On the Deployability of Augmented Reality Using Embedded Edge Devices

Ayoub Ben Ameur[§], Andrea Araldo
Télécom SudParis - Institut Polytechnique de Paris
{first_name}.{last_name}@telecom-sudparis.com

Francesco Bronzino[§]
Université Savoie Mont Blanc
francesco.bronzino@univ-smb.fr

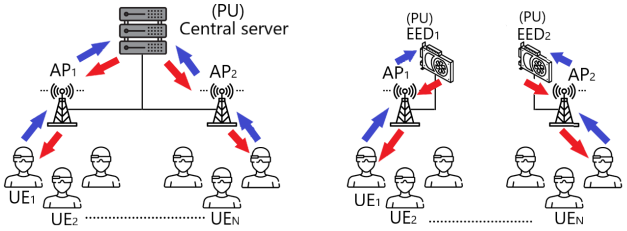
***Abstract*—Edge Computing exploits computational capabilities deployed at the very edge of the network to support applications with low latency requirements. Such capabilities can reside in small embedded devices that integrate dedicated hardware – e.g., a GPU – in a low cost package. But these devices have limited computing capabilities compared to standard server grade equipment. When deploying an Edge Computing based application, understanding whether the available hardware can meet target requirements is key in meeting the expected performance. In this paper, we study the feasibility of deploying Augmented Reality applications using Embedded Edge Devices (EEDs). We compare such deployment approach to one exploiting a standard dedicated server grade machine. Starting from an empirical evaluation of the capabilities of these devices, we propose a simple theoretical model to compare the performance of the two approaches. We then validate such model with NS-3 simulations and study their feasibility. Our results show that there is no one-fits-all solution. If we need to deploy high responsiveness applications, we need a centralized server grade architecture and we can in any case only support very few users. The centralized architecture fails to serve a larger number of users, even when low to mid responsiveness is required. In this case, we need to resort instead to a distributed deployment based on EEDs.**

I. Introduction

The massive growth in popularity of real time applications, e.g., Augmented Reality (AR) and the Internet of Things (IoT), has pushed service providers to part ways from the standard Cloud Computing paradigm and move towards decentralized solutions like Mobile Edge Clouds (MEC) [1]. MECs are more distributed and localized forms of cloud computing that aim to minimize the service response time of deployed applications by offering computing capabilities at the edge of the network instead that from a centralized remote location. Applications benefit from the processing computing advantages of cloud computing without compromising on their real time requirements.

To further carry forward the adoption of MECs, hardware providers have started to develop tiny embedded devices that integrate dedicated computing capabilities, e.g. a GPU or a TPU, at a reduced cost. NVIDIA's Jetson Nano [2] and Google's Coral Dev Unit [3] are just two examples of this trend, providing a good balance between cost and performance. This is in contrast to more expensive approaches that require the deployment of expensive server grade hardware.

[§]Work done while at Nokia Bell Labs France



(a) Centralized architecture (b) Distributed architecture

Fig. 1: The centralized vs. distributed architecture

On the other hand, adopting single board devices for computing intensive applications can introduce a new bottleneck in the edge computing architecture. Even if enabled with dedicated hardware capabilities, these devices provide only reduced computing power which might not be sufficient for computing intensive applications like Deep Neural Networks (DNNs).

In this paper we aim to answer the following question: *can EEDs be a viable solution to support applications with stringent computing and responsiveness requirements?* To understand the adoptability of these systems, we compare two different deployment strategies for edge cloud deployments (Fig.1): (a) a centralized approach that uses shared server grade equipment to support object recognition tasks and (b) a distributed system that exploits EEDs under the same application scenario. We focus our study on a realistic use case, in which User Equipment (UE) devices like Augmented Reality (AR) glasses or smartphones capture videos continuously and send frames to some Processing Units (PUs), which can either be a single server grade hardware or a set of EEDs. In particular, we use object recognition models based on DNN as the target computing task.

Our contributions can be summarized as follows:

1) We develop a simple mathematical model (§IV-A) to study how varying the number of users affects the total performance of the system.
2) To correctly parametrize such a model, we carry on a set of empirical measurements (§IV-B) to evaluate the latency and the accuracy of a state deep learning algorithm for video analytics (e.g., object detection and recognition) on different platforms (i.e. Coral Dev

Board, Jetson Nano, and GPU enabled server).

3) Finally, we validate the analytical model through simulations (§IV-C) in NS3 [4] and conduct the performance comparison (§IV-D) between centralized and distributed solutions in a real life scenario. Our results show that, if we want to support a high number of users, we need to deploy a certain number of EEDs instead of a single central server with high computational capacities when it is the case of an AR application requiring low/mid responsiveness (Fig. 5 and 6). For applications that demand high responsiveness, the only solution is the central server with high computational capacity, which anyways can support very few users.

II. System Description

We consider an AR application, whose aim is to capture videos of the environment, with **UEs** such as smart glasses or phones, and show to the user an appropriate description. This kind of application is envisaged in the context of Industry 4.0, where human operators are provided with information on their visual field, during maintenance or training. It can also be considered for cultural or leisure activities, as in smart museums, to identify artifacts.

A camera on the UE captures a video and continuously sends frames to processing units over the wireless link, via wireless **Access Points** (**APs**). The **Processing Units** (**PUs**) can be either a single **Central Server** or several **EEDs**, e.g., Jetson Nano or Google TPU Boards (Fig. 1). PUs host a pre-trained DNN, which performs object detection and recognition. Finally the classification result (and optionally additional information) is sent back to the users from the PUs through the wireless link, and is shown in the field of view in the UE.

We contrast two architectures: (i) a **Centralized Architecture**, where the PU is one single central server, equipped with a powerful GPU and (ii) a **Distributed Architecture**, where PUs are EEDs, and we deploy one EED per AP.

We want to show in which cases one is to be preferred to the other. We consider three kinds of applications (Table I). They are **Low Responsiveness** (**LR**), **Mid Responsiveness** (**MR**) and **High Responsiveness** (**HR**) applications. We also consider two types of wireless links, namely 802.11.ac [5] and 802.11.ax [6], which we denote with **Low Wireless Capacity** and **High Wireless Capacity**.

III. Related Work

We now review the different solutions proposed in the literature to enable AR. AR is demanding both in terms of complex computation and low latency. Indeed, computation is mostly performed by DNN and the result, e.g., classification, must be sent back to the User Equipment in few milliseconds (Table I). Recent work proposes optimizing DNNs in order to run them directly into the smartphones [9]. However, due to energy and computational constraints of smartphones, the latency is about 600 ms, which is too high for mid and high responsiveness applications (Table I). For this reason, most of the interest has now moved to *offloading* techniques: the computation takes place in a processing unit (PU), different than the UE. While this generally decreases the computation latencies, it also adds a network latency. In some work [10], the PU is represented by the Cloud. However, the latency remains above 400 ms (Fig. 9-12 of [10]), due to the high network latency to reach the Cloud. Moreover, the bandwidth consumed on Metro Area Networks risks to be unmanageable [11].

The Edge Computing (EC) paradigm promises to keep network latency small, thanks to the proximity of PUs and UE. An AR application with object detection, similar to our considered set-up, is studied in [12], where images from the User Equipment (UE) are matched against a database of images to find the most similar. Most of the computation is done by the edge servers, which also cache a part of the database. Only if the correct images are not found in such cache, the video frames are sent to the Cloud to be processed there. According to the authors, their system can achieve 300ms end-to-end latency when the edge server is used by 36 users simultaneously. The focus of [13] is instead a load-balancing algorithm that chooses in which of the available edge servers the video frames from UE should be sent. A limit of [13] is that it needs to solve an optimization problem. Between one resolution and the next, requests reside in a queue, for up to 100 ms. Therefore, this queue-and-optimize approach introduces a non-negligible additional latency which does not fit all the AR applications requirements as shown in Table I. Moreover, they assume to use Edge servers, on average 50 ms away, which, alone, exceeds the most stringent requirements. In our work we tackle instead a wider range of application requirements and, in order to satisfy the most stringent, we do not apply any load-balancing optimization and we place, in our distributed architecture, one processing unit (PU) per access point (AP). Liu et Al. [14] observe that, to adapt to the visual field on the UE, which continuously moving in AR applications, the overall latency must be below 20 ms. To achieve such objective, they propose a set of solutions consisting in performing part of the computation directly in the UE, in order to reduce the amount of information to offload toward the PUs. In this paper, instead, we do not consider any of such optimizations: our UE just needs to capture and sends the video and the entire processing is done in the PUs, in order to save computation and energy burden on battery-limited UE. A novelty of our work is that, while in all the previous work mentioned here, the PUs are fully-fledged servers, we instead consider a fully distributed architecture, where the processing is entirely done on small and cheap (~150$) embedded devices and contrast it with the use of a fully-fledged server as PU. In an alternative architecture proposed by [11], PUs are Cloudlets, i.e., virtual machines deployed at the edge of the network, i.e., in base stations. However, cloudlets can only run on fully-fledged servers. We want to understand, instead, the feasibility of low latency applications only relying on edge AI embedded devices as a replacement of fully-fledged servers.

TABLE I: Augmented Reality requirements

AR requirements	Latency
Low Responsiveness	500 ms [9]
Mid Responsiveness	100 ms
High Responsiveness	16 ms [14]

IV. Performance Evaluation

In order to assess the performance of the Centralized vs. Distribution architectures, we first provide a simple **analytical model** (§IV-A) to obtain equations for the latency and the maximum number of supported users (more complex models [17] will be considered in future work). In order to give realistic values to the model parameters and to assess the classification accuracy of the DNNs on the PU, we perform a **measurement campaign** (§IV-A) on a central server and two types of EEDs available in the market. We then validate our model in **simulation** (§IV-C). We finally find the system capacity and latency (§IV-D).

A. Analytical Model

Let us consider a network with N users and a processing unit (PU). User Equipment (UE) transmits a sequence of frames to one wireless Access Point (AP), which will then route it to the PU. For our analytic model we resort to [13], with the additional simplifying assumption that we do not consider the latency of sending-back the detection and recognition results, as in [15], as we assume they are mainly textual. Note that, as in [13], we do not consider the core network latency to transmit the frames from the AP to the PU, as it is negligible in practical scenarios (lower than 2 ms in [20]). As in [13], we also ignore the effect of congestion in the AP-PU network segment. Therefore, the system latency experienced by a user can be defined as :

$$L = L^w + L^p \tag{1}$$

where L^w is the wireless latency incurred by sending a video frame up to the AP and L^p is the processing latency for object detection and recognition in the PU. We denote with L^{required} the latency required by the application (Table I). Obviously, $L \leq L^{\text{required}}$ must be guaranteed.

1) Wireless Channel characterization: The wireless latency is determined by the user's video frame resolutions and wireless data rates. We denote with R the goodput, i.e., the number of TCP payload bits sent in a second. It has been observed (§ 11.5 of [18]) that when using TCP over a wireless network, the goodput R is approximately insensitive to the number of stations. This is why we keep R constant with respect of the number of UE devices . Moreover, a remarkable property of TCP flows on a wireless link is that data-rate is approximately equally shared among the N users (§ 11.5 of [18]). We can adopt such approximations, since we assume that UEs are relatively close to the AP and there are no obstacles. Therefore, the data-rate of a single UE is $r = \frac{R}{N}$.

A frame can be encoded in different resolutions $k \in \mathcal{K}$. Each resolution corresponds to s_k^2 pixels. The color depth σ is the number of bits employed to encode one pixel. Therefore, the size of a frame is $D_k = \sigma \cdot s_k^2$, in bits and to transmit a frame from the UE to the AP, the wireless latency is is [16]:

$$L^w = \frac{D_k}{r} = \frac{D_k \cdot N}{R} = \frac{\sigma \cdot s_k^2 \cdot N}{R} \tag{2}$$

2) Framerate and maximum number of users: Let us denote with f the framerate, i.e. the number of frames transmitted per second from the UE to the PU. Observe that f does not necessarily correspond to the framerate at which the video is captured, which can indeed be larger than f. Indeed, for some application, we may choose to subsample the video frames and send to the PU only a fraction $p \leq 1$ of the video frames (see §3.1 of [10]). Note also that the deep learning strategies considered in this paper and the related literature we refer to here, take entire frames as input, independent of the video encoding. For instance, if video is encoded in MPEG where there is only one complete frame (I-frame) in a Group of Picture (GoP), since DNNs are only capable of processing I-frames, f will refer to the number of I-frames sent to the PU, independent of the GoP and framerate of the video capture.

If we want to have a response within L^{required} to events detected in the environment, we need to send at least a frame each L^{required}. Therefore, $f \geq 1/L^{\text{required}}$. When N users send D_k bits in a shared wireless channel per each of their frames, then $f \cdot N \cdot D_k \leq R$ and thus $N \leq \frac{R}{D_k \cdot f} \leq \frac{R \cdot L^{\text{required}}}{D_k}$. Therefore, the number of users that can be supported simultaneously is:

$$N_{\max} = \frac{R \cdot L^{\text{required}}}{D_k} \tag{3}$$

3) Computational latency: As in [13], the computational latency to process a single frame is:

$$L^p = \frac{c_k}{F_{\text{pu}}} \cdot N \tag{4}$$

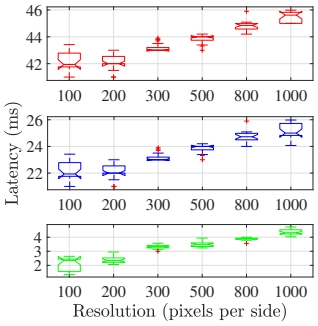
where c_k denotes the *complexity* of the inference for the frame resolution k, F_{pu} the amount of computation resources on the processing unit *pu* (which can be either a central server or a EED, as Jetson Nano or Google TPU board) and N the number of users sharing the PU. c_k depends on the frame resolution s_k^2 following a function $c_k = \psi(s_k)$. For simplicity, we set $\psi_{\text{pu}}(s_k^2) \triangleq \frac{\psi(s_k^2)}{F_{\text{pu}}}$ and rewrite (4) as:

$$L^p = \psi_{\text{pu}}(s_k^2) \cdot N \tag{5}$$

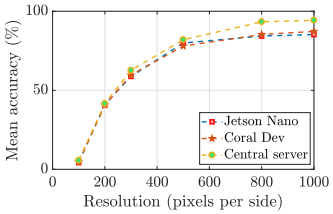
We experimentally measure in §IV-B and Fig. 2a the value of the function $\psi(s_k^2)$, for all resolutions $k \in \mathcal{K}$ and for the three types of PUs considered (Jetson Nano, Coral Dev, Central Server).

B. Measurement Campaign

In order to make our analytical model operational for our comparison of Centralized vs. Distributed architectures, we perform a set of measurements on real devices, in order to fix the value of the resolution k, the bits-per-pixel σ and find the complexity c_k. Since we want our work to be general enough and applicable to different scenarios, where different



(a) Inference time.



(b) Detection mean accuracy

Fig. 2: Experiments results on different platforms

objects may need to be detected, we use the Common Objects in COntext (COCO) dataset [19].

As for the Centralized Architecture, our Central Server is equipped with an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz CPU, 32 GB of RAM and NVIDIA GeForce GTX980 GPU. As for the distributed architecuture, instead,we experimented with two different EEDs: Jetson Nano Developer kit from Nvidia and Coral Dev Board by Google.

We deploy on the PUs a pre-trained DNN called SSD mobilenet v2 [21], which combines SSD for object detection and Mobilenet to classify the detected objects. In order to run such a DNN on Coral Dev and Jetson Nano in an effective way, we convert the regular model to a lighter version using Tensorflow Lite Converter and TensorRT, respectively, with their default configurations. Converting the regular model reduces its file size and introduces optimizations suitable for EEDs. The TensorFlow Lite Converter takes a trained TensorFlow (TensorFlow only) model as input and outputs a TFLite (.tflite) file, a FlatBuffer-based file containing a reduced, binary representation of the original model. On the other hand, TensorRT, built on CUDA, NVIDIA's parallel programming model, enables to optimize inference for all deep learning frameworks by fusing specific Layers and Tensors in one layer which improves the use of GPU memory and bandwidth.

All the measurements of this section are obtained with a color depth $\sigma = 8$ bits/pixel. In fact, we observe that increasing it does not considerably improve accuracy.

The frame resolution k chosen is one of the most important parameters: on the one hand, increasing it improves the classification accuracy, on the other hand it increases the amount of bits to send over the wireless link. Therefore, a good trade-off must be found.

In this paper, we consider 6 frame resolutions: $\mathcal{K} = \{100\text{x}100,\ 200\text{x}200,\ 300\text{x}300,\ 500\text{x}500,\ 800\text{x}800$ and $1000\text{x}1000\}$ pixels. For each frame resolution, we perform object detection+classification inference on 200 images.

Fig. 2a shows the inference latency, i.e., the time spent by the DNN between the instant it receives the frame up to the instant in which it provides the classification. As in [13], we

TABLE II: Inference time fitted curve parameters

Type of PU	a	b
Central server	3.23	$9.56\ e-10$
Coral Dev Board	20.98	$3.37\ e-09$
Jetson Nano	41.10	$7.15\ e-09$

fit the curve $\psi_{\text{pu}}(s_k^2)$ with a cubic regression, i.e., $\psi_{\text{pu}}(s_k^2) = a + bs_k^3$. The values of the coefficients a, b for with the types of PUs considered are given in Table II. The resolution has a negligible impact on the inference time (which is confirmed by the small b coefficients). It is instead very evident that the inference time greatly depends on the type of PU. The Central server is one magnitude faster than Jetson Nano, which remarkably slower than Coral Dev. The goal of this paper is however not to compare different Edge AI platforms. Here we want just to show that it is important to take into account the differences between them, and always starts the study of the performance in such scenarios with a measurement campaign on the real devices considered.

Fig. 2b shows that, in order to achieve a satisfying accuracy, we need a resolution larger than 500p. For this reason, we fix in what follows $k = 600p$. Note also that accuracy with the Central Server is better than with EDDs, which is due to the optimizations applied when converting the DNN to the format suitable for EEDs.

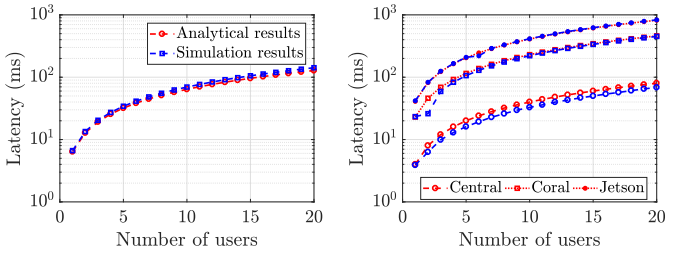
Having fixed k, σ and $c_k = \psi_{\text{pu}}(s_k^2)$, for all the PUs under consideration, we can now use the analytical model of §IV-A to obtain numerical values for the wireless latency (2), computational latency (5) and the maximum number of supported users (3).

Note that our measured accuracy matches the one observed in [13], while results are quite different in what concerns the inference time, which can be explained by the different Edge AI platform considered there. This confirms that a solid performance evaluation of Edge AI solutions should always start from a preliminary measurement campaign on the real devices, as we do here.

C. Simulation setup

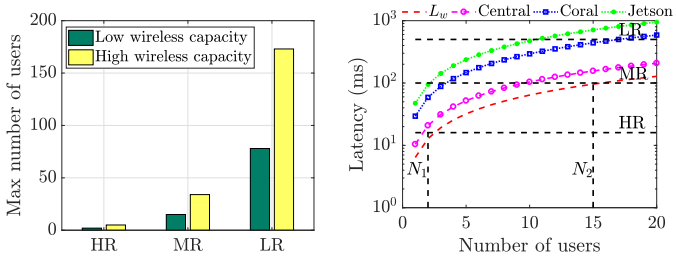
We now validate the analytical model via simulation in NS-3. We model Low and High Wireless capacity with the NS3 implementation of 802.11ac in MIMO 3x4 mode with a 80 Mhz channel and 802.11ax in MIMO 3x4 mode with a 160 Mhz channel width. The goodput is $R = 450$Mbps and $R =$ 1Gbps, respectively (Fig.3 of [7] and Table 1 of [8]).

We first focus on one single AP and N UEs connected to it, in both the Centralized and Distributed scenario, with Low Wireless Capacity. We defer to §IV-D the case of multiple APs and High Wireless Capacity. We model the UEs, the AP and the PUs server with the NS-3 class `NodeContainers` whose individual size corresponds to the number of elements from each class (EEDs, APs and PUs). We measure the wireless latency plus the inference time, varying the number of UEs. Since NS-3 does not model the computational capacity of a server, we model a PU as a queue. This queue has a constant



(a) Wireless performance validation (b) Computation validation

Fig. 3: Validation of the analytical model



(a) Maximum number of users (b) System Latency

Fig. 4: System capacity and latency.

serving time which corresponds, for each platform, to the inference latency for one frame, as measured in §IV-B. Since the timescales we are considering are very short (Table I), much faster than human movement, it is reasonable to simulate, every time, a static snapshot of the system, in which users do not switch from an AP to another. Our UEs send continuously frames to the PU, sending the next frame immediately after receiving the response. The simulation time is 17 minutes. Fig. 3 shows that simulation results match the analytical ones.

We then simulate two real scenarios where we have multiple users in the coverage of multiple APs. As in Fig. 1, for the distributed architecture, we deploy an EED at every AP (in a way that we have one EED per AP) while for the centralized architecture we deploy a single server connected to all of the APs. In each scenario, we let the total number of users vary, as well as the number of APs (the more the APs, the less the users-per-AP). Note that the number of APs corresponds, in the simplest case, to the number of rooms a the museum/exposition or industrial plant.

D. System Capacity and Latency

The capability of sustaining AR applications depends on two factors: the system capacity, i.e., the number of simultaneous users that can be supported, and the system latency, i.e., how fast can each single request be satisfied. The system capacity, on its turn, depends on two elements: the number of users supported by the wireless channel and the processing capability of the PUs. In what follows, we first study the limitations imposed by the wireless channel (§IV-D1), then the system latency (§ IV-D2). Considering system capacity and latency together, we finally study which requirements can be supported by the centralized and distributed architecture (§ IV-D3).

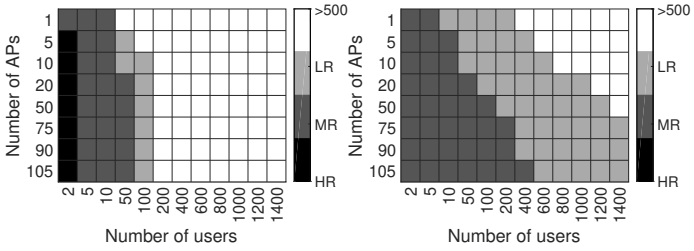
1) Limitations imposed by the wireless channel: The maximum number of users N_{max} per AP supported by the wireless channel can be obtained by (3). Fig. 4a shows N_{max} with Low and High wireless capacity technologies, namely providing $R = 450$ Mbps and $R = 1$ Gbps and with High, Mid and Low responsiveness requirements (HR, MR, LR), i.e., $L^{\text{required}} = \{16, 100, 500\}$ ms (see Table I). Observe that N_{max} only takes into account the limitation imposed by the wireless channel and not by the Processing Units (PUs). This limitation appears the same, in both distributed and centralized architectures. In other words, N_{max} indicates the maximum number of supported users in presence of ideal PUs of infinite capacity.

Fig. 4a shows that the number of users that can be supported by the wireless channel is very low when we need to send a frame every 16 ms (HR) e.g. 2 users with the low wireless capacity and 5 users with the high wireless capacity. The maximum number of users supported by the wireless channel goes to 15 and 34 for the low wireless capacity and the high wireless capacity respectively when we need to send a frame each 100 ms (MR). When it comes to LR (sending a frame every 500 ms), the wireless channel can support more than 75 users for low wireless capacity and more than 150 users for the higher wireless capacity.

2) System Latency: The system latency (see (1)) computed by the analytical model is depicted in Fig. 4b, where we focus on just one AP and we assume that there is one PU, namely a Coral Dev or Jetson Nano or Central server, behind the AP. We also assume Low wireless capacity $R = 450$ Mbps. Note that these results are a lower bound for the centralized architecture, in which there is one PU only for all the APs (see Fig. 1). N_1 and N_2 in Fig. 4b refer to the maximum number of users (see (3)) supported by the wireless channel for High Responsiveness (HR) and Mid Responsiveness (MR) applications, respectively (see Table I). For Low Responsiveness (LR) applications, instead, $N_{\text{max}} > 20$.

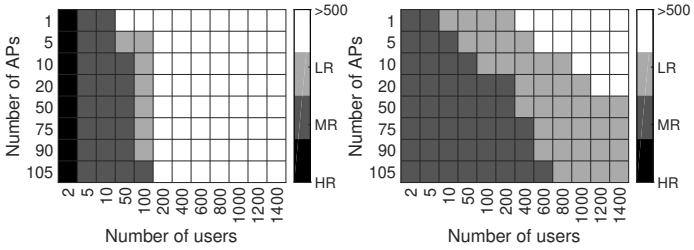
Fig. 4b shows that already by considering the wireless latency L^w only, with more than 2 users, it exceeds the requirements of HR. Adding also the processing time L^p, only one user is supported in HR (and only with the centralized server). However, we can achieve Mid Responsiveness and Low Responsiveness requirements with the distributed architecture but with a limited number of users compared to the centralized architecture.

3) Supported Applications: In Fig. 5 and 6, we show the requirements achievable (Table I) in 4 scenarios, assuming Low and High wireless data rates (§II) and the distributed vs. centralized architectures (Fig. 1). We consider a range of number of users $\#U$ going from 2 users up to 1400 users, which is the number of simultaneous visitors in a big museum as the Louvre. Such users are distributed across different APs, whose number $\#AP$ is varied between 1 and 105. The number of users sharing the wireless channel of a single AP is $N = \lceil \frac{\#U}{\#AP} \rceil$, i.e., the minimum integer bigger than $\frac{\#U}{\#AP}$.



(a) Centralized architecture (b) Distributed architecture

Fig. 5: Achievable requirements for $R = 450$ Mbps.



(a) Centralized architecture (b) Distributed architecture

Fig. 6: Achievable requirements for $R = 1$ Gbps.

A certain requirement L^{required} is satisfied if $N \leq N_{\max}$ (see (3)) and $L = L^w + L^p \leq L^{\text{required}}$. We check such conditions via our analytical model (§IV-A). In Fig. 5 and 6 we represent which of the requirements of Table I is satisfied. The results presented in Fig. 5 and 6 are obtained via NS3 simulation. Furthermore, we verified that they are also matched by the analytical model, which lets us compute the wireless latency L^w, based on (2), and the processing latency via (4):

$$L^p = \begin{cases} \frac{c_k}{F_{\text{pu}}} \cdot \#U & \text{for the centralized architecture} \\ \frac{c_k}{F_{\text{pu}}} \cdot N & \text{for the distributed architecture} \end{cases} \quad (6)$$

The results show that HR applications are only supported by the centralized architecture and only with few users. In all the other situations, the distributed architecture outperforms the centralized and can support MR and LR applications with a relatively large number of users, which is instead infeasible for the centralized architecture. Also observe that, as expected, increasing the number $\#AP$ of access points allows to support more demanding applications, as it reduces contention in the wireless channel in the centralized architecture. This also holds for the distributed architecture, in which the benefit of increasing $\#AP$, and thus also the PUs (see Fig. 1), are more evident, since it also reduces contention in processing (see (6)). Also note that adopting the latest wireless technology with high capacity helps to support more demanding applications.

V. Conclusion and Future Work

This paper presents a comparison of classic centralized systems vs. distributed systems based on Embedded Edge Devices (EEDs) for deploying AR applications with different requirements. We develop an analytical model to represent system capacity and latency of the two alternative systems. We evaluated the performance of deep learning algorithms for video analytics (e.g., object detection and recognition) on such devices via a measurement campaign. We parametrized our analytical model based on such measurements. We developed a NS3 simulator and verify that its results match the analytical model. The analytical model allowed us to study the performance with multiple users up to the order of a thousand.

We show in this paper that for a certain AR applications with high responsiveness requirements, the only solution is a powerful central server. Only few users can be supported. On the other hand, other AR applications with less stringent demands (LR and MR) can be well supported by a distributed architecture based on EEDs. More importantly, only the distributed architecture is able to support a large number of users.

In future work, we will introduce the monetary cost into the comparison of centralized vs. distributed architecture, based on the products available on the market. We will optimize the EED assignment to the AR user by introducing on-line strategies, which take decisions at every new user. Focusing on improving the power consumption management, the challenge is to develop strategies in order to optimize transmission power and offloading data. Another challenge is to explore the performance of other DNN architectures and probably combinations between CNNs and Recurrent Neural Networks.

References

[1] P. Mach, Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", IEEE Comm. Surveys & Tutorials, 2017.
[2] https://developer.nvidia.com/embedded/jetson-nano-developer-kit, 2019.
[3] https://coral.withgoogle.com/docs/dev-board/datasheet/, 2019.
[4] https://www.nsnam.org/
[5] https://standards.ieee.org/standard/802_11ac-2013.html
[6] https://standards.ieee.org/project/802_11ax.html
[7] H. Alakoca, M. Karaca, G. Karabulut Kurt, "Performance of TCP over 802.11ac based WLANs via testbed measurements", ISWCS, 2015.
[8] "Introduction to 802.11ax High-Efficiency Wireless", White Paper, National Instrument Corp., 2020.
[9] Huynh, L. N. et Al., "DeepMon: Mobile GPU-based deep learning framework for continuous vision applications", ACM MobiSys, 2017.
[10] T. Yu-han, L. Ravindranath, S. Deng, P. Bahl and H. Baoakrishnan, "Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors". ACM, 2015.
[11] Satyanarayanan, M. et Al., "Edge analytics in the internet of things", IEEE Pervasive Computing, 2015.
[12] J. Ren et Al., "An Edge-Computing Based Architecture for Mobile Augmented Reality", IEEE Network, 2019.
[13] Q. Liu, S. Huang, J. Opadere, T. Han, "An Edge Network Orchestrator for Mobile Augmented Reality", INFOCOM 2018.
[14] L. Liu, H. Li and M. Gruteser, "Edge Assisted Real-time Object Detection for Mobile Augmented Reality", MobiCom, 2019.
[15] J. Liu and Al. "Delay-optimal computation task scheduling for mobile-edge computing systems", IEEE International Symposium on Information Theory, 2016.
[16] S. Josilo and G. Dan, "Selfish Decentralized Computation Offloading for Mobile Cloud Computing in Dense Wireless Networks". IEEE Transactions on Mobile Computing, 2018.
[17] O. Tickoo and B. Sikdar, "Queueing analysis and delay mitigation in IEEE 802.11 random access MAC based wireless networks", INFOCOM, 2004.
[18] T. Bonald and M. Feuillet, "Performances des reseaux et des systemes informatiques", Institut Télécom et Lavoisier, Paris, 2011.
[19] T. Lin and Al., "Microsoft COCO: Common Objects in Context". CVPR, 2014.
[20] Li. and Al., "An efficient method for evaluating the end-to-end transmission time reliability of a switched Ethernet", JNCA, 2017.
[21] M. Sandler and Al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks", Google Inc, 2019.