Do We Need Sophisticated System Design for Edge-assisted Augmented Reality?

Jiayi Meng Z. Jonny Kong Y. Charlie Hu
Purdue University
{meng72, kong102, ychu}@purdue.edu

Mun Gi Choi Dhananjay Lal
Charter Communications
{Mun.Choi, Dhananjay.Lal}@charter.com

ABSTRACT

We revisit the performance of a canonical system design for edge-assisted AR that simply combines off-the-shelf H.264 video encoding with a standard object tracking technique. Our experimental analysis shows that the simple canonical design for edge-assisted object detection can achieve within 3.07%/1.51% of the accuracy of ideal offloading (which assumes infinite network bandwidth and the total network transmission time of a single RTT) under LTE/5G mmWave networks. Our findings suggest that recent trend towards sophisticated system architecture design for edge-assisted AR appears unnecessary. We provide insights for why video compression plus on-device object tracking is so effective in edge-assisted object detection, draw implications to edge-assisted AR research, and pose open problems that warrant further investigation into this surprise finding.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; **Ubiquitous and mobile computing design and evaluation methods**.

KEYWORDS

Edge-assisted Augmented Reality, Deep Neural Network, Object Detection, Video Compression

ACM Reference Format:
Jiayi Meng Z. Jonny Kong Y. Charlie Hu Mun Gi Choi Dhananjay Lal. 2022. Do We Need Sophisticated System Design for Edge-assisted Augmented Reality? . In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526267

1 INTRODUCTION

Augmented Reality (AR) promises unprecedented interactive and immersive experiences to users in a real-world environment, and along with VR/MR are the cornerstones of the emerging Metaverse [6].

A complex AR app often needs to perform a number of challenging tasks, *e.g.,* pose estimation, object detection, and depth estimation, in order to understand and interact with the physical environment. In recent years, Deep Neural Networks (DNN)-based solutions have been developed for these tasks and shown to achieve high accuracy. Since such DNN models are too computation-intensive to run on resource-constrained mobile devices in real time, offloading DNN inference tasks to edge servers, known as edge-assisted AR, has emerged as the de facto approach (*e.g.,* [7, 16]).

As offloading high-resolution, high-rate frames to edge servers conceptually requires ultra-high bandwidth, (edge-assisted) AR is widely hailed as a 5G "killer" app [1, 2], *e.g.,* in the AT&T and Microsoft alliance [3], and the Verizon and AWS alliance to showcase 5G edge computing solutions [5].

In this paper, we observe that previous work on edge-assisted AR (object detection) did not systemically explore several basic, legacy optimization techniques. We then present an in-depth evaluation of the potential and limit of a canonical edge-assisted design that simply combines off-the-shelf video encoding (H.264) with a fairly standard object tracking technique (the Lucas-Kanade algorithm [18]) running on mobile device in real time, denoted as local tracking [7] thereafter. Specifically, we compare it with three baselines (offloading raw frames, frames with image compression, and ideal offloading which assumes infinite network bandwidth and the total network transmission time of a single RTT), using three diverse AR video datasets which cover all the datasets used in previous work, and three representative DNN models for object detection. Our evaluation shows that such a canonical edge-assisted AR design can already achieve within 3.07%/1.51% of the accuracy (mean Average Precision) of ideal offloading, under LTE and 5G mmWave, respectively.

We further unravel the fundamental reasons for why the canonical design is so effective: (1) video encoding significantly reduces the offloaded frame size; (2) in the mean time lossy video compression at proper encoding bitrate causes negligible accuracy degradation of DNN models for object detection, due to the robustness of the DNN models; and (3) local tracking can uphold the estimation accuracy of periodic offloaded frames for all frames.

Our findings suggest that additional benefits of sophisticated system architecture designs for edge-assisted AR such as those proposed recently [16] will bring only marginal accuracy improvement over the canonical design. We conclude by drawing implications to edge-assisted AR research and pose several open problems that warrant further investigation as a consequence of this finding.

2 EDGE-ASSISTED AR: DESIGN OBJECTIVE AND PRIOR WORK

We give a brief background of the edge-assisted DNN-based AR problem, review optimization techniques in the literature, and discuss how two basic, legacy optimizations have not been systematically or properly explored.

EdgeSys'22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9253-2/22/04.
https://doi.org/10.1145/3517206.3526267

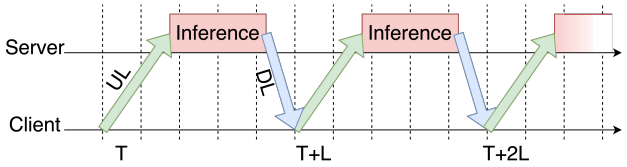

Figure 1: Frame offloading with E2E delay of L frame intervals.

2.1 Background of Edge-assisted AR

We consider a single DNN-based AR task, *e.g.,* object detection. Given a sequence of frames F_i captured in real time by the camera on the mobile device, and a DNN model M that implements the task, edge-assisted AR seeks to maximize the *overall accuracy* of the AR task, *i.e.,* the DNN inference, on all the frames F_i, by exploiting offloading the computation-intensive DNN inference to an edge server over the wireless network. The overall accuracy is typically measured by summarizing the accuracy per frame, *e.g.,* using mean average precision (mAP) [9] for object detection.

The end-to-end (E2E) latency of offloading the DNN inference task for frame i is simply

$$T_{e2e}(i) = T_{ul}(i) + T_{inf}(i) + T_{dl}(i) \quad (1)$$

where T_{ul} and T_{dl} denote the duration for the client to transmit the frame to the server and for the server to send the inference result to the client, respectively, and T_{inf} denotes the DNN inference time on the edge server. Specifically, the uplink/downlink data transfer time T_{ul}/T_{dl} consists of one-way network delay (half RTT) between the client and the server and the delay in transmitting data over the wireless network, *e.g.,* $T_{ul}(i) = \frac{RTT}{2} + \frac{D_{ul}(i)}{B_{ul}}$, where D_{ul} is size of uplink data and B_{ul} is uplink bandwidth.

In practice, the end-to-end offloading latency of the above simple design when running under today's wireless networks such as LTE and 5G can exceed the duration of a frame capture (*e.g.,* 16.7ms for 60 FPS). As a result, the client may only be able to offload frames periodically. Figure 1 shows an example where the E2E delay in offloading a single frame is L frame intervals.

The delay in offloading a frame in the simple design has two implications: the longer the offloading delay T_{e2e} (*e.g.,* L intervals), not only the estimation result (*e.g.,* for frame T) sent back by the server (received at T+L-1) will be *more stale*, but also *more frames* (*e.g.,* L frames) will be using the stale result. Both factors will contribute to reducing the overall estimation accuracy across all the frames of the video.

2.2 Towards Improving Edge-assisted AR Task Accuracy

The above analysis suggests the overall accuracy of an AR task in the simple edge-assisted AR design above can be improved by exploring two approaches: (1) reducing offloading latency T_{e2e}, and (2) improving the accuracy in using the result of last offloaded frame for new frames.

Optimization 1: reducing offloading latency by video/ image compression. A well-known technique in edge computing is to reduce data transfer latency via data compression (*e.g.,* [22]). In edge-assisted AR, an uploaded frame and its inference result can be encoded/decoded before/after data transfer to reduce the data transfer time T_{ul}/T_{dl}. When the result is small, the technique is typically applied to the uploaded frame only, *e.g.,* using video/image compression techniques, and the offloading latency becomes

$$T_{e2e} = T_{en}(i) + T_{ul}(i) + T_{de}(i) + T_{inf}(i) + T_{dl}(i) \quad (2)$$

where $T_{en}(i)$ and $T_{de}(i)$ denote the encoding and decoding latency on the mobile client and the server, respectively.

Optimization 2: reducing offloading latency by partial DNN offloading. In contrast to frame compression, a much more complex approach to reducing the offloading latency which was extensively studied in recent years is to partition the DNN model for inference between the mobile client and the server and find the optimal partitioning point that minimizes the end-to-end latency, *e.g.,* that has a small model feature size (*e.g.,* [8, 10, 12–15]). The idea can be further combined with data compression, *i.e.,* by compressing the model features of the partitioned layer offloaded to the edge server.

Optimization 3: improving estimation accuracy via local tracking. Since directly reusing the result of the last offloaded frame (T) for future frames (T+L, ... , T+2L-1) leads to poor estimation accuracy (due to staleness), local tracking [7] was proposed to improve the estimation accuracy by locally adjusting the estimation result of the last offloaded frame T to derive a better estimation for subsequent frames T+L, T+L+1, *etc.*, based on the difference between the features extracted from frames T and T+L+i. Since local tracking is geometry-based, it is generally light-weight and can be executed in a fraction of a frame interval.

Optimization 4: improving estimation accuracy via hybrid DNN offloading. Instead of local tracking, the mobile client can run a light-weight version of the DNN locally which may also provide better accuracy than directly reusing the result of the last offloaded frame.

2.3 Previous Work

We observe that the large amount of work on edge-assisted object detection in recent years failed to systematically explore the combination of two basic optimization techniques, frame compression and local tracking. In particular, we contrast prior work with what we denote as the *canonical design* of edge-assisted AR design where the mobile device simply applies an off-the-shelf, standard video compression technique (H.264) to the each offloaded, original frame output by the camera, offloads the next frame as soon as the result for the last offloaded frame comes back, and runs a standard local tracking technique (the Lucas-Kanade algorithm [18]) for new frames on the mobile device in real time.

First, the large body of work on DNN inference offloading (*e.g.,* [12, 13, 27]) studied reducing the latency of object detection on individual frames, disregarding the inter-frame arrival time in a real AR. It is unclear how much they can improve the accuracy of frames captured in real-time AR.

Second, other works on edge-assisted AR often resorted to much more complex designs, such as partial/hybrid DNN inference offloading, dissecting each frame into subframes and applying complex encoding, and leveraging specially-tuned DNN-based encoders and decoders, as summarized in Table 1. We note one of the first

Table 1: Comparison between previous work and canonical design on edge-assisted object detection.

Previous work	Video compression	Image compression	Partial DNN offloading	Local tracking	Hybrid DNN offloading	Compared with ideal offloading
e.g., [12, 13, 27]	No	Yes for some, complex	Yes, complex	No	Yes, complex	No
DeepDecision [22]	Yes	No	No	No	Yes, complex	No
Liu *et al.* [16]	Yes, complex	No	No	Yes	No	No
Canonical design	**Yes**	**No**	**No**	**Yes**	**No**	**Yes**

works on edge-assisted AR, Glimpse [7], did not consider DNN-based object detection and only considered two simple object types.

DeepDecision [22] proposed hybrid DNN inference offloading that uses a complex measurement-driven framework to decide whether to run a small DNN model locally or a large model remotely on the server. However, the study was limited, because (1) it did not consider local tracking, and (2) it did not compare the accuracy of the proposed solution with that of ideal offloading.

Liu *et al.* [16] also considered both local tracking and frame compression. However, the design significantly complicates the canonical design in three ways: (1) Instead of simply applying video encoding to whole frames, it resorts to a complex scheme where the mobile client dissects each frame into subframes of different importance and seeks to apply frame encoding of different bitrates. (2) Instead of applying a single encoding bitrate to all frames for diverse datasets and both LTE and 5G mmWave, it applies varying bitrates in frame encoding, *i.e.,* higher encoding bitrate to subframes that are "regions of interest" and lower encoding bitrates to other subframes. Such a heuristic does not directly optimize the AR task accuracy for a given network bandwidth constraint. (3) Instead of simply offloading frames back to back, it determines when to offload frames via a heuristic, which estimates the difference between two consecutive frames and decides to offload the new frame when the difference is larger than some predetermined threshold to save network bandwidth (as opposed to maximize the object detection accuracy). Such a heuristic does not directly try to maximize the overall object detection accuracy across the frames.

3 HOW GOOD IS THE CANONICAL OFFLOADING DESIGN?

Our central hypothesis is that *previous work on edge-assisted object detection did not systemically explore several basic, legacy optimization techniques, and that the* canonical design*, can already achieve accuracy close to that of ideal offloading which assumes infinite network bandwidth and the total data transfer time of a single RTT.*

We next validate the hypothesis via extensive experiments.

3.1 Prototypes of Edge-assisted AR

We first developed four edge-assisted object detection apps running on an Android device without local tracking:

Offload-RAW: This is the baseline offloading scheme where the client directly transmits camera-captured raw frames to the server for DNN inference as fast as it can. The server transmits inference results, *i.e.,* bounding boxes and confidence values, back to the client. We consider it as a baseline design as raw frames do not cause accuracy degradation in DNN inference.

Offload-IMG: The client first uses Android APIs to compress an offloaded raw frame to JPEG. It then offloads the JPEG image to the server for DNN inference.

Offload-VID: For each offloaded frame, the client first encodes it using the built-in hardware encoder of H.264 which is widely adopted on mobile devices. The client then offloads the encoded frame to the server. The server decodes the encoded frame to extract the RGB frame, executes DNN inference and then sends the inference result back to the client.

Offload-Ideal: This version is used to assess the upper-bound accuracy of edge-assisted AR. It assumes infinite network bandwidth and per-frame network transmission latency (sum of uploading and downloading duration) is the same as the RTT between the client and the edge server. To implement it, we first run the DNN model offline over all the frames, save the inference result and record the inference latency on the server. During the experiments, the client only sends the frame name to the server. The server sleeps for the inference latency and then sends the corresponding inference result back. In this way, we emulate ideal offloading with the network transmission latency of one RTT only.

We then extended all four versions to include the option for turning on local tracking: for every frame, the client runs the lightweight Lucas-Kanade algorithm to track objects from the last offloaded frame to the current frame.

3.2 Experimental Setup

Client and server. The mobile client runs on a Samsung Note 20 Ultra 5G phone equipped with Qualcomm Snapdragon 865 Plus 5G SoC. The edge server is equipped with an Intel Xeon W-2133 CPU and a GeForce 2080Ti GPU.

DNN models. We selected three popular DNN models developed for object detection running on the server, *i.e.,* FasterRCNN [24], SSD [17] and YOLOv3 [23].

Datasets. We chose three diverse video datasets which covers all the datasets used by prior works on edge-assisted AR, ImageNet 2017 [25], MOT 2017 [20] and Xiph [4], which represent typical AR scenarios. The videos we sampled from the three datasets have resolution of 1280×720, 1920×1080, and 3840×2160, respectively. Since the Xiph dataset does not come with ground truth labels, we manually labelled them. We extracted YUV420 frames (the standard camera output format) from the videos at 640×480 resolution, the closest resolution to SSD and YOLOv3's input resolution. Though Faster-RCNN can take variable-sized inputs, we found downsizing videos to 640×480 causes negligible accuracy degradation (1.61% mAP at most). During the experiments, we replayed videos at their original frame rate (*i.e.,* 30 FPS for ImageNet 2017 and MOT 2017 and 60 FPS for Xiph).

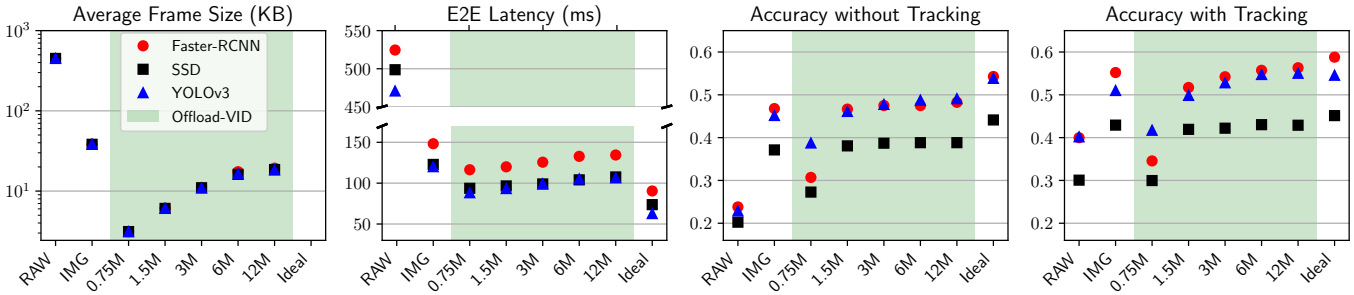

Figure 2: Comparison of average frame size, E2E latency and offloading accuracy without and with local tracking averaged across the three datasets, for the three DNN models under LTE.

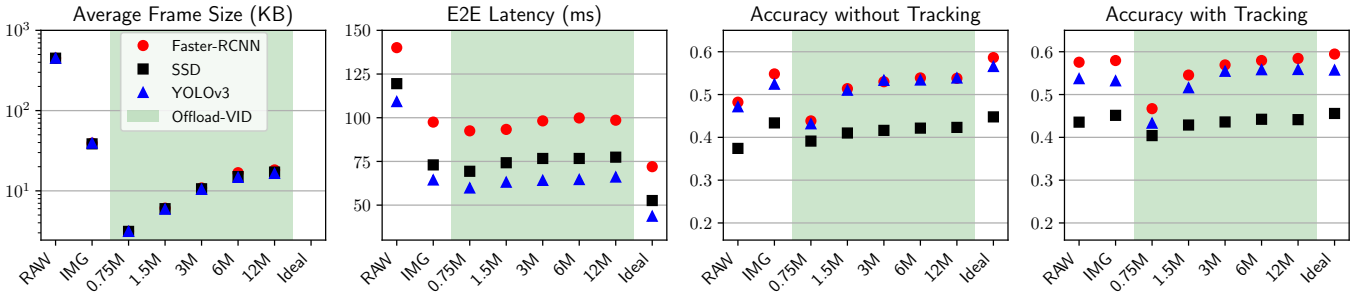

Figure 3: Comparison of average frame size, end-to-end latency and offloading accuracy without and with local tracking averaged across the three datasets, for the three DNN models under 5G.

Encoding parameters. For Offload-VID, we selected 5 different bitrates {0.75, 1.5, 3, 6, 12} (unit: Mbps). For Offload-IMG, we adopted the same JPEG quality level (*i.e.,* 70) used by [7].

Network conditions. We consider two network conditions, LTE and 5G mmWave, by emulating measured throughput and RTT using the Linux tc tool. For LTE, the TCP upload/download throughput are 10Mbps/100Mbps based on measurement using *speedtest.net*, and the client-server RTT is 30 ms as measured between the client and Verizon's Wavelength edge server in Downtown Boston using traceroute. For 5G, we used average upload and download bandwidth calculated using public traces for 5G mmWave [21], of around 73Mbps/520Mbps when the client is walking/driving, and an RTT of 12 ms based on our own measurement between the client and Verizon's Wavelength server under 5G mmWave. [1]

Metrics. We measure the following metrics for each offloading version: (1) *Accuracy:* the accuracy of object detection using the standard metric, mean Average Precision (mAP) [9] with the IoU threshold of 0.5; (2) *End-to-end latency:* the average per-frame offloading latency; (3) *Average frame size:* the average size of encoded frames offloaded to the edge server.

3.3 Results for LTE

We first compare the performance of the four online offloading schemes under LTE network condition.

Figure 2 (1st) shows that as encoding bitrate increases from 750 Kbps to 12 Mbps, video encoding increases the average frame size from 3.12 KB to 16.66 KB averaged across the three datasets for all the three models, which is smaller than the average frame size of image encoding (38.34 KB), and much smaller than raw frames (450 KB). The sublinear increase of the frame size is because of the default encoding technique used by Android, *i.e.,* VBR (variable bitrate) which applies higher compression ratio for less complex frame content.

The reduced frame size translates into lower end-to-end (E2E) offloading latency. Figure 2 (2nd) shows that Offload-VID achieves the lowest E2E delay, 98.20–143.88 ms under the encoding bitrate of 12 Mbps, which is smaller than the E2E delay of Offload-IMG (106.9–160.07 ms) and much smaller than that of Offload-RAW (465.98–528.57 ms). For Offload-VID, the latencies of encoding (on the mobile device) and decoding (on the edge server), 7.47 ms and 5.47 ms, respectively, regardless of the encoding bitrate, are not significant. For Offload-IMG, the latencies of encoding and decoding are slightly smaller, *i.e.,* 5.02 ms and 5.00 ms, respectively.

Figure 2 (3rd) shows that without local tracking, Offload-RAW achieves the lowest accuracy. For Offload-VID, as the encoding bitrate increases from 750 Kbps to 12 Mbps, the accuracy averaged across the three datasets increases from 30.70%, 27.28%, and 38.74%, to 48.25%, 38.82%, and 49.09%, for Faster-RCNN, SSD and YOLOv3, respectively. When the encoding bitrate reaches beyond 6 Mbps, the offloading accuracy increase slows down, as the model inference accuracy does not keep increasing (discussed in §4), but the E2E offloading latency increases because of the increased encoded frame size. Offload-VID with 6 Mbps bitrate achieves notably 6.80%, 5.31%, and 5.08% lower accuracy than Offload-Ideal for the three models. The gap is due to the higher E2E latency of Offload-VID (104.1–132.8 ms, or 4 frame intervals) compared to that of Offload-Ideal (62.7–90.4 ms, or 2–3 frame intervals). Offload-IMG achieves

[1] [26] reports similar median RTTs measured between a client and an Alibaba ENS edge server over LTE and 5G in China, of 34.2 ms and 10.4 ms, respectively.

worse accuracy than Offload-VID, *i.e.,* 7.49%, 7.00% and 8.67% lower accuracy than Offload-Ideal, because of lower model inference accuracy and much larger compressed frame size (by 1.75X−2.83X).

Figure 2 (4th) shows that with local tracking, the accuracy of Offload-RAW improves to 39.97%, 30.06% and 40.19%, for Faster-RCNN, SSD and YOLOv3, respectively. For Offload-VID, local tracking also increases the accuracy across all the bitrates for the three models. Offload-VID with 6 Mbps encoding bitrate achieves the accuracy of 55.73%, 43.03% and 54.74% for the three models, respectively, which is only 3.07%, 2.11%, and -0.17% smaller than Offload-Ideal. The much smaller gap is because local tracking from offloaded frames effectively captures the movement of objects and the camera. Finally, local tracking also improves the accuracy of Offload-IMG to 55.21%, 42.92% and 51.01% for the three models, which are 3.59%, 2.22%, 3.56% lower than that of Offload-Ideal.

In summary, combined with local tracking, video encoding achieves accuracy better than Offload-IMG and Offload-RAW, and within 3.07% of the accuracy of Offload-Ideal, under LTE.

3.4 Results for 5G mmWave

Figure 3 shows that 5G mmWave's lower RTT and higher bandwidth result in shorter E2E latency than LTE. For Offload-VID, at 6 Mbps bitrate, the E2E latency is 99.86 ms, 76.69 ms, and 64.62 ms, averaged across the three datasets for Faster-RCNN, SSD, and YOLOv3, respectively, which is significantly shorter than LTE's 132.80 ms, 104.05 ms, and 105.08 ms.

The reduction in E2E latency translates to higher accuracy. Without local tracking, Offload-VID with 6 Mbps bitrate achieves accuracy of 53.88%, 42.15%, and 53.39% for the three models, respectively, which are 6.40%, 3.33%, and 4.64% higher than that of LTE without tracking. Local tracking further improves the accuracy to 57.95%, 44.22%, and 55.80%, *i.e.,* 2.22%, 1.19%, and 1.06% higher than LTE with tracking.

The lower RTT and higher throughput of 5G also reduce the difference in E2E latency between different encoding schemes and ideal offloading, which in turn reduces the difference in their accuracy. With local tracking, Offload-VID's accuracy is 1.51%, 1.36%, and -0.08% lower than Offload-Ideal, while Offload-IMG's accuracy is 1.51%, 0.44% and 2.51% lower than Offload-Ideal and Offload-RAW's accuracy is 1.91%, 2.03% and 2.01% lower than Offload-Ideal.

In summary, unlike LTE, under 5G mmWave, with local tracking, both Offload-VID and Offload-IMG slightly outperform Offload-RAW, and practically tie with Offload-Ideal.

4 WHY DOES THE CANONICAL DESIGN WORK SO WELL?

We discuss the fundamental reasons why the canonical offloading design based on video encoding works so well.

1. Video encoding significantly reduces offloaded frame size and hence E2E offloading latency. Figure 2 shows even with the highest chosen bitrate of 6 Mbps, the average frame size of video encoding is only 18.72 KB, *i.e.,* 48.83% smaller than the image-encoded frames (38.34 KB), and only 4.16% of the uncompressed frame (450.00 KB). Its E2E latency thus drastically reduces to 113.98 ms, smaller than Offload-IMG (130.43 ms) and Offload-RAW (498.04 ms) over LTE. The E2E latency of Offload-VID, Offload-IMG and Offload-RAW further drops to 80.39 ms, 78.30 ms and 122.94 ms over 5G. The smaller E2E delay of Offload-IMG is mainly because Offload-IMG's encoding take 5.02 ms, lower than Offload-VID (7.47 ms) running on the mobile client.

Table 2: Comparison of frame size and average offline accuracy using original/compressed frames as input.

Method	Avg. Frame Size (KB)	Offline Accuracy		
		Faster-RCNN	SSD	YOLOv3
Original	**450**	**66.39%**	**47.21%**	**59.68%**
IMG	38.24	64.98%	47.03%	58.10%
VID (0.75M)	3.15	62.14%	45.29%	55.77%
VID (1.5M)	6.15	64.44%	46.18%	58.87%
VID (3.0M)	11.58	65.68%	46.67%	59.76%
VID (6.0M)	**17.69**	**66.32%**	**46.95%**	**60.13%**
VID (12.0M)	20.20	66.39%	47.30%	60.07%

2. Yet lossy compression does not cause accuracy degradation of DNN inference. Conceptually, video encoding is lossy; though it can reduce E2E offloading latency, it may comprise the accuracy of object detection for each offloaded frame running on the edge server and cancel out the benefits of frame compression. To evaluate this, we compare the accuracy of running the DNN models offline with three versions of each frame: the original frame, decoded frame after image encoding, and decoded frame after video encoding (following § 3.2). Table 2 shows that for video encoding, increasing the encoding bitrate from 0.75 Mbps to 12.0 Mbps increases accuracy from 62.14%, 45.29% and 55.77%, to 66.39%, 47.30% and 60.07% averaged over the three datasets for Faster-RCNN, SSD and YOLOv3, respectively. Compared with using original raw frames as input which achieves 66.39%, 47.21% and 59.68% accuracy for the three models, respectively, when the encoding bitrate reaches 6 Mbps, video encoding achieves 66.32%, 46.95% and 60.13% accuracy for the three models, *i.e.,* with almost no accuracy loss (0.26% at maximum) on DNN model inference.

3. Local tracking upholds estimation accuracy. We caculated the per-frame mean IoU (Intersect over Union) of Offload-VID without and with local tracking for all the models and videos, and found the gap reduces from 4.22% to 2.16% under LTE and from 3.34% to 1.65% under 5G, on average.

5 IMPLICATIONS TO EDGE-ASSISTED AR SYSTEM DESIGN

No need to design complicated offloading scheme. The key takeaway from our study is that there appears no need to develop complicated offloading system designs for edge-assisted AR, as the canonical design, already achieves accuracy very close to that of ideal offloading, and the gap further narrows with the improvement of the network technology, *e.g.,* within 3.07% for LTE and practically comparable under 5G mmWave. We argue the small and narrowing gap is well-justified by the simple offloading design which benefits from well-studied and well-engineered building blocks such as video encoding and local tracking. Further, it is unclear if complex offloading designs can bridge the small gap as the frame transfer

time is only 14.98 ms and 2.05 ms beyond the RTT under LTE and 5G mmWave. [2]

In this paper, we showed the canonical design works well under steady LTE and 5G throughput. In practice, the wireless networks may exhibit dynamics. We believe the canonical design will continue to work well as we have shown in §3 that it has low requirement on network bandwidth.

Improving and picking DNN models. Given the canonical design can already achieve accuracy close to that of ideal offloading, further effort on improving the accuracy of edge-assisted AR should focus on improving the accuracy of the DNN model and reducing DNN inference time which currently accounts for a major fraction of T_{e2e} (41.11% under LTE and 54.88% under 5G mmWave). A related question is how to pick the DNN models out of the many candidate models that offer different tradeoffs between accuracy and inference time (*e.g.,* [19]).

Reducing network RTT will play a bigger role for accuracy improvement. Our study shows that the accuracy improvement from LTE to 5G comes from reduced E2E latency by 28.00 ms, which is mainly caused by RTT reduction (from 30 ms to 12 ms) compared with bandwidth improvement (from 10 Mbps to 75 Mbps) which only cuts 10ms. Going forward, RTT reduction, *e.g.,* with the deployment of Standalone 5G, will more directly contribute to further E2E offloading latency reduction, compared to further bandwidth increase, given that the encoded frame size is relatively small.

Open questions. Our study also raises several open questions. (1) Will video encoding be equally effective in offloading other DNN-based AR tasks such as pose estimation and depth estimation? (2) A complex AR app or Mixed Reality app may need to perform multiple vision tasks on each frame. How should video encoding be applied in multi-task AR offloading? (3) Will the canonical design be equally effective in saving the mobile device power consumption in edge-assisted AR? (4) What is the limit of video encoding for edge-assisted object detection in non-AR context? For example, though the three video datasets used in our study are diverse, video captures in special video analytics systems may contain a large number of small objects [11]. How much will encoding such frames affect the accuracy of object detection DNN models?

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This project is supported in part by NSF grant 2112778 and by Charter Communications.

REFERENCES

[1] 2017. Augmented and Virtual Reality: the First Wave of 5G Killer Apps: Qualcomm – ABI Research. https://gsacom.com/paper/augmented-virtual-reality-first-wave-5g-killer-apps-qualcomm-abi-research/.
[2] 2018. The mobile future of augmented reality. https://www.qualcomm.com/media/documents/files/the-mobile-future-of-augmented-reality.pdf.
[3] 2019. AT&T integrates 5G with Microsoft Azure to enable next-generation solutions on the edge. https://www.business.att.com/learn/top-voices/at-t-integrates-5g-with-microsoft-azure-to-enable-next-generatio.html.
[4] 2019. Xiph. https://media.xiph.org/video/derf/.
[5] 2021. Verizon teams with NFL, AWS to showcase 5G edge. https://www.fiercewireless.com/operators/verizon-teams-nfl-aws-to-showcase-5g-edge.
[6] 2022. Metaverse. https://en.wikipedia.org/wiki/Metaverse.
[7] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.
[8] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* (2019).
[9] Mark Everingham, SM Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2015. The pascal visual object classes challenge: A retrospective. *International journal of computer vision* 111, 1 (2015), 98–136.
[10] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.
[11] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 559–572.
[12] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.
[13] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–15.
[14] En Li, Zhi Zhou, and Xu Chen. 2018. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*. 31–36.
[15] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. 2018. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 671–678.
[16] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.
[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.
[18] Bruce D. Lucas and Takeo Kanade. 1981. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2* (Vancouver, BC, Canada) *(IJCAI'81)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 674–679.
[19] Jiayi Meng, Zhaoning Kong, Qiang Xu, and Y Charlie Hu. 2021. Do Larger (More Accurate) Deep Neural Network Models Help in Edge-assisted Augmented Reality?. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*. 47–52.
[20] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. 2016. MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831* (2016).
[21] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand AK Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, et al. 2020. Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput. In *Proceedings of the ACM Internet Measurement Conference*. 176–193.
[22] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1421–1429.
[23] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015), 91–99.
[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y
[26] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. 2021. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*. 37–53.
[27] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 476–488.

[2] We could not directly compare to those complex designs (*e.g.,* [16]), as their implementation are not available.