Cost-effective Hardware Accelerator Recommendation for Edge Computing*

Xingyu Zhou, Robert Canady, Shunxing Bao, Aniruddha Gokhale
Dept of EECS, Vanderbilt University, Nashville,TN
(xingyu.zhou,robert.e.canady, shunxing.bao, a.gokhale)@vanderbilt.edu

Abstract

Hardware accelerator devices have emerged as an alternative to traditional CPUs since they not only help perform computations faster but also consume much less energy than a traditional CPU thereby helping to lower both capex (i.e., procurement costs) and opex (i.e., energy usage). However, since different accelerator technologies can illustrate different traits for different application types that run at the edge, there is a critical need for effective mechanisms that can help developers select the right technology (or a mix of) to use in their context, which is currently lacking. To address this critical need, we propose a recommender system to help users rapidly and cost-effectively select the right hardware accelerator technology for a given compute-intensive task. Our framework comprises the following workflow. First, we collect realistic execution traces of computations on real, single hardware accelerator devices. Second, we utilize these traces to deduce the achievable latencies and amortized costs for device deployments across the cloud-edge spectrum, which in turn provides guidance in selecting the right hardware.

1 Introduction

An issue that is often encountered by developers planning to deploy their applications, particulary those involving data-driven machine learning elements, is what device hardware is best suited (performance and cost-wise) for a given task under varying data processing demands. This question is even more pronounced [26] for resource-constrained edge computing/IoT.

This problem is particularly acute in scenarios where cameras with video streams are involved [1] because compute intensive tasks, especially deep learning based machine learning applications on these data, often require millions of operations for each inference [16]. Thus, to maintain low latencies and compliance with the power sensitivity of edge deployment, smaller models [8] operating on more efficient hardware are preferred [14]. To that end, hardware acceleration technologies, such as field programmable gate arrays (FPGAs), graphical processing units (GPUs) and application-specific integrated circuits (ASICs) among others, have shown significant promise for edge computing [20].

With the proliferation of different accelerator technologies, developers are faced with a significant dilemma: which accelerator technology is best suited for their application needs such that response times are met under different workload variations, the overall cost of the deployment fits their budget and the energy consumption for these devices are below a threshold. This dilemma stems from the fact that different accelerator technologies illustrate different performance and energy consumption traits for different application scenarios. Contemporary techniques that evaluate acceleration technologies are either too specific to a device type and incorporate only a single inference instance level [22,36] or comprise low-level circuit design optimization analysis [12, 30]. Consequently, there remains a significant lack of understanding on the applicability of these accelerator technologies in at-scale, edge-based applications [33] due to diversity of systems settings [24].

In a general sense, selecting a suitable hardware accelerator technology for a given computational task can be regarded as finding out a hardware device placement and optimization strategy for a given topology. Prior work has explored this general problem for service placement for edge computing [18, 27]. From a hardware perspective, workload-specific accelerator placement frameworks for automobiles and unmanned aircraft have been proposed [2, 17, 29]. In these workloads, energy usage is dominated by vehicles rather than computational overhead. From a higher cloud-level point of view, research on device and service placement also exists [3, 37]. Yet,

*This work was supported in part by AFOSR DDDAS FA9550-18-1-0126 program. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the sponsor.

widespread adoption of these different hardware platforms requires more systematic understanding of both spatial and temporal impacts in realistic deployments in a network with compute-intensive tasks.

To address these challenges, we present *HARE (Hardware Accelerator Recommender for the Edge)*, which is a framework to help users rapidly and cost-effectively select the right hardware accelerator for a given compute-intensive task. Our work focuses on a general framework to model compute latency and energy usage across accelerator devices, and makes the following contributions:

- We present a novel hardware-software co-evaluation framework that formalizes the performance comparison involving training samples obtained on real hardware.
- We develop a simple approximation method that can be utilized to implement long-term cost analysis for hardware-based machine learning inference behavior at the edge.
- We show case studies involving two realistic machine learning application settings and demonstrate HARE in these contexts.

The remainder of the paper is organized as follows: Section 2 describes the design of the HARE hardware accelerator evaluation framework; Section 3 provides empirical experiment design and evaluation results of applications across different hardware platforms; and Section 4 concludes the paper.

2 Recommender System Methodology

We now present technical details about HARE. In realizing it, we faced several challenges. The broad range of hardware options and settings makes performance quantification on the different device types very difficult. There are numerous hardware-related metrics and we had to decide which metrics were most significant and how to depict hardware acceleration patterns in a compact manner. Finally, for at-scale deployment of devices, the overall system performance is not just a simple addition of individual performance cases.

Our approach to realizing HARE uses the following steps: (1) Conduct performance and energy benchmarking in isolation per device type, and (2) Use these insights to approximate the performance, energy and cost metrics for at-scale deployments, which is then used as the recommendation engine.

2.1 Benchmarking in Isolation

For our first step, we have used machine learning-based applications to drive our benchmarking efforts. This choice stems from the fact that state-of-art deep learning advances have enabled higher demands for compute intensive data stream processing tasks from edge sources, especially video and image processing using deep learning [1]. We chose application cases belonging to classification (*ResNet-50* [9]) and detection (*Tiny Yolo* [25]) as test functions to explore and compare their performance across different hardware platforms. Note that machine learning comprises many more techniques and deployment on different hardware, such as random forests [32] or decision trees [21]. However, there is a general lack of comparison standards due to heterogeneous platform workflows and limited deployment scale. Further, although it is desirable for the same model to illustrate similar accuracy across different hardware [22] as is illustrated by TensorFlow [28], the performance across these hardware may be different.

To that end we consider four types of hardware including CPU, GPU, ASIC and FPGA. CPUs are the most general hardware platform. GPUs, particulary the NVIDIA products with CUDNN [4] support both desktop and embedded training and inference. For ASICs, we used two currently available neural network acceleration chips on the market: the Intel Neural Compute Stick (NCS) [10] and Google Coral with Tensor Processing Unit (TPU) [12]. For FPGAs, High-level synthesis (HLS) is a technology that can translate behavioral-level design descriptions using C/C++ into RTL descriptions [13] and make them easier for application developers to test edge applications [11]. For deep learning tasks, Xilinx provides a toolkit called DNNDK [7] based on HLS that is aimed at providing users with an end-to-end workflow to deploy a trained network on FPGA devices. The design flows for different hardware platforms that we have used are summarized in Table 1.

Single device executions aim at collecting time and power consumption data for the task under consideration. Time experiment records are series of data points of execution time length for each inference. From these data records, the mean and standard deviation of response time and its inverse (inference frequency) are determined to document the capability of this device. That is, we use a normal distribution conforming to $N(muT_{\text{dev}}, stdT^2_{\text{dev}})$ to approximate time consumption per inference for a device and $N(muFreq_{\text{dev}}, stdFreq^2_{\text{dev}})$ to denote the inference frequency. We use random variables rather than static values of average or maximum to allow uncertainty quantification for overall system performance. As the computation system is assumed to be active throughout the deployment cycle, for power consumption data both static and dynamic inference consumption is recorded.

2.2 Inferring the Desired Metrics At-Scale

The aim of this step is to infer the performance, energy and cost metrics for large-scale deployments of ML applications that span the cloud-to-edge spectrum using insights from benchmarking of isolated use cases and solv-

Table 1: Device-level Acceleration Deployment Workflows for Different Hardware Platforms

Design Flow	Edge CPU	Embedded GPU	FPGA	ASIC	Server GPU	Server CPU
Hardware	Raspberry Pi 3 b+	NVIDIA Jetson Nano	Avnet Ultra96	Intel NCS	NVIDIA GTX1060 6Gb	AMD FX-6300
ResNet-50	Tensorflow/Keras	TensorRT	DNNDK	OpenVINO	Tensorflow/Keras/Cuda	Tensorflow/Keras
Tiny Yolo	Darknet	Darknet/TensorRT	DNNDK	OpenVINO	Tensorflow/Keras/Cuda	Tensorflow/Keras

ing an optimization problem.

Our method for inferring the desired metric inference can be thought of as an optimistic upper bound approximation for the hardware device selection for application deployment across the cloud and edge without considering the underlying model-related error metrics like classification accuracy. Under a linear speedup assumption with additional device parallelism, further speed up with multiple devices would increase total inference capability without increasing uncertainty in performance (variance) [6]. To ensure nHW_{dev} that a specific type of device with total inference capability $R_{\text{dev}} \sim N(muFreq_{\text{dev}} * nHW_{\text{dev}}, stdFreq^2_{\text{dev}})$ can handle input data load $L_{\text{dev}} \sim N(muFreq_{\text{in}}, stdFreq^2_{\text{in}})$, there should be enough devices deployed for the given input pressure with a design confidence level $conf$:

$$Pr(R_{\text{dev}} - L_{\text{dev}}) > conf \tag{1}$$

For latency constraint, we have the latency as the sum of hardware running time and communication time in the inference loop. The average latency can be compared in a straightforward way:

$$T_{\text{app}}(dev) = T_{hw} + T_{comm} = muT_{dev} + t_{e2f} + t_{f2c} \tag{2}$$

where $t_{e2f} = S_{e2f}/B_{e2f}$ refers to the communication time from edge to fog (which is a layer between the edge and cloud) and $t_{f2c} = S_{fec}/B_{f2c}$ refers to the communication time from edge to cloud. The communication time is computed using the transfer package size S divided by the bandwidth B, where bandwidths are assumed to be stable. For different application scenarios, communication bandwidths vary [19].

We set the unit cost of electricity $costElec$ as a constant $\$0.1/kwh$ [31]. Moreover, the total electricity cost can be denoted as:

$$costP(dev, T_{cycle}) = P_{\text{app}}(dev, T_{cycle}) * costElec \tag{3}$$

Likewise, the total power consumption for a given deployment cycle length T_{cycle} can be computed by the sum of idle power and working power:

$$\begin{aligned} P_{\text{app}}(dev, T_{\text{cycle}}) &= P_{\text{idle}}(dev) * T_{\text{cycle}} \\ &+ P_{\text{perinf}}(dev) * muFreq_{\text{in}} * T_{\text{cycle}} \end{aligned} \tag{4}$$

Based on the definitions given above, we formulate the problem of hardware accelerator selection as minimizing the total deployment cost while lowering time and power consumption to a lower target level. We consider performance requirements at the application level using hardware commercial cost, latency T_{app} for each inference loop and total working power consumption $P_{\text{app}}(dev, T_{cycle})$ over the design deployment cycle T_{cycle}.

$$\min_{dev \in ListHW} \sum costHW_{\text{dev}} * nHW_{\text{dev}} + costP(dev, T_{cycle})$$

subject to:

$$T_{\text{app}}(dev) \leqslant t_{\text{target}} \tag{5}$$

3 Empirical Validation of HARE

We now validate the effectiveness of the HARE framework. Based on the functional descriptions and hardware design workflows discussed above, in this part we describe experiments involving hardware deployment and other metric evaluations on test applications.

3.1 Testbed Configuration

The testbed and commercial price ($costHW_{\text{dev}}$) of deployment platforms at the time we conducted the experiments are presented below.

1. **CPU:** CPU is the most general option. Two options from both server and edge side are considered in our test framework.
    1. Raspberry Pi 3B ($40): Edge deployment with a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
    2. AMD FX-6300 CPU ($70): Server deployment with 6 cores, 6 threads @ 3.5-3.8GHz
3. **GPU:** For GPU, two options from both server and edge side are considered in our test framework.
    1. Jetson Nano ($99): Edge deployment with embedded GPU 128-core Maxwell and CPU Quad-core ARM A57 @ 1.43 GHz
    2. GTX 1060 6Gb ($180): Server deployment with 1280-core Pascal architecture
3. **ASIC:** ASIC stands for application specific integrated circuits. We choose Neural Compute Stick(NCS) from Intel ($75). It is equipped with Movidius Myriad 2 Vision Processing Unit (VPU) with nominal 600 MHz operations at 0.9V.
4. **FPGA:** Machine learning inference tasks require both general-purpose computations and

application-specific computations. We choose the relative high-end Ultra96 ($250) Zynq Ultrascale+ Development board (`www.zedboard.org`).

As a result, the list of potential hardware in our problem setting can be shown as: $ListHW = \{RPi, FX6300, JetsonNano, GTX1060, NCS, Ultra96\}$. In this set of devices, the Raspberry Pi (*RPi*) could be regarded as the baseline for acceleration comparisons due to its lowest performance (being a CPU). Some devices need to work with a host. NCS needs to be plugged into a USB port. And GTX1060 needs to be connected to a PCIE port with external power support.

3.2 Per-Device Benchmarking Results

Experiments on classification tasks are conducted on a set of 500 images with a resolution of $640 * 480$. Experiments on detection tasks are conducted on a road traffic video consisting of 874 frames with a resolution of $1280 * 720$. We primarily gather power and time consumption on these high-dimensional data compatible with ImageNet [16] to guarantee higher generalization than cifar-10 [15] or other lower-dimensional datasets. The power consumption data is gathered using a specific power measurement instrument called Watts Up Pro. For time consumption, it is worth pointing out that the time metric we are using here is the total response time running on hardware for inference tasks. As a result, $T_{\text{hw}} = T_{\text{preprocess}} + T_{\text{inference}}$. That is, the total time consumption includes both the input data preprocessing time and the inference time. Tables 2 and 4 show response times for each Resnet50 and Tiny Yolo inference, respectively, while Tables 3 and 5 show both their idle and execution power consumptions.

Table 2: Response Time (T_{hw}) for Object classification Task using *ResNet-50* (Unit: Second)

Time	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
mean	2.089	0.133	0.029	0.218	0.039	0.268
std	0.058	0.016	0.001	0.003	0.005	0.006

Table 3: Power Consumption for Object classification using *ResNet-50* (Unit: Watt)

Power	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
Idle	1.8	2.2	6.2	0.4	10	72
Infer	4.8	5.6	7.6	1.9	122	145

Table 4: Response Time (T_{hw}) for Traffic Detection Task using *Tiny Yolo* (Unit: Second)

Time	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
mean	2.874	0.096	0.023	0.238	0.059	0.217
std	0.068	0.008	0.001	0.003	0.002	0.076

Table 5: Power Consumption for Traffic Detection using *Tiny Yolo* (Unit: Watt)

Power	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
Idle	1.8	2.3	7.4	0.4	10	72
Infer	4.8	11.7	9.2	2.1	122	150

3.3 At-Scale Approximation Results

We make evaluations for time, power and cost for at-scale deployments using our approximation approach.

3.3.1 Application Topology

We present a prototypical smart application case [5] comprising distributed camera networks that can be generalized to scenarios like unmanned shopping using object classification and surveillance using detection as shown in Figure 1. Computation burdens are conducted on edge nodes. There would be different design standards for various scenarios. Without loss of generality, we set the system deployment goal as computation resources should guarantee to handle no less than half (2 of 4) of input loads from every fog group (3 groups) with an overall confidence level of 99%. We set the input with relatively high uncertainty when $stdFreq_{\text{in}} = muFreq_{\text{in}}$.

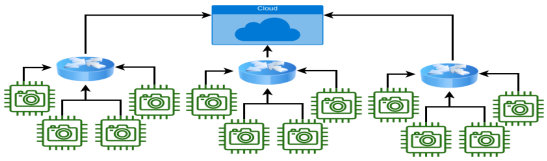


Figure 1: Three-level Design Topology Layout: (1) Top:Cloud servers; (2) Intermediate:3 Fog groups include communication control and some computation power; (3) Bottom:4 Edge nodes in each fog group closest to sensors and data needs to be processed.

3.3.2 Latency Estimation

A straightforward comparison for response times per inference can be conducted using inference time data from Tables 2 and 4. From the time point of view, Rapsberry Pi 3 b+ consumes the most for each inference task. Data for bandwidth between edge/fog and cloud is retrieved from standard IEEE802.11n Wifi setting [19] as the optimal upper bound of $135Mbps$. The size of control signal is set as $1kb$ and data signal using JPEG [34] 100% with $24bit/pixel$. Based on the definitions from Section 2, we show latency estimations of $T_{\text{app}}(dev)$ in Table 6.

3.3.3 Power Consumption

Based on the method discussed, we can approximate the total power consumption holding application accelerations for a given cycle length. We use the idle power of one device to denote the total idle power of this type

Table 6: Inference Cycle Time $T_{\text{app}}(dev)$(Unit:Second)

Time	RPi	JetNano	Ultra96	NCS	GTX1060	FX6300
Loc	Edge	Edge	Edge	Edge	Cloud	Cloud
Res	2.088	0.131	0.029	0.218	0.046	0.275
Yolo	2.869	0.096	0.023	0.238	0.081	0.190

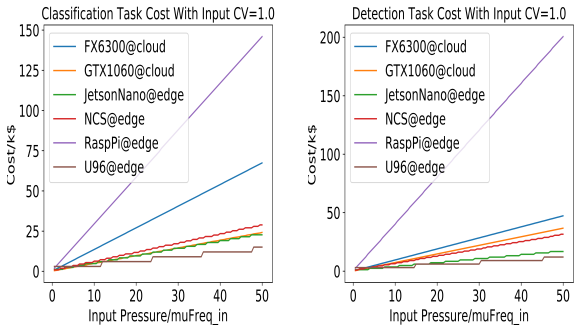
of devices in a network based on an ideal time-division setting. We set the deployment time length as 24 months for our simulated at-scale setup and the total power consumption for both classification and detection tasks is computed for this period as shown in Table 7.

Table 7: Total Working Power Consumption (Unit:kWh)

Power	RPi	JetNano	Ultra96	NCS	GTX1060	FX6300
Res	3720	372.6	87.2	144.8	2243	14236
Yolo	5040	485.7	87.1	173.9	2660	27685

3.3.4 Cost Evaluation

Figures 2a and 2b show results for devices with *ResNet-50* classification and *Tiny Yolo* detection applications, respectively. Under increasing input pressure, the number of devices required steps up like stairs. For a given design cycle, the total cost consisting of device and power cost would be proportional to the number of devices.



(a) *ResNet-50* Classification (b) *Tiny Yolo* Detection

Figure 2: Cost (Unit:k\$) for a 24 Month Deployment Cycle Under Increasing Data Input Pressure (Frequency) and Medium Complexity($muFreq_{in} = stdFreq_{in}$)

Traditional CPUs like Raspberry Pi on the edge and FX6300 on the cloud both show worst performances. This explains why hardware accelerators are necessary for such use cases involving compute-intensive tasks like ML inference. For long-term deployments, power consumption would play a significant role in the total cost. For classification and detection tasks, the Ultra96 (FPGA) and JetsonNano (embedded GPU) illustrate the two most cost-efficient options for the edge side at high input pressures. NCS (ASIC) could be easily deployed to existing network topology but relatively lower individual device performance becomes its limitation. Edge accelerators show more obvious advantage for video detection case; this indicates the necessity for offloading streaming data processing in the edge computing pattern.

4 Conclusions

This paper presents a simple recommendation system to help users select an accelerator hardware device for their applications deployed across the cloud to edge spectrum. Our approach first measures realistic execution traces of computations on individual hardware accelerator devices. We then use these data to approximate latency, power consumption and long-term cost for at-scale deployments, which provides guidance on device selection.

Summary: For the hardware devices used, FPGAs show both highest absolute computation power and highest energy efficiency. This makes it a good option for flexible acceleration tasks at the edge. However, for relatively low pressure scenarios, the device costs could be dominant and FPGAs are harder to program. Furthermore, the number of devices needed is based only on an ideal assumption and a large number of connected nodes might still lead to a larger number of devices needed and higher usage threshold for seeking the desired parallelism. For machine learning inference tasks, server-side GPUs can have the most computation power and higher energy efficiency than CPUs and some lightweight edge devices. Thus, embedded GPUs like Jetson Nano could have much potential for more general computation tasks, particularly those related to graphics and vision. ASIC devices like NCS do not show the highest time efficiency or power efficiency but considering its low cost and low threshold to use, it is still competitive for neural network-specific and relatively low input frequency tasks.

Limitations: So far we have only considered a device selection strategy where only one type of device is considered at one time. In addition, we only consider an ideal minimum number of devices needed for a given input pressure without accounting for the cost of task scheduling. Moreover, we only consider a relatively simple application scenario where only one type of acceleration task is executed at a time. More experiments beyond classification and detection would be needed. We plan to investigate the possibility of at-scale deployment of RNN [23] and GAN [35] in edge scenarios. Another potential issue is the cost and power requirements to keep switches and gateways operational, which are not negligible especially in cases where commercial cloud environments are involved. Finally, we have not taken interference effects between device executions into consideration. This will shed light on further research in design and optimization of fog/edge topologies involving heterogeneous hardware accelerators targeting heterogeneous tasks.

5 Discussion Topics

This paper describes a device selection problem that must be overcome for the use of hardware accelerators in edge computing. The main challenge lies in the heterogeneity of hardware devices and in contrast the lack of systematic research about their impact on application development, operational costs and performance. We hope that our ideas will stimulate discussion on several open issues as follows regarding the applicability of hardware acceleration techniques at the edge:

- How far can the hardware acceleration pattern reach in edge application scenarios? We discuss the state-of-art deep learning accelerations but what other applications can benefit from accelerator technologies where our framework will be useful?
- Is our objective function correct or do we need to incorporate additional or different cost function and constraints? Some applications may need mixed accelerators. How does this change the approach?
- Should hardware acceleration techniques just be used for inference computations for already learned models, or can it also be used for model re-learning?
- Anonymous reviews pointed out that the power models are relatively straightforward and require a lot of profiling. Can we develop models that cut across accelerators? That is, can we find a mathematical function that subsumes everything from a GPU to an FPGA?

Some of our claims may be considered controversial, and we are looking forward to hearing different points of view on these issues:

- We have used simulations to scale hardware device deployments. This was based on parameters obtained using devices at hand and executing two deep learning tasks. We have not experimented with other lower level computations like matrix multiplications and do not know if their performance accelerations are similar.
- Our experiments used single processing functions. This is based on a full computation task offloading which put all task burden on the target device. We need additional experimentation with more integrated complex applications like streaming processing applications.
- The edge is a highly dynamic environment and there may be a need to dynamically schedule and migrate applications. For multiple different types of tasks, how can this be made feasible given that the circuitry devices like FPGA and ASIC need to be programmed and how can this be achieved on-the-fly?
- We call this a recommendation system. Is this appropriate or what more needs to be done?

References

[1] ANANTHANARAYANAN, G., BAHL, P., BODÍK, P., CHINTALAPUDI, K., PHILIPOSE, M., RAVINDRANATH, L., AND SINHA, S. Real-time video analytics: The killer app for edge computing. *computer 50*, 10 (2017), 58–67.

[2] BOUBIN, J. G., BABU, N. T., STEWART, C., CHUMLEY, J., AND ZHANG, S. Managing edge resources for fully autonomous aerial systems. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (2019), pp. 74–87.

[3] CHEN, F., SHAN, Y., ZHANG, Y., WANG, Y., FRANKE, H., CHANG, X., AND WANG, K. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers* (2014), ACM, p. 3.

[4] CHETLUR, S., WOOLLEY, C., VANDERMERSCH, P., COHEN, J., TRAN, J., CATANZARO, B., AND SHELHAMER, E. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[5] DOC-GROUP,VANDERBILT UNIVERSITY. Edge Hardware Accelerator Recommender Demo. `https://github.com/dustinjoe/EdgeHardwareAcceleratorRecommender/`, 2020. [Online; accessed 01-May-2020].

[6] FOSTER, W. Leverage Multiple Intel® Neural Compute Sticks with Intel® Distribution of OpenVINO™ Toolkit. `https://software.intel.com/en-us/articles/leverage-multiple-intel-neural-compute-sticks-with-intel-distribution-of-openvino-toolkit`, 2019. [Online; accessed 20-Feb-2020].

[7] GUO, K., HAN, S., YAO, S., WANG, Y., XIE, Y., AND YANG, H. Software-hardware codesign for efficient neural network acceleration. *IEEE Micro 37*, 2 (2017), 18–25.

[8] HAN, S., MAO, H., AND DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[9] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[10] INTEL INCORPORATION. Intel® Movidius™ Neural Compute Stick. `https://software.intel.com/en-us/movidius-ncs`, 2018. [Online; accessed 19-Jan-2020].

[11] JIANG, S., HE, D., YANG, C., XU, C., LUO, G., CHEN, Y., LIU, Y., AND JIANG, J. Accelerating mobile applications at the network edge with software-programmable fpgas. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (2018), IEEE, pp. 55–62.

[12] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA, S., BODEN, N., BORCHERS, A., ET AL. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (2017), IEEE, pp. 1–12.

[13] KASTNER, R., MATAI, J., AND NEUENDORFFER, S. Parallel programming for fpgas. *arXiv preprint arXiv:1805.03648* (2018).

[14] KHONA, C. Key Attributes of an Intelligent IIoT Edge Platform. `http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/`, 2017. [Online; accessed 19-July-2019].

[15] KRIZHEVSKY, A., NAIR, V., AND HINTON, G. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html 55* (2014).

[16] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[17] LIN, S.-C., ZHANG, Y., HSU, C.-H., SKACH, M., HAQUE, M. E., TANG, L., AND MARS, J. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (2018), pp. 751–766.

[18] MAHMUD, R., SRIRAMA, S. N., RAMAMOHANARAO, K., AND BUYYA, R. Profit-aware application placement for integrated fog–cloud computing environments. *Journal of Parallel and Distributed Computing 135* (2020), 177–190.

[19] MAO, Y., YOU, C., ZHANG, J., HUANG, K., AND LETAIEF, K. B. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials 19*, 4 (2017), 2322–2358.

[20] NAJAFI, M., ZHANG, K., SADOGHI, M., AND JACOBSEN, H.-A. Hardware acceleration landscape for distributed real-time analytics: Virtues and limitations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), IEEE, pp. 1938–1948.

[21] NARAYANAN, R., HONBO, D., MEMIK, G., CHOUDHARY, A., AND ZAMBRENO, J. An fpga implementation of decision tree classification. In *2007 Design, Automation & Test in Europe Conference & Exhibition* (2007), IEEE, pp. 1–6.

[22] NURVITADHI, E., SHEFFIELD, D., SIM, J., MISHRA, A., VENKATESH, G., AND MARR, D. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic. In *2016 International Conference on Field-Programmable Technology (FPT)* (2016), IEEE, pp. 77–84.

[23] NURVITADHI, E., SIM, J., SHEFFIELD, D., MISHRA, A., KRISHNAN, S., AND MARR, D. Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)* (2016), IEEE, pp. 1–4.

[24] REDDI, V. J., CHENG, C., KANTER, D., MATTSON, P., SCHMUELLING, G., WU, C.-J., ANDERSON, B., BREUGHE, M., CHARLEBOIS, M., CHOU, W., ET AL. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549* (2019).

[25] REDMON, J., DIVVALA, S., GIRSHICK, R., AND FARHADI, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[26] SATYANARAYANAN, M. The emergence of edge computing. *Computer 50*, 1 (2017), 30–39.

[27] SKARLAT, O., NARDELLI, M., SCHULTE, S., AND DUSTDAR, S. Towards qos-aware fog service placement. In *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)* (2017), IEEE, pp. 89–96.

[28] SZE, V., CHEN, Y.-H., YANG, T.-J., AND EMER, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE 105*, 12 (2017), 2295–2329.

[29] TANG, J., LIU, S., LIU, L., YU, B., AND SHI, W. Lopecs: A low-power edge computing system for real-time autonomous driving services. *IEEE Access 8* (2020), 30467–30479.

[30] UMUROGLU, Y., FRASER, N. J., GAMBARDELLA, G., BLOTT, M., LEONG, P., JAHRE, M., AND VISSERS, K. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017), ACM, pp. 65–74.

[31] U.S. ENERGY INFORMATION ADMINISTRATION. Electric Power Monthly with Data for November 2019. `https://www.eia.gov/electricity/monthly/current_month/epm.pdf`, 2020. [Online; accessed 06-Feb-2020].

[32] VAN ESSEN, B., MACARAEG, C., GOKHALE, M., AND PRENGER, R. Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines* (2012), IEEE, pp. 232–239.

[33] VARGHESE, B., WANG, N., BERMBACH, D., HONG, C.-H., DE LARA, E., SHI, W., AND STEWART, C. A survey on edge benchmarking, 2020.

[34] WALLACE, G. K. The jpeg still picture compression standard. *IEEE transactions on consumer electronics 38*, 1 (1992), xviii–xxxiv.

[35] YAZDANBAKHSH, A., BRZOZOWSKI, M., KHALEGHI, B., GHODRATI, S., SAMADI, K., KIM, N. S., AND ESMAEILZADEH, H. Flexigan: An end-to-end solution for fpga acceleration of generative adversarial networks. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2018), IEEE, pp. 65–72.

[36] ZHOU, Y., GUPTA, U., DAI, S., ZHAO, R., SRIVASTAVA, N., JIN, H., FEATHERSTON, J., LAI, Y.-H., LIU, G., VELASQUEZ, G. A., ET AL. Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2018), ACM, pp. 269–278.

[37] ZHU, H., LO, D., CHENG, L., GOVINDARAJU, R., RANGANATHAN, P., AND EREZ, M. Kelp: Qos for accelerated machine learning systems. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2019), IEEE, pp. 172–184.