Do Larger (More Accurate) Deep Neural Network Models Help in Edge-assisted Augmented Reality?

Jiayi Meng*
Purdue University

Zhaoning Kong*
Purdue University

Qiang Xu
Purdue University

Y. Charlie Hu
Purdue University

ABSTRACT

Edge-assisted Augmented Reality (AR) which offloads compute-intensive Deep Neural Network (DNN)-based AR tasks to edge servers faces an important design challenge: how to pick the DNN model out of many choices proposed for each AR task for offloading. For each AR task, *e.g.,* depth estimation, many DNN-based models have been proposed over time that vary in accuracy and complexity. In general, more accurate models are also more complex; they are larger and have longer inference time. Thus choosing a larger model in offloading can provide higher accuracy for the offloaded frames but also incur longer turnaround time, during which the AR app has to reuse the estimation result from the last offloaded frame, which can lead to lower average accuracy.

In this paper, we experimentally study this design tradeoff using depth estimation as a case study. We design optimal offloading schedule and further consider the impact of numerous factors such as on-device fast tracking, frame downsizing and available network bandwidth. Our results show that for edge-assisted monocular depth estimation, with proper frame downsizing and fast tracking, compared to small models, the improved accuracy of large models can offset its longer turnaround time to provide higher average estimation accuracy across frames under both LTE and 5G mmWave.

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, Monocular Depth Estimation

ACM Reference Format:
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 *ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI'21), August 27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3472727.3472807

*Both authors contributed equally to this research.

1 INTRODUCTION

Augmented Reality (AR) promises unprecedented interactive and immersive experiences to users in a real-world environment, where physical objects that reside in the real world are enhanced by computer-generated perceptual information, and has applications across diverse areas such as retail, education, and entertainment. The global mobile AR market is forecast to reach 21 billion dollars with 2 billion mobile AR users by 2024 [1].

A complete AR app needs to perform a number of challenging tasks to understand and interact with the physical environment. These include pose estimation, object detection, and depth estimation [3]. Using the popular Pokemon GO app as an example, pose detection allows the AR app to track the 6 DoF (degrees of freedom) of the camera relative to the physical environment; object detection allows the AR app to detect objects in the physical world such as trees and buildings in order to decide *where* to position a Pokemon; and depth estimation determines the distance of the physical objects from the camera in order to decide *how* to place the Pokemon relative to a physical object, *e.g.,* in front of or behind a tree.

Performing each of the tasks with high accuracy is challenging. In the past few years, Deep Neural Networks (DNN)-based solutions have been developed for these tasks that can achieve reasonably high accuracy. However, such DNN models are also too computation-intensive to run on resource-constrained mobile devices in real time. As a result, offloading such DNN inference tasks to edge cloud servers has become the de facto approach to leveraging such DNN-based solutions (*e.g.,* [7, 13]).

Offloading DNN-based AR tasks to edge servers, known as edge-assisted AR, faces an important design challenge: how to pick a DNN model out of many choices proposed for each AR task to run on the edge server. For a given AR task, *e.g.,* depth estimation, many DNN-based models have been proposed over time. In general, more accurate models are also more complex; they are larger, require larger input, and have longer inference time. Thus choosing a larger model in offloading can provide higher accuracy for the offloaded frames but also incur longer turnaround time for a given network and an edge server, during which the AR app has to reuse the estimation result from the last offloaded frame, which can lead to lower average accuracy.

In this paper, we experimentally study this important design tradeoff in edge-assisted AR using depth estimation as a case study.

State-of-the-art DNN-based depth estimation, known as *monocular depth estimation*, performs accurate depth estimation using a single RGB camera which is low cost and widely available on mobile devices. A number of monocular depth estimation models have

Table 1: Specifications and performance of the three DNN models for monocular depth estimation.

Model Size	**Model Name**	**Input Res.**	**Output Size**	**MACs** ($\times 10^9$)	**Abs. Rels. Error**	**Inf. Time on Server** (ms)	**Inf. Time on Pixel 2** (s)
Small	SC-SfmLearner [6]	832×256	832×256	18.81	13.4%	8.18	1.57
Medium	MonoDepth2 [10]	1024×320	1024×320	21.45	11.6%	12.34	1.64
Large	DenseDepth [5]	1248×384	1248×384	191.45	10.5%	41.80	14.08

been proposed in the past few years [5, 6, 10]. As in many other areas of deep learning research, the progress has focused on improving accuracy. As a result, more accurate depth estimation models generally are more complex and larger and have longer inference time. In our study, we picked three state-of-art DNN models for monocular depth estimation, SC-SfmLearner [6], MonoDepth2 [10] and DenseDepth [5]. Table 1 summarizes the characteristics of the three models. We see that they have increasing input frame sizes and model sizes (measured in the number of multi-and-add operations, or MACs), and correspondingly longer inference time, of 8.18ms, 12.34ms and 41.80ms, on a server GPU (GeForce 2080 Ti), but also increasingly lower average estimation errors, of 13.4%, 11.6% and 10.5%, respectively (using the KITTI dataset [15]). We denote the three models as Small, Medium, and Large, respectively, in the rest of the paper.

We proceed with studying the accuracy-delay tradeoffs of depth estimation DNN models by experimentally comparing the average estimation accuracy across all frames in a video (offloaded and not offloaded) in offloading the three monocular depth estimation models, in four steps. We emulate offloading to an edge server on a PC equipped with a GeForce 2080 Ti GPU. We use 12 videos from the widely used KITTI dataset (that are not used for training the three models) in our evaluation.

First, we compare the three models under a baseline offloading scheme under the typical cellular bandwidth available today (*i.e.,* LTE with measured uplink and downlink bandwidth of 25Mbps and 110Mbps, respectively), where the frames are offloaded as frequently as the network and the edge server can accommodate, and in between two offloaded frames, the estimation result for the last offloaded frame is reused for the current frame. To determine the maximal offloading frequency, we develop optimal offloading scheduling for the two cases where either the network transmission or the server inference time is the bottleneck. Our results show that Small, Medium and Large models suffer long network transmission latency under LTE, causing increasingly lower offloading frequencies, once every 11, 15 and 23 frames, which translate into high average errors, of 21.7%, 22.5% and 22.2%, respectively.

Second, we study how incorporating on-device fast tracking for the non-offloaded frames affect the design tradeoff. Specifically, we exploit warping, a lightweight geometric image reconstruction technique, to synthesize the depth map for a target frame (a non-offloaded frame) from the reference depth map for a reference frame (the last offloaded frame) predicted by the DNN model, using the relative pose change between those two frames. Surprisingly, our results show that warping slightly increases the average error to 22.1%, 22.6% and 25.3% for Small, Medium and Large models, respectively. The reason is that warping works well when the two frames have similar content but cannot handle new content in the target frame that did not exist in the reference frame, *e.g.,* during fast camera movement or scene change.

Third, we study if downsizing frames to reduce network transmission latency will improve the accuracy of larger DNN models. We explore frame downsizing, a lightweight compression algorithm by dropping pixels uniformly [1]. Our results show that frame downsizing with carefully chosen target size can significantly reduce the end-to-end offloading delay of large models without compromising their accuracy; the average estimation accuracy across the three models are 17.2%, 19.6%, and 17.0% without fast tracking, and 15.9%, 17.0%, and 15.5% with fast tracking, respectively.

Finally, we study whether improved network bandwidth, *i.e.,* the emerging 5G, will help larger DNN models to achieve better accuracy. We repeat the offloading experiments with frame downsizing over 5G mmWave, by emulating the average uplink and downlink bandwidth of 73Mbps/520Mbps measured in the wild [14]. Our results show that as expected, 5G mmWave significantly increases offloading frequency for all models, to once every 2, 3, and 3 frames, respectively. However, the improved offloading frequency helps the larger model much more than the smaller one; the errors become 15.1%, 17.1% and 15.6% without fast tracking, and 14.3%, 15.0% and 13.5% with fast tracking, respectively.

In summary, our evaluation study shows that for edge-assisted monocular depth estimation, with proper frame downsizing and fast tracking, compared to small models, the improved accuracy of large models can offset its longer turnaround time to provide higher average estimation accuracy across frames under both LTE and 5G mmWave.. Our findings highlight the importance of carefully choosing the right DNN model out of the myriad (and ever increasing number) of DNN models proposed by the machine learning researchers in edge-assisted AR.

2 MOTIVATION

We give a brief overview of monocular depth estimation DNN models and motivate the need for offloading them to edge servers.

2.1 Monocular Depth Estimation using DNN Models

Depth estimation aims at estimating a depth map of the surrounding real-world environment of the mobile device, which stores a distance value between the camera and the real-world object for every frame pixel. Depth information has many use cases in AR, *e.g.,* accurately blending virtual objects with physical objects on the screen, or generating shadows of virtual objects and enabling realistic physics simulations. Directly measuring depth data using

[1] The downsizing operation takes only a few milliseconds even on mobile devices (*e.g.,* Pixel 2), which does not reduce the offloading frequency.

sensors has not been widely adopted on commodity mobile devices, because of their high cost or limited measurement range and operating conditions, *e.g.,* up to 5m for LiDAR in iPhone 12 Pro.

In the past few years, monocular depth estimation based on CNN models and using a single RGB camera widely available on commodity mobile device has been proposed and shown to achieve high accuracy [5, 6, 8–12, 16].

More recently, several scale-consistent[2] monocular DNN models have been proposed which lend themselves readily usable for AR on any mobile device with a single RGB camera. We select three representative scale-consistent monocular depth estimation models in our study: (1) DenseDepth follows a standard encoder-decoder network architecture and leverages high-performance pre-trained CNN models to improve accuracy via supervised learning, using large-scale datasets [5]. (2) MonoDepth2 is a self-supervised approach without the need of ground-truth labels but requires training using data collected from stereo cameras [10]. (3) SC-SfmLearner is a fully unsupervised framework where the depth network can be solely learned from monocular videos without using ground-truth depth or stereo image pairs [6].

2.2 Need for Edge Offloading

Table 1 summarizes the characteristics of the three models. To characterize their complexity, we calculate the number of multiply-and-accumulate operations (MACs) per inference (using ptflops [2]) for each DNN model. We also measure the per-frame inference time of each model when executing on a server equipped with Intel Xeon W-2133 CPU and Nvidia GeForce RTX 2080 Ti (used throughout the paper). All three models are trained using the same subset of videos in the KITTI dataset [15] for fair comparison. To measure the estimation error, we use the remaining 12 videos of the KITTI dataset, and calculate the *average depth estimation error* across all frames of each video when applying each model. The per-frame estimation error is calculated as the average pixel-wise absolute relative difference between the ground truth and estimated depth maps.

Table 1 shows that larger models tend to have longer inference time but also higher estimation accuracy. In particular, DenseDepth is the largest model with 191.45×10^9 MACs, compared to MonoDepth2 with 21.45×10^9 MACs and SC-SfmLearner with 18.81×10^9 MACs. As a result, it has the longest inference time of 41.80ms but also the lowest average error of 10.5%; MonoDepth2 has a lower inference latency of 12.34ms but higher average error of 11.6%; and finally SC-SfmLearner has the lowest inference time of 8.18ms but also the highest average error of 13.4%. In the rest of this paper, we refer to the three DNNs as "Large", "Medium" or "Small" DNNs, respectively.

Running DNN models on mobile devices. To evaluate the performance of executing the DNN models on mobile devices, we ported the models to use PyTorch Android API [4], and built a simple application that runs TorchScript to estimate the depth maps from RGB images using the ported models, on a Pixel 2 phone. Table 1 shows that the inference time of the Small, Medium and Large DNNs are 1.57s, 1.64s and 14.08s, respectively. Such long per-frame inference latency severely limits the frame rate of inference and their realtime usage in AR, and motivates the need for offloading to edge servers for potentially much reduced inference time.

[2]Scale-consistent models can predict depth maps for a sequence of RGB frames in a video with the same scale, while scale-inconsistent models predict depth maps with unknown or varying scales and hence are generally not applicable to AR.

Roadmap. Comparing the accuracy of the three DNN models in edge-assisted depth estimation is challenging since the performance of offloading each DNN model to an edge server can be affected by two major factors, (1) offloading frequency which in turn is affected by offloading scheduling, frame downsizing and available network bandwidth, and (2) local optimization for non-offloaded frames. To navigate through these factors, we proceed with our model size/end-to-end accuracy tradeoffs of depth estimation DNN models in three steps, by incrementally considering more factors.

3 BASELINE OFFLOADING SCHEME OVER LTE

We start with a baseline offloading scheme over today's cellular network, LTE, assuming there is only one edge server (GPU) available to one mobile device at a time.

3.1 Tradeoff between Model Accuracy and Offloading Latency

Offloading a computation-intensive task, *e.g.,* a DNN model, for mobile AR requires three steps: (1) uploading a single frame from the mobile device to the edge server over the wireless network, (2) performing DNN inference for the uploaded frame, and (3) transmitting the estimation result back to the mobile device. The end-to-end offloading latency using DNN model m can be modelled as:

$$T_{end-to-end} = T_{ul}(m) + T_{inf}(m) + T_{dl}(m) \qquad (1)$$

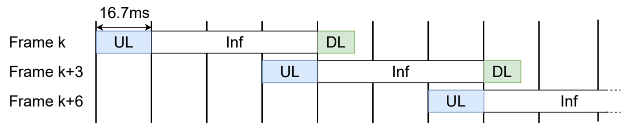
where T_{ul} and T_{dl} denote the time taken in uploading the RGB frame to the server and sending the estimation result to the client, respectively, and T_{inf} is the inference time on the server.

In practice, the end-to-end offloading time can exceed the per-frame duration (*e.g.,* 16.7ms for 60 FPS). For example, if end-to-end offloading takes L frame intervals, then the AR app can only offload every L-th frame to the edge server, and the result of each offloaded frame will be used as the estimate for L future, non-offloaded frames. For example, if frames T, T+L, etc, are offloaded, the result of frame T will come back at T+L-1, and used as the estimation for frames T+L, ... , T+2L-1. As a result, the longer the offloading delay, the more stale the estimation result will be, and more frames will be using that stale result, and hence the lower the average estimation error across the frames of a video will be.

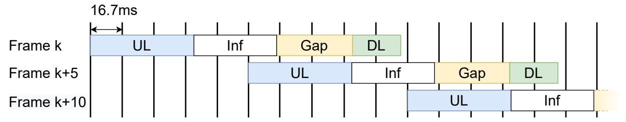
In comparing different-sized DNN models, we expect that a larger DNN model to have longer inference time and network transmission time and hence offloading delay, but also higher accuracy for each offloaded frame. Since their tradeoff on the average error can not be modelled analytically, we resort to experimental evaluation.

3.2 Optimal Offloading Scheduling

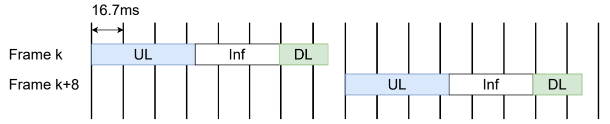
The above discussion suggests that minimizing offloading delay and maximizing offloading frequency will minimize the average error from offloading. Since network transmission and server inference exploit different resources (network and GPU, respectively), pipelining can be used to maximize the offloading frequency. However,



(a) Inference-dominant offloading (average reuse distance: 5)



(b) Network-dominant offloading with pipelining (average reuse distance: 11)



(c) Network-dominant offloading sequentially (average reuse distance: 10.5)

Figure 1: Different offloading scenarios have different offloading scheduling policies (Inf: inference).

depending on the relative latency of the two tasks, pipelining them may also increase the offloading delay. In the following, we separately consider two possible scenarios, and study how to minimize the average error from offloading.

In the first scenario, denoted as *inference-dominant offloading*, the inference latency is longer than the network transmission delay, *i.e.,* $T_{inf} > T_{ul} + T_{dl}$. As shown in Figure 1a, inference for frame k can be stably pipelined with downlink traffic for the last offloaded frame and uplink traffic for the next offloaded frame without affecting the per-frame end-to-end offloading delay, and the maximum offloading frequency is simply determined by the per-frame inference duration on the server, *i.e.,* $f = \left\lceil \frac{T_{inf}}{T_{frame}} \right\rceil^{-1}$, where T_{frame} is per-frame interval, *e.g.,* 16.7ms under 60 FPS.

In the second scenario, denoted as *network-dominant offloading*, the network transmission delay is longer than inference delay, *i.e.,* $T_{ul} + T_{dl} \geq T_{inf}$. To pipeline network transmission with inference, downlink traffic for frame k has to wait for uplink traffic for next offloaded frame to complete, which leads to a gap between inference and downlink transmission for frame k, as shown in Figure 1b. To stably pipeline network-dominant offloading, the total duration of gap and inference should equal that of uplink and downlink transmission for each frame, *i.e.,* $T_{gap}+T_{inf} = T_{ul}+T_{dl}$, where T_{gap} is the duration of the gap, and the maximum offloading frequency f is determined by the total network transmission duration for a single frame, *i.e.,* $f = \left\lceil \frac{T_{ul}+T_{dl}}{T_{frame}} \right\rceil^{-1}$.

However, pipelining network transmission and server inference in this way in network-dominant offloading can result in longer end-to-end offloading delay. An alterative offloading schedule, simply offloading without pipelining, achieves the lowest possible per-offloading end-to-end delay of $T_{ul}+T_{inf}+T_{dl}$, as shown in Figure 1c, which also dictates its maximum offloading frequency to be $f = \left\lceil \frac{T_{ul}+T_{inf}+T_{dl}}{T_{frame}} \right\rceil^{-1}$.

To estimate which offloading schedule (pipelining vs. no pipelining) achieves the minimal average error from offloading for network-dominant offloading, we define *reuse distance* which measures the frame distance between an offloaded frame F_i and each frame F_j that reuses the result of frame F_i. The reuse distance can be calculated using offloading frequency and end-to-end offloading latency. Since one offloaded frame F_i may be reused by multiple future frames, we calculate the *average reuse distance* for each offloading schedule. For a given DNN model, the larger the average reuse distance, the worse the average estimation accuracy across all frames.

In summary, the optimal offloading scheduling works as follows. First, it determines the type of offloading based on estimated latency T_{ul}, T_{dl} and T_{inf}, given network uplink/downlink bandwidth, server compute capability, and the DNN model to be offloaded. If it is inference-dominant offloading, we pipeline network transmission and server inference; if it is network-dominant offloading, we compare the average reuse distance between pipelining and no pipelining, and select the schedule that has the smaller reuse distance.

Table 2: Performance of offloading DNN models under LTE and 5G mmWave.

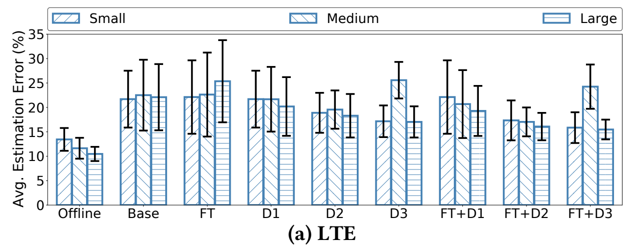
Model	Frame Res. for Net.	Net. Latency (UL/DL) (ms)	Inf. Time (ms)	Offloading Freq.
LTE				
Small	832×256	114.24/45.62	8.18	11
Medium	1024×320	167.68/62.10	12.34	15
Large	1248×384	244.03/85.65	41.80	23
LTE + best frame downsizing				
Small	208×64	21.20/16.91	8.18	3
Medium	416×128	39.81/22.65	12.34	5
Large	208×64	21.20/16.91	41.80	3
5G mmWave				
Small	832×256	40.33/12.39	8.18	4
Medium	1024×320	58.81/15.83	12.34	6
Large	1248×384	85.22/20.75	41.80	9
5G mmWave + best frame downsizing				
Small	416×128	14.58/7.60	8.18	2
Medium	416×128	14.58/7.60	12.34	3
Large	416×128	14.58/7.60	41.80	3

3.3 Experimental Results

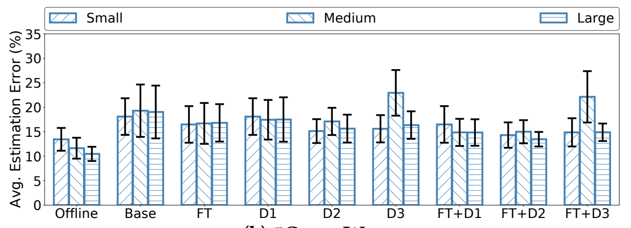
Experimental setup. We emulate offloading RGB frames of the 12 videos to the edge server over LTE by setting the upload/download throughput to be 25Mbps/110Mbps (measured using iperf in the wild) and the RTT to be 30ms (measured between an edge server and the mobile client using traceroute).

Results. We first calculate the optimal offloading frequency. Table 2 shows that Small, Medium and Large models have increasingly smaller offloading frequencies, of once every 11, 15 and 23 frames, respectively, which are primarily determined by the long network transmission latency.

We next evaluate the average accuracy of offloading the three DNN models to the edge across the 12 KITTI videos. Figure 2a shows that compared to the offline model accuracy (Table 1), offloading incurs significant error increase for all three models. The Small DNN achieves the lowest average error of 21.7%, 1.6X higher than



(a) LTE



(b) 5G mmWave

Figure 2: Average absolute relative errors of different offloading approaches to single edge server over LTE and 5G mmWave. Base: offloading alone. FT: offloading with fast tracking. D1, D2 and D3: downsizing frames to 832×256, 416×128 and 208×64, respectively.

its offline accuracy. The Medium and Large DNNs achieve average errors of 22.5% and 22.2%, respectively, which are 1.9X and 2.1X higher than the corresponding offline DNN accuracy. The results suggest that directly reusing the result from the last offloaded frame leads to poor accuracy for non-offloaded frames and hence low average accuracy for edge-assisted AR.

4 HOW DOES LOCAL FAST TRACKING HELP?

Since directly reusing the result from the last offloaded frame leads to poor estimation accuracy for non-offloaded frames, we next study how the idea of local fast tracking (*e.g.,* [7, 13]) which can improve the estimation accuracy of non-offloaded frames affects the relative performance of different-sized DNN models.

Fast tracking. we use a geometry-based method called "warping" to synthesize the depth map of a target frame d_{tgt}, *i.e.,* a non-offloaded frame, using the depth map of a reference frame d_{ref}, *i.e.,* the last offloaded frame, and the relative pose change of the camera between the two frames $p(ref, tgt)$. In detail, d_{ref} is first mapped to points in the 3D space using its pixel values, forming a point cloud that represents the 3D surfaces of the captured objects. d_{tgt} is then inferred from the point cloud via simple linear algebra, using $p(ref, tgt)$ which identifies the viewpoint change from the reference frame to the target frame.

Experimental results. Figure 2a shows that even though fast tracking does not change offloading frequencies, surprisingly, it increases the average error to 22.1%, 22.6% and 25.3% for the Small, Medium, and Large DNNs, respectively. To understand why, we look into the 12 videos. We observe that warping actually increased the average error for about 6 out of 12 videos for each DNN model, which have many frames with new content in the non-offloaded frames not seen in the last offloaded frame, due to fast camera movement or frequent scene change. Since warping relies on the depth in the reference frame to infer the depth for the target frame, it cannot estimate the depth for such new content in non-offloaded frames.

5 HOW DOES FRAME DOWNSIZING HELP?

Since network transmission latency dominates end-to-end offloading latency and hence offloading frequency, we next study how downsizing frames, which reduces network transmission latency for all models, affects the relative performance of different-sized DNN models.

Frame downsizing. Frame downsizing is a lightweight compression algorithm that simply drops pixels uniformly, *e.g.,* every other row and every other column of the image. For every frame to be offloaded, the client first subsamples it to some resolution, then transmits the downsized frame to the server, and then the server upsamples the frame back to the original resolution of the input expected by the DNN model for inference. After the inference, the server downsizes the depth map before sending it back to the client. Frame downsizing reduces the transmitted frame size, but may degrade the frame quality. It is lightweight and does not add to the end-to-end offloading delay.

Since different downsizing resolutions may affect offloading differently, we select three different resolutions, *i.e.,* 832×256, 416×128 and 208×64 (denoted as D1, D2 and D3).

Experimental results. Table 2 shows downsizing frames to the carefully chosen resolution, *i.e.,* 208×64, 416×128 and 208×64, for Small, Medium and Large DNNs, respectively, significantly reduces network transmission latency and improves the offloading frequencies to once every 3, 5 and 3 frames, which result in the highest accuracies without and with fast tracking under LTE for the three DNN models, respectively.

Figure 2a shows that (1) without fast tracking, the Large DNN achieves the lowest error of 17.0% when downsizing frames to 208×64, lower than the best accuracy achieved by the Medium and Small models (19.6% and 17.2%, respectively); (2) With fast tracking, the Large DNN achieves the lowest error of 15.5% when downsizing frames to 208×64, lower than the best accuracies achieved by the Medium and Small models (17.0% and 15.9%, respectively). The results suggest that different DNN models have different sensitivity to frame downsizing. With carefully chosen downsample size, frame downsizing can significantly reduce the end-to-end offloading delay of the Large model without compromising its accuracy, and making it achieve the highest average accuracy among the DNN models.

6 HOW DOES 5G MMWAVE HELP?

Since network transmission latency dominates end-to-end offloading latency and hence offloading frequency, we next study how the emerging faster network, *i.e.,* 5G mmWave, affects the relative performance of different-sized DNN models.

Experimental setup. We repeat the offloading experiments by changing the wireless network from LTE to 5G mmWave. We use the calculated average upload and download bandwidth using traces measured for 5G mmWave [14], of around 73Mbps/520Mbps when the client is walking/driving, and our measured RTT under 5G mmWave using traceroute of 12ms in Downtown Boston.

Results. Table 2 shows that without frame downsizing, 5G mmWave significantly improves the offloading frequencies compared to LTE for all three DNN models, to once every 4, 6 and 9 frames, respectively. With carefully chosen target size for downsizing frames, the offloading frequencies further increase to once every 2, 3 and 3 frames. Figure 2b shows that the improved offloading frequencies from 5G mmWave and downsizing translate into improved average estimation error for all three models: (1) without fast tracking , the average errors of Small, Medium and Large DNNs drop to 15.1%, 17.1% and 15.6%; (2) fast tracking further reduces the average errors to to 14.3%, 15.0% and 13.5%, respectively. In other words, the Large model achieves the highest average accuracy from edge offloading.

7 CONCLUSION

In this paper, we experimentally studied the design tradeoff in picking from different-sized DNN models in edge-assisted monocular depth estimation, an important task of AR. Our results show that for edge-assisted monocular depth estimation, with proper frame downsizing and fast tracking, compared to small models, the improved accuracy of large models can offset its longer turnaround time to provide higher average estimation accuracy across frames, under both LTE and 5G mmWave. In ongoing work, we are studying the same design challenge in offloading other AR tasks, including object detection and pose estimation.

ACKNOWLEDGMENTS

We thank our shepherd Kai Gao and the anonymous reviewers for their helpful comments. This project is supported in part by NSF grant 2112778.

REFERENCES

[1] 2020. Mobile augmented reality (AR) market revenue worldwide from 2019 to 2024. https://www.statista.com/statistics/282453/mobile-augmented-reality-market-size/.

[2] 2021. Flops Counter for Convolutional Networks in Pytorch Framework. https://github.com/sovrasov/flops-counter.pytorch

[3] 2021. Fundamental concepts of ARCore. https://developers.google.com/ar/discover/concepts.

[4] 2021. PYTORCH MOBILE. https://pytorch.org/mobile/android/

[5] Ibraheem Alhashim et al. 2018. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941* (2018).

[6] Jiawang Bian et al. 2019. Unsupervised scale-consistent depth and ego-motion learning from monocular video. *Advances in neural information processing systems* 32 (2019), 35–45.

[7] Tiffany Yu-Han Chen et al. 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] David Eigen et al. 2014. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283* (2014).

[9] Huan Fu et al. 2018. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2002–2011.

[10] Clément Godard et al. 2019. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3828–3838.

[11] Iro Laina, , et al. 2016. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 239–248.

[12] Bo Li et al. 2015. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1119–1127.

[13] Luyang Liu et al. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[14] Arvind Narayanan et al. 2020. Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput. In *Proceedings of the ACM Internet Measurement Conference*. 176–193.

[15] Jonas Uhrig et al. 2017. Sparsity Invariant CNNs. In *International Conference on 3D Vision (3DV)*.

[16] Tinghui Zhou et al. 2017. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1851–1858.