprehensive understanding of different objects and instances in the real world, as well as more computation resources for rendering high quality elements. Reports forecast that 99 million AR/VR devices will be shipped in 2021 [1], and that the market will reach 108 billion dollars [2] by then.

Existing mobile AR solutions such as ARKit and ARCore enable surface detection and object pinning on smartphones, while more sophisticated AR headsets such as Microsoft HoloLens [3] and the announced Magic Leap One [4] are able to understand the 3D geometry of the surroundings and render virtual overlays at 60fps. However, most existing AR systems can detect surfaces but lack the ability to detect and classify complex objects in the real world, which is essential for many new AR and MR applications. As illustrated in Figure 1, detecting surrounding vehicles or pedestrians can help warn a driver when facing potentially dangerous situations. Detecting human body key points and facial landmarks allow render virtual overlays on the human body, such as a virtual mask on the face or a Pikachu sitting on the shoulder. Such



(a) Dangerous traffic warning.

(b) Pickachu sits on her shoulder.

Figure 1: New AR applications supported by object detection algorithms.

capabilities could be enabled with CNN, who have shown superior performance in the object detection task [5], but it remains difficult to execute large networks on mobile devices with low latency, for example, TensorFlow Lite [6] requires more than one second to execute an accurate CNN model (e.g. ResNet Faster RCNN model) on one single frame.

Offloading object detection to the edge or cloud is also very challenging due to the stringent requirements on high detection accuracy and low end-to-end latency. High quality AR devices require the system to not only successfully classify the object, but also localize the object with high accuracy. Even detection latencies of less than 100ms can therefore significantly reduce the detection accuracy due to changes in the user's view—the frame locations where the object was originally detected may no longer match the current location of the object. In addition, as mixed reality graphics approach the complexity of VR, one can also expect them to require less than 20ms motion-to-photon latency, which has been found to be necessary to avoid causing user motion sickness in VR applications [7]. Furthermore, compared to traditional AR that only renders simple annotations, mixed reality requires rendering virtual elements in much higher quality, which leaves less latency budget for the object detection task.

Most existing research has focused on enabling high frame rate object detection on mobile devices but does not consider these end-to-end latency requirements for high quality AR and mixed reality systems. Glimpse [8] achieves 30fps object detection on a smartphone by offload trigger frames to the cloud server, and tracks the detected bounding boxes on remaining frames locally on the mobile devices. DeepDecision [9] designs a framework to decide whether to offload the object detection task to the edge cloud or do local inference based on current network conditions. However, they all require more than 400ms offloading latency and also require large amounts of local computation, which leaves little resources to render high-quality virtual overlays. No prior work, can achieve high detection accuracy in moving scenarios or finish the entire detection and rendering pipeline under 20ms.

To achieve this, we propose a system that significantly reduces the offloading detection latency and hides the remaining latency with an on-device fast object tracking method. To reduce offloading latency, it employs a *Dynamic RoI Encoding* technique and a *Parallel Streaming and Inference* technique. The *Dynamic RoI Encoding* technique adjusts the encoding quality on each frame to reduce the transmission latency based on the Regions of Interest (RoIs) detected in the last offloaded frame. The key innovation lies in identifying the regions with potential objects of interest from candidate regions on prior frames. It provides higher quality encodings in areas where objects are likely to be detected and uses stronger compression in other areas to save bandwidth and thereby reduce latency. The *Parallel Streaming and Inference* method pipelines the streaming and inference processes to further reduce the offloading latency. We propose a novel *Dependency Aware Inference* method to enable slice-based inference of CNN object detection models without affecting the detection result. On the AR device, the system decouples the rendering pipeline from the offloading pipeline instead of waiting for the detection result from the edge cloud for every frame. To allow this, it uses a fast and lightweight object tracking method based on the motion vector extracted from the encoded video frames and the cached object detection results from prior frames processed in the edge cloud to adjust the bounding boxes or key points on the current frame in the presence of motion. Taking advantage of the low offloading latency, we find this method can provide accurate object detection results and leave enough time and computation resources for the AR device to render high-quality virtual overlays. Besides, we also introduce an *Adaptive Offloading* technique to reduce the bandwidth and power consumption of our system by deciding whether to offload each frame to the edge cloud to process based on the changes of this frame compare to the previous offloaded frame.

Our system is able to achieve high accuracy object detection for existing AR/MR system running at 60fps for both the object detection and human keypoint detection task. We implement the end-to-end AR platform on commodity devices to evaluate our system. The results show that the system increases the detection accuracy by 20.2%-34.8%, and reduce the false detection rate by 27.0%-38.2% for the object detection and human keypoint detection tasks. And the system requires only 2.24ms latency and less than 15% resources on the AR device, which leaves the remaining time between frames to render high quality virtual elements for high quality AR/MR experience.

The contributions of this work can be summarized as follows:

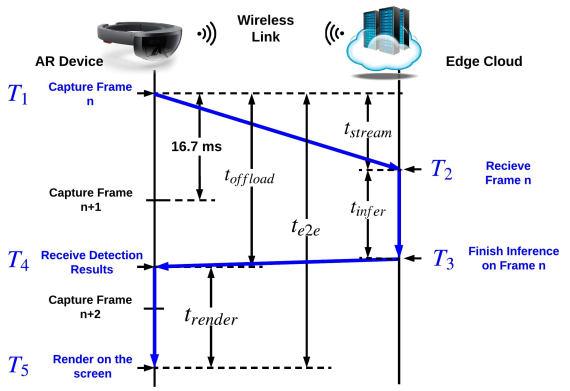


Figure 2: Latency Analysis.

- Quantifying accuracy and latency requirements in an end-to-end AR system with the object detection task offloaded.
- Proposing a framework with individual rendering and offloading pipelines.
- Designing a *Dynamic RoI Encoding* technique to dynamically determine the Regions of Interest in order to reduce the transmission latency and bandwidth consumption in the offloading pipeline.
- Developing a *Parallel Streaming and Inference* method to pipeline the streaming and inference processes to further reduce the offloading latency.
- Creating a *Motion Vector Based Object Tracking* technique to achieve fast and lightweight object tracking on the AR devices, based on the embedded motion vectors in the encoded video stream.
- Implementing and evaluating an end-to-end system based on commodity hardware and showing that the proposed system can achieve 60fps AR experience with accurate object detection.

2 CHALLENGES AND ANALYSIS

Offering sophisticated object detection in mobile augmented reality devices is challenging because the task is too computationally intensive to be executed on-device and too bandwidth intensive to be easily offloaded to the edge or cloud. Most lightweight object detection models require more than 500ms processing time on current high-end smartphones with GPU support. Even on a powerful mobile GPU SoCs (such as the Nvidia Tegra TX2 which is reportedly used on the Magic Leap One), object detection on an HD frame still takes more than 50ms. This is too long to process every frame on a 60Hz system and likely to lead to energy consumption and heat dissipation issues on the mobile device.

Latency Analysis. When offloading the detection tasks to more powerful edge or cloud platforms the image encoding and transfer steps add significant latency. Longer latency not only reduces the detection accuracy but also degrades the AR experience. To better understand these challenges, we model the end-to-end latency of a baseline AR solution with offloading as follows:

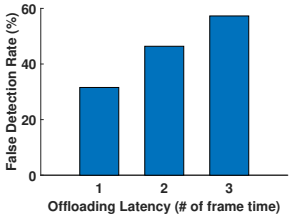
$$\begin{aligned} t_{e2e} &= t_{offload} + t_{render} \\ t_{offload} &= t_{stream} + t_{infer} + t_{trans\_back} \\ t_{stream} &= t_{encode} + t_{trans} + t_{decode} \end{aligned} \tag{1}$$

As shown in Figure 2, the AR device (i.e. smartphone or AR headset) is assumed to be connected to an edge cloud through a wireless connection (i.e. WiFi or LTE). The blue arrow illustrates the critical path for a single frame. Let t_{e2e} be the end-to-end latency, which includes the offloading latency $t_{offload}$ and the rendering latency t_{render}. $t_{offload}$ is determined by three main components: (1) the time to stream a frame captured by the camera from the AR device to the edge cloud $t_{stream} = T_2 - T_1$, (2) the time to execute the object detection inference on the frame at the edge cloud $t_{infer} = T_3 - T_2$, and (3) the time to transmit the detection results back to the AR device $t_{trans\_back} = T_4 - T_3$. To reduce the bandwidth consumption and streaming latency t_{stream}, the raw frames are compressed to H.264 video streams on the device and decoded in the edge cloud [10]. Therefore, t_{stream} itself consists of encoding latency (t_{encode}), transmission latency (t_{trans}) and decoding latency (t_{decode}).

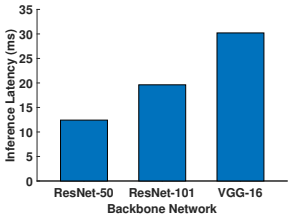
We conduct an experiment to measure the latency and its impact on detection accuracy in the entire pipeline, and find that it is extremely challenging for existing AR system to achieve high object detection accuracy in 60fps display systems. In the experiment, we connect a Nvidia Jetson TX2 to an edge cloud through two different WiFi protocols (WiFi-5GHz, WiFi-2.4GHz) and stream encoded frames of a video [11] at 1280x720 resolution from the Jetson to the edge cloud for inference. The edge cloud is a powerful PC equipped with a Nvidia Titan Xp GPU.

Detection Accuracy Metrics. To evaluate the detection accuracy in terms of both object classification and localization, we calculate the IoU of each detected bounding box and its ground truth as the accuracy of this detection. We also define the percentage of detected bounding boxes with less than 0.75 IoU [12] (the strict detection metric used in the object detection task) as false detection rate. Similarly, we use the Object Keypoint Similarity (OKS) [13] metric to measure the accuracy of each group of keypoints in the human keypoint detection task.

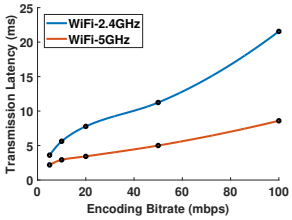
We find that low latency object detection is highly beneficial for achieving a high detection accuracy. Figure 3(a) shows the impact of $t_{offload}$ on the false detection rate. We



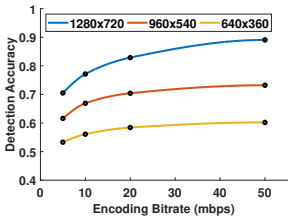
(a) False detection rate with respect to different offloading latency.



(b) Inference latency of Faster R-CNN for different backbone CNN networks.



(c) Transmission latency of different encoding bit rate in two different WiFi networks.



(d) Encoding Bitrate vs Inference Accuracy for two different resolutions.

Figure 3: Latency and accuracy analysis.

can find that even a latency of a frame time (16.7ms) will increase the false detection rate from 0% to 31.56%. This is because during the time that the detection result is sent back to the AR device, the user's view may have changed due to user motion or scene motion.

However, it is very challenging to achieve very low latency object detection with commodity infrastructures. We first measure the latency spend on inference (t_{infer}), and show the result in Figure 3(b). To push the limit of t_{infer} on the edge cloud, we use TensorRT [14] to optimize three pre-trained Faster R-CNN models[1] using INT8 precision. These three models use three different backbone CNN networks (ResNet-50, ResNet-101, and VGG-16) for feature extraction. As shown in Figure 3(b), we can observe that all three models require more than 10ms for object detection.

Figure 3(c) shows the additional latency imposed by transmitting a single HD frame with different encoding bitrate from the AR device to the edge cloud (t_{trans}) through two different WiFi connections (WiFi-2.4GHz and WiFi-5GHz). Here, bitrate is a codec parameter that determines the quality of video encoding. Encoding with small bitrate will result in a lossy frame after decoded. We can observe that the average t_{trans} requires to transmit an encoded frame with 50mbps bitrate is 5.0ms on 5GHz WiFi and 11.2ms on 2.4GHz WiFi.

[1]We choose Faster R-CNN because it is much more accurate than other alternatives, such as SSD and R-FCN.

Inference plus transmission latency therefore already exceeds the display time for one frame. One may think that decreasing resolution or encoding bitrate may reduce the transmission latency, however, this also reduces the detection accuracy of an object detection model.

To validate this issue, we show the detection accuracy of the ResNet-50 based Faster R-CNN model under different encoding bitrate and resolution in Figure 3(d). In this case, we use the detection result on raw video frames (without video compression) as the ground truth to calculate the IoU. The result shows that it requires at least 50Mbps encoding bitrate to achieve a high detection accuracy (i.e. 90). We also compare the detection result on two lower resolution frames (960x540 and 640x320), and show that lower resolution has much worse detection accuracy than the original 1280x720 frame. Lowering resolution therefore also does not improve detection accuracy. Note that this accuracy drop can be stacked together with the drop caused by the long offloading latency to get a much lower detection accuracy.

Based on the above analysis, we find that it is extremely challenging for existing AR system to achieve high object detection accuracy in 60fps display systems. This can lead to poor alignment of complex rendered objects with physical objects or persons in the scene.

3 SYSTEM ARCHITECTURE

To overcome these limitations, we propose a system that is able to achieve high accuracy object detection with little overhead on the rendering pipeline of mobile augmented reality platforms, by reducing the detection latency with low latency offloading techniques and hiding the remaining latency with an on-device fast object tracking method. Figure 4 shows the architecture of our proposed system. At a high level, the system has two parts connected through a wireless link: a local tracking and rendering system on a mobile device (a smartphone or an AR headset) and a pipelined objected detection system on the edge cloud. To hide the latency caused by offloading the object detection task, our system decouples the rendering process and the CNN offloading process into two separate pipelines. The local rendering pipeline starts to track the scene and render virtual overlays while waiting for object detection results, and then incorporates the detection results into the tracking for the next frame when they arrive.

As shown in Figure 4, both pipelines start with a *Dynamic RoI Encoding* technique that not only compresses raw frames for the CNN offloading pipeline (yellow arrow), but also provides its meta data for the on-device tracking module in the tracking and rendering pipeline (green arrow). *Dynamic RoI Encoding* is an efficient video encoding mechanism that is able to largely reduce the bandwidth consumption and thereby reduce the transmission latency to the edge

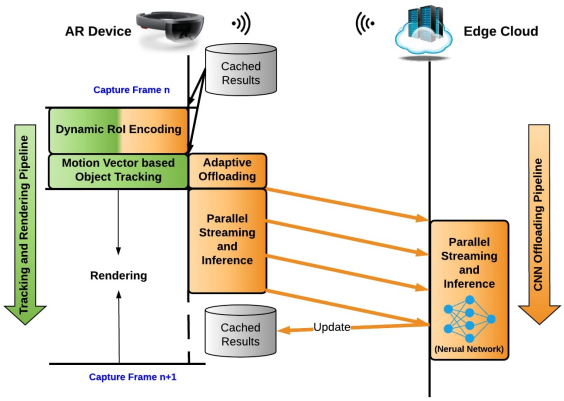


Figure 4: System Architecture.

cloud, while maintaining detection accuracy. The key idea of *Dynamic RoI Encoding (DRE)* is to decrease the encoding quality of uninteresting areas in a frame and to maintain high quality for candidate areas that may contain objects of interest based on earlier object detection results. Due to the spatiotemporal correlation over subsequent video frames, the system uses the intermediate inference output of the last offloaded frame as candidate areas. These candidate areas are where it maintains high encoding quality and are also referred to as regions of interest (RoIs).

In the CNN offloading pipeline as illustrated by the yellow blocks and arrow, we propose an *Adaptive Offloading* and a *Parallel Streaming and Inference (PSI)* technique to further reduce the latency and bandwidth consumption of the offloading task.

Adaptive Offloading is able to reduce the bandwidth and power consumption of our system by deciding whether to offload each frame to the edge cloud based on whether there are significant changes compared to the previous offloaded frame. For efficiency, this technique reuses the macroblock type (inter-predicted blocks or intra-predicted blocks) embedded in the encoded video frame from the *Dynamic RoI Encoding* to identify significant changes that warrant offloading for object detection.

Once the frame is marked for offloading, the *Parallel Streaming and Inference (PSI)* method parallelizes the transmission, decoding and inference tasks to further reduce the offloading latency. It splits a frame into slices and starts the convolutional neural network object detection task as soon as a slice is received, rather than waiting for the entire frame. This means that reception, decoding, and object detection can proceed in parallel. To solve the dependency issues across slices during object detection, we introduce a novel *Dependency Aware Inference* mechanism that determines the region on each feature map that has enough input features to calculate after each slice is received, and only calculates features lie in this region. The detection results are sent back to the AR device and cached for future use.

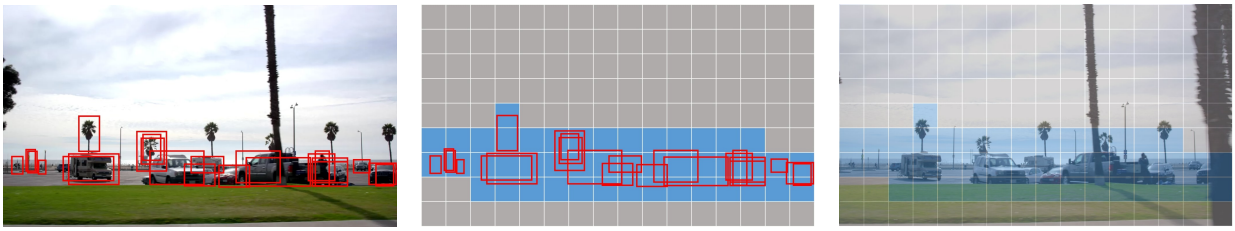
In the tracking and rendering pipeline (green blocks and arrow in Figure 4), instead of waiting for the next detection result, we use a fast and light-weight *Motion Vector based Object Tracking (MvOT)* technique to adjust the prior cached detection results with viewer or scene motion. Compared to traditional object tracking approaches that match image feature points (i.e. SIFT and Optical Flow) on two frames, this technique again reuses motion vectors embedded in the encoded video frames, which allows object tracking without any extra processing overhead. Given the aforementioned optimizations to reduce offloading latency, tracking is needed only for shorter time frames and a lightweight method can provide sufficiently accurate results. Using such a lightweight method leaves enough time and computational resources for rendering on the device, in particular to render high-quality virtual overlays within the 16.7ms (for 60Hz screen refresh rate) latency requirement.

4 DYNAMIC ROI ENCODING

Dynamic RoI Encoding reduces the transmission latency of the offloading pipeline while maintaining a high object detection accuracy. Transmitting the frames with high visual quality from the mobile to the edge/cloud leads to a high bandwidth consumption and thereby transmission latency. *Dynamic RoI Encoding* selectively applies higher degrees of compression to parts of the frame that are less likely to contain objects of interest and maintains high quality in regions with candidate objects. This largely reduces the size of encoded frames with only a small tradeoff in object detection accuracy. The key lies in identifying the regions with potential objects of interest, which we will refer to as regions of interest. The design exploits candidate regions that have been generated internally by the convolutional neural network on prior frames. Note that *Dynamic RoI Encoding* leverages the existing RoI encoding technique that is widely used in video streaming standards but adds a novel, effective mechanism to dynamically determine the RoIs for each frame.

4.1 Preliminaries

RoI Encoding. While the building block of RoI encoding has been used in other applications, current methods to select regions of interest are not suitable for this augmented reality object detection task. RoI encoding is already supported by most video encoding platform, which allows the user to adjust the encoding quality (i.e. Quantization Parameter - QP) for each macroblock in a frame. It has been largely adopted



(a) Detect RoIs on the last offloaded frame. (b) Mark macroblocks that overlap with RoIs. (c) Change encoding quality on the current frame.

Figure 5: Three main procedures of RoI encoding.

in surveillance camera video streaming and 360-degree video streaming, where the RoIs are pre-defined or much easier to predict based on user's field of view. For example, the RoI can be derived as the area that a user chooses to look at. This region would then receive near-lossless compression to maintain quality while lossier compression is used for the background or non-RoI area. Augmented reality includes use cases that should draw users attention to other areas of the view and therefore regions of interest cannot just be based on the current objects a user focuses on.

Object Detection CNNs. Due to impressive performance gains of state-of-the-art object detection is largely based on CNN. While several networks exist (e.g., Faster-RCNN, Mask-RCNN), they share a similar architecture, which firstly utilizes a CNN network to extract the features of the input image, then internally propose candidate regions (also called regions of interest) and their corresponding possibilities through a region proposal network, and finally perform and refine the object classification. The CNN network is also called backbone network and there are multiple options for its implementation, including VGG, ResNet, and Inception. The region proposal network usually generates hundreds of regions of interest which are potential objects locations in the frame.

Note that the term **RoIs** is used both in object detection and video compression. For the object detection task, RoIs are usually the output proposals of the region proposal network. While in the field of video compression, RoIs are the areas inside video frames that may contain more visual information and will be encoded with fewer losses. This presents an opportunity to exploit this similarity and tie these concepts together.

4.2 Design

In order to reduce the bandwidth consumption and data transmission delay, we design a dynamic RoI encoding mechanism that links internal RoI generated in the object detection CNNs to the image encoder. Specifically, it uses the CNN candidate RoIs generated on the last processed frame for determining encoding quality on the next camera frame. It accommodates a degree of motion by slightly enlarging each region of interest by one macroblock but largely benefits from the similarity between two frames captured a short moment apart in time. While one may expect that even greater bandwidth savings are possible by choosing RoIs only in areas where object were detected on the previous frame, this approach frequently misses new objects that appear in the scene because the image areas containing these objects end up too heavily compressed. Changes in such a heavily compressed area, however, are often still identified as part of the much larger set of candidate RoIs of the CNN, the outputs of the region proposal network. We therefore use the RoIs from the region proposal network, filtered with a low minimum prediction confidence threshold (i.e., 0.02). A sample output of our RoI detection method is shown in Figure 5(a).

In order to use these selected RoIs to adjust the encoding quality on the current frame, we calculate a QP map that defines the encoding quality (QP) for each macroblock on the frame. The QP map indicates for each macroblock whether it overlaps with any RoI. In the example in Figure 5(b), all overlapping macroblocks are marked in blue and non-overlapping ones in grey. Since object detection is offloaded to the edge, cloud the object detection pipeline sends this QP map back to the AR device, which uses it for the next captured frame. As shown in Figure 5(c), the encoder applies lossy compression on those non-overlapping (grey) regions, while maintaining high visual quality on overlapping (blue) regions.[2] Specifically, our implementation reduces the QP value by 5 for lossy encoding.

5 PARALLEL STREAMING AND INFERENCE

We offload the heavy deep neural network computation to the edge cloud. This requires transmitting the camera frames

[2]Note that Figure 5(b) and 5(c) uses a grid of 16x9 macroblocks for illustration purposes. In the H.264 standard, a macroblock is usually 16x16 pixels, so a 1280x720 resolution frame has 80x45 macroblocks.

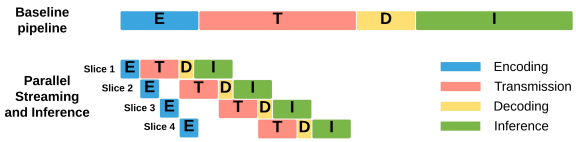


Figure 6: Parallel Streaming and Inference.

from the mobile side to the edge cloud. Conventional architectures, however, can only start the object detection process when the entire frame is received, as the deep neural networks are designed with neighborhood dependency. This will add to the latency, since both the streaming and the inference process take considerable time and run sequentially, as discussed in section 2. To mitigate this long latency, we propose a *Parallel Streaming and Inference* technique which enables inferences on slices of a frame, so that the streaming and inference can be effectively pipelined and executed in parallel. Since streaming and inference consume different resources that do not affect each other: transmission consumes bandwidth on the wireless link, decoding uses edge cloud hardware decoders, and the neural network inference mainly consumes GPUs or FPGAs resources on the edge cloud, this technique effectively use multiple resources to execute different tasks in parallel, which can significantly reduce the latency.

The challenge for deep neural networks to execute on a slice of frame is the dependency among inputs, which is caused by the neuron operations that take neighborhood values as input. To address this problem, we propose **Dependency Aware Inference** to automatically analyze the dependencies of each layer, and only infer on the regions which have enough neighbor values. Figure 6 shows how the Parallel Streaming and Inference method reduces the offloading latency. Compared with encoding and inference on the entire frame, we encode the whole image into multiple slices, each slice will be sent to the edge cloud immediately after it is encoded. The edge cloud will start to infer once it receives and decodes the first slice of the frame.

5.1 Dependency Aware Inference

Due to the computational dependency among neighbor values of the input frame, simply running inference and then merging based on slices of a frame will cause significant wrong feature values near boundaries. To solve this problem, we design a *Dependency Aware Inference* technique which only calculates the regions of feature points in each layer with enough input feature points available. Dependencies are caused by the convolutional layers (as well as pooling layers sometimes), where the feature computation around the boundary of each frame slice requires also adjacent slices.

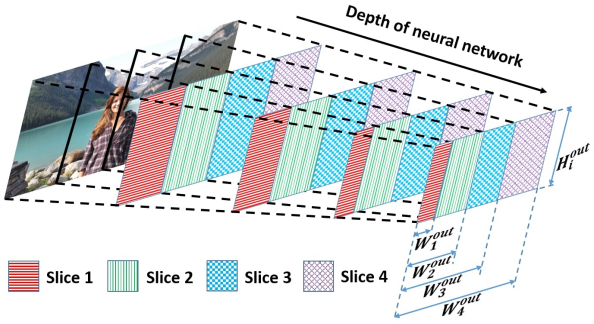


Figure 7: Dependency Aware Inference.

This effect propagates for the standard convolutional layers and pooling layers structure. We experimentally find that the boundary feature computation of the last convolutional layer on VGG-16, Resnet-50, and Resnet-101, requires 96, 120, 240 pixels respectively. One naive solution for parallelizing inference is to recompute such regions when the next slice arrives at the edge cloud. However, this requires significant extra computations for every convolutional layer, which inflates the inference latency.

To solve this dependency issue, we calculate the size of the *valid region* for the output feature map of each layer, and only infer based on valid regions. Valid regions are defined as the areas of each convolutional feature map that have enough input features available and their sizes can be determined in equation 2.

$$H_i^{out} = (H_i^{in} - 1)/S + 1$$
$$W_i^{out} = \begin{cases} \frac{W_i^{in} - (F-1)/2 - 1}{S} + 1, & i = 1, 2, ..., n-1 \\ \frac{W_i^{in} - 1}{S} + 1, & i = n \end{cases} \quad (2)$$

H_i^{out} and W_i^{out} are the height and width of *valid region* of the output feature map of a convolutional layer after slice i arrives at the edge cloud (i is the number of slice, n is the number of slices we divided.). Similarly, H_i^{in} and W_i^{in} are the *valid region* on the input feature map of this convolutional layer. We also define the spatial extent and stride of this conv layer as F and S correspondingly[3]. Note that we empirically set n to 4 in our system to archive a balance between transmission and inference.

Figure 7 illustrates the concept of the *Dependency Aware Inference* technique. Since our system cuts the whole frame into 4 slices with 1/4 of the original width, H_i^{out} of one conv layer is constant and only affected by H_i^{in} and S as shown in the first equation, while W_i^{out} keeps increasing as more

[3]Note that we assume the number of zero padding of a conv layer is equal to $(F-1)/2$ in most cases.

(a) Cached detection result of the last offloaded frame.

(b) Motion vectors extracted from the current encoded frame.

(c) Shift the bounding box based on the motion vectors.

Figure 8: Main procedures of RoI encoding.

slices arrive at the edge cloud. For example, in the case of a standard 3x3 convolutional layer with stride 1, we will not calculate the very right column of features for slice 1,2 and 3, due to those features requiring inputs from the next slice of the frame. As shown in Figure 7, our system only calculates the red regions in each conv layer after slice 1 arrives at the edge cloud. As more slices arrive, the *valid region* keeps increasing on each feature map, and the system continuously calculates those new features included in the *valid region*. We can observe that the number of features that can be calculated for slice 1 keeps decreasing as the network goes deeper. Slice 2 and 3 are able to compute more features than slice 1, and all the remaining features will be calculated after slice 4 arrives. Note that we also defined similar logic to process pooling layers, which will not calculate the rightmost column in the output feature map for slice 1,2 and 3 if the input feature map is an odd number.

6 MOTION VECTORS BASED OBJECT TRACKING

In this section, we introduce Motion Vector Based Object Tracking that is able to estimate the object detection result of the current frame using the motion vector extracted from the encoded video frames and the cached object detection result from the last offloaded frame.

Motion vectors are broadly used by modern video encoding approaches (e.g. H.264 and H.265) to indicate the offset of pixels among frames to achieve a higher compression rate. Commodity mobile devices are usually equipped with specific hardware to accelerate video encoding and compute the motion vectors. Figure 8 shows the key steps of the Motion Vector based Fast Object Tracking technique. For each new frame captured by the camera, the system passes the frame to the *Dynamic RoI Encoding* session. The encoder uses the frame corresponding to the last cached detection result (Figure 8(a)) as its reference frame for inter-frame compression. After that, the system extracts all motion vectors from the encoded frame, as illustrated in Figure 8(b). To track the object in the current frame, we get the bounding box of this object in the last offloaded frame, calculate the mean of all motion vectors that reside in the bounding box, and use it to shift the old position (in blue) to the current position (in yellow), as illustrated in Figure 8(c). Similarly, we also apply this technique to the human keypoint detection task, in which we calculate the mean motion vector in the closest 9x9 macroblock region of each keypoint, and use it to shift each keypoint.

In our experiment, we find that the accuracy of the motion vector decreases as the time interval between the current frame and reference frame increases. However, due to the low offloading latency achieved by the proposed latency optimization techniques, we found that this method can provide accurate object detection results with very short latency. The system we implemented on Nvidia Jetson TX2 requires only 2.24ms for this motion tracking process, which leaves enough time and computation resources for the AR device to render high-quality virtual overlays within the 16.7ms latency requirement. Note that this technique cannot hide the latency to first detection of an object. Since this is already well under the response time that human observers notice, this technique focuses on accurate tracking so that virtual objects can follow the motion of physical ones.

7 ADAPTIVE OFFLOADING

To effectively schedule the offloading pipeline, we propose an *Adaptive Offloading* mechanism to determine which encoded frame should be offloaded to the edge cloud. The *Adaptive Offloading* mechanism is designed based on two principles: (1) a frame will only be eligible to be offloaded if the previous offloaded frame has been completely received by the edge cloud, (2) a frame will be considered for offloading if it differs significantly from the last offloaded frame. The first principle eliminates frames queuing up to avoid network congestion, while the second principle ensures that only necessary views

with enough changes will be offloaded to minimize communication and computing costs. Therefore, if a frame satisfies both principles, it will be offloaded to the edge cloud.

The first principle requires the system to be aware of the transmission latency of previous offloaded frames. The edge cloud therefore signals the AR device once it receives the last slice of the offloaded frame. Based on this time difference between the reception time and the transmission time, the AR calculates the transmission latency and uses it to decide whether to offload the next encoded frame.

To fulfill the second principle, it is necessary to estimate the differences between two frames. We evaluate such differences from two perspectives with either of them satisfying the second principle: (1) whether large motions (including both user's motion and objects' motion) occur among the frames, (2) whether there are considerable amounts of changed pixels appearing in the frame. The motion of a frame is quantified by the sum of all the motion vectors, and the number of new pixels is estimated by the number of intra-predicted macroblocks within an encoded frame. Between the two types of macroblocks (inter-predicted block and intra-predicted block) within an encoded frame, we experimentally find that intra-predicted macroblocks usually refer to newly appeared regions, since these macroblocks fail to find reference pixel blocks in the reference frame during encoding.

8 IMPLEMENTATION

Our implementation is entirely based on commodity hardware and consists of around 4000 lines of code.

8.1 Hardware Setup

In the hardware setup, we use a mobile development board Nvidia Jetson TX2 as the AR device, which contains the same mobile SoC (Tegra TX2) as the Magic Leap One AR glass. The Jetson board is connected to a TP-Link AC1900 router through a WiFi connection. We emulate an edge cloud with a PC equipped with an Intel i7-6850K CPU and a Nvidia Titan XP GPU, which connects to router through a 1Gbps Ethernet cable. Both the AR device and the desktop PC run an Ubuntu 16.04 OS.

8.2 Software Implementation

We implement our proposed techniques based on Nvidia JetPack[15], Nvidia Multimedia API [16], Nvidia TensorRT [14], and the Nvidia Video Codec SDK [17].

Client Side. We implement the client side functions on the Nvidia Jetson TX2 with its JetPack SDK. The implementation follows the design flow in Figure 4. We first create a camera capture session running at 60fps using the JetPack Camera API, and register a video encoder as its frame consumer using the Multimedia API. To realize the RoI encoding module, we use the `setROIParams()` function to set the RoIs and their QP delta value for encoding the next frame, based on the RoIs generated on the edge cloud. We also enable the external RPS control mode to set the reference frame of each frame to the source frame of the current cached detection results, so that the extracted Motion Vectors can be used to shift the cached detection results. To implement the Parallel Streaming and Inference module, we enable the slice mode for the video encoder and use the `setSliceLength()` function with a proper length to let the encoder split a frame into four slices. After frame slices are encoded, the system extracts motion vectors and macroblock types from each slice using the `getMetadata()` function. This information is used as the input for Adaptive Offloading and MvOT in two different threads (Rendering thread and offloading thread). In the offloading thread, if the Adaptive Offloading module decides to offload this frame, its four slices will be sent out to the server through the wireless link one by one. In the rendering thread, the Motion Vector based Object Tracking module uses the extracted motion vectors and cached detection results to achieve fast object tracking. The system then renders virtual overlays based on the coordinates of the detection result.

Server Side. The server side implementation contains two main modules: Parallel Decoding and Parallel Inference, which are designed to run in two different threads to avoid blocking each other. In the Parallel Decoding thread, the system keeps waiting for the encoded frame slices from the AR device. Once a slice is received, it immediately passes it to the video decoder for decoding in asynchronous mode, which won't block the system to continue receiving other slices. We use Nvidia Video Codec SDK to take advantage of the hardware accelerated video decoder in the Nvidia Titan Xp GPU. After each slice is decoded, the system passes it to the parallel inference thread in a callback function attached to the decoder. The Parallel Inference module is implemented using the Nvidia TensorRT, which is a high-performance deep learning inference optimizer designed for Nvidia GPUs. To push the limit of inference latency on the server side PC, we use the INT8 calibration tool [18] in TensorRT to optimize the object detection model, and achieves 3-4 times latency improvement on the same setup. To achieve the proposed *Dependency Aware Inference* method, we add a `PluginLayer` before each convolutional layer and pooling layer to adjust their input and output regions based on Equation 2. After the inference process of a whole frame, the edge cloud sends the detection results as well as the QP map back to the AR device for future processing.

9 EVALUATION

In this section, we evaluate the performance of the system in terms of detection accuracy, detection latency, end-to-end tracking and rendering latency, offloading latency, bandwidth consumption, and resource consumption. The results demonstrate that our system is able to achieve both the high accuracy and the low latency requirement for AR headsets and hand-held AR system running at 60fps, under different network background traffic loads. The result shows that the system increases the detection accuracy by 20.2%-34.8%, and reduce the false detection rate by 27.0%-38.2% for the object detection and human keypoint detection tasks, respectively. To achieve this high accuracy, the system reduces the offloading latency by 32.4%-50.1% and requires only an average of 2.24ms to run the MvOT method on the AR device, which leaves the remaining time between frames to render high quality virtual elements.

9.1 Experiment Setup

We use the setup and implementation described in Section 8 to conduct experiments. Two different detection tasks are designed to evaluate the performance of our system: an object detection task and a keypoint detection task. Both of them follow the flow in Figure 4. In the first task, the edge cloud runs a Faster R-CNN object detection model with ResNet-50 to generate bounding boxes of objects for each offloaded frame. In the second task, the edge cloud runs a Keypoint Detection Mask R-CNN model with ResNet-50 to detect the human body keypoints. Based on the detection result, the AR device renders a complex 3D cube on the user's left hand, as shown in Figure 10. Both detection tasks run local object tracking and rending at 60fps on the AR device. We use two different WiFi connections (2.4GHz and 5GHz) as the wireless link between the AR device and the edge cloud. The bandwidths measured with `iperf3` are 82.8Mbps and 276Mbps correspondingly. Compared to the first task, the second task incurs higher rendering loads on the AR device.

For repeatable experiments, we extract raw YUV frames at 1280x720 resolution from ten videos[4] in the Xiph video dataset [19] as the camera input for evaluation. In total, 5651 frames have been processed in our evaluation. Note that we use pre-recorded videos instead of the raw camera feed because the pre-recorded video frames usually contains complex scenes with multiple objects and different camera motions, which are much more challenging than normal camera feed. The experiments strictly follow the same work flow as shown in Figure 4 running in real-time, without any pre-encoding or profiling on each frame.

[4]DrivingPOV, RollerCoaster, BarScene, FoodMarket, and SquareAndTimelapse for object detection task. Crosswalk, BoxingPractice, Narrator, FoodMarket, as well as SquareAndTimelapse for the human keypoint detection task.

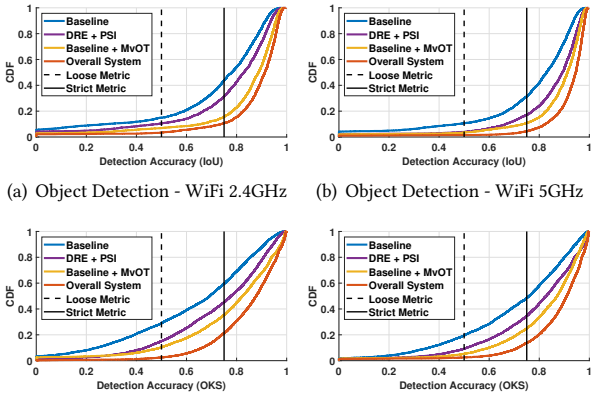
Detection Model	Approaches	WiFi 2.4GHz	WiFi 5GHz
Faster R-CNN Object Detection	Baseline	0.700	0.758
	DRE + PSI	0.758	0.837
	MvOT only	0.825	0.864
	Overall System	0.864	0.911
Mask R-CNN Keypoint Detection	Baseline	0.6247	0.6964
	DRE + PSI	0.7232	0.7761
	MvOT only	0.7667	0.8146
	Overall System	0.8418	0.8677

Table 1: Mean Detection Accuracy (IoU/OKS) of two models with two WiFi connections.

9.2 Object Detection Accuracy

Our system is able to achieve high detection accuracy and low false detection rate under various network conditions. We first measure the object detection accuracy in four approaches: the baseline solution (Baseline), our solution with only the two latency optimization techniques (DRE + PSI), our solution with only the client side motion vector based object tracking method (Baseline + MvOT), and our overall system with all three techniques (DRE + PSI + MvOT). The baseline approach follows the standard pipeline we introduced in Section 2. We evaluate the detection accuracy of our system with two key metrics: mean detection accuracy and false detection rate. Specifically, we feed extracted frames of each video to the client side video encoder at 60fps to emulate a camera but allow experiments with repeatable motion in the video frames. To calculate the detection accuracy for each frame, we calculate the mean Intersection over Union (IoU) or Object Keypoint Similarity (OKS) between the detection result from the MvOT and the ground truth detection result of each frame (without frame compression and emulating no latency). Recall that IoU is 0 when the detected object labels do not match (e.g., vehicle vs pedestrian) and otherwise represent the degree of position similarity within the frame. More precisely, it is the intersection area over the union area of the detection bounding box and ground truth bounding box. Similar to IoU, OKS also varies from 0 to 1, describing the normalized Euclidean distances between detected positions of keypoints and groundtruth labels. In the experiment, we connect the server and the client devices through two WiFi connections: WiFi-2.4GHz and WiFi-5GHz.

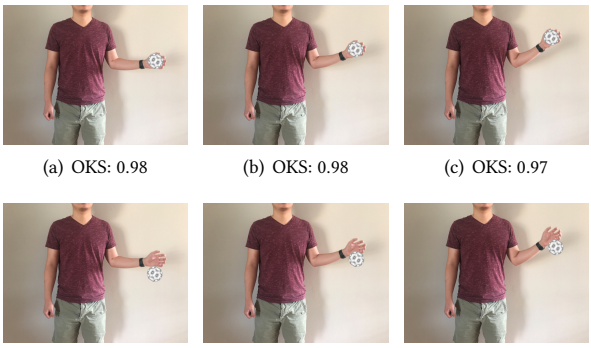
Table 1 shows the mean detection accuracy of two models with two different WiFi connections. In the object detection case, we can observe that our system achieves a 23.4%



(a) Object Detection - WiFi 2.4GHz (b) Object Detection - WiFi 5GHz

(c) Keypoint Detection - WiFi 2.4GHz (d) Keypoint Detection - WiFi 5GHz

Figure 9: CDF of detection accuracy (IoU/OKS) for object detection and keypoint detection task.



(a) OKS: 0.98 (b) OKS: 0.98 (c) OKS: 0.97

(d) OKS: 0.83 (e) OKS: 0.76 (f) OKS: 0.73

Figure 10: (a)-(c) Rendering results based on our system. (d)-(f) Rendering results based on the baseline approach.

improvement for the WiFi-2.4GHz connection and a 20.2% improvement for the WiFi-5GHz connection. In the human keypoint detection case, our system achieves a 34.8% improvement for WiFi-2.4GHz and a 24.6% improvement for WiFi-5GHz. The results also show that the three main techniques (DRE, PSI, and MvOT) are able to effectively increase the detection accuracy of the system. By comparing the DRE + PSI approach with the Baseline approach, we find that the low latency offloading solution helps to achieve high detection accuracy. By comparing the Baseline + MvOT with the Baseline approach, we also see that our fast object tracking technique increases accuracy. The gains of these two approaches accumulate in the overall system accuracy.

In addition, we show the CDF of the measured detection accuracy results in Figure 9. To determine acceptable detection accuracy, we adopt two widely used thresholds in the computer vision community: 0.5 as a loose accuracy threshold and 0.75 as the strict accuracy threshold [20]. A detected bounding box or a set of keypoints with a detection accuracy less than the accuracy metric is then considered a false detection. Due to the high quality requirement of AR/MR system, we mainly discuss the false detection rate in terms of the strict accuracy metric, but we also mark the loose metric in each figure with the black dashed line.

Figure 9(a) and Figure 9(b) show the CDF of IoU for the object detection task. Result shows that our system only has 10.68% false detection rate using WiFi-2.4GHz and 4.34% using WiFi-5GHz, which reduce the false detection rate of the baseline approach by 33.1% and 27.0% correspondingly. Figure 9(c) and Figure 9(d) show the CDF of OKS for the human keypoint detection task. Compared to object detection task that only tracks the position of each object bounding box, this task requires to track 17 human keypoints of each human using embedded motion vector, which is much more challenging. However, our system can still reduce the false detection rate by 38.2% with WiFi-2.4GHz and 34.9% with WiFi-5GHz.

To understand how the detection accuracy affects the AR experience, we show several frames with their detection accuracy (OKS) from a sample AR the human keypoint detection task in Figure 10. In this sequence, the person is moving the left hand while the system seeks to render virtual object in the palm of the hand. The three frames in the first row are the rendering results based on our system, while the bottom three frames are based on the baseline approach. We can observe that the rendered cube is well-positioned in our system but trailing behind the palm due to delayed detection results in the baseline approach.

Impact of background network traffic. Results further show that our system is less affected by the background network load, and accuracy degrades more gracefully even in congested networks. Figure 11 shows our measurement results of the false detection rate in WiFi networks with different background traffic loads. In the experiment, we gradually increase the background traffic in the network, and record the corresponding false detection rate with both WiFi-5GHz and WiFi-2.4Hz connections. When raising the traffic load from 0% to 90%, the false detection rate for baseline increases by 49.84% and 35.60% in WiFi-2.4GHz and WiFi-5GHz, respectively. For our system, the increase is only 21.97% and 15.58%, which shows the higher tolerance of our system to network congestion.

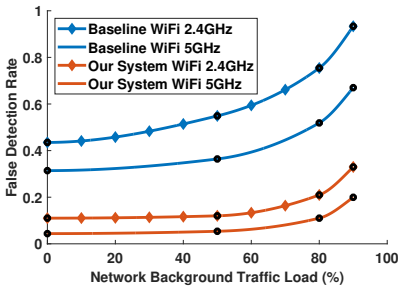


Figure 11: The false detection rate of our system is less affected by the background network load.

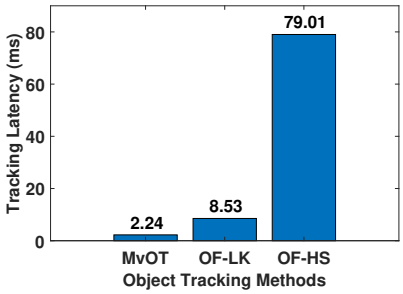


Figure 12: Latency of MvOT compare with two different optical flow tracking methods.

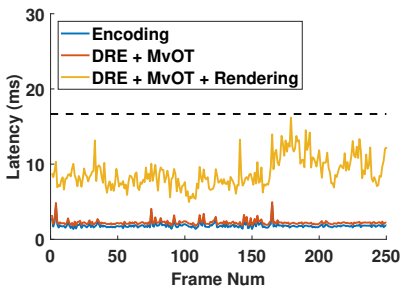


Figure 13: Raw latency traces of our system running a keypoint detection task.

9.3 Object Tracking Latency

Our system only requires 2.24ms to adjust the positions of previously detected objects in a new frame, which leave enough time and computation resources for the AR device to render high-quality virtual overlays with the time between two frames. Figure 12 compares our MvOT method with two standard optical flow based object tracking approaches—the Lucas Kanade and Horn Schunck methods. Both methods have been optimized to take advantage of the on-board GPU of Nvidia Jetson TX2. We can observe that our 2.24ms MvOT method is significantly faster than traditional optical flow approaches and requires 75% less GPU resources compared to the Lucas Kanade based optical flow method. While their tracking may be more accurate, the delay would mean missing the frame display time, which leads to lower accuracy because objects can have moved even further in the next frame.

9.4 End-to-end Tracking and Rendering Latency

Our system is able to achieve an end-to-end latency within the 16.7ms inter-frame time at 60fps to maintain a smooth AR experience. To validate this, we run the keypoint detection task with 3D cube rendering on the *BoxingPractice* video and plot the raw latency traces in Figure 13. The black dashed line in the figure is the 16.7ms deadline for 60fps AR devices, and the yellow curve is the end-to-end latency of this application. Due to our low latency object detection method (Encoding + MvOT) requires an average latency of only 2.24ms, we leave more than 14ms for the AR device to render high quality elements on the screen. We can find that our system is able to finish the detection and rendering tasks within 16.7ms for all 250 test frames.

9.5 Offloading Latency

Our RoI Encoding and Parallel Streaming and Inference techniques can effectively reduce the offloading latency. Figure 14 shows the offloading latency of three methods (Baseline, DRE, and DRE + PSI) with two different WiFi connections. We divide the offloading latency into streaming latency and inference latency for the first two methods, and use a PSI latency for the third method, because the streaming and inference processes run in parallel. The streaming latency contains time spending on encoding, transmission, and decoding tasks. The mean encoding latency to encode an HD frame on Jetson TX2 is 1.6ms and the mean decoding latency on our edge cloud server is less than 1ms.

In the baseline approach, the mean offloading latency is 34.56ms for WiFi-2.4G and 22.96ms for WiFi-5G. With the RDE technique, our system is able to reduce the streaming latency by 8.33ms and 2.94ms, respectively. Combine the techniques of both RDE and PSI, the system further reduces the offloading latency to 17.23ms and 15.52ms. We find that our latency optimization techniques are especially effective to reduce the offloading latency on lower bandwidth connections, such as on the 2.4GHz WiFi network.

9.6 Bandwidth Consumption

Our system is able to reduce the bandwidth consumption of the offloading task through the *Dynamic RoI Encoding (DRE)* and *Adaptive Offloading* techniques. We conduct an experiment to measure the bandwidth consumption of three different offloading approaches (Baseline, DRE only, and DRE plus Adaptive Offloading) in the object detection task. In all three approaches, we use seven different QPs (5, 10, 15, 20, 25, 30, and 35) to control the base quality to encode each frame. The approaches with the RoI Encoding technique will adjust the encoding quality based on the detected RoIs, and the adaptive offloading approach further makes the decision whether to offload each frame to the edge cloud. We record

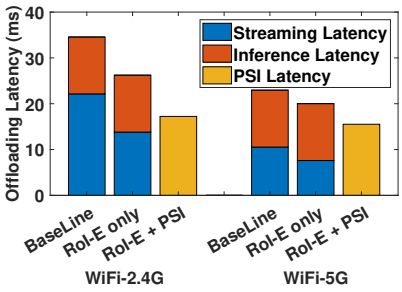


Figure 14: Offloading latency of three approaches using WiFi.

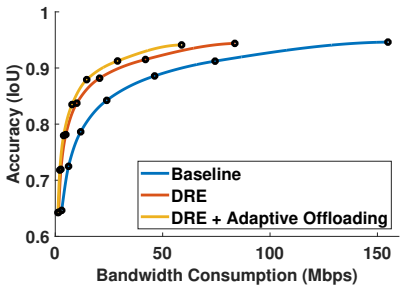


Figure 15: Bandwidth consumption of three approaches.

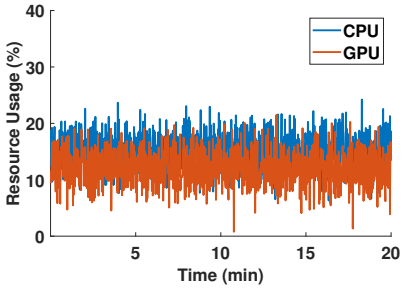


Figure 16: CPU/GPU Resource consumption of our system.

the mean detection accuracy and the bandwidth consumption of these approaches for each QP.

Figure 15 shows how the mean detection accuracy changes with the bandwidth consumption for the object detection task, with the comparison of these three approaches. For the same bandwidth consumption, our RoI Encoding plus Adaptive Offloading approach can achieve the highest mean detection accuracy. Similarly, we can observe that this approach also requires the least bandwidth consumption given a mean detection accuracy, e.g. to achieve the mean detection accuracy of 0.9, our system reduces 62.9% bandwidth consumption compared to the baseline approach.

9.7 Resource Consumption

Our solution consumes very few computation resources on the AR devices. To calculate the resource consumption of our system, we run an object detection task without any local rendering tasks on the DrivingPOV video repeatedly for 20 minutes and use the *tegrastats* tool from JetPack to measure the resource CPU and GPU usage. Figure 16 shows the raw resource usage traces for 20 minutes. Results show that our system requires only 15% of the CPU resource and 13% of the GPU resource, which leaves all the remaining resources to rendering rich graphic overlays for AR/MR system.

10 RELATED WORKS

Mobile AR. Designing mobile Augmented Reality system has attracted strong interest from both industry and academia. ARCore [21] and ARKit [22] are two mobile Augmented Reality platforms, while HoloLens [3] and Magic Leap One [4] further promise to achieve an experience called Mixed Reality. However, none of these platforms support object detection due to the high computation demands. To enable such experience, Vuforia [23] provides an object detection plugin on the mobile devices based on the traditional feature extraction approach. Overlay [24] uses sensor data from the mobile device to reduce the number of candidate objects. VisualPrint [25] aims to reduce the bandwidth consumption of image offloading by only transmit the extracted feature points to the cloud. However, none of them is able to run in real-time (e.g. 30fps or 60fps). Glimpse [8] is a continuous object recognition system on the mobile device designed for cloud offload under challenging network conditions. It runs at 30fps, only offloads trigger frames to the cloud, and uses an optical flow based object tracking method to update the object bounding boxes on the remaining frames. However, Glimpse yields higher end-to-end latency and requires more significant computational resources on the smartphone for object tracking, which leaves less resources for rendering high quality AR/MR . In comparison, we explore a different point in the design space that realizes higher quality AR/MR under more benign network latencies to nearby edge servers. It proposes several techniques to significantly reduce the offloading latency so that it frequently completes within a single frame interval and the low-cost motion vector based tracking method also leaves most of the computation resources available for heavy rendering tasks on the device. Other mobile AR work [26–28] also provide useful insights for us.

Deep Learning. In recent year, Convolutional Neural Network (CNN) has been proven to achieve better performance than traditional hand-crafted feature approaches on various detection tasks. Huang et al. [29] compare the speed and accuracy trade-offs for modern CNN object detection models, including Faster R-CNN [30], R-FCN [31] and SSD [32]. Thanks to the idea of multitask learning, current CNN can further reuse the deep features inside the object bounding box for more fine-grained detection, such as instance segmentation [33], human key points detection [33], facial landmark detection [34], etc. There have been extensive works on how to efficiently run these CNN models on mobile devices [35–42]. However, none of them can satisfy the latency requirement for high quality AR/MR system.

Mobile Vision Offloading. Offloading computation intensive tasks to cloud or edge cloud infrastructures is a feasible way to enable continuous vision analytics. Chen et al. [43] evaluate the performance of seven edge computing applications in terms of latency. DeepDecision [9] designs a framework to decide whether to offload the object detection task to the edge cloud or do local inference based on the network conditions. Lavea [44] offloads computation between clients and nearby edge nodes to provide low-latency video analytics. VideoStorm [45] and Chameleon [46] achieve higher accuracy video analytics with the same amount of computational resources on the cloud by adapting the video configurations. Most of these works focus on a single aspect in the whole vision offloading pipeline, while we focus more on improving the performance of the entire offloading and rendering process.

Adaptive Video Streaming. Adaptive video streaming techniques have been largely exploited to achieve better QoE. Several 360-degree video streaming works [47–50] also adopt the idea of RoI encoding to reduce the latency and bandwidth consumption of the streaming process. Adaptive video streaming techniques have also been adopted by mobile gaming [51, 52] and virtual reality system [53–55] to achieve high quality experience on mobile thin clients. Other video adaptation techniques [56–59] are also complementary to our work.

11 DISCUSSION

In this section, we discuss the following three issues: (1) advantages and the generality of our system, (2) comparison with existing AR tools, (3) limitations of our system.

Generality. Our system is a software solution that can be extended to different hardware and operating systems. The video streaming modules on both the server side and client side can be implemented using various hardware codec APIs, such as Nvidia Video Codec [17], Intel QuickSync [60], Android MediaCodec [61], etc. The inference module on the edge cloud is developed using Nvidia TensorRT [14] platform, which is compatible with most servers equipped with Nvidia GPUs. As shown in Figure 11, our system better tolerates higher background network traffic loads than a baseline offloading scheme, which makes it usable over a wider range of network conditions.

Comparison with Existing AR Tools. Most existing AR tools are not capable of detecting 3D objects continuously in real-time. ARCore [21] and ARKit [22] currently only support 2D image detection such as posters, artwork, or signs on recent high-end smartphones. HoloLens [3] can achieve 3D object detection through the Microsoft Cognitive Services [62] but not in real-time. Besides, several deep learning frameworks designed for mobile and embedded devices, such as TensorFlow Lite [6], Caffe 2 [63], and TensorRT for Jetson [14], claim to achieve low latency inference on existing mobile devices. However, they do not achieve the low latency and high quality requirement for sophisticated mobile AR. Such frameworks typically run compressed CNN models (e.g. Mobilenet SSD model) on low resolution frames, and can only achieve maximum 20 fps performance. TensorFlow Lite requires more than one second to execute an accurate CNN model (e.g. ResNet Faster RCNN model) on one single frame [64]. Instead of processing all inference tasks on mobile devices, our system is able to achieve real-time continuous 3D object detection on existing mobile devices with the support of edge cloud platforms. Note that our system is not an entire AR solution, but complements existing AR systems.

Limitations. This project has not addressed the network challenges in outdoor scenarios or with significantly varying channel conditions, which requires further evaluations in the future. In our experiment, we use WiFi-2.4GHZ and WiFi-5GHz transmissions between the server and the client. Although we have addressed the system's robustness under increasing background network traffic load, we did not yet study other challenges such as network jitters and dropped packets that are possible with higher wireless channel variations.

12 CONCLUSION

In this paper, we design a system that enables high accuracy object detection for AR/MR systems running at 60fps. To achieve this, we propose several low latency offloading techniques that significantly reduce the offloading latency and bandwidth consumption. On the AR device, the system decouples the rendering pipeline from the offloading pipeline, and uses a fast object tracking method to maintain detection accuracy. We prototype an end-to-end system on commodity hardware, and the results show that the system increases the detection accuracy by 20.2%-34.8%, and reduce the false detection rate by 27.0%-38.2% for two object detection tasks. The system requires very few resources for object tracking on the AR device, which leaves the remaining time between frames for rendering to support high quality AR/MR experiences.

ACKNOWLEDGEMENTS

We sincerely thank our shepherd Karthik Dantu and anonymous reviewers for their valuable comments. This material is based in part upon work supported by the National Science Foundation under Grant Nos. 1329939 and PAWR/COSMOS Grant Nos. 1827923.

REFERENCES

[1] Virtual Reality and Augmented Reality Device Sales to Hit 99 Million Devices in 2021. http://www.capacitymedia.com/Article/3755961/VR-and-AR-device-shipments-to-hit-99m-by-2021.

[2] The reality of VR/AR growth. https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/.

[3] Microsoft HoloLens. https://www.microsoft.com/en-us/hololens/.

[4] Magic Leap One. https://www.magicleap.com/.

[5] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[6] TensorFlow Lite. https://www.tensorflow.org/lite/.

[7] Kevin Boos, David Chu, and Eduardo Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 291–304. ACM, 2016.

[8] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM, 2015.

[9] Xukan Ran, Haoliang Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *INFOCOM. IEEE*, 2018.

[10] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[11] Netflix DrivingPOV Video. https://media.xiph.org/video/derf/Chimera/Netflix_DrivingPOV_Copyright.txt/.

[12] Intersection over Union (IoU). http://cocodataset.org/#detection-eval/.

[13] Object Keypoint Similarity (OKS). http://cocodataset.org/#keypoints-eval/.

[14] Nvidia TensorRT. https://developer.nvidia.com/tensorrt/.

[15] Nvidia JetPack. https://developer.nvidia.com/embedded/jetpack/.

[16] Nvidia Multimedia API. https://developer.nvidia.com/embedded/downloads/.

[17] Nvidia Video Codec. https://developer.nvidia.com/nvidia-video-codec-sdk.

[18] Fast INT8 Inference with TensorRT 3. https://devblogs.nvidia.com/int8-inference-autonomous-vehicles-tensorrt/.

[19] Xiph Video Dataset. https://media.xiph.org/video/derf/.

[20] Detection Evaluation for Microsoft COCO. http://cocodataset.org/#detection-eval/.

[21] Google ARCore. https://developers.google.com/ar/.

[22] Apple ARKit. https://developer.apple.com/arkit/.

[23] Vuforia Object Recognition. https://library.vuforia.com/articles/Training/Object-Recognition/.

[24] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 331–344. ACM, 2015.

[25] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Low bandwidth offload for mobile ar. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 237–251. ACM, 2016.

[26] Wenxiao Zhang, Bo Han, and Pan Hui. On the networking challenges of mobile augmented reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 24–29. ACM, 2017.

[27] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. Cars: Collaborative augmented reality for socialization. 2018.

[28] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. Avr: Augmented vehicular reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 81–95. ACM, 2018.

[29] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[31] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[34] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[35] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[36] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, page 23. IEEE Press, 2016.

[37] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[38] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[39] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95. ACM, 2017.

[40] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–81. ACM, 2017.

[41] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 255–266. IEEE Press, 2016.

[42] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 389–400. ACM, 2018.

[43] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai,

Roberta Klatzky, et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 14. ACM, 2017.

[44] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 15. ACM, 2017.

[45] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, pages 377–392. USENIX Association, 2017.

[46] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodík, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 ACM SIGCOMM Conference*. ACM, 2018.

[47] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. pages 482–494, 2018.

[48] Xiufeng Xie and Xinyu Zhang. Poi360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 336–349. ACM, 2017.

[49] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6. ACM, 2016.

[50] Xing Liu, Qingyang Xiao, Vijay Gopalakrishnan, Bo Han, Feng Qian, and Matteo Varvello. 360 innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2017.

[51] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 151–165. ACM, 2015.

[52] Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 121–135. ACM, 2015.

[53] Eduardo Cuervo, Krishna Chintalapudi, and Manikanta Kotaru. Creating the perfect illusion: What will it take to create life-like virtual reality headsets? 2018.

[54] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 68–80. ACM, 2018.

[55] Ruiguang Zhong, Manni Wang, Zijian Chen, Luyang Liu, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Thomas Moscibroda. On building a programmable wireless high-quality virtual reality system using commodity hardware. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*, page 7. ACM, 2017.

[56] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[57] Hyunho Yeo, Sunghyun Do, and Dongsu Han. How will deep learning change internet video delivery? In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 57–64. ACM, 2017.

[58] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. Mp-dash: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 129–143. ACM, 2016.

[59] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285. ACM, 2016.

[60] Intel Quick Sync. https://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html.

[61] Android MediaCodec. https://developer.android.com/reference/android/media/MediaCodec.html.

[62] Microsoft Cognitive Services. https://azure.microsoft.com/en-us/services/cognitive-services/.

[63] Caffe2. https://caffe2.ai/.

[64] TensorFlow Lite Performance. https://www.tensorflow.org/lite/models.