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., lower procurement costs) and opex (i.e., lesser 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 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 accelertor devices. Second, we utilize these traces to deduce the achievable latencies and amortized costs of device deployments at scale, which serves as the 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 [21] 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 of machine learning applications on these data, often require millions of operations for each inference [14]. Thus, for the sake of maintaining low latencies and compliance with the power sensitivity of edge deployment, smaller models [7] working on more efficient hardware are preferred [13]. 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 [17].

With the proliferation of different accelerator technology alternatives, developers are faced with a significant dilemma: which accelerator technology is best suited for their application needs such that application 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 specific to a device type and incorporate only a single inference instance level [18, 28] or comprise low-level circuit design optimization analysis [11, 24]. Correspondingly, there remains a significant lack of understanding on the applicability of these accelerator technologies for at-scale, edge-based applications.

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 strategy for the given topology. Likewise, even though there is much research on service placement and optimization for edge computing [15, 22], these efforts are focused primarily on software services rather than hardware devices. Yet, these prior works show a general path to formulate an optimization problem to reach a design objective under evolving constraints [27]. Similarly, research on cloud-level device and service placement also exists [3]. But more widespread use of these different hardware platforms requires more systematic under-

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

standing of both spatial and temporal impacts in realistic deployments of the hardware 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 technology for a given compute intensive task. Our literature search did not reveal any prior work that takes into account the impact of heterogeneous hardware acceleration devices on the cost and power consumption for long-term deployments, and provides recommendation of the suitable technology to use. A related effort of ours [2], whose focus is on the user interface to our ecosystem that uses HARE as its recommendation system, has been submitted to the collocated OpML 2020 conference. This paper on HARE 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.
- Using these samples, 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 applications 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 on the HARE recommender system. In realizing HARE, 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 ML 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 choose application cases belonging to classification (*ResNet-50* [8]) and detection (*Tiny Yolo* [20]) as test functions to explore and compare their performances across different hardware platforms. Although the deployment workflow for each device differs yet it is desirable for the same neural network architecture to illustrate similar inference performance [18]. Fortunately, the application of hardware-oriented high-level language and common computation frameworks like Tensorflow make it possible for the same neural network applications to execute on different hardware platforms without significant performance deviation [23].

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) [9] and Google Coral with Tensor Processing Unit(TPU) [11]. For FPGAs, High-level synthesis (HLS) is a technology, which can translate behavioral-level design descriptions using C/C++ into RTL descriptions [12] and make them easier for application developers testing edge applications [10]. For deep learning tasks, Xilinx provides a toolkit called DNNDK [6] 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.

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

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 solving an optimization problem.

Our method for inferring the desired metrics can be thought of as an optimistic upper bound approximation for the hardware device selection for application deployment across the cloud and edge when the underlying model-related error metrics like classification accuracy are not considered. 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) [5]. 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 number of 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 [16] and it is worth pointing out with the emergence of more advanced communication techniques such as 5G, latency for data communication would become lower.

We set the unit cost of electricity $costElec$ as a constant $\$0.1/kwh$ [25]. Moreover, the total electricity cost can be denoted as:

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

We use the three-sigma assumption [19] for initialization and finishing time for each inference cycle. In this way, the power consumption per inference can be approximated as

$$P_{\text{perinf}}(dev) = muT_{\text{dev}} * Pinf_{\text{dev}} + 3 * stdT_{\text{dev}} * (Pinf_{\text{dev}} + Pidle_{\text{dev}})/2 * 2 \tag{4}$$

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:

$$P_{\text{app}}(dev, T_{\text{cycle}}) = P_{\text{idle}}(dev) * T_{\text{cycle}} + P_{\text{perinf}}(dev) * muFreq_{\text{in}} * T_{\text{cycle}} \tag{5}$$

Based on the definitions given above, we can formulate our problem for the hardware accelerator selection as minimizing the total deployment cost while lowering time and power consumption to a lower target level. We consider performance requirements from specific 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:

$$T_{\text{app}}(dev) \leqslant t_{\text{target}} \tag{6}$$

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 propose 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 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 Ultra96 ($250) Zynq Ultrascale+ Development board (`www.zedboard.org`) with a relatively high-end chip.

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).

3.2 Results of Per-Device Benchmarking

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 test data. 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.

Table 2 and Table 4 show response times for each inference. Table 3 and Table 5 show both idle and execution power consumptions. In the devices under test, the Intel Neural Compute Stick (NCS) and GTX1060 cannot operate on their own: NCS needs to be plugged into a USB port and GTX1060 needs to be plugged onto a motherboard.

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 Results for the Approximation Approach

Based on the methods described in Section 2, we make evaluations for time, power and cost for at-scale deployments using our approximation approach.

3.3.1 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 [16] as the optimal upper bound of $135Mbps$. The size of control signal is set as $1kb$ and data signal using JPEG [26] 100% with $24bit/pixel$. Based on the definitions from Section 2, we show latency estimations of $T_{\text{app}}(dev)$ in Table 6.

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

3.3.2 Power Consumption

Based on the method discussed, we can approximate the total power consumption holding application accelerations for a given cycle length. We set this 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

3.3.3 Cost Evaluation

Figures 1 and 2 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. For a given design cycle, the total cost consisting of device and power cost would be proportional to the number of devices in the system.

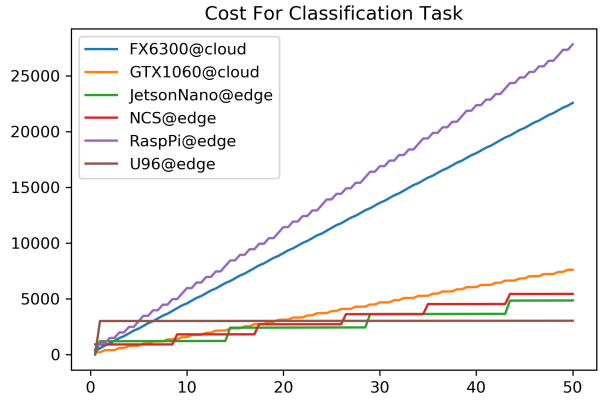

Figure 1: Classification Task Cost (Unit:$) using *ResNet-50* for a 24 Month Deployment Cycle Length Under Increasing Data Input Pressure (Frequency)

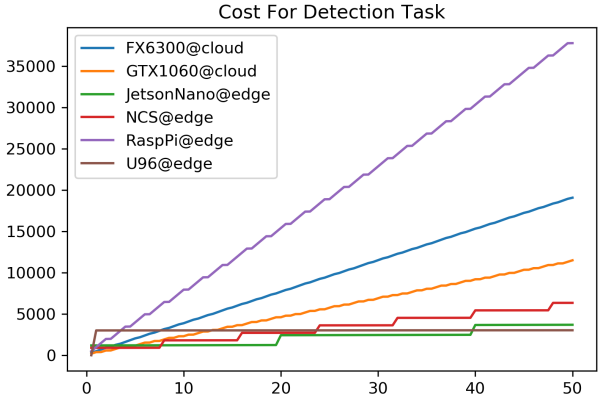

Figure 2: Detection Task Cost (Unit:$) using *Tiny Yolo* for a 24 Month Deployment Cycle Length Under Increasing Data Input Pressure (Frequency)

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 utilize these data to approximate latency, power consumption and long-term cost for at-scale deployments. These approximation results can then be used for device selection.

Summary: 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 machine learning inference. For long-term deployments, power consumption would play a significant role in the total cost. For classification and detection tasks used in this work, the Ultra96 (FPGA) and JetsonNano (embedded GPU) illustrate the two most cost-efficient options for the edge side for high input pressures. NCS (ASIC) could be easily deployed to existing network topology but relatively lower individual device performance becomes its application constraint.

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, it is worth noticing here that the number of devices 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 effiency than CPUs and some lightweight edge devices. Thus, embedded GPUs like Jetson Nano could have much potential for more general computation tasks especially 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 relative low input frequency tasks.

Limitations: So far we have only considered a pure 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 for task scheduling. Moreover, we only consider a relatively simple application scenario where only one type of acceleration task is conducted. 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.

5 Discussion Topics

This paper describes a device selection problem that must be overcome for the wider application of hardware accelerators in edge computing. The main challenge lies in the heterogeneity of hardware device options in market and in contrast the lack of systematic research about their impact on application development and operation costs and performance. We hope that our ideas will stimulate discussion of several open issues regarding the applicability of hardware acceleration techniques to 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 a mix of accelerator devices. How does this change the approach?
- Should hardware acceleration techniques just be used for inference computations for learned models, or can it also be used for model re-learning?

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 streamline processing applications.
- The edge is a highly dynamic environment and there may be a need to dynamically schedule and migrate applications. For multiple different type of tasks on the schedule how can this be made feasible given that the circuitry devices like FPGA and ASIC needs 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] CANADY, R., ZHOU, X., BAO, S., BARVE, Y., BHATTACHARJEE, A., AND GOKHALE, A. Harp: A model-driven engineering approach to automate hardware acceleration for edge-based machine learning applications. In *In submission to the 2020 2nd USENIX Conference on Operational Machine Learning(OpML)* (2020), Usenix, p. 2.

[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] 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].

[6] 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.

[7] 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).

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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].

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] PUKELSHEIM, F. The three sigma rule. *The American Statistician 48*, 2 (1994), 88–91.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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].

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

[27] ZENG, D., GU, L., GUO, S., CHENG, Z., AND YU, S. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers 65*, 12 (2016), 3702–3712.

[28] 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.