A Reality Check on Inference at Mobile Networks Edge

Alejandro Cartas*[1], Martin Kocour°[1], Aravindh Raman‡[1], Ilias Leontiadis★, Jordi Luque★, Nishanth Sastry‡, Jose Nuñez-Martinez★, Diego Perino★, Carlos Segura★

★Telefonica I+D, Research, Spain, *University of Barcelona, Spain, ‡King's College London, UK,
° Brno University of Technology, Czech Republic, [1] Joint first author while intern at Telefonica I+D

Abstract

Edge computing is considered a key enabler to deploy Artificial Intelligence platforms to provide real-time applications such as AR/VR or cognitive assistance. Previous works show computing capabilities deployed very close to the user can actually reduce the end-to-end latency of such interactive applications. Nonetheless, the main performance bottleneck remains in the machine learning inference operation. In this paper, we question some assumptions of these works, as the network location where edge computing is deployed, and considered software architectures within the framework of a couple of popular machine learning tasks. Our experimental evaluation shows that after performance tuning that leverages recent advances in deep learning algorithms and hardware, network latency is now the main bottleneck on end-to-end application performance. We also report that deploying computing capabilities at the first network node still provides latency reduction but, overall, it is not required by all applications. Based on our findings, we overview the requirements and sketch the design of an adaptive architecture for general machine learning inference across edge locations.

CCS Concepts • **Networks → Network measurement**; **Cloud computing**; **Mobile networks**.

Keywords Edge computing, Artificial Intelligence

ACM Reference Format:
Alejandro Cartas, Martin Kocour, Aravindh Raman, Ilias Leontiadis, Jordi Luque, Nishanth Sastry, Jose Nuñez-Martinez, Diego Perino, and Carlos Segura. 2019. A Reality Check on Inference at Mobile Networks Edge. In *2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '19), March 25, 2019, Dresden, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301418.3313946

1 Introduction

Sophisticated Artificial Intelligence (AI) enabled platforms are widely used by several popular real-time services like visual object detection and speech recognition. Despite their high computational and communication requirements, these services are poised to become a critical part of next generation applications such as augmented and virtual reality (AR/VR) or cognitive assistance, to cite a few examples. Such applications usually require low latency for both inference and communication over the network, commonly in the range of few hundreds of milliseconds (e.g., cognitive assistance) or even tens of milliseconds (e.g., AR/VR).

However, inference tasks are still compute intensive, and a low end-to-end delay requirements mean that this inference would need to be performed close to the user, to stay under overall end-to-end delay requirements of such applications. Several recent works [6, 7, 35] have shown that moving computation capabilities and application hosting support towards the edge of the network can significantly reduce network delays for a number of important inference applications. However, most evaluations typically assume that the edge is located on the node closest to the user, typically the eNodeB [1]. Further, they either show that given current hardware, improvement from moving to the edge is negligible for compute-intensive applications [6, 35]. Or, they show that significant gains can be obtained when compute resources are available [7].

In this work, we question these simplistic assumptions. First, from a machine learning perspective, we show that performance tuning, leveraging recent advances can lead to significantly faster inference (§2). This decreased time budget for computation allows increased budget for network latency even with stringent application requirements. In practice, Mobile Network Operators (MNOs) can use this flexibility to deploy computing capabilities at different network sites, from eNodeB to service and packet gateways, or MNO datacenters.

Second, it is naïve to assume uniformly rich computational resources everywhere: For example, it would be extremely capital intensive to equip every eNodeB with high-end servers. Therefore, sites may have different processing capabilities with different deployment costs, and may support different processing frameworks, provide a given latency, and might serve a different number of users. Also, the latency from the end user device to centralized cloud infrastructures can vary from one MNO to another [8].

To evaluate the end-to-end latency, we select a reference network architecture based on the network innovation testbed of a major mobile European operator and perform a measurement analysis for the two use cases of object detection and speech recognition (§3). The locations we consider are part of the internal backbone of MNOs, typically connected via bandwidth overprovisioned links. Our results show that there is indeed a good amount of flexibility, and that the "edge" for some delay-sensitive applications can be more central and better provisioned location than the eNodeB. Based on our results, we advocate a novel adaptive architecture for an edge-based Delivery Network for Applications (eDNA) that we briefly overview (§4). eDNA uses active measurements to dynamically allocate processing resources across different edge locations while meeting application latency requirements.

2 Inference at the network edge

In this section, we first review requirements of applications that could benefit from adding inference capabilities to the edge of the mobile network. A wide range of applications, ranging from smart

EdgeSys '19, March 25, 2019, Dresden, Germany
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6275-7/19/03…$15.00
https://doi.org/10.1145/3301418.3313946

agriculture [32] to connected cars [4] have been proposed. These applications have different needs, with some requiring only low-frequency sampling and little computational complexity, to others that rely on sensor networks intricately tied to a geography, and having heavy computational or network requirements. Even the network requirements can vary, with some (e.g., data analytics) having heavy bandwidth requirements, whereas others (e.g., detecting pedestrians in autonomous cars) are latency sensitive.

Our principal interest is to establish to what extent the "edge" of today's networks can support such diverse machine learning *inference* applications. We, therefore, selected two inference tasks – object detection and speech recognition – which form critical building blocks in several important applications. For edge computing to be practical, tasks such as these need to be supportable within "reasonable" time budgets. In this section, we investigate the computational latency of these inference tasks with different kinds of hardware availabilities. In §3, we consider the network latency imposed by different edge locations.

Based on the network design of a large European ISP, we identify a fundamental trade off when offloading tasks: edge locations that are "more central" are likely to be better provisioned (for example, with more expensive GPU-enabled servers) as compared with computational facilities at the very edge of the network (e.g., eNodeBs). Therefore, we investigate the extent to which computational requirements can be brought down with performance tuning, as well as the adaptability of current inference libraries to a range of hardware capabilities.

2.1 State of the art benchmarks

Most previous works [14, 15, 20, 22] have focused on computer vision (CV) tasks that only process single frames. [20, 22] perform image classification on mobile devices by distributing the computing load locally and on the edge. However, their experiments required hardware such as external mobile GPUs that may not be commonly available. In [14, 15], a personal assistant system is introduced that performs image and speech recognition. Additionally, it leverages a sophisticated natural language processing (NLP) to accommodate a question-and-answer system. In [6, 13], a set of simple applications make use of different CV algorithms. In particular, the Sandwich app [6] processes a video stream using a more computational intensive task than image classification. Although their results show competitive latencies (*i.e.,* few seconds), the use case does not take into account realistic network settings and most recent algorithms (*i.e.,* [6] uses Faster R-CNN on Caffe [18]).

2.2 Can we bring down the processing time?

Consider the task of object detection as an example. Given an image, the task is to find the enclosing boxes of instances from a set of possible categories. For example, enclosing boxes for all flowers and faces in an image. Performance is measured in terms of the resulting mean average precision (mAP) and time taken. In modern Deep Neural Network (DNN) architectures, there is a fundamental trade-off between the two metrics, which depends on the employed meta-architecture and its CNN feature extractors [17]. However, the processing latency could still be too high for real-time inference For example, in the Sandwich app [6], the processing latency of detecting six categories using Faster R-CNN on Caffe [18] is on the order of a few seconds.

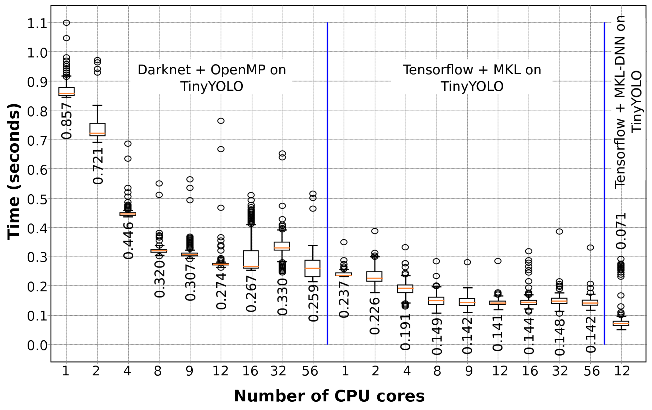


Figure 1. CPU computing time statistics for object detection on 1,796 frames (1 min. video) using different implementations of Tiny YOLO

	TensorFlow	DarkNet
	Tiny YOLO (v1) [9]	Tiny YOLO (v2) [28]
mAP	57.1	23.7
FPS (GPU)	-	244
#Categories	20	80
Dataset	Pascal VOC [12]	COCO [23]

Table 1. Description of the pre-trained object detectors used in our experiments

We set out to improve this with a series of low-level performance fine-tuning. Fig. 1 shows the three main steps, separated by a blue vertical line for each improvement. As a benchmark, we first start by using state-of-the-art object detection system YOLO v2 [28]. Table 1 charts out the accuracy of models and number of categories for object detection as used in Fig. 1. As can be seen, this system is faster than Sandwich, with sub-second latencies. It also scales well, with increased numbers of CPU cores resulting in better performance. However, as noted before, most eNodeB edge nodes are not equipped with 32 or 56 core machines. Investigating the bottleneck, we find that Yolo v2 uses OpenMP, which does not scale well on recent Intel machines. We replace this by using TensorFlow, which by default uses the Intel Math Kernel Library (MKL), which in turn is based on Intel's Threading Building Blocks (TBB) which results in massive performance improvements, even for many fewer CPU cores. Finally, we use a version of MKL that is especially optimised for DNNs, resulting *sub-centisecond* processing times for a reasonably equipped 12-core machine.

In case of automatic speech recognition (ASR), in recent years industry and academia have also moved towards DNN architectures due to its higher accuracy and model training times [29]. Our ASR component relies on a Temporal Delay DNN acoustic model, included as nnet2 recipe in Kaldi speech recognition software [26, 27]. We trained the model for Spanish language using several internal and publicly available databases, accounting for more than 300 hours of speech, an evolution from our previous systems in [25, 33]. Having a deep understanding of the ASR technology allows us to tune specific components, such as the beam-search within the Viterbi hypothesis decoding or the target lexicon, at the expense of word accuracy. Nonetheless, word error rates obtained ensures usability by most NLP applications such as topic detection or intent

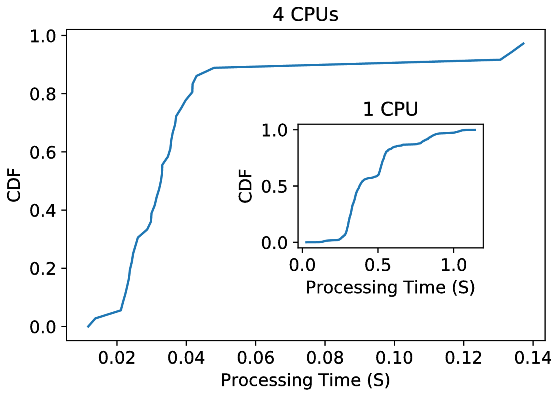


Figure 2. Performance of speech recognition for isolated words in terms of processing delays, when four CPU cores (onset) or just one CPU core (inset) are allocated

classification and entity detection, common tasks in conversational agents. We constrain our experiments just to CPU-based decoding. Fig. 2 depicts number of CPU cycles assigned to a speech worker for decoding an isolated word, a thousand in total, lasting within [0.01, 1] seconds range. The figure reports acceptable delays for conversational or dictation applications [16], lower than one second, just by using one core. Furthermore, comparable delays to those from network are also reported, that is, lower than 100ms by using several CPU cores.

3 Network edge location

We now investigate how network edge location impacts end-to-end latency of applications.

3.1 Experimental Setup

Network architecture We leverage the multi-site mobile testbed of the innovation division of a large European ISP for edge deployments and Amazon web services for conventional cloud-based deployments (Fig. 3).

The setup comprises of an *edge* infrastructure, composed of set of eNodeBs hosted on Intel NUC, connected via optical fiber to a Network Function Virtualization Infrastructure (NFVI) where the mobile core is deployed. Such NFVI represents a Central Office Re-architected as a Datacenter (CORD) architecture, which brings cloud agility to the current Telco central office. The central office is connected via the transport network to the core datacenter, and to the Internet via an edge router after traversing part of the ISP network. Note that the eNodeB, central office and ISP datacenter (ISP-DC) are located in three different locations in Spain. This is representative of a rural deployment where eNodeB and Central Office are in different locations with about 10 ms latency. We focus on this scenario as it is more challenging and diverse than urban environments. Indeed, in an urban deployment the eNodeB is co-located with the Central Office (or very close); in this case the latency between Central Office and eNodeB is negligible and all latency results would be reduced of about 10 ms.

A Nexus-5x Android client placed inside a Faraday Cage (FC) is used to load the application and is connected to the eNodeB through a Universal Software Radio Peripheral (USRP B210) equipped with an 30dBm attenuator and RF cables to reach the FC. FC is used to avoid interference from other signals, making the measurements reliable. Our cloud infrastructure is comprised of Amazon web services (AWS) instances located at three different locations, based on the distance from the edge in Spain - one location is close by (Paris), one far away (US, Ohio) and one mid-way (Ireland) and are hosted on Intel Xeon Platinum processors (4 Cores, 8GB Memory).

Object detection application We developed a web application for object detection that can be used on different devices for several applications, e.g., wearable devices, mobile phones. In this application, a (live) video stream is enriched with labels and bounding boxes of the detected objects. *NewVision [21]* an android client for object detection for visually impaired people, but it is not used for our experiments.

Text-to-Speech application We employed the Kõnele [19] android text2speech application for speech recognition experiments. The application records a user's voice and displays translated text based on trained nnet2 Kaldi online models (§2). The application can replace a classic keyboard for typing a text and is a typical application for cloud-assisted machine learning. For recognition experiments, we also make use of command-line and web clients from Kaldi-GStreamer. Both the clients accept raw audio (2 bytes per sample at 8KHz rate) files as an input and displays transcribed text in real time.

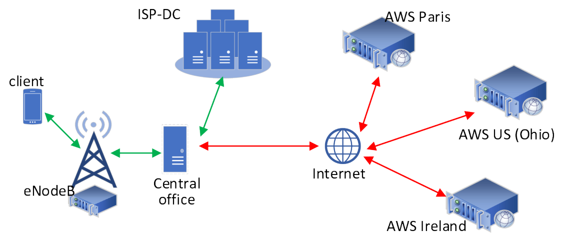


Figure 3. Testbed set-up

Ping application ICMP ping round-trip times (RTT) are unreliable under certain conditions, especially when there is a differential treatment of ICMP traffic. To avoid this issue, we build a simple application which sends ping-pong frames using web-sockets. A client sends a ping message along with the timestamp to the designated server and gets a pong message with the same timestamp as response. RTT is calculated as a difference between these times.

Assumptions and Limitations In order to generate consistent measurements, on every node we run Ubuntu 16.04 and allocate four cores for every container. However, we notice a variation in processing times on different hosts, as a result of differences in clock speed (2.9 GHz on the AWS cloud; 2.5Ghz on edge nodes). Also it is worth noting that processing load is quite a variable factor. While the AWS nodes just host our application, ISP-DC is deployed in a production network and includes load from the other services as well. Further, the testbed does not support local breakout at eNodeBs; latencies reported for eNodeB are inferred as the difference in latency between eNodeB and the central office that we experimentally evaluate. Also, GPUs are available at ISP datacenter and AWS only: latencies reported for GPU processing at other locations are inferred from those measures. Finally, we do not evaluate the bandwidth cost of different services deployed at different network locations. We plan to address these limitations as future work.

3.2 WebSockets Ping-Pong Latency

We start measuring the round-trip times (RTT) from the mobile client to all the listed end-points (cf. Table 2). We observe a 30% RTT difference between the eNodeB and the Central Office. This is a significant difference that could affect application performance if the eNodeB is not co-located at the Central Office. Further, we notice a 35% latency difference between Central Office and centralized cloud infrastructures. Again this is a non negligible latency gap that could affect user experience. The difference is more evident when centralized cloud VMs are located in another continent (*i.e.,* Ohio). Finally, ISP datacenters does not provide latency reduction with respect to geographically close centralized cloud servers.

3.3 Inference Latency

Object Detection Fig. 4(a) shows the split of time-taken between sending the frame and receiving the bounding box (delivery delay) and the time taken to process the frames. It is quite evident from the figure that the overall latency can be brought down to nearly real-time (roughly 400ms). The figure also shows that except edge deployments, the delivery delay exceeds the processing delay (for example, by 10% in case of a Ireland server), indicating that the processing delay plays a vital role in the overall latency. This becomes even more evident when there are additional processing resources. Fig 4(b) shows the overall delay can be reduced to half by using GPU, with a better performance being a result of optimized processing. In this case, the bottleneck is clearly the network delay.

We now focus on two examples of services leveraging object detection to understand how latency impacts perceived user Quality of Experience (QoE). First, we consider a service providing guidance to users to achieve a given goal (e.g., sandwich preparation in [6]). We notice that, independently from the location of the network where processing is performed, the latency is lower than service requirements (The service requirement for this is an upper bound of 0.6s [6], shown as a dashed line in Fig. 4(a)). In this case, processing can be deployed either in ISP datacenters or even in centralized clouds where the deployment cost is minimized while providing good QoE to users. A second example is autonomous car driving, where cars need to identify objects on the street. In this case, network location and resources play a critical role. Indeed, in order to meet the stringent requirements of the service (150ms round trip latency [31]; shown as dashed lines in Fig. 4(b)), processing should run on GPU installed at eNodeBs or Central Office only when co-located with eNodeBs.

Speech Recognition We now focus on the speech recognition task. Fig. 4(c) shows the network delay and processing time for transcribing an audio to text using 4 CPU cores. We keep the total audio length to 5 minutes, and send such audio to the compute engine in a real-time fashion (by sending chunks lasting 250ms and Kaldi-Gstremer [3]). We calculate delivery delay as average latency in sending the audio chunk and receiving the constructed text. Again, it is quite clear that network delay dominates processing delay (even though the decoding of speech happens in CPU and does not use GPU). Indeed, even when the edge is located at the eNodeB, network delay accounts for 90.6% of total delay. The overall delay to US is more than twice of eNodeBs/Central office, even though the processing delay is in sub milliseconds, indicating network as a key factor.

	eNodeB	Central Office	ISP-CDN	Paris	Ireland	Ohio (US)
RTT (ms)	28.13 ±3.66	41.15 ±2.82	62.57 ±8.74	64.10 ±6.37	77.40 ±2.33	151.17 ±7.48

Table 2. Round-trip time from mobile client to different network locations

In order to evaluate latency impact on user QoE, we take into consideration that interacting with an intelligent voice-based assistant (e.g., Siri/Alexa) located at the network edge would require the client transmitting voice samples to a bot in the edge cloud; and the bot transmitting voice packets back which are then played out on speakers by the client. Thus, the *network* latency required for a fluent and sophisticated two way conversation with such bots would be similar to a VoIP conversation between two humans in today's networks. Note that this is a *lower* bound on the delay requirement as the inference task at the bot would also need to perform speech recognition (which we consider here) and also both natural language understanding and synthesis (which are task dependent, and not considered by us). Following [34], we take the maximum delay allowed to be 250ms. We notice service requirements are met only if processing is deployed within the MNO network. However, the service does not require to deploy processing in a specific location, and MNO can thus perform processing placement to minimize deployment cost.

Takeaway: *We empirically verify that network latency is currently the main bottleneck in end-to-end latency in applications performing machine learning inference. We also show that deployment of processing at first network node is not required by all services, and MNOs can leverage other network locations or centralized cloud servers for some use cases.*

4 System design

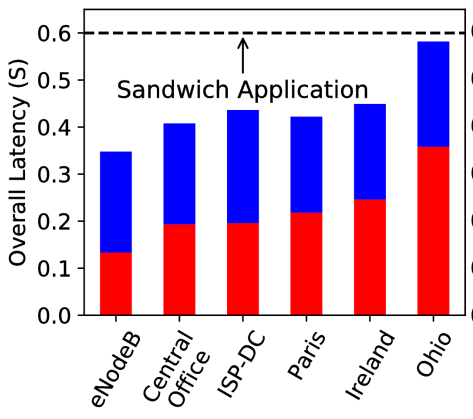
Based on previous findings, we develop an architecture for scalable edge processing. We first motivate our design by discussing the key requirements needed to be satisfied both from an application perspective as well as an infrastructure administration perspective. Then we describe the architecture itself.

4.1 System requirements

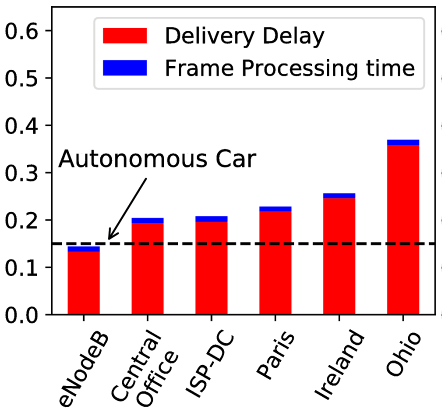
To meet the requirements of applications such as the ones discussed in Sec. 2, and considering findings of Sec. 2-3 we require:

Autoscaling The system should scale automatically (in the order of milliseconds) to allocate and deallocate resources as required by the workload. Furthermore, we require a **lightweight** system deployable in a variety of edge locations, ranging from eNodeBs with low-end hardware to multi-core rackable servers with GPUs.
Real-time processing Application requests should be satisfied instantaneously e.g., the system cannot leverage request batching to perform inference scalably.
Multi user sharing As the number of users increase, it will become difficult, if not impossible, to serve each user with a separate edge computing instance, especially for inference-based applications because sophisticated and accurate machine learning models can require large memories.

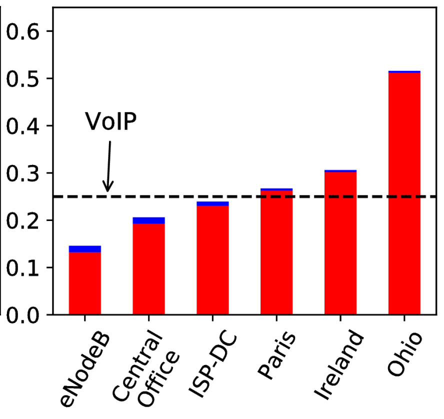
Also, for the diversity of inference applications we consider, we require the ability to process both **audio** and **video** streams. As shown in Table 3, the main existing approaches fall short in



(a) Video Frame Latency (CPU)



(b) Video Frame Latency (GPU)



(c) Speech Frame Latency (CPU)

Figure 4. Overall Latency (stacked as frame delivery and processing delays) for object detection and speech recognition in various locations in the network. Dashed lines stand for the stringent latency requirements for corresponding applications. Frame time for speech correspond to audio chunks of 250ms

Feature	Gabriel	Kaldi	Lucida/ DJNN	Serverless	eDNA
Real time	Yes	Yes	No	Yes	Yes
Multi user	No	Yes	Yes	Yes	Yes
Autoscaling	No	No	No	Yes	Yes
Video stream	Yes	No	Yes	No	Yes
Audio stream	No	Yes	Yes	No	Yes

Table 3. Existing architectures compared to eDNA

one or more aspects: Gabriel [6, 13] uses a VM-based offloading framework that supports video processing efficiently, but being VM-based, cannot be shared across multiple users. It also does not support audio. Lucida [15, 24] and DJNN & Tonic [10, 14] collectively support a large number of applications including image and speech recognition. However, they scale by using batch processing and are therefore not real time. Also, they are optimized for large warehouse-scale computers, and face problems on more modest hardware. Kaldi-Gstremer server [3] does efficient speech recognition but is not designed for real time autoscaling, but research purposes, nor does it support video. However, we use it as a building block for our speech recognizer application and also as a starting point for our object detection.

Serverless computing [5] is an emerging paradigm for deploying autoscaling cloud applications under dynamic workload requirements. Further, recent work suggests that this approach may need tweaking to support network intensive applications [2, 30]. Sand [2] is a recent proposal that by creating fine-grained application sandboxing mechanisms that decrease startup times, and using the sequentiality of edge functions to improve resource utilisation.

4.2 eDNA: (Edge) Delivery Network for Applications

Fig. 5 shows our architecture. Similar to Kaldi [3] and Sand [2], eDNA has a central **Compute Engine** based around a master/worker architecture. Each audio or video service is wrapped as a Docker [11] container. Workers are responsible for processing either audio or video stream and can be chained according to the processing pipelines described in Sec. 2.2. Each physical machine has one master that maintains list of available workers on that machine, and

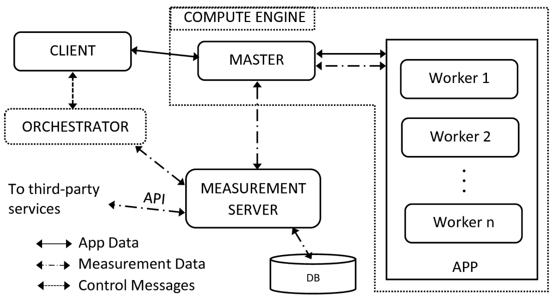


Figure 5. Design of eDNA architecture

can run either in the same container or inside a separate container on remote servers accessible from the worker's host. Building on Kaldi, master-client and master-worker connections use full-duplex communication over WebSockets.

Our improvements over Kaldi and Sand come from two core elements: a *measurement server (MS)* and the *orchestrator*. The MS maintains a database of network locations or hosts where each service can be provided (eg., a service requiring a GPU may need to be in one particular location; a service which needs to be close to a particular user may be constrained geographically, *etc.*). It also takes a network-wide view and collects and stores real-time information about available resources, which are then used by the orchestrator for creating a service function chain to suit the application needs. The MS is built using the Python Flask microframework and uses MongoDB for data storage and retrieval.

The orchestrator keeps track of the client application's requirements, and performs load balancing, assigning a given client to a given master; increasing or decreasing the number of workers associated with each master (depending on resource availability on physical hosts), and managing the resource allocation within each container by starting new processes or shutting down unused ones to suit the current mix of service requirements from clients.

Service flow. Our architecture follows a client-server model. The client (e.g., an object detection or speech recognition application) initiates communication, by retrieving the static content items of the service from the application server, and contacting the orchestrator to register the user for the given service. The orchestrator redirects the user to the best network location or host matching its service requirements, based on resource availability. The orchestrator may also scale the resources allocated to the service container of that host and/or initiate or shut down processes as needed. The client then registers itself with the master of the host selected by the orchestrator. The master then retrieves an available worker from its list and assigns it to the user. Once the connection is successfully established, the client sends the content (e.g., image to be recognised or audio file to be transcribed) to the worker, which then accomplishes the task and returns the answer to the client.

Takeaway: eDNA *achieves **autoscaling** by dynamically scaling the resources allocated to container. There are also no machine learning model startup costs as the model is loaded into memory when the container starts, allowing **real-time processing**. It scales to **multiple users**, and remains **lightweight**, using (typically) a single container per service and one process per user. It can therefore run on low end hardware but can also leverage higher end hardware without architectural changes.*

5 Conclusion

In this paper we revisited some assumptions made by previous works focusing on machine learning inference and edge computing. Specifically, by means of experiments, we show recent advances in Machine Learning, especially on Deep Learning, has enabled inference in few tens/hundreds of milliseconds even for computing intensive applications (e.g., object detection). As a consequence, we showed network latency is now the main bottleneck in end-to-end latency especially when GPU are used. Further, we highlight not all interactive applications require deployment at first network hop. Indeed, some services can be deployed in centralized cloud infrastructures, while others can be served from any location inside the network of Mobile Operators.

Based on these findings, we argue operators should dynamically take advantage of heterogeneous deployment, from less expensive network locations (*i.e.,* Central Office, ISP datacenters or even centralized cloud services), to highly distributed nodes close to users. Further we advocated and designed a novel architecture for dynamic resource/function allocation in such heterogeneous deployments. This work is a first step towards an understanding of machine learning workloads in heterogeneous edge infrastructures that we plan to extend along multiple direction. First, we plan to consider more applications, machine learning functions and network topologies to fully understand different trade-offs. Second, we plan to design and implement algorithms for the automatic orchestration of machine learning functions across heterogeneous edge deployment. Orchestration algorithms will be implemented on the top of existing open source tools, and will drive the definition of metrics to be monitored by the measurement server. Finally, we plan to release the code of eDNA as open source once finalized.

Acknowledgments

The work is partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC) via the Internet of Silicon Retinas Project, Grant No. EP/P022723/1.

References

[1] 3GPP. TS23.501, V15.3.0 (2018-09), Technical Specification Group Services and System Aspects; Study on Architecture for the 5G System; Stage 2, Sept. 2018.
[2] I. E. Akkus, R. Chen, I. Rimac, M. Stein, et al. SAND: Towards high-performance serverless computing. In *Proceedings of the USENIX ATC*, 2018.
[3] T. Alumäe. Full-duplex Speech-to-text System for Estonian. In *Baltic HLT 2014*, Kaunas, Lithuania, 2014.
[4] C. E. Andrade, S. D. Byers, V. Gopalakrishnan, E. Halepovic, D. J. Poole, L. K. Tran, and C. T. Volinsky. Connected cars in cellular network: A measurement study. In *Proceedings of the 2017 IMC*, pages 235–241. ACM, 2017.
[5] I. Baldini, P. Castro, K. Chang, P. Cheng, et al. *Serverless Computing: Current Trends and Open Problems*, pages 1–20. Springer Singapore, Singapore, 2017.
[6] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, et al. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the SEC 2017*, New York, NY, USA. ACM.
[7] J. Cho, K. Sundaresan, R. Mahindra, et al. ACACIA: Context-aware Edge Computing for Continuous Interactive Applications over Mobile Networks. In *Proceedings of the CoNEXT '16*, pages 375–389, New York, NY, USA, 2016. ACM.
[8] CloudHarmony. Transparency for the cloud. https://cloudharmony.com/.
[9] Daniyal Shahrokhian. Tensorflow implementation of You Only Look Once. https://github.com/dshahrokhian/YOLO_tensorflow.
[10] DjiNN and Tonic. DNN as a service. http://djinn.clarity-lab.org.
[11] Docker, Inc. Docker: Enterprise container platform. https://www.docker.com.
[12] M. Everingham, L. Van Gool, C. K. I. Williams, et al. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
[13] K. Ha, Z. Chen, W. Hu, W. Richter, et al. Towards Wearable Cognitive Assistance. In *Proceedings of the MobiSys '14*, pages 68–81, New York, NY, USA, 2014. ACM.
[14] J. Hauswald, Y. Kang, M. A. Laurenzano, et al. DjiNN and Tonic: DNN As a Service and Its Implications for Future Warehouse Scale Computers. *SIGARCH Comput. Archit. News*, 43(3):27–40, June 2015.
[15] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, et al. Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. *SIGPLAN Not.*, 50(4):223–238, Mar. 2015.
[16] T. Holtgraves, S. Ross, C. Weywadt, and T. Han. Perceiving artificial social agents. *Computers in Human Behavior*, 23(5):2163 – 2174, 2007.
[17] J. Huang, V. Rathod, C. Sun, M. Zhu, et al. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *Proceedings of the CVPR 2017*, July 2017.
[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, et al. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.
[19] K. Kaljurand. Kõnele (v1.6.78). http://kaljurand.github.io/K6nele, 2018.
[20] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, et al. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the ASPLOS '17*, pages 615–629, New York, NY, USA, 2017. ACM.
[21] M. Kocour, M. Gabonay, K. Mihalíková, and T. Juřica. Newvision (v1.0). https://gitlab.com/xkocou08/newvision, 2018.
[22] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, et al. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *Proceedings of the IPSN 2016*, pages 1–12, April 2016.
[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, et al. Microsoft coco: Common objects in context. In *Computer Vision – ECCV*. Springer International Publishing, 2014.
[24] Lucida. Speech and vision based intelligent personal assistant. https://github.com/claritylab/lucida.
[25] J. Luque, C. Segura, A. SÃąnchez, M. Umbert, and L. A. Galindo. The role of linguistic and prosodic cues on the prediction of self-reported satisfaction in contact centre phone calls. In *Proc. Interspeech 2017*, pages 2346–2350, 2017.
[26] V. Peddinti, D. Povey, and S. Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Proc. Interspeech*, 2015.
[27] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer, and K. Veselý. The Kaldi Speech Recognition Toolkit. In *Proc. of IEEE ASRU*, 2011.
[28] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242*, 2016.
[29] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.
[30] A. Singhvi, S. Banerjee, Y. Harchol, A. Akella, et al. Granular Computing and Network Intensive Applications: Friends or Foes? In *Proceedings of the Workshop HotNets-XVI*, pages 157–163, New York, NY, USA, 2017. ACM.
[31] S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *nature*, 381(6582):520, 1996.
[32] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. N. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman. FarmBeats: An IoT Platform for Data-Driven Agriculture. In *NSDI*, pages 515–529, 2017.
[33] K. Veselý, C. Segura, I. Szöke, J. Luque, and J. Cernocký. Lightly supervised vs. semi-supervised training of acoustic model on luxembourgish for low-resource automatic speech recognition. In *Proc. Interspeech*, 2018.
[34] VOIP. Qos. https://www.voip-info.org/qos/, 2018.
[35] W. Zhang, B. Han, and P. Hui. On the Networking Challenges of Mobile Augmented Reality. In *Proceedings of the Workshop on VR/AR Network '17*, pages 24–29, New York, NY, USA, 2017. ACM.