Accelerating Mobile Applications at the Network Edge with Software-Programmable FPGAs

Shuang Jiang*, Dong He†, Chenxi Yang†, Chenren Xu*, Guojie Luo*, Yang Chen†, Yunlu Liu‡, Jiangwei Jiang‡
*Peking University, Beijing, China †Fudan University, Shanghai, China ‡Alibaba Inc., Hangzhou, China

Abstract—**Recently, Edge Computing has emerged as a new computing paradigm dedicated for mobile applications for performance enhancement and energy efficiency purposes. Specifically, it benefits today's interactive applications on power-constrained devices by offloading compute-intensive tasks to the edge nodes which is in close proximity. Meanwhile, Field Programmable Gate Array (FPGA) is well known for its excellence in accelerating compute-intensive tasks such as deep learning algorithms in a high performance and energy efficiency manner due to its hardware-customizable nature. In this paper, we make the first attempt to leverage and combine the advantages of these two, and proposed a new network-assisted computing model, namely FPGA-based edge computing. As a case study, we choose three computer vision (CV)-based interactive mobile applications, and implement their backend computation parts on FPGA. By deploying such application-customized accelerator modules for computation offloading at the network edge, we experimentally demonstrate that this approach can effectively reduce response time for the applications and energy consumption for the entire system in comparison with traditional CPU-based edge/cloud offloading approach.**

I. Introduction

Edge Computing is an emerging network-assisted computing model for optimizing cloud computing by placing computing (and storage) resources at the network edge (*e.g.* WiFi access point, cellular base station) which is in close proximity to mobile or sensor devices. Over the years multiple concepts including "cloudlets" [1], "fog computing" [2], "mobile edge computing" [3] have been proposed under different context but share the common core idea of reducing the dependency on the remote cloud. This computing model has the native advantage of reducing the response time – by reducing the round trip time (RTT) from a multi-hop long end-to-end session across the Internet to a 1-hop wireless segment, it makes the unpredictable network condition not the performance bottleneck any more. This property is very important for today's interactive applications (*e.g.* augmented reality, intelligent personal assistant) which not only rely on compute-intensive algorithms such as deep learning but also request (near) real-time performance. Recent work has experimentally quantified the benefits of edge computing. For instance, by placing VM-based cloudlets [1] at the network edge to accelerate the computational engine, one can achieve lower response time by up to 4.9x compared with cloud offloading, enhance the user experience and save energy consumption for mobile applications that take visual/speech signals as input [4], [5].

On the other side, Field Programmable Gate Array (FPGA) has been proven to be an appealing solution to accelerate compute-intensive workloads. Because of the customizable hardware architecture, all its on-chip available logic blocks can be (re)configured with dedicated pipeline and parallelism design for performance (*i.e.* latency and throughput) enhancement and optimization. As an generic example for data modeling and representation, FPGA-based accelerators for convolutional neural networks (CNN) can achieve 2.0x∼2.5x improvement on AlexNet for image classification in terms of processing time over CPUs [6], [7], [8]. As another example for a specific application, in *Sirius*, an intelligent personal assistant [9], FPGA is used for accelerating the visual/speech signal based workloads running in data centers and reducing the query latency by 16x. Moreover, FPGA has already been employed for computation acceleration in cloud computing as well. For instance, Microsoft has deployed FPGAs in its data centers to improve the throughput of the ranking portion of the Bing web search engine by nearly 2x [10].

Motivated by the advantages of edge offloading and FPGA-based acceleration, we seek to combine the two technologies to further boost the responsiveness performance in edge computing, *i.e.*, deploy FPGA-based accelerators at the network edge to accelerate the mobile applications from the computation offloading perspective. As a case study, we choose three mobile computer vision (CV)-based interactive applications. By offloading the computation from CPU in mobile/cloud to FPGA at the network edge, the FPGA-based edge computing model effectively reduces response time for the applications and energy consumption for the entire system.

Contributions.

- To our best knowledge, this work makes the first attempt to propose the FPGA-based edge computing model. By combining the native advantage of edge offloading and FPGA-based computational acceleration, this model can effectively reduce the response time and energy consumption, and thus benefit mobile interactive applications.
- We design and implement a proof-of-concept FPGA-based edge computing system. Our experimental results demonstrate that our solution can reduce the response time and execution time by up to 3x and 15x respectively over CPU-based edge/cloud offloading in a case study using three CV-based interactive applications.

- We exploit the benefits of energy efficiency in both end device and edge nodes. Our results showcase that our system can achieve up to 29.5% and 16.2% energy efficiency on mobile device and edge nodes respectively.

II. System Description

In this section, we introduce the design and setup of our FPGA-based edge computing system. The overall system architecture is shown is Fig. 1. It consists of a mobile device, an edge node (representing the edge network) and a cloud, which specifications are detailed in Tab. I.

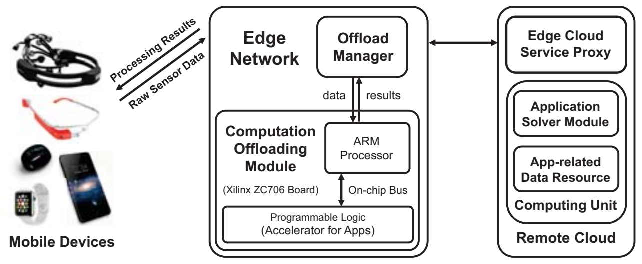


Figure 1. System Architecture for FPGA-based Edge Computing.

Mobile Device (Huawei Mate 9) is connected to the edge node via 802.11n WiFi. It primarily runs the front-end part of the application to interface with the *Computation Offloading* module for sending requests (*e.g.* raw sensor data) and receiving application-specific responses via a UI.

Edge Network is composed of a wireless router (Xiaomi MiWiFi Mini [11]) and an ARM-FPGA board (Xilinx ZC706 [12]) connected via Ethernet. It has two main components, namely *Offload Manager* module and *Computation Offloading* module. The *Offload Manager* module is implemented on the router and used to interface with the frond-end application and route the data to the offloading target (the local *Computation Offloading* module or the remote cloud). In this work, the local *Computation Offloading* module is implemented using FPGA – the ZC706 board has a 2-core ARM Cortex-A9 processor (runs Xilinx Linux 4.4 [13]) and a Zynq-7000 XC7Z045 FPGA connected via a high-speed on-chip bus. The ARM processor runs simple programs to forward data between *Offload Manager* module and the FPGA-based *Computation Offloading* module. The *Computation Offloading* module is implemented using Xilinx SDSoC 2016.2 Tool. Specifically, we analyze the C/C++ implementation of the applications to localize the bottleneck parts, and optimize them by inserting pragmas to configure SDSoC to generate pipelines, unrolled loops and other optimization techniques. As SDSoC only supports a subset of the C/C++ functions, we also need to modify the code to fit this constraint. Then SDSoC invokes High-Level Synthesis (HLS) toolchains to synthesize hardware code (*e.g.* Verilog) from our C/C++ implementation and generates the data motion network between the ARM processor and the FPGA. The core algorithms of the applications are accelerated on the FPGA part.

Table I
Hardware specification of the system participants.

Mobile Device (Huawei Mate 9)	**Edge Network** (Xilinx ZC706 Board)	**Cloud** (in-lab Server)
ARM Cortex A73*	ARM Cortex-A9	Intel Xeon E5-2650
2.4 GHz, 4 cores	1.2 GHz, 2 cores	2.3 GHz, 20 cores**
6 GB RAM	1 GB RAM	64 GB RAM
128 GB Flash	8 GB Flash	2 TB HDD
802.11a/b/g/n/ac WiFi	1 Gbps Ethernet	1 Gbps Ethernet

* The device also has 4 ARM Cortex A53 cores of 1.8 GHz.
** It has 20 physical cores and 40 logic CPUs in total.

Cloud is used to provide the baseline performance of traditional CPU-based cloud offloading. We use our in-lab server to emulate the remote cloud in our experiments. It has an Intel Xeon processor with 40 logic CPUs and the memory size is 64 GB, which is more powerful than most virtual machines provided in commercial cloud computing service. The *Edge Cloud Service Proxy* interfaces with the *Offload Manager* in the edge network for offload the computation task to the *Computing Unit*, which contains both the computation engine and the app-related data (*i.e.* CNN model parameters). To further consider the dynamic network condition (CDN location, load balancing, *etc.*), we use the Linux *netem* tool on the cloud to add delay between mobile devices and cloud.

III. Experimental Approach

In this section, we first introduce the three interactive applications we choose for our case study. Then, we describe our experimental setup for FPGA-based edge offloading, CPU-based cloud offloading and CPU-based edge offloading for performance evaluation and comparison.

A. Applications

Recent years have witnessed the proliferation of mobile interactive applications (*e.g.* Apple's Siri, AiPoly Vision [14], Snapseed [15], MAR AR [16]) powered by the advanced R&D progression in low-power hardware design. For instance, Google glass [17] successfully embeds CPU, image sensor, wireless connectivity, display, battery and anything else into a single glasses temple – this system-level integration effort reaches a milestone that brings mobile computer vision (*e.g.* object detection/recognition)-based application into people's daily life. For this purpose, we primarily choose the following three applications in this domain for our case study. Each application is partitioned into a front-end mobile application for taking input data and a back-end server program that performs the computation. It processes one image at a time.

Digit [18] is an application that recognizes the handwritten digit number in a given image. The program on the back-end server is based on a convolution neural network (CNN)

Table II
ARCHITECTURE OF THE CNN FOR *Digit*

Layer	Input Fmaps	Output Fmaps	Output Dim
Conv1	1	6	28
Pool1	6	6	14
Conv2	6	16	10
Pool2	16	16	5
Conv3	16	120	1
FC	120	10	1

model[1]. It takes a grey-scale image which is resized to 28 $\times$ 28 as input and finds the most probable digit class. The CNN model we employ here is similar to the LeNet-5 [18] architecture, as shown in Tab. II. The parameters have been pre-trained on the MNIST dataset [18]. We implement the server program using C++ and OpenCV [19].

Object [8] is an application that recognizes different objects in a given image. It takes a 3-channel RGB image as input and returns the class name of object in the input image. The recognition algorithm is based on a binarized neural network (BNN) model [20]. A BNN is essentially an extreme CNN whose weights and values of its feature maps are binarized to -1 or +1, which reduces both the storage size and time for inference. Before calculating, the input image is resized to 32 $\times$ 32. The architecture of th neural network model is similar to AlexNet [21], as shown in Tab. III. The parameters have been pre-trained on the CIFAR-10 dataset [22]. Besides the BNN version (we call it *Object-BNN* in the rest of this paper), we also implement a CNN version on back-end server which adopts the same parameters and model architecture (we call it *Object-CNN* in the rest of this paper). The recognition error rate of the CNN version is about 1% lower than BNN. As the binarized operations in BNN can be regarded as an accelerating method for CNN on hardware, we likewise compare the performance of the CPU-based CNN with the FPGA-based BNN.

Face [23] is a face detection application that identifies the coordinates of the rectangles containing human faces. The server takes a 320x240 greyscale image as input and returns an image that marks human faces with rectangles. It implements the Viola Jones algorithm [24] which is based on Haar-like features. This algorithm scans the image in different scales and calculates similarity between the scanning region and the pre-trained Haar feature templates to determine whether there is a face in the region. The server program is implemented using C++ and OpenCV.

These applications are all compute-intensive and are expected to benefit from the edge offloading method. *Digit* applies a relatively simple deep learning model, while *Object* adopts a relatively complex deep learning model; *Face* does not use a deep learning method but is based on a traditional computer vision algorithm. To reduce the network traffic, many applications about computer vision resize the images on the mobile device before sending them to back-end server. Thus size of images sent to the back-end server is much smaller than the initial size, which is about 1 MB or larger. The average request data size and response data size of *Face* are both about 75 KB. The other two applications are similar on the network load, whose average request size is about 1 KB and the response size is less than 15 bytes.

Table III
ARCHITECTURE OF THE BNN AND CNN FOR *Object*

Layer	Input Fmaps	Output Fmaps	Output Dim
Conv1	3	128	32
Conv2	128	128	32
Pool	128	128	16
Conv3	128	256	16
Conv4	256	256	16
Pool	256	256	8
Conv5	256	512	8
Conv6	512	512	8
Pool	512	512	4
FC1	8192	1024	1
FC2	1024	1024	1
FC3	1024	10	1

B. Experimental Setup

The purpose of computation offloading is to accelerate the applications to provide users with better experience. Thus, our experiments focus on one of the most important metrics to mobile application users, namely *response time* (time between the request is sent and the response is received by the mobile device). We also measure the *execution time* (time between the server starts processing and it finishes computing) to observe the speedup of FPGA-based offloading over CPU-based offloading.

1) Edge versus Cloud: First of all, we compare FPGA-based edge offloading with the CPU-based cloud offloading method. In the former case, the server program runs on the FPGA-based edge node and the *Offload Manager* will route the application request towards that. In the latter case, the server program runs in the cloud (*i.e.* our in-lab server). We emulate the cloud latencies by using Linux *netem* tool on the cloud to add delay of 0, 5, 25, 50, 75 (as default[2]), 100 and 150 ms between mobile devices and cloud. We evaluate both of the two offloading methods on the three applications. For each application, we record the response time and execution time of 500 interactions when offloading computation to edge node. In the case of cloud offloading, we also conduct the same measurement with different cloud latencies. We record the response times and execution times of 500 requests for different cloud latencies respectively.

[1]CNN is one type of neural network, a multi-layer machine learning model and is usually used for image classification. It uses pre-trained parameters and input data to accomplish the classification tasks. A CNN consists of different types of layers – *Convolutional Layer* (Conv) does convolution operations to extract features from images. *Pooling Layer* (Pool) performs statistical functions (*e.g.* max, min) to refine the features. *Full-Connected Layer* (FC) uses the features to classify the image. The input and output of each layer are called *Feature Maps* (Fmaps).

[2]Li et al. reported in [25] that the average RTT to their optimal Amazon EC2 instance is 74 ms, which is similar to the typical network condition reported in [5]

2) FPGA versus CPU: To further investigate the efficacy of FPGA-based offloading, we compare FPGA-based edge offloading with the CPU-based edge offloading. We use a laptop with an Intel Core i7-7700 CPU and 8 GB memory as our CPU-based edge node, which is more powerful than the Cloudlet configuration in [5]. We run the same server programs used in the cloud offloading cases on our CPU-based edge node. We conduct the same experiments mentioned before for all the applications.

In addition, we measure the energy consumption of FPGA-based edge and CPU-based edge as well. For the FPGA-based edge, we connect a power monitor to the FPGA board to observe the power usage during the experiments and record a stable value as the result. For the CPU-based case, we use the *powertop* tool to measure the energy consumption when server programs are running.

IV. Experimental Results

In this section, we first demonstrate the performance of FPGA-based edge offloading and CPU-based cloud offloading on the three applications, and then show the comparison between FPGA-based edge offloading and CPU-based edge offloading on response time, execution time and energy consumption. Finally, we present and analyze the comparison between the offloading method and the no-offloading method.

A. Edge versus Cloud

We calculate the average response time and execution time of 500 interactions for all cases of the three applications and the results are shown in Fig. 2.

1) Response Time: We observe that for all the three applications, using the FPGA-based edge as the offloading target brings shorter response time than the CPU-based cloud in general. For the *Digit* application, although it does not show any advantages with FPGA-based acceleration due to the simplicity of computation in terms of execution time, its response time is still 2.9x shorter than the cloud offloading case because of the shorter RTT, even with a delay of 0 ms, as shown in Fig. 2(a). As for the typical cloud offloading case in which delay is 75 ms, the FPGA-based edge shortens the response time by 5.1x. For the *Object* application, Fig. 2(b) and Fig. 2(c) show that with the same network latency, the response time of the BNN server is very close to that of the CNN server. This is reasonable because there is no specific acceleration for the binarized operation on general CPUs. Hence the BNN only reduces storage size of the models but can not be expected to run faster than the traditional CNN. In this application, the FPGA-based edge offloading gains a speedup of 2.3x compared with the 0-ms cloud offloading case and 4.3x compared with the 75-ms case. For the *Face* application (Fig. 2(d)), response time using the FPGA-based edge is 1.5x shorter than the 0-ms case and 4.6x shorter than the 75-ms case.

We also see in Fig. 2 that with the network delay increasing, the response time of CPU-based cloud offloading increases significantly, which may cause bad user experience. By contrast, the network speed between mobile device and the edge node is usually stable and fast. Thus the edge offloading method can also work well even if the network condition in core network is poor.

Table IV
Execution time of edge and cloud (msec)

	Digit	**Face**	**Object-CNN**	**Object-BNN**
FPGA-based edge	2.021	41.63	13.99	13.99
CPU-based cloud	2.137	153.97	202.47	207.52

2) Execution Time: From Fig. 2 we can also see that for each application, the average execution time of CPU-based cloud remains nearly the same with different network delays. This is obvious because network communication should not affect the time for computing. Tab. IV lists the details of execution time for the applications, in which the execution time of CPU-based cloud presented is the case with 0-ms delay. We see that the execution time of *Digit* on the FPGA-based edge is almost the same as that on CPU-based cloud. The main reason is that the neural network architecture in this application is relatively simple and the CPU is efficient enough. The FPGA-based edge offloading brings about an evident speedup of execution time for the other two applications. The *Object* application runs 14.5x faster on the FPGA-based edge than the CPU-based cloud, and the acceleration ratio is 3.2x for the *Face* application. With a well-designed accelerator for a relatively complex algorithm, FPGA can improve the performance of the applications greatly.

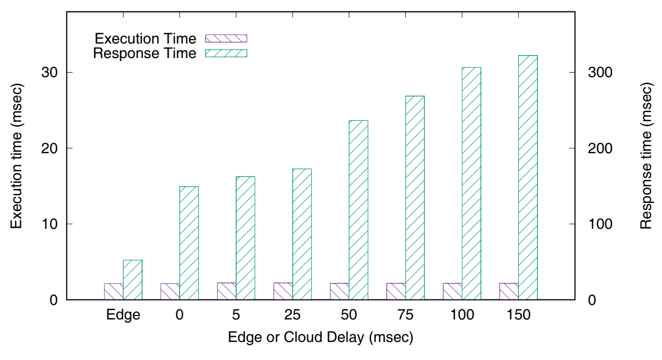
B. FPGA versus CPU

To further demonstrate the advantages of using FPGA, we replace the FPGA-based edge node with a laptop that runs the same server programs and evaluate the performance of the applications in the same way we do in FPGA-based edge offloading case. Also, we measure the energy consumption of the FPGA-based edge and the CPU-based edge. The configuration of the laptop is described in Section III.

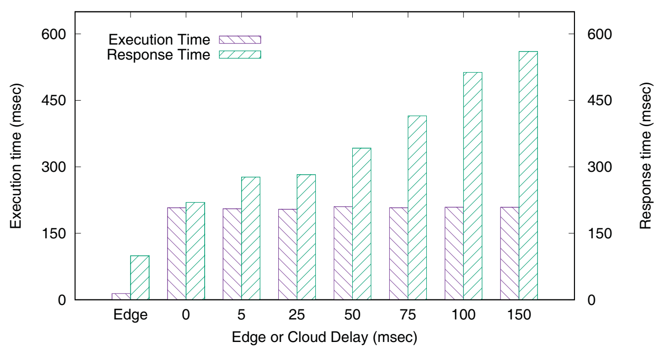
1) Response Time: Fig. 3 shows the results of execution time and response time. We observe that the FPGA-based edge offloading performs better than the CPU-based edge offloading in general. The response time and execution time are nearly the same for *Digit* application using different kinds of edge nodes, for the same reason mentioned before that the server program is relatively simple. Also, the BNN and CNN version server program for *Object* have similar performance on the CPU-based edge as they perform on the CPU-based cloud. The FPGA-based edge reduces the response time by 1.62x for the *Object* application and 1.14x for the *Face* application.

Table V
Execution Time of FPGA and CPU(msec)

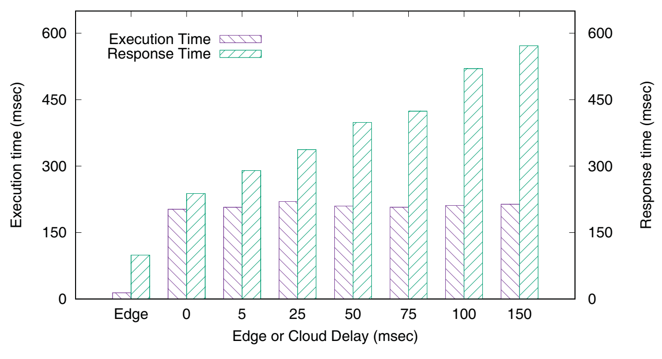
	Digit	**Face**	**Object-CNN**	**Object-BNN**
FPGA	2.021	41.63	13.99	13.99
CPU	2.014	85.12	97.32	99.47



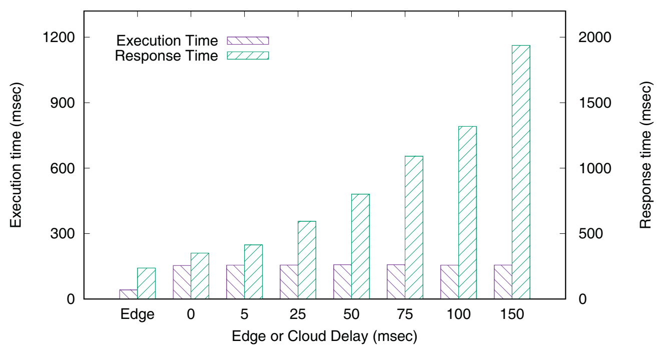
(a) Digit



(b) Object-BNN

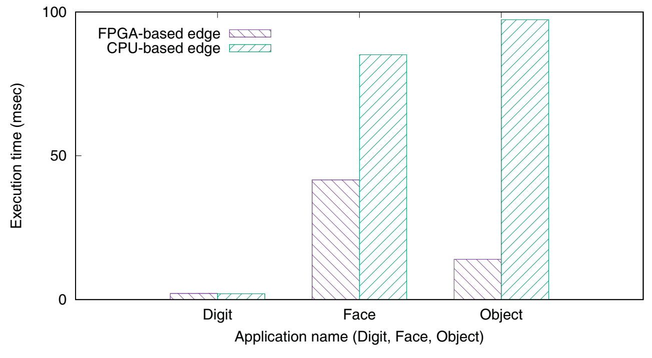


(c) Object-CNN

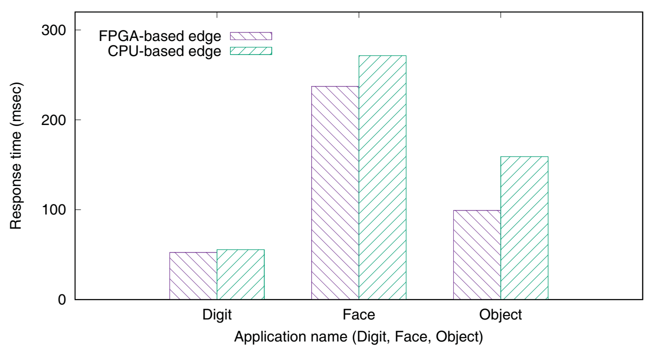


(d) Face

Figure 2. Response time and execution time for the applications in our case study.



(a) Execution Time



(b) Response Time

Figure 3. Response Time and Execution Time for Computation with FPGA-based and CPU-based Edge

2) Execution Time: Tab. V lists the execution time of FPGA and CPU for all the applications. It is reasonable that the execution time of FPGA and CPU for application *Digit* are very short and nearly the same. The execution time of FPGA-based edge is 2.04x shorter than the CPU-based edge for application *Face* and 6.96x for application *Object*. Although our CPU-based edge is powerful, the FPGA-based edge still performs better. We observe from the synthesis report generated by Xilinx SDSoC Tool that for all the accelerators we implement in our experiments, the utilizations of FPGA on-chip resources is less than 35%, which means that when further optimization by leveraging more FPGA on-chip resources, the execution time on FPGA will be further reduced.

3) Energy Consumption: We assume that the energy consumption on the edge router remains unchanged in both the FPGA-based case and the CPU-based case for the same application, because the edge router transfers equal size of data under the same edge network condition during one interaction in both cases. Thus, we do not add energy consumption of the edge router to the total energy consumption. When measuring the energy consumption of the CPU-based edge using the *powertop* tool, we terminate all irrelevant processes to reduce external influence as much as possible.

Fig. 4 shows the energy consumption of both the FPGA-

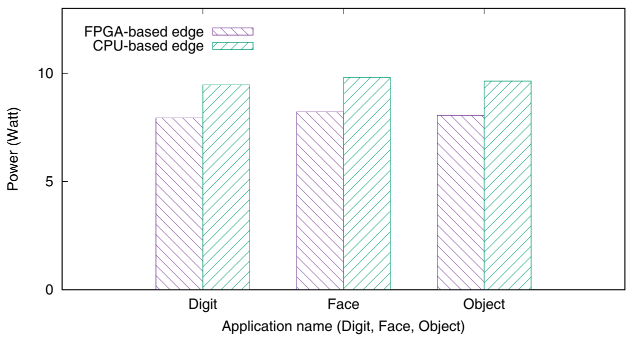


Figure 4. Energy Consumption of the FPGA-based and CPU-based edge

based edge and CPU-based edge for the applications. We observe that for each application, the power consumption of the FPGA-based edge is smaller than that of the CPU-based edge. In particular, using FPGA saves 16.2% of energy for the *Digit* application, 16.1% for *Object* and 15.7% for *Face* compared with CPU. As shown before, the FPGA-based edge runs faster than CPU, so that the FPGA-based approach is more energy-efficient.

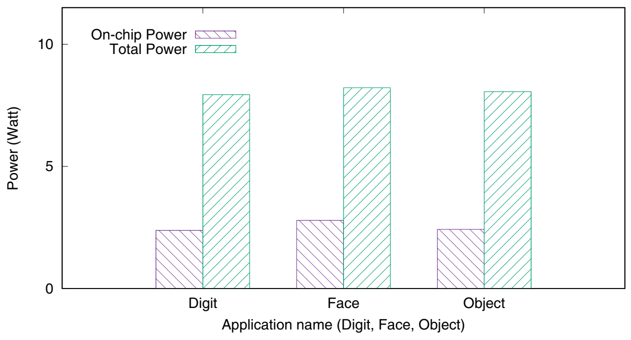


Figure 5. Energy Consumption on the FPGA Board

To more accurately estimate energy consumption of the FPGA-based edge offloading solution, we obtain the FPGA on-chip power using Xilinx Power Estimator 2017.2 which analyzes the synthesis report of Xilinx SDSoC and estimates the on-chip power of FPGA programs. We can see in Fig. 5 that the on-chip power is much lower than the total value, only accounting for 29.8%∼33.9% of the total amount. This is because there are many extra unused components running on our ZC706 board that consume a lot of energy, which is unavoidable on most of today's FPGA development boards. If the unnecessary components can be removed (*e.g.* by designing a new board only with the components for edge computing), the total energy consumption will be further reduced.

C. No-Offloading versus Offloading

1) Response Time: We do not conduct experiments in the scenario that the server program runs locally on the mobile devices (the No-Offloading case). On the one hand, previous research such as [1], [26] and [5] has clearly shown that, in comparison with the No-Offloading case, the response time of computation shortens when offloading them to the CPU-based edge. On the other hand, our work points out that the FPGA-based edge offloading outperforms the CPU-based edge offloading in regard to the response time. Therefore, we can safely conclude that the FPGA-based edge offloading solution will outperform the non-offloading case.

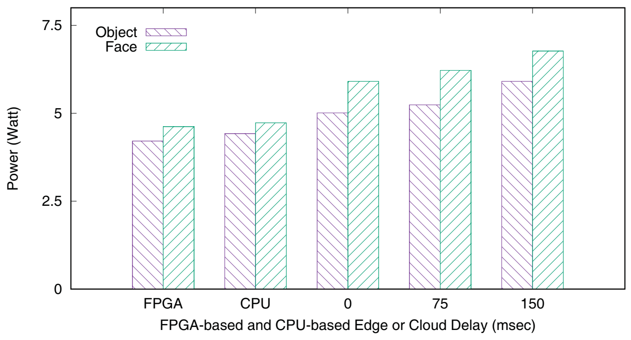


Figure 6. Energy Consumption of the Mobile Device

2) Energy Consumption: When it comes to the energy consumption of mobile devices, the study [5] has pointed out that the energy consumption of the mobile devices is greatly reduced by offloading computation to any either edge or cloud. We measure the power of the mobile device in different offloading cases when running *Object* and *Face* whose response time are relatively longer among the three applications. We use a Rockchip RK3288 board that runs Android 5.1 as the mobile device in this case. The board has a 1.8 GHz ARM Cortex-A17 processor and 2 GB memory. As shown in Fig. 6, edge offloading reduces more energy consumption than cloud offloading for mobile devices. Specifically, FPGA-based edge offloading saves 22.4% and 34.6% of energy for *Object* and *Face* respectively, compared with the 75-ms cloud offloading case. As the network delay increases, offloading to cloud makes the mobile device consume more energy. We also observe that FPGA-based edge offloading and CPU-based edge offloading exert similar influence on the energy consumption of the mobile device.

V. Discussion and Limitations

The experimental results show that FPGA can be a desirable choice in the context of edge computing. FPGA-based edge offloading can effectively reduce the response time of interactive applications. Compared with CPU-based edge offloading, FPGA-based edge offloading also performs better in terms of processing speed and energy consumption. It is convincing that FPGA is an attractive choice for edge computing.

Our core target is to improve user experience for the interactive applications, so we need to know whether the response time we achieved is good enough in terms of human perception. Previous work has derived "tight and loose bounds" of target response time for different types of interactive applications [27]. The tight bound is an ideal target, below which the users are “satisfied”. Above the loose bound, the users become aware of slowness, which means the user experience is significantly impacted. Specifically, for a object recognition task, the response time range should be about 370$\sim$1000 ms to meet the average human performance [27]. We use this result as the target response time range for our three applications, because they are all detection or recognition-related. Fig. 2 shows that all the three applications meet the tight bound in the FPGA-based edge offloading and CPU-based edge offloading case, and FPGA-based edge offloading performs better. As for the typical 75-ms case of CPU-based cloud offloading, *Face* and *Object* only meet the loose bound. With the network delay increasing, user experience will dramatically decrease and finally unable to meet the loose bound. We can conclude that FPGA-based edge offloading meet the requirements of user experience for our applications very well.

In our experiments, there is no comparison between FPGAs and GPUs, another widely-used hardware. In general, GPUs can achieve higher throughput and the peak speed is ususally faster than FPGAs in most cases. However, FPGAs can bring about lower latency for a single request and consume less energy at the same time. Previous work has already shown the comparison of performance between GPUs and FPGAs. The energy consumption of Tesla K40 GPU when running a BNN is about 50 times of FPGA, while the speed to process one image of GPU to run a BNN is only 8 times faster than FPGA [8]. For sliding-window applications, FPGAs can achieve speedup of up to 11x compared to GPUs, while also using less energy [28]. The newest Intel FPGAs can deliver 60% speedup and 2.3x better in performance/watt compared with Titan X GPUs [29]. We can see that FPGA is more energy-efficient in most cases. Besides, FPGA is more flexible than GPU because of its feature of reconfiguration. Considering these aspects, we believe that FPGA is a better choice for edge offloading.

The applications in our experiments are all concerned with computer vision. Yet nowadays, applications involving audio and speech processing are springing up and becoming an important part of interactive applications. The state-of-art solutions for such applications are mostly based on deep learning and machine learning algorithms. And there has already been research on accelerating audio and speech processing applications using FPGAs [30], [9]. Therefore, we believe the FPGA-based edge offloading is also capable of accelerating applications involving audio and speech.

There are still some limitations about our FPGA-based edge offloading method at this stage. First of all, our work does not consider much about the unique features of the network edge. We attempt to “use” FPGA at the network edge to make applications run faster and validate the efficacy of it, rather than optimize the workload considering the unique situation of edge computing. Thus, the technical contribution in this paper is rather limited. We leave trying to optimize performance of the FPGA-based edge considering the unique characteristics of the network edge in our future work. Second, developing efficient FPGA accelerators is difficult. CPU programs are familiar to most programmers and there is a lot of existing CPU-based work for interactive applications. By contrast, developing FPGA programs requires the programmer to have good knowledge on both the application and FPGA. The development cycle is much longer and it is hard to debug hardware programs due to the poor code readability. Although new tools like Xilinx SDSoC greatly reduce the difficulty, there still exists a “gap” between the development of FPGA and CPU programs. Third, the frequency of processors on today’s FPGA boards (usually lower than 1.2 GHz) is much lower than CPUs in laptops or VMs (higher than 2.3 GHz). That is to say, the on-board co-processors may become a bottleneck of FPGA-based edge offloading. Fortunately, these problems can be addressed with the progression of FPGA design tools and hardware performance. For instance, Intel has already developed the “Intel Xeon + FPGA Platform” for data centers [31], which use the powerful Intel Xeon processor as FPGA co-processors for acceleration. We believe that there will be more powerful FPGA-based hardware suitable for edge computing in the future, and that FPGA-based edge offloading will be an important part of edge computing.

VI. Related Work

In general, computation offloading for interactive applications can be accomplished on cloud, network edge and client side (mobile device) using different hardware, including CPU, GPU, FPGA, *etc.*. There are several combinations of the offloading target and the hardware.

Cloud Offloading. Cloud computing is a great step in the history of computer. The MAUI system [32] enables fine-grained offload of mobile code to the infrastructure. Nowadays, many big firms serve as large cloud providers, such as Microsoft, Amazon, Alibaba, *etc.*. Apple’s Siri converges CPU-based cloud offloading for speech recognition, hinting at the rich commercial opportunities in this fast-developing field. However, as the network condition between mobile devices and remote cloud is unreliable, today’s compute-intensive interactive applications with complex machine learning algorithms may lead to high latency and result in bad user experience. Considering this, edge computing can be a better choice. Prior work has investigated the acceleration performance of FPGA and the FPGA-based cloud has been transformed into actual production. The design of *Sirius* shows that GPU-accelerated and FPGA-accelerated servers improve the query latency on average by 10x and 16x respectively. Microsoft has been deploying FPGAs in its Azure cloud [10], creating cloud servers that can be reconfigured to optimize a divergent set of applications and functions.

Edge Offloading. There have been great efforts leveraging the benefits of edge offloading on a variety of applications using the CPU-based edge. The proof-of-concept prototype of VM-based cloudlets [1] suggests that this technology can help meet the requirements for the rapid customization of infrastructure for increasingly diverse applications. Work in [33] rigorously explores the impact of mobile multimedia applications on data center consolidation and offered experimental evidence to support the claim in [1] and [26], where different applications leveraged the VM-based cloudlets to accelerate the processing progress, getting a relatively short response time and good performance.

To the best of our knowledge, no other work provides the detailed investigation of using FPGA for edge offloading. Wenlu *et al.* conduct an overall investigation of edge computing by offloading different compute-intensive applications to CPU-based cloudlets both in WiFi and 4G LTE network [5]. Their work has built up a not merely intensive but also extensive comprehension on edge offloading. On the other hand, there has been a lot of work on accelerating complex algorithms using FPGA, such as [8], [34], [7], *etc.*. Our work leverages FPGA as the target device of edge offloading for the first time and conducts an investigation of its performance.

VII. Conclusion

Edge offloading is attractive to improving the user experience of today's interactive applications, and FPGAs perform very well on accelerating computationally intensive workloads like deep learning algorithms because of its strong computing abilities and energy-efficiency. This paper attempts to combine the advantages of edge offloading and FPGA by deploying FPGAs at the network edge to accelerate interactive mobile applications, and proposed a new network-assisted computing model, namely FPGA-based edge computing. Using the Xilinx SDSoC Tool, we can effectively synthesize C/C++ code into hardware programs to implement the FPGA accelerators for different applications. Our experimental results show that FPGA-based edge offloading can reduce the response time and execution time by up to 3x and 15x respectively compared with CPU-based edge/cloud offloading. What is more, our system can save up to 29.5% and 16.2% of energy consumption on mobile device and edge nodes respectively. While FPGA-based edge offloading is still in its infant stage, we believe that this paper sheds lights on considering to leverage new devices and technologies to improve mobile applications in the context of *Edge Computing*.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *ACM MCC*, 2012.

[3] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[4] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards Wearable Cognitive Assistance," in *ACM MobiSys*, 2014.

[5] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the Impact of Edge Computing on Mobile Applications," in *ACM APSys*, 2016.

[6] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-Layer CNN Accelerators," in *IEEE/ACM MICRO*, 2016.

[7] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *ACM DAC*, 2017.

[8] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," in *ACM FPGA*, 2017.

[9] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, Austinand Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers," in *ACM ASPLOS*, 2015.

[10] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *ACM/IEEE ISCA*, 2014.

[11] "Xiaomi Mi WiFi Router Mini," https://www.mi.com/miwifimini/.

[12] "Xilinx Zynq-7000 All Programmable SoC ZC 706 Evaluation Kit," https:www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html.

[13] "The Official Linux Kernel from Xilinx," https://github.com/Xilinx/linux-xlnx.

[14] "AiPoly - Vision Through Artificial Intelligence," http://www.aipoly.com/.

[15] "Snapseed - Review," https://snapseed.apportal.co/.

[16] G. Takacs, M. E. Choubassi, Y. Wu, and I. Kozintsev, "3D Mobile Augmented Reality in Urban Scenes," in *IEEE ICME*, 2011.

[17] "Google Glass," https://en.wikipedia.org/wiki/Google_Glass.

[18] Y. Lecun, L. Bottou, Y. Bengio, and Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[19] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, 2000.

[20] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *ACM NIPS*, 2016.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *ACM NIPS*, 2012.

[22] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features From Tiny Images," *Technical Report*, 2009.

[23] N. Srivastava, S. Dai, R. Manohar, and Z. Zhang, "Accelerating Face Detection on Programmable SoC Using C-Based Synthesis," in *ACM FPGA*, 2017.

[24] P. Viola and M. Jones, "Robust Real-time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, 2004.

[25] A. Li, X. Ynag, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *ACM IMC*, 2010.

[26] M. Satyanarayanan, P. Simoes, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge Analytics in the Internet of Things," *IEEE Pervasive Computing*, vol. 14, no. 2, 2015.

[27] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance," in *ACM SEC*, 2017.

[28] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-Window Applicatons," in *ACM FPGA*, 2012.

[29] E. Nurvitadhi, G. Venkatesh, S. Jaewoong, D. Marr, and R. e. a. Huang, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?" in *ACM FPGA*, 2017.

[30] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "FPGA-based Low-power Speech Recognition with Recurrent Neural Networks," in *IEEE SiPS*, 2016.

[31] P. Gupta, "Intel Xeon+ FPGA Platform for the Data Center," in *Workshop presentation, Reconfigurable Computing for the Masses, Really*, 2015.

[32] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *ACM MobiSys*, 2010.

[33] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The Impact of Mobile Multimedia Applications on Data Center Consolidation," in *IEEE IC2E*, 2013.

[34] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *ACM FPGA*, 2015.