Keep Clear of the Edges
An Empirical Study of Artificial Intelligence Workload Performance and Resource Footprint on Edge Devices

Kun Suo, Tu N. Nguyen, Yong Shi, Jing (Selena) He, and Chih-Cheng Hung
Department of Computer Science, Kennesaw State University, Marietta, GA 30060, USA
Email: {ksuo, tu.nguyen, yshi5, she4, chung1}@kennesaw.edu

***Abstract*—Recently, with the advent of the Internet of everything and 5G network, the amount of data generated by various edge scenarios such as autonomous vehicles, smart industry, 4K/8K, virtual reality (VR), augmented reality (AR), etc., has greatly exploded. All these trends significantly brought real-time, hardware dependence, low power consumption, and security requirements to the facilities, and rapidly popularized edge computing. Meanwhile, artificial intelligence (AI) workloads also changed the computing paradigm from cloud services to mobile applications dramatically. Different from wide deployment and sufficient study of AI in the cloud or mobile platforms, AI workload performance and their resource impact on edges have not been well understood yet. There lacks an in-depth analysis and comparison of their advantages, limitations, performance, and resource consumptions in an edge environment. In this paper, we perform a comprehensive study of representative AI workloads on edge platforms. We first conduct a summary of modern edge hardware and popular AI workloads. Then we quantitatively evaluate three categories (i.e., classification, image-to-image, and segmentation) of the most popular and widely used AI applications in realistic edge environments based on Raspberry Pi, Nvidia TX2, etc. We find that interaction between hardware and neural network models incurs non-negligible impact and overhead on AI workloads at edges. Our experiments show that performance variation and difference in resource footprint limit availability of certain types of workloads and their algorithms for edge platforms, and users need to select appropriate workload, model, and algorithm based on requirements and characteristics of edge environments.**

***Index Terms*—edge computing, artificial intelligence, performance, resource footprint**

I. Introduction

In recent years, edge computing has become a research hotspot in both academia and industry. In cloud computing, all the resources are concentrated in the cloud or datacenter. The data generated on terminal devices is transmitted to the cloud through network, and both computing and data storage are performed on the cloud platform. In edge computing, computing and storage resources are mostly deployed on the edge (i.e., edge servers or IoT devices), and local data can be processed nearby without transferring the data to the remote cloud. A report from AlefEdge [3] predicts that the edge computing market will exceed $4 trillion by 2030. The emergence of edges can greatly save the network bandwidth and reduce the transmission delay between the cloud and the terminal, and their advantages can significantly help to achieve better application performance and quality-of-service (QoS).

Meanwhile, artificial intelligence (AI) is being increasingly deployed on edge computing. The emergence of intelligent edge computing is the result of the continuous development of computer hardware and software as well as the requirements for new applications or scenarios. First, the rapid innovation on hardware such as various AI dedicated chips enables to deploy artificial intelligence models on modern edges. Next, with the development of Internet-of-Everything, especially the artificial intelligent Internet-of-Things (AIoT), various new devices are constantly emerging and generating massive data. For example, ubiquitous surveillance cameras produce huge amount of video data every day, and a self-driving vehicle could generate terabytes of data daily on the road. Sending all the data onto cloud for processing is unaffordable for today's infrastructure. Besides, new scenarios and applications also require local data processing. For example, autonomous driving and industrial automation have high requirements for real-time data processing. Finally, privacy is another concern for edge computing as many data (i.e., pictures, audios, videos, etc.) contain a lot of personal information. The best way to protect privacy is to process data on device instead of transmitting personal data to the cloud.

Over the last decade, many studies have been proposed to study the AI performance in the cloud. As smartphones become more popular, the research related to mobile AIs also developed and improved rapidly. As AI or machine learning (ML) workload becomes increasing complex and more data is generated at the edge, it is also interesting and critical to study different artificial intelligence application performance on the edges and their impact on the system, especially in the resource perspective. However, we find that recent research in the edge AI area focuses more on the AI model compression and optimization, or AI privacy protection and model security on edge platforms. There lacks research on the resource consumption and performance analysis of different AI models or applications on various edge computing platforms, which is important for the improvement of edge platforms and the deployment of AI workloads and services.

In this paper, we present a detailed analysis of representative AI workloads on edges and perform an empirical study of their performance and resource footprints. To the best of our knowledge, this paper is the first to explore these aspects of AI workloads on edges. We have important findings that

978-1-6654-4331-9/21/$31.00

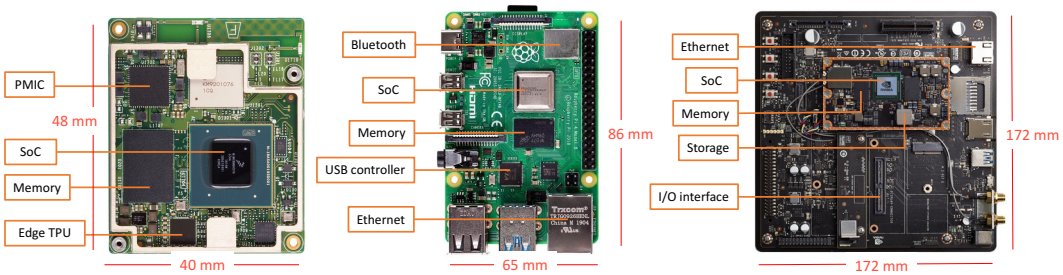

Fig. 1: Typical edge hardware. From left to right are Google Coral, Raspberry Pi, and Nvidia Jetson TX2.

could help user select the appropriate edge platforms for their workloads and guide the optimization of existing edge hardware. The rest of this paper is organized as follows. In Section II, we investigate the edge hardware and most widely used AI edge workloads. Then we present our methodology, the software and hardware we used, and the machine learning models and algorithms in Section III. We use representative applications to evaluate the performance, measure the resource consumption, analyze the overhead in Section IV, V and VI, and review related works in Section VII. Finally, Section VIII concludes this paper with insights.

II. Modern Edge Hardware and Artificial Intelligence Workloads

A. Edge Hardware

Processor with Accelerator. Similar to traditional cloud hardware, there exist two types of processors or system-on-a-chip (SoC) in edge devices. As shown in Figure 1, the most popular design of edge chips is to separate the functions to adopt heterogeneous tasks, where the CPU is mainly used for logic and control while the majority of computing workloads are executed on a specific accelerator (i.e., GPU, TPU, FPGA, etc.). Despite sacrificing general-purpose computing power, heterogeneous processor with dedicated acceleration units is more efficient in many scenarios, which can significantly reduce the execution time of workloads, and improve the performance-to-power ratio on the hardware. For instance, Han *et al.* [12] proposed an efficient speech recognition (ESE) solution using FPGA to improve the efficiency of sparse long short-term memory network on mobiles and accelerate speech recognition applications. Compared to solutions implemented with CPU or GPU, ESE built on FPGA can achieve up to 40× and 11.5× on energy efficiency, respectively.

General Purpose SoC. In contrast to chips equipped with specific accelerators, another type of SoC on edges is the general processor, such as ARM chips on the Raspberry Pi. By extending functions from data centers to edges, service providers can process data much closer to terminal devices and data sources. For instance, Intel Xeon D-2100 provides an independent SoC which integrates all functions including computing, storage, and networking inside the chip. ARM recently announced two new processors, Cortex-M55 and Ethos-U55, which provide built-in AI processing capabilities for IoT devices. In recent years, in order to improve the computing power of machine learning (ML) and AI, general processors also add new instruction support for tensor computing and increase the parallelism. For instance, Intel added SIMD (single instruction stream multiple data stream) and VLIW (very long instruction word) into its Habana chip, which allows processing large amounts of parallel data with only one instruction. Different from the processor with specific accelerators, this type of solution makes the SoC architecture more versatile and generalized.

Memory. The memory on edges also has a huge impact on AI workload performance. As the memory speed is far slower than processors, inefficiency on memory will not only limit SoC performance, but also cause memory wall effect for many applications. Based on statistics [2], 90% of the data stored today was produced within past two years. Therefore, data generation, calculation, and fast access have become an inevitable part of the user experience on modern AIoTs. As the size of most neural network models continue to grow, it introduces significant challenges to the AI deployment on edges. From the hardware perspective, we can use high-density memory or speed up read and write through non-volatile memory. Another strategy is to make neural network small enough through model compression. If the model is difficult to diminish, the data could also be directly computed in memory without having to fetch data into processors. Normally, in-memory calculation can greatly reduce the power consumption and dramatically increase the speed of inference. However, in-memory computing is essentially simulation calculation and the algorithm accuracy might be limited.

Storage & Network. Besides the processor and memory, storage is also critical for ML performance on edge devices, especially for unstructured data such as pictures and videos. Network connections, including 5G, Bluetooth, Wi-Fi, Ultra-wideband, etc., also play an important role in edge AI of distributed and parallel systems. As this paper focuses on AI workload execution and resource footprint on the single edge node, the above hardware will not be discussed in this research.

B. Edge AI Workloads

To find the most popular and widely used AI workloads on edge infrastructure, we surveyed a selection of services and categorized them based on their scenarios: Target Recognition (*Type* α), Image Processing (*Type* β), and Object Segmentation (*Type* γ). We present an overview of these applications along

TABLE I: Specifications of the measured devices.

Specification	Raspberry Pi Model 4B (General edge architecture)	NVIDIA Jetson TX2 (Heterogeneous edge architecture)	Amazon EC2 t2.xlarge (General cloud architecture)
CPU	A quad-core cortex-A72 ARM v8 64-bit SoC 1.5GHz	A quad-core 2.0GHz 64-bit ARM v8 A57 processor; a dual-core 2.0GHz superscalar ARM v8 Denver processor	A quad-core 3.0GHz Intel scalable processor (vCPU)
GPU	No GPU	56-core 1.33 TFLOPS NVIDIA Pascal	No GPU
Cache	32 KB L1 data cache 48 KB L1 instruction cache 1MB L2 cache	64KB L1 data cache 128KB L1 instruction cache 2MB L2 cache	32KB L1 data cache 32KB L1 instruction cache 256KB L2 cache
Memory	4 GB LPDDR4-2400 RAM	8 GB 128-bit LPDDR4 1866 MHz	16 GB RAM (vMemory)
Storage	32 GB MicroSD Card	32 GB eMMC 5.1 disk	80GB HDD

with brief descriptions of their characteristics. Overall, edge AI workloads have the following common features: First, the application generates large amounts of data at the edge, which is not necessary to upload all of them to the cloud. Instead, most of the data is processed locally in real time, where the network has little impact while SoC as well as memory determines its performance. Second, compared to cloud services, edge applications are highly distributed, more fragmented and miniaturized, which responses with less latency and data transmission time. Third, the software and hardware of edge computing is more diversified than cloud with fixed usage. Although the edge can normally expand, it does not provide universal functions as cloud, and the edge scenes are often dominated by a single node. Lastly, each module of edge computing also needs to continually iterate and update, but the frequency is much less than cloud scenarios. Therefore, the update on the edge is often unreliable and uncontrollable, and cannot be as neat as cloud services to ensure the same version and status in real time. Different from cloud, the execution of edge service is more local data-driven.

(α.1) Object Recognition. In computer vision, object recognition aims to use image processing theories and pattern recognition methods to determine whether there are objects of interest in given image or video. In recent years, object recognition has received more attention, and has been applied in many fields at edge devices. For instance, security companies deployed object recognition in surveillance systems to track objects and pedestrians. Smart city systems adopt monitoring systems for traffic tracing, vehicle counting, plate detection and recognition. Also, smartphones use AI for content-based image retrieval, automatically collecting metadata for albums.

(α.2) Classification. Classification is one of the fundamental tasks of many AI workloads. An example is to label input images with fixed categories under the smallest classification error. Generally, image classification can only tell categories and their probabilities that appear in the picture, and limited by categories that have been trained. Although many algorithms and models can implement image classification, it still faces many difficulties and challenges in reality, such as unbalanced categories, fine-grained classification with high intra-class variance, etc. Besides, image classification is also limited by real-time inference ability of edge hardware.

(α.3) Facial Recognition. Facial recognition is a process that identifies or verifies the identity of a subject in images or videos based on facial characteristics (such as statistical or geometric features). Right now, facial recognition is widely used in life, such as iPhone FaceID. Other common applications of face recognition include access control, monitoring systems, fraud detection, identity authentication, and social media. The traditional facial recognition transmits the figure to the cloud for recognition. This process takes large amount of time. Moreover, in some cases where the network environment is poor, traditional algorithms will be much slower. The facial recognition in edge computing places the recognition process locally, which saves time for repetitive transmission, improves efficiency and security at low cost.

(β.1) Image Super-resolution. Super-resolution is to improve the resolution of original image through algorithms. Low-resolution images are generally obtained through a series of degradation operations, which might lose lots of details but also introduce much noise. The super-resolution process based on deep learning is essentially using paired training data for supervised learning, and then performing the inverse operation to obtain the reconstructed high-definition image. Currently, there exist two types of image reconstruction. One synthesizes a high-resolution image from multiple low-resolution images. The other obtains a high-resolution image from a single low-resolution image. Image super-resolution has a wide range of applications in movies, satellite images, medical imaging and other fields. Through image super-resolution, reconstruction of high-quality multimedia content can be completed without consuming many resources, especially at the edges.

(β.2) Photo Enhancement. Photo enhancement is to improve the overall or local characteristics of images, including meliorating the color, brightness, and contrast, making original illegible parts clear, or highlighting the important objects inside. Image enhancement has been adopted with many AI tasks, such as target detection and item classification. In edge computing, image enhancement is also widely used such as identification of severe weather and night images in autonomous driving. Images and videos captured under dim light conditions often suffer from visibility, with significant reductions in brightness and contrast. Therefore, industries usually enhance images first, and then proceed to the detection and recognition process.

(β.3) Bokeh Simulation. Recently, image effects have received increasing attention in different applications. Bokeh is one popular example which blurs the image at unfocused area.

TABLE II: Specifications of AI workloads.

Task	Neural Network	Description	Resolution(px)
Object Recognition	MobileNet v2 [31]	A depthwise separable convolution network with linear bottlenecks between the layers and linear bottlenecks between the layers	224x224
Classification	Inception v3 [39]	A widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset	346x346
Classification	Inception v4 [37]	Built on Inception v3 but with specialized grid reduction	346x346
Facial Recognition	Inception-ResNet v2 [14]	A variation of Inception V3 with residual networks	346x346
Classification	ResNet-50 v2 [14]	A 50 layers deep convolutional neural network, consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers	346x346
Classification	ResNet-152 v2 [14]	Similar as ResNet-50 v2 but contains 152 layers	256x256
Classification	VGG-16 [34]	A CNN with 16 layers including 13 convolutional layers and 3 dense layers	224x224
Super-Resolution	VGG-19 [21]	Similar as VGG-16 but with 19 layers including 16 convolutional layers and 3 dense layers	512x512
Super-Resolution	ResNet-SRGAN [22]	Super-resolution Using a Generative Adversarial Network (GAN). Residual block designed using ResNet	512x512
Image Deblurring	SRCNN 9-5-5 [11]	A deep learning method for single image super-resolution	512x512
Image Enhancement	ResNet-DPED [17]	A residual CNN improving color resolution and image sharpness	256x256
Bokeh Simulation	U-Net [30]	A convolutional network for fast and precise segmentation of biomedical images	512x512
Semantic Image Synthesis	Nvidia-SPADE [28]	A semantic image synthesis system with spatially-adaptive normalization	128x128
Image Segmentation	ICNet [43]	A network for real-time semantic segmentation on high-resolution images	1024x1024
Image Segmentation	PSPNet [44]	Pyramid scene parsing network with pixel level prediction	720x720
Image Segmentation	DeepLab v1 [9]	A deep convolutional network for semantic image segmentation	512x512

Generally, this effect is achieved by large aperture of the fast lens, which is difficult for mobile platform and edge cameras. However, with artificial intelligence develops, it is much easier to divide the focused area and background to achieve the blurred effect in images, such as portrait mode on portable cameras. With pre-trained neural networks, it can be applied to any image to have an artistic blur.

(β.4) Semantic Image Synthesis (SIS). SIS is to semantically control and modify scenes in images. Unlike traditional methods, data-driven graphics can use multiple image regions to synthesize a new image and extract typical semantic features based on large-scale image datasets. This method allows users to specify structure of scene, modify contents, and effectively eliminate discontinuities and artifacts generated during the synthesis process. The current SIS system mainly generates a realistic image based on user-specified semantic including colors, sketches, textures and others. For example, Nvidia's GauGAN can not only generate good visual effects, but also control the style and semantic structure of the generated results, which is significantly useful in the modern virtual reality (VR) or augment reality (AR) applications.

(γ.1) Semantic Segmentation. Object segmentation is another key task in AI, which divides the image into several disjoint areas based on grayscale, color, spatial texture, geometric shape and other characteristics. A common example is to separate the target from the background image. The features in the same area show consistency or similarity. Semantic segmentation is extremely important in areas such as identifying buildings, roads, and forests in satellite images, or locating lesions and measuring areas in medical images, etc. In recent years, with development of deep learning, image segmentation technology has evolved into scene object segmentation, human background segmentation, human face parsing, 3D reconstruction, etc., which has been widely applied in self-driving, augmented reality, security monitoring and other industries.

III. Methodology

A. Hardware Platforms

We conduct our study on several representative edge devices. For general purpose edge, we choose Raspberry Pi 4B with quad-core CPU and 4GB of RAM. For edge hardware with heterogeneous accelerators, we use Nvidia Jetson TX2, which has a quad-core A57 processor, a dual-core Denver processor, a 56-core GPU, and 8GB of RAM. In order to highlight the performance difference between edge devices and cloud servers, we also evaluate the workloads on an Amazon EC2 `t2.xlarge` VM, which is equipped with a quad-core vCPU and 16GB of memory. All processor cores in SoC have a three-level cache hierarchy that the L1 and L2 caches are private to each core, while the last level cache (LLC) is shared among all cores. Table I summarizes the key architectural parameters of the hardware systems.

B. Framework, Benchmarks and Measurement

Based on the popularity of machine learning frameworks, this paper mainly evaluates the performance of TensorFlow Lite, which is widely adopted on mobile platforms, e.g., iOS, Android, as well as IoT device operating system, e.g., Snappy, TinyOS, Raspbian, etc. The performance discrepancy with other frameworks such as Caffe2, MXNet, PyTorch can be learned from Zhang's report [41]. Keras is also extensively used but it is still built based on TensorFlow.

TensorFlow Lite. TensorFlow is developed by Google in 2015, which has integrated most of the common units into the ML framework. Typically, when a ML model is implemented and trained using TensorFlow, it usually generates a model file that requires ample storage space and a GPU to run

inference. However, luxuries such as large storage and specific accelerators are not available on most edge devices. Instead, TensorFlow Lite is an open-source deep learning framework for mobile and edge device and was presented in 2017. As a successor of TF Mobile library, it provides better performance and smaller binary size due to optimized kernels, pre-fused activations and fewer dependencies. The version of TensorFlow Lite we used in this paper is 1.14.

Workloads. AI Benchmark [1] is an open-source python library for evaluating AI performance of various hardware platforms. The benchmark relies on TensorFlow ML library, and provides a lightweight and accurate solution for assessing inference and training speed of key deep learning models. AI Benchmark is distributed as a Python package and supports any OSes. The version of AI Benchmark we are using is 0.1.2. In total, it consists of 42 tests and 16 sections. The details of workload and their neural network models can be found in Table II. The tests cover all major deep learning tasks and algorithms on modern systems and platforms.

Experimental Setup and Methodology. We execute workloads in AI Benchmark to evaluate the AI and ML performance on the above edge devices. Data from these workloads was captured using Linux *nmon* [4] and the output files were then converted into HTML files using *nmonchart*. These files were further edited to highlight the duration of each test. The CPU and memory statistics of each device were obtained, and their efficiencies were analyzed. In addition, we also measured the process context switch on edges, which was calculated by counting number of processes that are enqueued or dequeued from the run queue on each core every second.

C. Machine Learning Models and Algorithms

Convolutional neural networks (CNNs) have been deployed successfully in a variety of AI applications, including image classification, face recognition, and object detection. In 2012, AlexNet was proposed and a variety of neural networks have been invented since then. In 2014, researchers from Oxford proposed VGG proving that increasing the depth of networks can affect final results. Compared to AlexNet, VGG16 [34] used several consecutive 3×3 convolution kernels to replace the larger convolution kernels (e.g., 7×7) in AlexNet. VGG19 [21] has three more convolutional layers than VGG16, which performs better than VGG16 but requires more computation. The most direct way to improve network performance is to increase the network depth (number of network layers) and width (number of neurons). However, this method could introduce too many parameters, produce overfitting, increase the calculation complexity and make it difficult for optimization.

Another solution to improve neural network performance is to add more customized functions on convolutional layers. In 2014, GoogLeNet proposed Inception structure [38] to build a high-performance network. Inception V1 has a sparse network structure but can generate dense data, which not only improves the neural network performance, but also guarantees the resource efficiency. Inception V2 [18] updates the internal calculation logic of V1 and proposes factorizing convolutions. Inception V3 [39] adds factorization to speed up the calculation and increase the depth and nonlinearity of the network. Inception V4 [37] uses residual connection and further evolved to Inception-ResNet-v1 and Inception-ResNet-v2 networks. With the popularity of mobile and IoTs, it is also increasingly critical to study small and efficient CNN models. In 2017, Google announced MobileNet V1 [15], a lightweight deep neural network with depthwise separable convolution for embedded devices. Next year, Google released MobileNet V2 [31], which integrated inverted residual block with V1. The experiment proves that V2 has similar amount of calculation but V2 achieves better accuracy than MobileNet V1.

Although CNN has been widely used in image-level tasks like object recognition and classification, many other AI workloads such as image enhancement and object segmentation require pixel-level detection and analysis. Because CNN loses image details during convolution and pooling, it could not well indicate the specific outline of the object and thus many new neural networks were designed based on CNN by adding new features and modules. U-Net [30] is a segmentation network that stitches features together in the channel to achieve semantic segmentation. PSPNet [44] introduces more context information on fully convolutional network (FCN) algorithm and adopts global average pooling and feature fusion to avoid false segmentation. ICNet [43] is built on PSPNet and uses low-resolution semantic information as well as high-resolution image details to achieve image semantic segmentation. DeepLab [9] combines deep convolutional neural networks and dense conditional random field, and achieves good results in semantic segmentation. In terms of image super-resolution, many neural networks are also proposed in combination with knowledge in other fields. For instance, SRCNN [11] proposed using deep learning and traditional sparse coding to apply in single-image super-resolution reconstruction. SRGAN [22] found that the image would be too smooth and lack of realism if the image magnification is larger than four. Therefore, it proposed to take advantage of generative adversarial network (GAN) to generate more details in the super-resolution image.

IV. Object Recognition and Classification Performance and Analysis

We first evaluated the recognition and classification workloads on edge devices and cloud VM. We used MobileNet-V2 model to recognize a 224×224-pixel photo from 1000 different object classes. As Figure 2(a) shows, MobileNet is fast and lightweight, which consumes very little CPU and memory. For instance, compared to the Inception-V3 CNN with same workloads, MobileNet only requires less than half of RAM and the average CPU is nearly an order of magnitude lower, which is more suitable to deployed on the edge and IoT devices with resource constraints. Next, we executed Inception-V3 network, which is comprised of 11 inception blocks that mainly consist of 1×1, 1×3+3×1, 1×7+7×1 and 3×3 con-

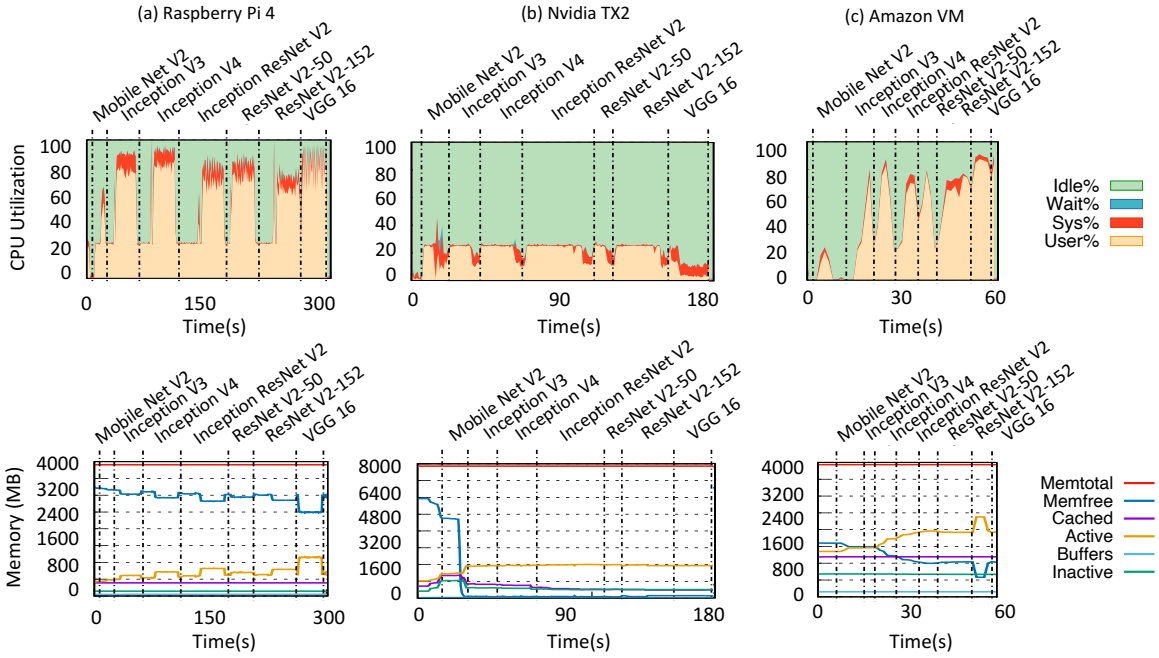

Fig. 2: CPU and memory utilization of object recognition applications on edge devices and cloud VMs.

volutional layers, to recognize the same image set but with larger input. Compared to MobileNet, Inception-V3 is larger and more accurate, but consumes a much higher average CPU as well as more memory. Inception-V4 also uses the similar deep convolutional network and the performance and resource footprint difference are relatively small compared to Inception-V3. Lastly, we evaluated face recognition performance using Inception-Resnet-V2 neural network. The size of input images is 346×346 pixels, and the dimensionality of feature vectors is 128. Different from Inception-V3 and V4, Inception-Resnet-V2 adopts a residual connection, which significantly improves recognition performance. As depicts in Figure 2, the CPU utilization is much less than Inception-V4 but the Inception-Resnet-V2 consumes more memory than the general deep convolutional network models. We also evaluated neural network ResNet V2-50 with input 346×346, and ResNet V2-152 with input 256×256. The peak CPU of ResNet V2-50 is higher as the input picture is larger. However, ResNet V2-152 runs longer because it has more layers to calculate. VGG is by far the most expensive architecture in terms of computing requirements and number of parameters, although it has been widely adopted in many applications. It can be observed from Figure 2 that the average CPU, peak CPU and execution time are similar as Inception and ResNet models even though the input picture of VGG16 is much smaller. In addition, VGG16 consumes more memory, even if the input data is the same as MobileNet. Therefore, it is highly recommended not deploying VGG model on edge or IoT devices.

Besides above, we also observed that the most of image recognition and classification workloads were CPU-intensive. Therefore, we evaluated the same AI workloads on Nvidia TX2, another edge device but equipped with an accelerator. As Figure 2(b) shows, the CPU consumption are reduced significantly due to the computation offloading onto GPUs. For example, under Nvidia TX2, the CPU utilization of all applications does not exceed 25% in comparison with more than 80% on the Raspberry Pi. However, there also exist additional overhead with external accelerators. First, the GPU computation consumes lots of memory for input data, temporary value and program instructions, and the free memory drops significantly when the workload executes, as shown in Figure 2(b). For instance, when Inception-V3 executes the same load, Nvidia TX2 requires 2GB memory while Raspberry Pi only needs 600MB. Second, we also found the process context switch increases dramatically with GPUs. As shown in Figure 3, the process switch number per second increases with an order of magnitude compared to the computation with only CPUs. This is due to the fact that the CPU has to interact with the accelerators during the computation. For instance, the CUDA framework function *cudaMemcpy* is frequently processed during the data transmission between CPUs and GPUs. In addition, we also evaluate the same workload on cloud VM of Amazon EC2 and compare the results with edges. As depicted in Figure 2(c) and Figure 3, the resource footprint of edge devices (such as Raspberry Pi) performs similarly to that of VMs on the cloud. However, the speed of cloud servers far exceeds that of edge nodes.

V. Image-to-image Performance and Analysis

Next, we evaluated the performance and resource footprint of image-to-image workloads. As compute-intensive workloads, the performance of image-to-image tasks on general edges is similar as cloud VMs. Like the results in Section IV, the heterogeneous architecture with accelerators reduces the CPU utilization while consuming more memory. However, such the hardware could effectively strengthen the system stability. We

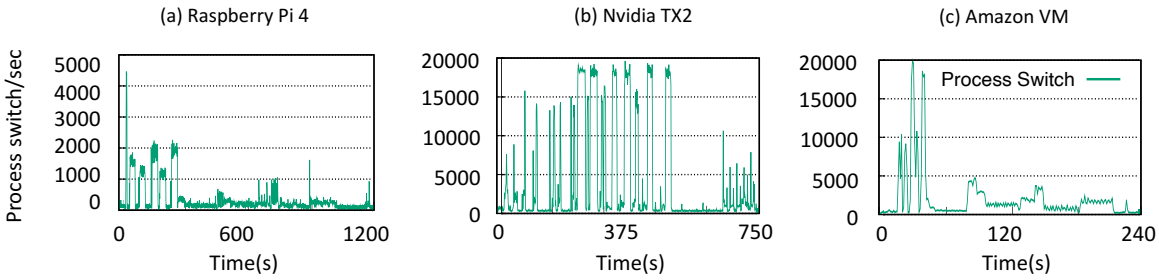

Fig. 3: Process context switches of AI workloads on edge devices and cloud VMs.

also observed similar results on Google coral with edge TPUs. As depicted in Figure 4, compared to the Nvidia TX2, the AI workload performance on cloud VMs is much close with that on Raspberry Pi but with smaller variation. Instead, as the OS is heavier and the cloud is equipped with thicker software stack, the VM consumes more memory executing the same AI workloads. In the following sections, we analyze individual application with their neural network models.

Image deblurring is built on one of the most primitive and lightweight neural network SRCNN. The structure of SRCNN is very simple and only has three convolutional layers. For a low-resolution image, this method first uses bicubic interpolation to enlarge to the target size, and then performs nonlinear mapping through a three-layer convolution network to output a high-resolution image. The size of the convolution kernel of three layers are 9×9, 5×5 and 5×5. As shown in Figure 4, SRCNN model is much faster than VGG19 and DPED. In our results, we also note that the processing time of SRCNN is highly linear with the resolution of the test image. The size of the highest image pixel that SRCNN can handle increases linearly with the total RAM size of the device.

Compared to VGG16 in Section IV, there is no essential difference of VGG19 except the network depth. Based on AlexNet, VGG network uses a continual 3×3 small convolution kernel to replace the larger convolution kernel in AlexNet (e.g., 11×11), which can better extract deep features of data. Different from the previous models, VGG has a deeper number of layers (19 layers in VGG19) and more parameters (138 million parameters in VGG19), and thus consumes more computing resources and executes much longer. We found that most of its parameters are from the first fully connected layer.

SRGAN adopts generative adversarial network (GAN) for image super resolution. The traditional methods focus on a smaller image magnification. However, once the magnification rate is above four, the generated images would become too smooth and are lack of details. For the same input size, the CPU utilization of SRGAN is much less than SRCNN and VGG19, and the memory consumption is also smaller. DPED also used GAN for image enhancement. The generation network has 12 layers, which used three-channel image patch as input, then performed by four residual blocks with two convolution layers in each block, another three convolutional layers and one three-channel convolutional layer. The discriminator network has five convolutional layers and finally generates a 2-dimensional probability vector. DPED proposes a variety of loss functions, including content loss, textures loss, detail enhancement, etc., to effectively evaluate the image quality. As shown in Figure 4, DPED is also a compute bound workload with similar CPU usage and memory consumption as other neural network models.

U-Net is designed for semantic segmentation and here the workload adopts U-Net to simulate the bokeh in the photo. U-Net is a fully convolutional neural network where the input and output are both images without fully connected layer. The shallow high-resolution layer is used for locating pixel while the deep layer is designed for pixel classification. Because the adjacent pixel blocks are repeatedly processed, there exists a lot of repetitive calculations and peak CPU is always high. Unlike the classic convolutional neural network that uses fully connected layers to obtain fixed-length feature vectors, U-Net can accept input images of any size and restore it to the same size of input image using deconvolution layers for sampling. Therefore, its memory utilization, as illustrated in Figure 4, is stable during the execution.

In recent years, many works have been proposed in image generation, such as spatially-adaptive normalization (SPADE) displayed by Nvidia at GTC 2019. SPADE is built on BatchNorm [18] and the input of generator is a vector which is calculated by an image encoder and a specific style image. After processing, SPADE generates a synthetical image based on user scribble. Our measurements found that the CPU and memory usage fluctuated during the SPADE processing. We speculated that it is due to the fact that generators are implemented by stacking multiple residual blocks. In the SPADE architecture, the vector is processed by encoders through inverse normalization. When using the trained model to generate images, it can directly use the vector as generator input. As each generator is compute-intensive, the performance fluctuation can be observed on general edge devices as well as the cloud VMs.

VI. Segmentation Performance and Analysis

Lastly, we evaluated the performance and system footprint of various segmentation workloads. PSPNet is one of the most widely used semantic segmentation algorithms. First, the features of input image are extracted through a feature extraction network. The features of different depths are obtained by pooling operations of different scales. Next, an 1×1 convolutional layer reduces the feature dimension to a quarter of the original one. Finally, these features are directly

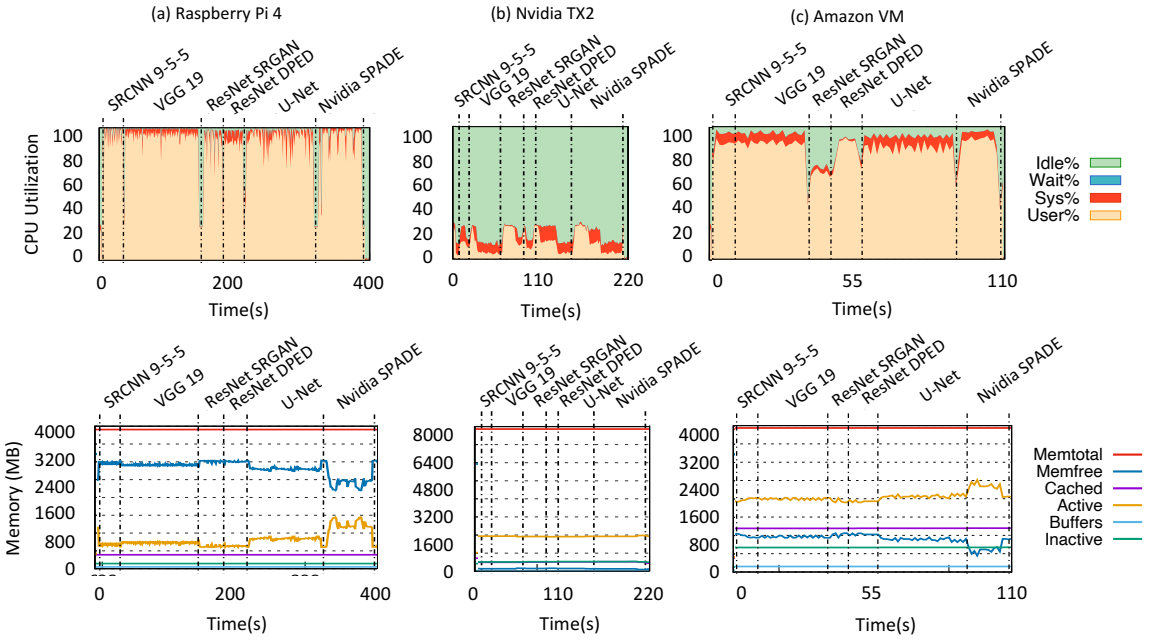

Fig. 4: CPU and memory utilization of image-to-image applications.

sampled to the same size and merged with the input features. The process of feature merging is to fuse the details of the target (shallow features) and the context information (global features). As there needs to provide global context, feature receptive field outputted by the feature extraction network must be large enough. From resource perspective shown in Figure 5, PSPNet has to run long enough and consumes much CPU and memory resources.

ICNet is a real-time semantic segmentation network based on PSPNet, which is designed to reduce the inference time. It combines the efficiency of processing low-resolution images and the inference quality of high-resolution images. Specifically, the idea is to first obtain prediction based on low-resolution images. Then, ICNet provides cascading feature fusion unit and label to integrate high-resolution features and gradually refine semantic graph. Experiments show that ICNet can achieve a fast and high-quality segmentation model with low computational cost. As depicted in Figure 5, although the input of ICNet is larger, it still runs faster than PSPNet and the peak CPU and memory consumption is much lower, which is more suitable for deployment on edge or IoT devices.

DeepLab is a semantic segmentation network proposed by Google in 2016. When processing semantic segmentation, the previous DCNN (Deep Convolutional Neural Network) either makes the output resolution smaller in multiple pooling or loses important information when continuous downsampling. DeepLab used hollow convolution, which can effectively control the receptive field and adjust the resolution. Meanwhile, by setting the step size, DeepLab can reduce the resolution while increasing the receptive field of hollow convolution. DeepLab consists of DCNN and conditional random field (CRF). The DCNN model uses pre-trained VGG16. Therefore, its resource consumption is relatively high in both peak CPU and memory usage. In addition, the predicted object location of DCNN is not accurate and the object outline is rough. The process of input images is gradually abstracted. Therefore, we can observe lots of CPU and memory usage fluctuates in Figure 5. Similar to the previous evaluations, the Nvidia TX2 offloads most of the computation onto GPUs and consumes more memory. Due to the more powerful hardware, the cloud server performs similar to the Raspberry Pi but is more stable and faster than the edges. The CPU variation of execution on the cloud is only 6% while edge CPU variation is nearly 15%.

VII. Related Work

Artificial intelligence on edges. As increasing AI and ML applications are deployed on the IoTs, a growing number of studies start to focus on analyzing the efficiency at edges. Canziani *et al.* [8] report an analysis of deep neural networks (DNN) with their accuracy, memory footprint, parameters, power consumption, etc. Bianco *et al.* [6] presents case studies of more than 40 state-of-the-art DNN architectures trained on ImageNet-1k. However, these works only focused on specific types of applications such as image recognition and our work is more in-depth and comprehensive for all representative workloads. In 2017, Shanthamallu *et al.* [32] assessed various ML models and methods in IoT data analytics, and presented different algorithms and their efficiencies in extracting higher level data from IoT devices. In 2018, Dong *et al.* [23] presented research that focused on processing capabilities for deep learning at the edge and Zhang *et al.* [41] studied various edge devices with different ML packages. Different from the above research working on efficient deep learning algorithms or artificial intelligence models, our study focused on the comparison and analysis of the system resources level and how well they are processed and utilized by different models.

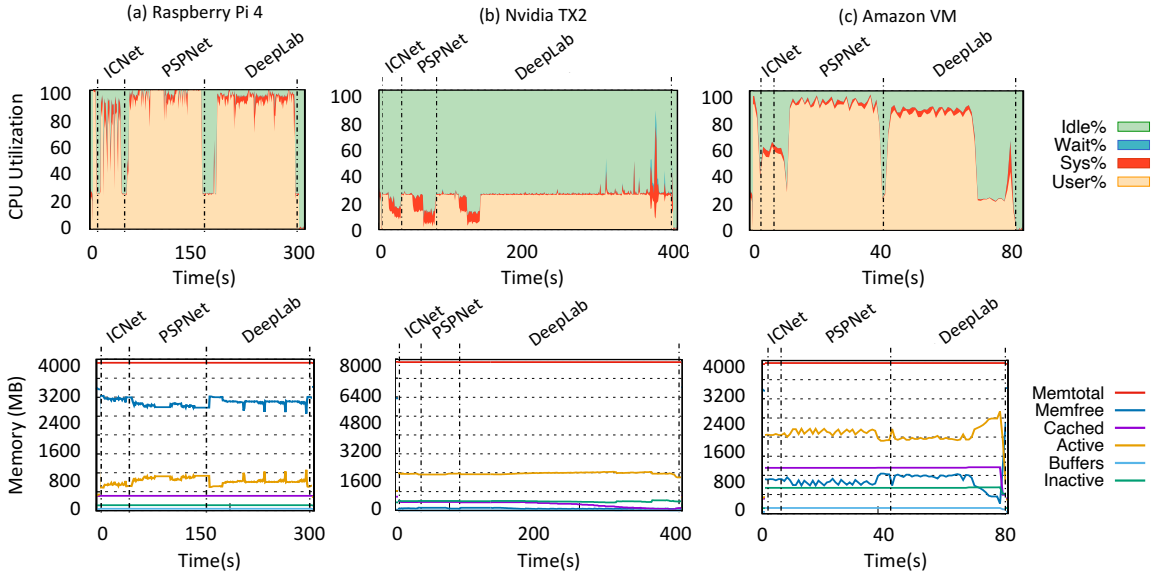

Fig. 5: CPU and memory utilization of image segmentation applications.

Infrastructure efficiency on edges. Recently, many studies explored how to optimize infrastructure performance to improve edge computing efficiency for AI or ML workloads. Morabito *et al.* [26] proposed to consolidate edge computing with lightweight virtualization including containers and unikernels. Son *et al.* [35] proposed a framework with SDN and NFV at edge environments, which greatly improve end-to-end delay, network traffic, and power consumption in different scenarios. Several other papers studies computation offloading [36], frequency scaling [10], cloud-edge assisted [23] to improve the energy efficiency, application performance and system security. For instance, Li *et al.* [23] proposed to divide DNN model into two parts and deployed the initial layers computation on edge servers and higher layers on the cloud. DeepDecision [29] and MCDNN [13] take an optimization-based offloading approach with constraints such as network latency and bandwidth, energy, and cost. Those studies are orthogonal to our work.

Edge infrastructure resource management. Much effort [5], [7], [19], [20], [24], [27], [33], [45] has been dedicated to analyzing factors that affect edge application performance and proposing effective solutions. Zhu *et al.* [45] designed and implemented Slim, a low-overhead container overlay network in which packets inside only traverse the network stack exactly one time. Khalid *et al.* [19] reported that a container with heavy network traffic can decrease the compute resource available to its neighbors on the same server, and thus proposed a scheme, named Iron, to precisely account the consumed CPU time and enforce fair resource allocation. Other works, including the virtual routing, resource management [16], redistribution and reassignment [42], hardware offloading or bypassing the inefficient parts inside kernel [40], focus on optimizing the data path and improving system network processing. There is also a large body of work dedicated to elastic resource management and high resource utilization. For instance, Mohan *et al.* [25] proposed optimizing cold start through pre-allocating virtual network interfaces that are later bound to new function containers. Different from the above works, this research focuses on investigating the efficiency and system footprint of AI workloads on edge infrastructure.

VIII. Conclusion and Insights

In this paper, we present a detailed analysis of representative AI workloads on various edge devices. Specifically, we perform a comprehensive empirical study and quantitatively evaluate the classification, image-to-image, and segmentation applications with different neural network models. We compared the performance discrepancy on different types of edge hardware as well as cloud platforms. To the best of our knowledge, this paper is the first to explore the AI workloads performance and their resource footprint on the edge platforms. Our analysis and findings could help the users deploy the appropriate models and workloads on their scenarios, and shed light on guiding the optimization of AI workloads on modern edge environments. To summarize, this paper has the following findings and insights:

- General and heterogeneous edge architecture with accelerators present a significant performance difference of AI workloads. Good performance can be attained by offloading computation to accelerators while it might sacrifice with massive memory consumption.
- For the same type of workload, cloud and general edge perform similarly and have analogous resource footprints. For AI applications on edges, local hardware such as SoC and memory determine its performance while the cloud is faster and more stable.
- Same types of workloads perform comparably but algorithm models such as neural networks could impose certain overhead on resources. This is due to complex

interactions between the AI workloads and the software stacks in edge devices.

- AI processing in edge devices is also concerned with other factors such as response time, model volume and cost. Overall, only a few AI workloads and related models are suitable to execute or deploy on edges.

ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their comments and suggestions on this paper. This work was supported in part by U.S. NSF grants CPS-2103459, SHF-2210744, AMPS-2229073 and CNS-2103405.

REFERENCES

[1] *AI Benchmark*. http://ai-benchmark.com/.

[2] *Big Data and What it Means*. https://www.uschamberfoundation.org/bhq/big-data-and-what-it-means.

[3] *Edge Internet Economy*. https://bit.ly/2TyZbRw.

[4] *nmon*. http://nmon.sourceforge.net.

[5] D. Bermbach, A.-S. Karakaya, and S. Buchholz. Using application knowledge to reduce cold starts in faas services. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020.

[6] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. Benchmark analysis of representative deep neural network architectures. In *Proceedings of IEEE Access*, 2018.

[7] J. Cadden, T. Unger, Y. Awad, H. Dong, O. Krieger, and J. Appavoo. Seuss: skip redundant paths to make serverless fast. In *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020.

[8] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. In *arXiv preprint arXiv:1605.07678*, 2016.

[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[10] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen. Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing. In *Proceedings of IEEE TCC*, 2019.

[11] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *arXiv preprint arXiv:1501.00092*, 2015.

[12] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of ACM FPGA*, 2017.

[13] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of ACM MobiSys*, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[16] Y. Hu, M. Song, and T. Li. Towards full containerization in containerized network function virtualization. In *Proceedings of ACM ASPLOS*, 2017.

[17] A. Ignatov, N. Kobyshev, K. Vanhoey, R. Timofte, and L. Van Gool. Dslr-quality photos on mobile devices with deep convolutional networks. *arXiv preprint arXiv:1704.02470*, 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[19] J. Khalid, E. Rozner, W. Felter, C. Xu, K. Rajamani, A. Ferreira, and A. Akella. Iron: Isolating network-based cpu in container environments. In *Proceedings of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[20] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan. Freeflow: Software-based virtual RDMA networking for containerized clouds. In *Proceedings of USENIX NSDI*, 2019.

[21] J. Kim, J. K. Lee, and K. M. Lee. Accurate image super-resolution using very deep convolutional networks. *arXiv preprint arXiv:1511.04587*, 2016.

[22] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2017.

[23] H. Li, K. Ota, and M. Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. In *Proceedings of IEEE Network*, 2018.

[24] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018.

[25] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov. Agile cold starts for scalable serverless. In *Proceedings of USENIX HotCloud*, 2019.

[26] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott. Consolidate iot edge computing with lightweight virtualization. In *Processing of IEEE Network*, 2018.

[27] H. D. Nguyen, C. Zhang, Z. Xiao, and A. A. Chien. Real-time serverless: Enabling application performance guarantees. In *Proceedings of the 5th International Workshop on Serverless Computing*, 2019.

[28] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. Semantic image synthesis with spatially-adaptive normalization. *arXiv preprint arXiv:1903.07291*, 2019.

[29] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *Proceedings of IEEE INFOCOM*, 2018.

[30] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE CVPR*, 2018.

[32] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley. A brief survey of machine learning methods and their sensor and iot applications. In *Processing of IISA*, 2017.

[33] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse, and H. Weatherspoon. X-containers: Breaking down barriers to improve performance and isolation of cloud-native containers. In *Proceedings of ACM ASPLOS*, 2019.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015.

[35] J. Son, T. He, and R. Buyya. Cloudsimsdn-nfv: Modeling and simulation of network function virtualization and service function chaining in edge computing environments. In *Proceedings of Software: Practice and Experience*, 2019.

[36] H. Sun, F. Zhou, and R. Q. Hu. Joint offloading and computation energy efficiency maximization in a mobile edge computing system. In *Proceedings of IEEE Transactions on Vehicular Technology*, 2019.

[37] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE CVPR*, 2015.

[39] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE CVPR*, 2016.

[40] J. Weerasinghe and F. Abel. On the cost of tunnel endpoint processing in overlay virtual networks. In *Proceedings of International Conference on Utility and Cloud Computing (UCC)*, 2014.

[41] X. Zhang, Y. Wang, and W. Shi. pcamp: Performance comparison of machine learning packages on the edges. In *Processing of HotEdge*, 2018.

[42] Y. Zhang, Y. Li, K. Xu, D. Wang, M. Li, X. Cao, and Q. Liang. A communication-aware container re-distribution approach for high performance vnfs. In *Proceedings of IEEE ICDCS*, 2017.

[43] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. Icnet for real-time semantic segmentation on high-resolution images. *arXiv preprint arXiv:1704.08545*, 2018.

[44] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *arXiv preprint arXiv:1612.01105*, 2017.

[45] D. Zhuo, K. Zhang, Y. Zhu, H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson. Slim: OS kernel support for a low-overhead container overlay network. In *Proceedings of USENIX NSDI*, 2019.