Circuits, Systems, and Signal Processing (2020) 39:1059–1070
https://doi.org/10.1007/s00034-019-01226-7

# A Hardware Acceleration Platform for AI-Based Inference at the Edge

Kimon Karras[1] · Evangelos Pallis[1] · George Mastorakis[1] ·
Yannis Nikoloudakis[1] · Jordi Mongay Batalla[3] ·
Constandinos X. Mavromoustakis[2] · Evangelos Markakis[1]

Received: 2 February 2019 / Revised: 17 July 2019 / Accepted: 29 July 2019 / Published online: 8 August 2019

**Abstract**
Machine learning (ML) algorithms are already transforming the way data are collected and processed in the data center, where some form of AI has permeated most areas of computing. The integration of AI algorithms at the edge is the next logical step which is already under investigation. However, harnessing such algorithms at the edge will require more computing power than what current platforms offer. In this paper, we present an FPGA system-on-chip-based architecture that supports the acceleration of ML algorithms in an edge environment. The system supports dynamic deployment of ML functions driven either locally or remotely, thus achieving a remarkable degree of flexibility . We demonstrate the efficacy of this architecture by executing a version of the well-known YOLO classifier which demonstrates competitive performance while requiring a reasonable amount of resources on the device.

**Keywords** Acceleration · YOLO · EDGE · Fog · ML · Acceleration of machine learning · AI · Computing · PCP

## 1 Introduction

The omnipotence and omniscience of cloud computing has been one of the mantras of the last decade and continues unabated until today. At the same time, voices warning of a forthcoming collision between cloud computing and the other big trend of recent years, the Internet of Things, are also multiplying, slowly but steadily.

The solution to this predicament seams to be edge computing, which tries to move part of the workload from the cloud to the edge of the network. Under this overarching theme, one finds many different concepts such as Cloudlets, fog computing and the recently announced multi-access edge computing (MEC). These efforts all share a

✉ Evangelos Markakis
markakis@pasiphae.hmu.gr

Extended author information available on the last page of the article

basic premise, namely the use of virtualized resources at the edge of the network, to perform at least part of the data analysis and storage there.

The main drive behind this is twofold. Its first component is the need to stem the tide of data threatening to overflow the hyperscaled data centers on which the cloud is based as trillions of devices will come online and connect to the Internet over the next years. These devices will all produce an enormous amount of information which will need to be processed in various ways. Some of that processing will need to be done over an aggregated set of data on a large scale (e.g., collecting and analyzing vehicle data over an entire state), but some will have much more limited scope and significance (using the same example, data regarding traffic in specific areas can be analyzed to optimize traffic light patterns). The latter is an example of processing that can be performed locally, close to the data source, thus reducing the data that need to be transferred to a data center and back and thus the network's load.

## 2 Related Work

With the evolution of fifth-generation mobile networking (5G) to be down the wire, bringing a variety of new and old services, the ability to handle and stabilize the "churn" problem of network operators becomes critical. This problem of network operators is strictly interconnected with the understanding of customers and their wide demand for service consumption that until recently was hard to acquire in specific requirements. Following, it becomes evident that we need to support the operator with a decision near the edge that will be assisted by technology which can be cooperative and trusted [19] and at the same time can be helpful to the operator. Subsequently, we need to support 5G services with an assistive decision technology that will be able to assist user intervention in delay-sensitive applications [3,8]. Specific algorithms try to predict utility functions without user intervention [20]. They follow appropriate paradigms to guarantee statistically significant results[15]. There are several available solutions for churn prediction to exploit machine learning techniques [11]. However, these solutions have severe limitations in terms of forecast accuracy and real applicability. Applicability is related to the understanding of the forecast model. The problem is of relevance, if QoE inference is derived as a function of all the information available, such as network state, results of marketing campaigns, contractual, demographic, billing, handset, market, customer survey data and other factors. For instance, in [16], 750 features are jointly analyzed. The amount of information may become even larger if other critical factors are considered, such as cyber-security indicators [1], or in case network monitoring data [5] are directly put as an input to the forecast model. These complex models with a wide variety of features are the rationale behind the so-called big data trend, recently emerged in both the scientific community [22] and the ICT market [4,13]. Mapping this kind of big data structures into intelligible models is a formidable problem. Due to the intensity of the sources [2], it is obvious that we firstly need to process the sources near the data generation "springs" and secondly to be able to handle the overload generated at the edge. The authors in [9,10] and [6] presented some architectures that propose the exploitation of fog installations at the network edge, setting the basis for low end-to-end latency, by providing the necessary computing resources required to deploy and maintain the 5G grade services close to their consumers. Moreover, identifying the ongoing need for extra processing, storage and bandwidth capacity, they go some steps further and propose the use of a field programmable gate array (FPGA) system-on-chip (SoC), which is an integrated circuit that combines processors, programmable fabric and potentially additional logic[7]. This combination allows us to optimally balance the task load by allowing the processors to handle control-dominated tasks, like managing a Fog network and delegating all compute-intensive tasks to the programmable logic. To accomplish this, the programmable fabric needs to be virtualized so that the orchestration environment can deploy the appropriate application on it at any given time. This is accomplished by executing cloud software on the processors of the FPGA SoC, which, together with the specialized hardware, enables the deployment of hardware virtual machines on the programmable logic. These accelerated Fog nodes can help in various services, e.g., health [12] mobile [6] but following and borrowing from the Cloud domain [7] the real support can be in the acceleration of the decision in the edge. In this paper, we present an edge acceleration platform architecture that can be used for accelerating AI-based inference at the edge.

## 3 Edge Acceleration Platform Architecture

Our acceleration platform is based on a small FPGA SoC, a Zynq 7030 device, that presents the optimal trade-off between the flexibility of a CPU and the compute resources provided by an FPGA. Our platform is divided into software and hardware parts with the former running on the ARM-based CPU and the latter on the FPGA fabric. Each of these accommodate a static part and a dynamic part.

The static part is the software and hardware required to enable our acceleration platform to perform its task. This entails the OpenStack worker that is required to receive and deploy tasks, perform any network interface virtualization required and interface with the VM deployed on the platform. The hardware includes all the interconnect required to access data on the shared memory between the CPU and the programmable logic, so as to avoid costly data transfers between CPU and FPGA memory.

The software running on the FPGA SoC's ARM CPUs allows the integration of the platform into OpenStack, which is explained in Sect. 3.2. Finally, the changes that were performed in the OpenStack controller, in order to allow it to utilize programmable logic-based devices, are highlighted in Sect. 3.3. Our edge processing platform strikes one important compromise in order to streamline its design and make it suitable for portable, power-constrained devices. This is to allow for the deployment of only a single virtual machine at any given point in time, thus making the platform single tenant. This was done for several reasons, the foremost of which are to avoid the complications of multi-tenancy in terms of designing and managing the appropriate hardware cores but also since there little incentive to do so in a device as small as the one selected here. In the results section it is proved that the SoC cannot handle in full an entire application much less accommodate more of them.

### 3.1 Hardware

The platform allows for the deployment of tasks onto the programmable logic, which can be swapped in and out by the OpenStack controller as required. To accomplish this, we utilize a feature present in modern FPGA called dynamic partial reconfiguration (DPR). DPR allows part of the programmable fabric to be reprogrammed on the fly, while the remainder of the device continues to operate as intended. We use DPR to update the HW VM that is executed on the programmable logic on the fly. As such, the programmable logic in the programmable cloud platform is further sub-divided into two parts, the static and dynamic ones, a typical arrangement in systems that utilize dynamic partial reconfiguration to change the functionality of a section of the device. This division is shown in Fig. 1.

The static area is ancillary and enables the reconfiguration of the dynamic one, as well as the interconnection of the processor system (PS) with the dynamic area. It contains an AXI interconnection which connects the PS to the programmable logic (PL) as well as the PCAP that performs the reconfiguration of the dynamic area. The AXI Interconnect module contains two channels, one for data traffic, to and from the HW VM and one used for sending monitoring data from the VM to the monitoring SW on the CPU. The AXI interconnect runs at 250MHz and has a width of 64 bits, thus providing enough bandwidth to and from the dynamic area.

The dynamic area is the larger part of the programmable fabric and is where the HW VM is deployed after receiving it over the OpenStack worker. The deployed HW VM contains the user accelerator and a monitoring component as mentioned previously.

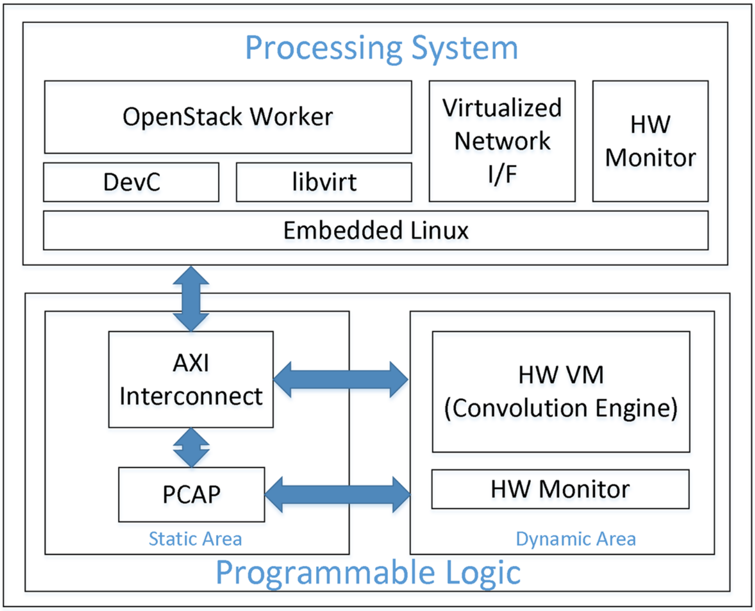

**Fig. 1** Overview of the edge acceleration platform architecture

This component is provided as part of the platform; however, the accelerator designer must feed it with the appropriate monitoring data which will then be sent to the PS.

### 3.2 Software

A large part of the innovation in the programmable cloud platform comes from its software. At the core of the software is the modified programmable-logic-aware OpenStack (compute) worker node which is assisted by low-level software that writes data into the shared memory, the DevC driver that feeds data into the PCAP, an FPGA component that is responsible for performing the actual DPR, and a pseudo-libvirt library that poses as a counterpart to the worker and answers queries regarding the available resources of the device, through an extended API as described in Sect. 3.3. As the FPGA SoC is single tenant, the libvirt is a, respectively, simple component. From a high-level perspective, workers executing a modified version of the compute service client are responsible for managing the VMs. Quite similar to its original purpose, workers can deploy, terminate and reboot VMs, as well as run monitoring services to provide valuable information such as FPGA resource utilization.

An important aspect that must be taken under consideration in the software stack is the synchronization between data traffic and the deployment of a new VM. It is imperative that when the VM is retired and the programmable fabric is to be reconfigured, all pending transactions between the VM and the SW have been completed. This includes all memory transactions for both the actual accelerator and any monitoring data that might be transferred at any given time. If stale data remain in the AXI interconnection, then this will lead to unpredictable behavior from which it might be impossible to recover. The SW stack takes special care of this by synchronizing the threads passing the data from the OpenStack agent on to the fabric. Thus, when a new deployment request comes down for execution, the software is notified and stops receiving new transfers from that point on, while at the same time locking the deployment thread until all outstanding transactions have been completed. The VM image (essentially the bitstream) can then be passed on to DevC driver, which will perform the reconfiguration. The locks will then be released, so traffic can start flowing again to and from the accelerator.

Finally, the software stack includes a monitoring component which reads the monitoring data off the dedicated AXI interconnect, writes them into a local file and transmits them to a remote server over the well-known curl utility, which has been cross-compiled for the Zynq platform.

### 3.3 Making OpenStack FPGA-Aware

When it comes to deploying a VM using a multi-tenant OpenStack topology, the OpenStack compute worker node is only one side of the coin. The other side is the OpenStack controller node which is responsible for running services that direct and manage the deployment of tasks in the pool of worker nodes it supervises. It also provides a single point of access through API services for communication with the other components of OpenStack. Deploying a HW VM correctly requires not only an adapted worker but also and an FPGA-aware OpenStack controller node. Related aspects of the controller's functionality that were taken into account and modified in order to make it FPGA-aware for the needs of the proposed solution are listed below:

- Adaptation of the Scheduler service to recognize, locate and manage programmable logic-based devices and assign HW VMs to them. The allocation of VMs to tenants is done based on several filtering functions that apply criteria to select the most suitable tenant that will host a VM. The selection is done based on a pre-defined set of metrics and information gathered from the monitoring services running on the worker nodes.
- The OpenStack Nova API service has been extended in order to include the calls to platform-specific functions. Toward this direction, additional commands have been added to the API that allow the management of the HW VM images, by identifying and deploying them. So far, the changes are command-line only. It goes without saying the extensions on the Nova API are backward compatible and were implemented with respect to the Nova developers policies. This is accomplished taking advantage of the API microversions framework that allows for specification of the API version that will be used in any circumstance.
- The conductor service, which also runs on the controller node, was slightly adapted in order to maintain access to the default relational database used by OpenStack, for storing stateful information regarding the status of the platform. OpenStack's messaging queue is put to use to an extend that allows the brokerage of our system-specific messages between worker nodes, API service, scheduler and the network service described below.

Another important aspect of deploying VMs over OpenStack is the provision and management of basic networking services. For the allocation of IP addresses, setup and configuration of the virtual networks of the host nodes , a Nova-network service runs on the controller node, which provisions services such as NAT and DHCP. A Nova-network client which executes on the worker node is responsible for configuring the network interface of the respective host. The Nova-network service was selected over the more advanced OpenStack network service (neutron) due to neutron's added complexity and due to the fact that our requirements at this stage of the platform development do not require advanced networking topologies, or services such as load balancers and VPN which are offered by neutron.

Since our platform is a compute-only node, our modifications focused on the Nova component which is responsible for managing compute tasks but also extend to Glance and administers the VM images used to deploy the VMs. Thus, the Glance component was modified to be able to manage HW VMs. Glance can now store these VMs and identify which ones are destined for FPGA SoCs and which are not, avoiding mishaps during deployment.

## 4 YOLO Accelerator

This section briefly introduces the YOLO algorithm, one of the most common and efficient algorithms for object detection and classification and our implementation, which is tailored to suit the needs of our platform. It should be noted that this paper does not aim at detailing the best possible implementation of the YOLO algorithm ever presented on an FPGA. We have purposefully selected a smaller device which is better suited for edge devices than one of the bigger, but much more expensive and power-hungry FPGAs, which have been used in other studies. Furthermore, the highlight of this paper is the platform and not the implementation of the YOLO algorithm itself.

### 4.1 You Only Look Once Primer

Object detection is one of the core visual detection algorithms where the challenge is to not only identify objects on the image but to also locate them, commonly through a bounding box.

YOLO [17] is a well-known neural network that represents a new approach to object detection. Until YOLO came along, object detection was based on existing work that utilized classifiers. YOLO approaches object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Both the classification problem and the detection of the classified objects on the image are handled as one task by one neural network. This allows across the board optimizations which lead to a very fast architecture. The original YOLO network could achieve 45 frames per second, while an streamlined version called Fast YOLO could reach 155 frames per second. It outperforms all other detection methods, including DPM [21] and RCNN [18], by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the People-Art Dataset. We defer further description to the related publications as this does not form the core of our papers.

### 4.2 Algorithm Implementation

Our implementation of the YOLO algorithm is based on [14]. Thus, we apply similar techniques where we reduce the convolution layers of the algorithm to a binary weight format (− 1 to 1) and a 3-bit feature map. These layers form the bulk of the processing as 97% of all the operations in the YOLO algorithm are part of these convolutional layers. The remaining 3% are part of the input and output layers which are not amenable to quantization and can only be trimmed down to an 8-bit fixed-point representation without significant loss in accuracy.

Even at this reduced accuracy, implementing all of the convolution layer of the algorithm in a parallel manner on a small device such as the Zynq 7030 is practically impossible. Thus, we opt to time multiplex the convolution engines in the programmable logic. We have thus created optimized modules for each required convolution layer, and we use dynamic partial reconfiguration, a feature endemic to our platform, to swap them in and out as required. As we will see in Results section, the reconfiguration time is really short (since in this case we are not deploying a new virtual machine each time, but simply adjusting the functionality of the programmable fabric) and thus it is worth optimizing each circuit to reduce power consumption even further in comparison with a more generic convolution engine.

In contrast to the convolution layers, the input and output layers have been implemented as software kernels and run on the A9 processors using the NEON SIMD

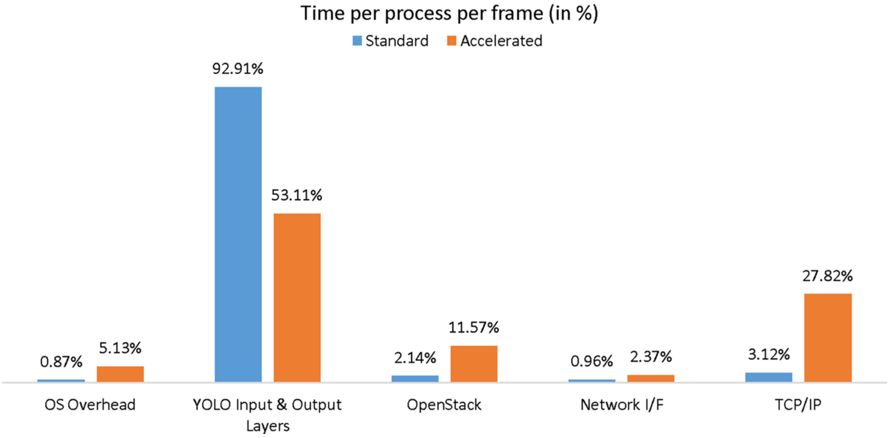

**Fig. 2** Amount of time spent in core SW functions for the accelerated version of YOLO versus the original version of the code

instruction set which offers reasonable performance for this small part of the target application. It was decided that designing these layers in the programmable fabric would not yield sufficient benefit to justify the effort.

## 5 Results

The comprehensive evaluation of our platform entails two aspects. The first ensures that the infrastructure performs its intended task in an efficient manner, and the second is that the implemented solution highlights the efficacy of the architecture.

### 5.1 Platform Results

In order to verify that our platform performs efficiently as the basis for the deployment of ML algorithms on programmable devices, we measured a set of metrics related to the overhead the platform adds, both in software and in the hardware.

The first aspect we need to investigate is whether the overhead caused by the software platform is acceptable. This is critical since we are using a rather weak, dual-core A9 processor whose processing power should be reserved for software component of the deployed VM. Figure 2 illustrates the share of CPU resources when the system is running the YOLO accelerator compared to when the system is running the pure SW version of YOLO. The results illustrate that the bulk of the system resources is available for the accelerator. The next biggest component is the TCP/IP network stack which is part of the Linux OS running on the CPU, followed by the OpenStack worker. Reducing this overhead meaningfully would mean moving some of the TCP functionality of the CPU into the programmable logic. This should be done with caution however since this would eat up resources that could be used for the hardware part of the accelerator.

Next we have to look into the overhead caused by the hardware part of our acceleration platform. This is measured in FPGA compute resources (LUTs, FFs, BRAMs,

**Table 1** Static area resource use

| | LUTs | FFs | BRAMs |
|---|---|---|---|
| Static area | 6738 | 5384 | 2 |
| Total available | 78600 | 157200 | 500 |
| % Used | 8.57 | 3.4 | 0.4 |

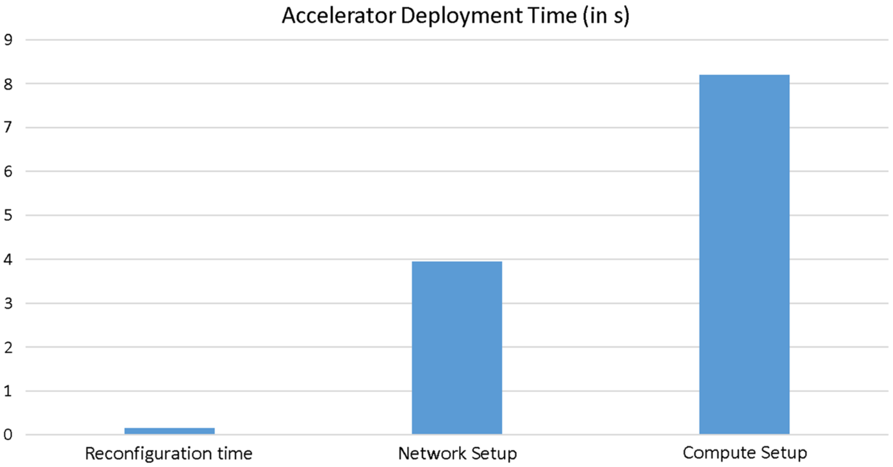

**Fig. 3** Time required to set up a HW/SW virtual machine

etc.) that are used for the static area of the programmable logic and are thus unavailable for use in the dynamic part. Table 1 provides an overhead of the resource use of the static area. It is evident that even in a small FPGA SoC like the Zynq 7030, the overhead incurred is minimal which allows for ample space for the acceleration logic.

The final aspect of the platform metrics we need to evaluate is the total time it takes to setup the VM before it can start performing useful work. Maintaining this as low as possible is important to allow for quick swap in and out of VMs, which enhances the flexibility of the platform. Figure 3 shows a breakdown of the time it takes to set up a VM into its constituent components. The data show clearly that while the total time is not negligible, most of that time is dominated by the network and compute setup in OpenStack and only a small fraction is due to the actual reconfiguration of the FPGA logic. Still, it is clear that reducing this time even further is an important area where future optimization could be performed.

### 5.2 Algorithm Implementation Results

While the focus of this work is not on the algorithm itself but on the platform, it is important to highlight that one is able to implement even a very complex machine learning algorithm on a small FPGA SoC with reasonable results in terms of performance and power consumption.

After applying all our optimizations, the maximum frame rate our implementation of YOLO could achieve was 8fps. This is a significant improvement over executing the same code purely on the A9 CPU which resulted in a meager 0.02fps. While this is not close to 25fps which is the normal video frame rate, it should be sufficient to perform the task required of many embedded edge devices like cameras, where it is not necessary to process all frames to identify the necessary information.

In terms of power consumption, our platform draws a mere 6.57W which is comparable to other solutions found in the literature [14].

## 6 Conclusions

This paper presented a flexible FPGA SoC-based platform that can be used for the remote deployment of VMs that include a programmable logic component onto a small, low-cost, low-power device. This device comprises software and hardware components and allows for the efficient processing of complicated algorithm on our platform. The concept is validated through the deployment of a modified version of the well-known YOLO algorithm and the measurement of both platform-specific and algorithm-related metrics that show that we can remotely deploy a VM to the platform in a timely manner and with low overhead in the range of several milliseconds and that a complex algorithm running on the platform can deliver significant performance benefits (X speed up while consuming Y Watts). Future work will focus on reducing platform overhead, especially in the software area and on expanding it to support other forms of task deployment (e.g., containers).

**Acknowledgements** This work has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation, with Title H2020-FORTIKA "cyber-security Accelerator for trusted SMEs IT Ecosystem" under Grant Agreement No. 740690.

## References

1. A. Abbasi, R.Y. Lau, D.E. Brown, Predicting behavior. IEEE Intell. Syst. **30**(3), 35–43 (2015)
2. J.G. Andrews, S. Buzzi, W. Choi, S.V. Hanly, A. Lozano, A.C. Soong, J.C. Zhang, What will 5G be? IEEE J. Sel. Areas Commun. **32**(6), 1065–1082 (2014)
3. I. Farris, T. Taleb, H. Flinck, A. Iera, Providing ultra-short latency to user-centric 5G applications at the mobile network edge. Trans. Emerg. Telecommun. Technol. **29**(4), e3169 (2018)
4. J. Gazda, P. Tóth, J. Zausinová, M. Vološin, V. Gazda, On the interdependence of the financial market and open access spectrum market in the 5G network. Symmetry **10**(1), 12 (2018)
5. Y. He, F.R. Yu, N. Zhao, H. Yin, H. Yao, R.C. Qiu, Big Data Analytics in Mobile Cellular Networks. IEEE Access **4**, 1985–1996 (2016)
6. S. Jiang, D. He, C. Yang, C. Xu, G. Luo, Y. Chen, Y. Liu, J. Jiang. Accelerating mobile applications at the network edge with software-programmable FPGAs, in *Proceedings—IEEE INFOCOM*, vol. 2018 (IEEE, 2018), pp. 55–62
7. K. Karras, O. Kipouridis, N. Zotos, E. Markakis, G. Bogdos. Enabling virtualized programmable logic resources at the edge and the cloud, in *Hardware Accelerators in Data Centers* (Springer, Cham, 2018), pp. 149–162
8. E. Markakis, E. Pallis, C. Skianis, V. Zacharopoulos. Exploiting peer-to-peer technology for network and resource management in interactive broadcasting environments, in *GLOBECOM—IEEE Global Telecommunications Conference* (IEEE, 2010), pages 1–5

9. E.K. Markakis, K. Karras, A. Sideris, G. Alexiou, E. Pallis, Computing, caching, and communication at the edge: the cornerstone for building a versatile 5G ecosystem. IEEE Commun. Mag. **55**(11), 152–157 (2017)
10. E.K. Markakis, K. Karras, N. Zotos, A. Sideris, T. Moysiadis, A. Corsaro, G. Alexiou, C. Skianis, G. Mastorakis, C.X. Mavromoustakis, E. Pallis, EXEGESIS: extreme edge resource harvesting for a virtualized fog environment. IEEE Commun. Mag. **55**(7), 173–179 (2017)
11. K. Mishra, R. Rani. Churn prediction in telecommunication using machine learning, in *International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017* (IEEE, 2018), pp. 2252–2257
12. R.K. Pathinarupothi, P. Durga, E.S. Rangan, IoT-based smart edge for global health: remote monitoring with severity detection and alerts transmission. IEEE Internet Things J. **6**(2), 2449–2462 (2019)
13. A.R. Prasad, S. Lakshminarayanan, S. Arumugam, Market dynamics and security considerations of 5G. J. ICT Standard. **5**(3), 225–250 (2018)
14. T.B. Preußer, G. Gambardella, N. Fraser, M. Blott. Inference of quantized neural networks on heterogeneous all-programmable devices, in *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, vol. 2018 (IEEE, 2018), pp. 833–838
15. R. Rackwitz, Structural reliability analysis and prediction. Struct. Saf. **23**(2), 194–195 (2002)
16. D. Radosavljevik, P. Van Der Putten. Preventing churn intelecommunications: the forgotten network, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8207LNCS (Springer, Berlin, 2013), pp. 357–368
17. J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You only look once: unified, real-time object detection, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016 (2016), pp. 779–788
18. S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017)
19. M. T. Ribeiro, S. Singh, C. Guestrin. “Why Should I Trust You?”, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD ’16*. (ACM Press, New York, 2016), pp. 1135–1144
20. F. Ricci, B. Shapira, L. Rokach. Recommender systems: introduction and challenges, in *Recommender Systems Handbook*, 2nd edn. chap. 1.2 (Springer, Boston, 2015), pages 1–34
21. J. Yan, Z. Lei, L. Wen, S. Z. Li. The fastest deformable part model for object detection, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 2497–2504
22. K. Zheng, Z. Yang, K. Zhang, P. Chatzimisios, K. Yang, W. Xiang, Big data-driven optimization for mobile networks toward 5G. IEEE Netw. **30**(1), 44–51 (2016)

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Affiliations

**Kimon Karras[1] · Evangelos Pallis[1] · George Mastorakis[1] · Yannis Nikoloudakis[1] · Jordi Mongay Batalla[3] · Constandinos X. Mavromoustakis[2] · Evangelos Markakis[1]**

Kimon Karras
karras@pasiphae.hmu.gr

Evangelos Pallis
pallis@pasiphae.hmu.gr

George Mastorakis
mastorakis@pasiphae.hmu.gr

Yannis Nikoloudakis
nikoloudakis@pasiphae.hmu.gr

Jordi Mongay Batalla
jordim@tele.pw.edu.pl

Constandinos X. Mavromoustakis
mavromoustakis@unic.ac.cy

[1] Hellenic Mediterranean University, Heraklion, Greece

[2] University of Nicosia, Nicosia, Cyprus

[3] National Institute of Telecommunications, Warsaw, Poland