OpenEI: An Open Framework for Edge Intelligence

Xingzhou Zhang*†, Yifan Wang*†, Sidi Lu*, Liangkai Liu*, Lanyu Xu* and Weisong Shi*
*Department of Computer Science, Wayne State University, Detroit, MI, USA, 48202
†Institute of Computing Technology, University of Chinese Academy of Sciences, Beijing, China, 100190
{zhangxingzhou, wangyifan2014}@ict.ac.cn, {lu.sidi, liangkai, xu.lanyu, weisong}@wayne.edu

[1] ***Abstract*—In the last five years, edge computing has attracted tremendous attention from industry and academia due to its promise to reduce latency, save bandwidth, improve availability, and protect data privacy to keep data secure. At the same time, we have witnessed the proliferation of AI algorithms and models which accelerate the successful deployment of intelligence mainly in cloud services. These two trends, combined together, have created a new horizon: Edge Intelligence (EI). The development of EI requires much attention from both the computer systems research community and the AI community to meet these demands.**

However, existing computing techniques used in the cloud are not applicable to edge computing directly due to the diversity of computing sources and the distribution of data sources. We envision that there missing a framework that can be rapidly deployed on edge and enable edge AI capabilities. To address this challenge, in this paper we first present the definition and a systematic review of EI. Then, we introduce an Open Framework for Edge Intelligence (OpenEI), which is a lightweight software platform to equip edges with intelligent processing and data sharing capability. We analyze four fundamental EI techniques which are used to build OpenEI and identify several open problems based on potential research directions. Finally, four typical application scenarios enabled by OpenEI are presented.

***Index Terms*—Edge intelligence, edge computing, deep learning, edge data analysis, cloud-edge collaboration**

I. Introduction

With the burgeoning growth of the Internet of Everything, the amount of data generated by edge increases dramatically, resulting in higher network bandwidth requirements. Meanwhile the emergence of novel applications calls for lower latency of the network. Based on these two main requirements, EC arises, which refers to processing the data at the edge of the network. Edge Computing (EC) guarantees quality of service when dealing with a massive amount of data for cloud computing [1]. Cisco Global Cloud Index [2] estimates that there will be 10 times more useful data being created (85 ZB) than being stored or used (7.2 ZB) by 2021, and EC is a potential technology to help bridge this gap.

At the same time, Artificial Intelligence (AI) applications based on machine learning (especially deep learning algorithms) are fueled by advances in models, processing power, and big data. Nowadays, applications are built as a central attribute, and users are beginning to expect near-human interaction with the appliances they use. For example, since the sensors and cameras mounted on an autonomous vehicle generate about one gigabyte of data per second [3], it is hard to upload the data and get instructions from the cloud in real-time. As for mobile phone applications, such as those related with face recognition and speech translation, they have high requirements for running either online or offline.

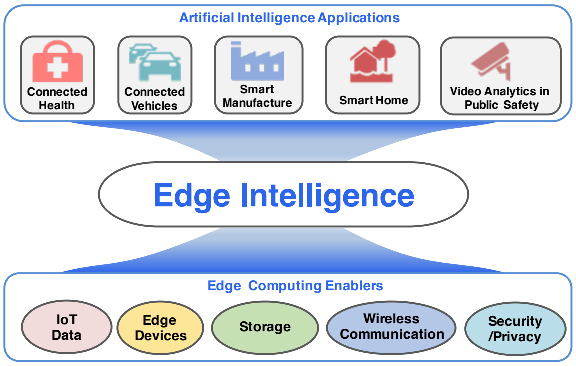


Fig. 1. Motivation of Edge Intelligence.

Pushed by EC techniques and pulled by AI applications, Edge Intelligence (EI) has been pushed to the horizon. As is shown in Figure 1, the development of EC techniques, including powerful IoT data, edge devices, storage, wireless communication, and security and privacy make it possible to run AI algorithms on the edge. AI applications, including connected health, connected vehicles, smart manufacturing, smart home, and video analytics call for running on the edge. In the EI scenario, advanced AI models based on machine learning algorithms will be optimized to run on the edge. The edge will be capable of dealing with video frames, natural speech information, time-series data and unstructured data generated by cameras, microphones, and other sensors without uploading data to the cloud and waiting for the response.

Migrating the AI functions from the cloud to the edge is highly regarded by industry and academy. Forbes listed the convergence of IoT and AI on the edge as one of five AI trends in 2019 [4]. Forbes believes that most of the models trained in the public cloud will be deployed on the edge and edge devices will be equipped with special AI chips based on FPGAs and ASICs. Microsoft provides Azure IoT Edge [5], a fully managed service, to deliver cloud intelligence locally by deploying and running AI algorithms and services on cross-

[1] This paper has been accepted by ICDCS 2019. Please cite: Xingzhou Zhang, Yifan Wang, Sidi Lu, Liangkai Liu, Lanyu Xu, Weisong Shi, *OpenEI: An Open Framework for Edge Intelligence*, in Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS), July 7-10, 2019, Dallas, USA.

platform edge devices. Similar to Azure IoT Edge, Cloud IoT Edge [6] extends Google Cloud's data processing and machine learning to billions of edge devices by taking advantage of Google AI products, such as TensorFlow Lite and Edge TPU. AWS IoT Greengrass [7] has been published to make it easy to perform machine learning inference locally on devices, using models that have been trained and optimized in the cloud.

However, several challenges when offloading state-of-the-art AI techniques on the edge directly, including

- *Computing power limitation*. The edge is usually resource-constrained compared to the cloud data center, which is not a good fit for executing DNN represented AI algorithms since DNN requires a large footprint on both storage (as big as 500MB for VGG-16 Model [8]) and computing power (as high as 15300 MMA for executing VGG-16 model [9]).
- *Data sharing and collaborating*. The data on the cloud data center is easy to be batch processed and managed, which is beneficial in terms of the concentration of data. However, the temporal-spatial diversity of edge data creates obstacles for the data sharing and collaborating.
- *Mismatch between edge platform and AI algorithms*. The computing power on the cloud is relatively consistent while edges have diverse computing powers. Meanwhile, different AI algorithms have different computing power requirements. Therefore, it is a big challenge to match an existing algorithm with the edge platform.

To address these challenges, this paper proposes an Open Framework for Edge Intelligence, *OpenEI*, which is a lightweight software platform to equip the edge with intelligent processing and data sharing capability. To solve the problems that the EC power limitation brings, *OpenEI* contains a lightweight deep learning package (*package manager*) which is designed for the resource constrained edge and includes optimized AI models. In order to handle the data sharing problem, `libei` is designed to provide a uniform RESTful API. By calling the API, developers are able to access all data, algorithms, and computing resources. The heterogeneity of the architecture is transparent to the user, which makes it possible to share data and collaborate between edges. In order to solve the mismatch problem, *OpenEI* designs a model selector to find the most suitable models for a specific targeting edge platform. The model selector refers to the computing power (such as memory and energy) that the algorithm requires and the edge platform provides. The contributions of this paper are as follows:

- A formal definition and a systematic analysis of EI are presented. Each EI algorithm is defined as a four-element tuple *ALEM* $< Accuracy, Latency, Energy, Memory\ footprint >$.
- *OpenEI*, an Open Framework for Edge Intelligence, is proposed to address the challenges of EI, including computing power limitations, data sharing and collaborating, and the mismatch between edge platform and AI algorithms.
- Four key enabling techniques of EI and their potential directions are depicted. Several open problems are also identified in the paper.

The remainder of this paper is organized into six sections. In Section II, we define EI and list the advantages of EI. We present *OpenEI* to support EI in Section III. Four key techniques that enable EI are explained in Section IV, including algorithms, packages, running environments, and hardware. EI is designed to support many potential applications, such as live video analytic for public safety, connected and autonomous driving, smart home, and smart and connected health, which are illustrated in Section V. Finally, Section VI concludes the paper.

II. The Definition of Edge Intelligence

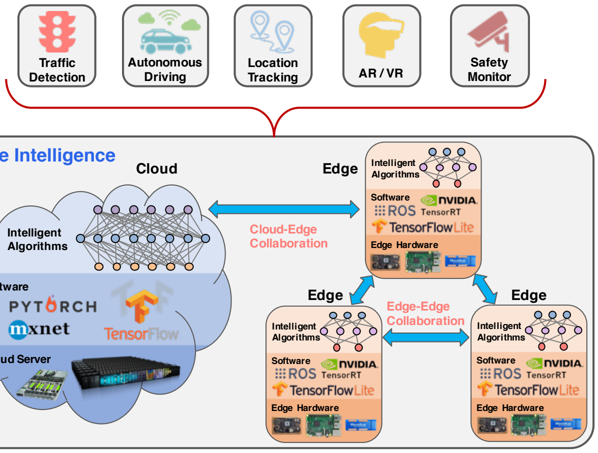


Fig. 2. Edge Intelligence.

A. Motivation

The development of EI comes from two aspects. On the one hand, the burgeoning growth of the IoT results in a dramatically increasing amount of IoT data, which needs to be processed on the edge. In this paper, **we define IoT as the billions of physical devices around the world that are securely connected to the Internet, individually or collaboratively, collecting and sharing data, applying intelligence to actuate the physical world in a safe way.** On the other hand, the emergence of AI applications calls for a higher requirement for real-time performance, such as autonomous driving, real-time translation, and video surveillance. EI is presented to deal with this massive edge data in an intelligent manner.

B. Definition

Currently, many studies related to EI are beginning to emerge. International Electrotechnical Commission (IEC) defines EI as the process of when the data is acquired, stored and processed with machine learning algorithms at the network edge. It believes that several information technology and operational technology industries are moving closer to the edge of the network so that aspects such as real-time networks, security capabilities, and personalized/customized connectivity are addressed [10]. In 2018, [11] discussed the challenges and the opportunities that EI created by presenting a use-case showing that the careful design of the convolutional neural networks (CNN) for object detection would lead to real-time performance on embedded edge devices. [12] enabled EI for activity recognition in smart homes from multiple perspectives, including architecture, algorithm and system.

In this paper, **we define EI as the capability to enable edges to execute artificial intelligence algorithms.** The diversity of edge hardware results in different in AI models or algorithms they carry; that is, edges have different EI capabilities. Here the capability is defined as a four-element tuple $< Accuracy, Latency, Energy, Memory\ footprint >$ which is abbreviated as *ALEM*. *Accuracy* is the internal attribute of AI algorithms. In practice, the definition of *Accuracy* depends on specific applications; for example, it is measured by the mean average precision (mAP) in object detection tasks and measured by the BLEU score metric in machine translation tasks. To execute the AI tasks on the edge, some algorithms are optimized by compressing the size of the model, quantizing the weight and other methods that will decrease accuracy. Better EI capability means that the edge is able to employ the algorithms with greater *Accuracy*. *Latency* represents the inference time when running the trained model on the edge. To measure the *Latency*, we calculate the average latency of multiple inference tasks. When running the same models, the *Latency* measures the level of performance of the edge. *Energy* refers to the increased power consumption of the hardware when executing the inference task. *Memory footprint* is the memory usage when running the AI model. *Energy* and *Memory footprint* indicate the computing resource requirements of the algorithms.

EI involves much knowledge and technology, such as AI algorithm design, software and system, computing architecture, sensor network and so on. Figure 2 shows the overview of EI. To support EI, many techniques have been developed, called EI techniques, which include algorithms, software, and hardware. There is a one-to-one correspondence between the cloud and the single edge. From algorithms perspective, the cloud data centers train powerful models and the edge does the inference. With the development of EI, the edge will also undertake some local training tasks. From the software perspective, the cloud runs the cluster operating system and deep learning framework, such as TensorFlow [13] and MXNet [14]. On the edge, both the embedded operating system and the stand-alone operating system are widely used. The lightweight deep learning package is used to speed up the execution, such as TensorFlow Lite [15] and CoreML [16]. From the hardware perspective, cloud data centers are deployed on high-performance platforms, such as GPU, CPU, FPGA, and ASIC clusters while the hardware of the edge are heterogeneous edges, such as edge server, mobile phone, Raspberry Pi, *etc.*

C. Collaboration

As shown in Figure 2, there are two types of collaboration for EI: cloud-edge and edge-edge collaboration. In the cloud-edge scenario, the models are usually trained on the cloud and then downloaded to the edge which executes the inference task. Sometimes, edges will retrain the model by transfer learning based on the data they generated. The retrained models will be uploaded to the cloud and combined into a general and global model. In addition, researchers have also focused on the distributed deep learning models over the cloud and edge. For example, DDNN [17] is a distributed deep neural network architecture across the cloud and edge.

Edge-edge collaboration has two aspects. First, multiple edges work collaboratively to accomplish a compute-intensive task. For example, several edges will be distributed when training a huge deep learning network. The task will be allocated according to the computing power. Second, multiple edges work together to accomplish a task with different divisions based on different environments. For example, in smart home environments, a smartphone predicts when a user is approaching home, triggering and the smart thermostat will be triggered to set the suitable temperature for the users. Individually, every task is particularly difficult, but the coordination within the edge makes it easy.

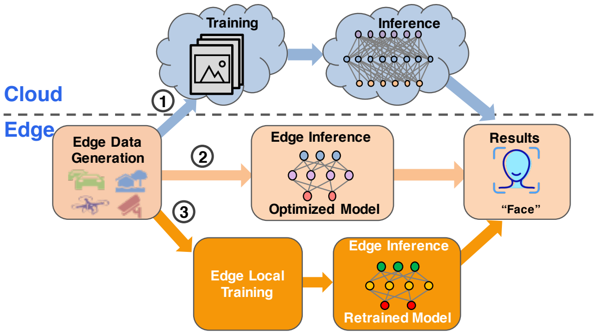


Fig. 3. Dataflow of Edge Intelligence.

D. Dataflow of EI

As is shown in Figure 3, the data generated by the edge comes from different sources, such as cars, drones, smart homes, *etc.* and has three data flows:

- First is uploading the data to the cloud and training based on the multi-source data. When the model training is completed, the cloud will do the inference based on the edge data and send the result to the edge. This dataflow is widely used in traditional machine intelligence.
- Second is executing the inference on the edge directly. The data generated by the edge will be the input of the edge model downloaded from the cloud. The edge will do the inference based on the input and output the results. This is the current EI dataflow.
- Third is training on the edge locally. The data will be used to retrain the model on the edge by taking advantage of transfer learning. After retraining, the edge will build a personalized model which has better performance for the data generated on the edge. This will be the future dataflow of EI.

III. *OpenEI*: Open Framework for Edge Intelligence

In this section, we introduce an Open Framework for Edge Intelligence (*OpenEI*), a lightweight software platform to equip the edge with intelligent processing and data sharing capability. The goal of *OpenEI* is that any hardware, ranging from Raspberry Pi to a powerful Cluster, will become an intelligent edge after deploying *OpenEI*. Meanwhile, the EI attributes, accuracy, latency, energy, and memory footprint, will have an order of magnitude improvement comparing to the current AI algorithms running on the deep learning package.

A. *Requirements*

Let us use an example of building an EI application to walk through the requirements of *OpenEI*. If we want to enable a new Raspberry Pi EI capability, deploying and configuring *OpenEI* is enough. After that, the Raspberry Pi is able to detect multiple objects directly based on the data collected by the camera on board and meet the real-time requirement. It also has the ability to execute the trajectory tracking task collaborated with other *OpenEI* deployed edges. Several questions may arise: how does Raspberry Pi collect, save, and share data? How does Raspberry Pi run a powerful object detection algorithm in the real-time manner? How does Raspberry Pi collaborate with others?

To realize the example above, *OpenEI* should meet the following four requirements: ease of use, optimal selection, interoperability, and optimization for the edge. The detailed explanations are as follows:

Ease of use. Today, it is not very straightforward to deploy the deep learning framework and run AI models on the edge because of the current complicated process to deploy and configure. Drawing on the idea of plug and play, *OpenEI* is deploy and play. By leveraging the API, *OpenEI* is easy to install and easy to develop third-party applications for users.

Optimal selection. The biggest problem is not the lack of algorithms, but how to choose a matched algorithm for a specific configuration of the edge. The model selector is designed to meet the requirements.

Interoperability. To collaborate with the cloud and other heterogeneous edges, *OpenEI* is designed as a cross-platform software. `libei` provides RESTful API for the edge to communicate and work with others.

Optimization for the edge. To run heavy AI algorithms on the edge, being lightweight is the core feature as well as a significant difference between *OpenEI* and other data analyze platforms. Two methods are used to optimize the algorithm for the edge. One is adopting the *package manager* which has been optimized for the edge and cutting out the redundancy operations unrelated to deep learning. The other is running lightweight algorithms which have been co-optimized with the package.

The answers will be found in the design of *OpenEI*. Figure 4 shows the overview of *OpenEI*, which consists of three components: a *package manager* to run inference and train the model locally, a *model selector* to select the most suitable model for the edge, and `libei`, a library including a RESTful API for data sharing.

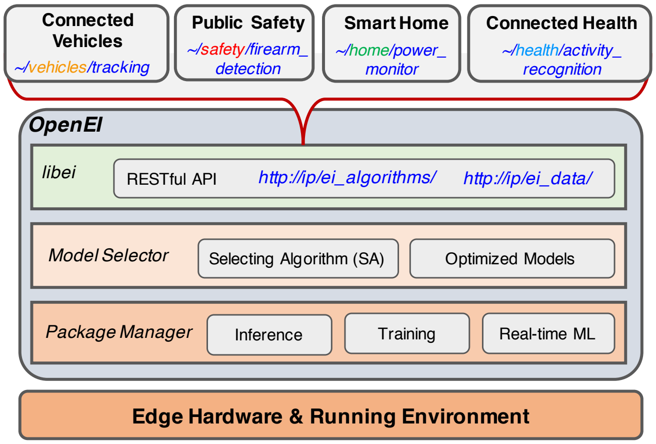


Fig. 4. The overview of *OpenEI*.

B. *Package Manager*

Similar to TensorFlow Lite [15], *package manager* is a lightweight deep learning package which has been optimized to run AI algorithms on the edge platform, which guarantees the low power consumption and low memory footprint. *package manager* is installed on the operating system of edge hardware and provides a running environment for AI algorithms. In addition to supporting the inference task as TensorFlow Lite does, *package manager* also supports training the model locally. By retraining the model based on the local data, *OpenEI* provides a personalized model which performs better than general models.

Emerging computing challenges require real-time learning, prediction, and automated decision-making in diverse EI domains such as autonomous vehicles and health-care informatics. To meet the real-time requirement, *package manager* contains a real-time machine learning module. When the module is called, the machine learning task will be set to the highest priority to ensure that it has as many computing resources as possible. Meanwhile, the models are optimized for the *package manager* since the co-optimization of the framework and algorithms is capable of increasing the system performance and speedup the execution. That is why Raspberry Pi has the ability to run a powerful object detection algorithm smoothly.

C. *Model Selector*

Currently, neural network based models have started to trickle in. We envision that the biggest problem is not the lack of models, but how to select a matched model for a specific edge based on different EI capabilities. The *model selector* includes multiple optimized AI models and a selecting algorithm (SA). The optimized models have been optimized to present a better performance on the *package manager* based on the techniques which will be discussed in detail in Section IV.A. Model selecting can be regarded as a multi-dimensional space selection problem. As is shown in Figure 5, there are at least three dimensions to choose, e.g., AI models, machine learning packages, and edge hardware. Taking image classification as an example, more than 10 AI models (AlexNet, Vgg, ResNet, MobileNet, to name a few), 5 packages (TensorFlow, PyTorch, MXNet, to name a few), and 10 edge hardware platforms (NVIDIA Jetson TX2, Intel Movidius, Mobile Phone, to name a few) need to be considered.

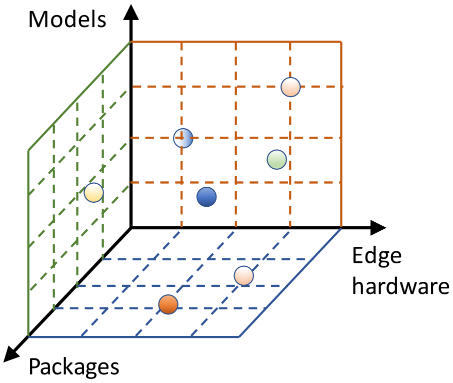


Fig. 5. Model selector.

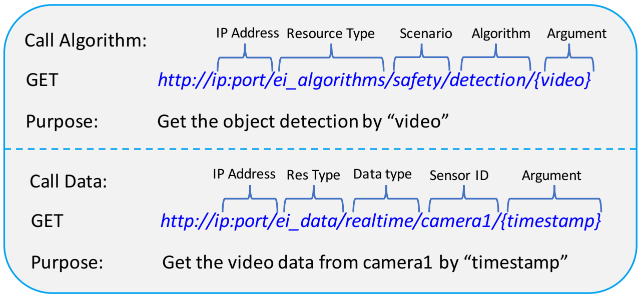
SA in *model selector* is designed to find the most suitable models for the specific edge platform based on users' requirements. It will first evaluate the EI capability of the hardware platform based on the four-element tuple *ALEM* $< Accuracy, Latency, Energy, Memory\ footprint >$ and then selecting the most suitable combinations, which is regarded as an optimization problem:

$$\begin{aligned} &\underset{m\in Models}{\arg\min} && L \\ &s.t. && A \geq A_{req}, E \leq E_{pro}, M \leq M_{pro} \end{aligned} \tag{1}$$

where A, L, E, M refer to *Accuracy, Latency, Energy, Memory footprint* when running the models on the edge. A_{req} denotes the lowest accuracy that meet the application's requirement. E_{pro} and M_{pro} are the energy and memory footprint that the edge provides. m refers to the selected models and $Models$ refers to all the models. Equation 1 depicts the desire to minimize $Latency$ while meeting the $Accuracy$, $Energy$ and $Memoryfootprint$ requirements. Meanwhile, if users pay more attention to $Accuracy$, the optimization target will be replaced by maximize A and the constraints are L, E, and M. The same is true of other requirements, i.e. $Energy$ and $Memoryfootprint$. Deep reinforcement learning will be leveraged to find the optimal combination.

D. `libei`

`libei` provides a RESTful API which makes it possible to communicate and work together with the cloud, other edges, and IoT devices. Every resource, including the data, computing resource, and models, are represented by a URL whose suffix is the name of the desired resource. As is shown in Figure 6, the RESTful API provided by `libei` consists of four fields. The first field is the IP address and port number of the edge. The second field represents the type of recourse, including the algorithm whose suffix is $ei\_algorithms$ and the data whose suffix is ei_data. If users call for the algorithm, the third field indicates the application scenario that *OpenEI* supports, including connected vehicles, public safety, smart home, and connected health. The last field is the specific algorithm that the application scenario needs. The argument is the parameter required for algorithm execution. In terms of calling for data APIs, the third field indicates the data's type, including real-time data and historical data and the last field represents the sensor's ID. Developers will get the data over a period of time by the start and end which are provided by the timestamp argument.


Call Algorithm:

IP Address, Resource Type, Scenario, Algorithm, Argument

GET http://ip:port/ei_algorithms/safety/detection/{video}

Purpose: Get the object detection by "video"

Call Data:

IP Address, Res Type, Data type, Sensor ID, Argument

GET http://ip:port/ei_data/realtime/camera1/{timestamp}

Purpose: Get the video data from camera1 by "timestamp"

Fig. 6. The RESTful API of *libei*

E. Summary

At last, let echo the original example of building an object detection application on the Raspberry Pi to introduce the programming model and summarized the processing flow of *OpenEI*. When *OpenEI* has been deployed on the Raspberry Pi, the developer is able to visit http://ip:port/ei_data/realtime/camera1/timestamp=present_time to get the real-time video frames which could save on the Raspberry Pi. Subsequently, the URI http://ip:port/ei_algorithms/safety/detection/video=video will be visited to call for the object detection function and response the detection results to the developer.

In terms of the processing flow of *OpenEI*, when `libei` receives the instruction of object detection, the model selector will choose a most suitable model from the optimized models based on the developer's requirement (the default is accuracy oriented) and the current computing resource of the Raspberry Pi. Subsequently, *package manager* will call the deep learning package to execute the inference task. If the application is urgent, the real-time machine learning module will be called to guarantee the latency.

IV. Key Techniques

Many techniques from AI and EC promote the development of EI. In this section, we summarize the key techniques and classify them into four aspects: algorithms, packages, running environments and hardware. The implement of *OpenEI* will leverage the latest techniques in algorithms and packages.

OpenEI will be installed on the running environments and the performance of the hardware will offer significant references when designing the model selector of *OpenEI*.

A. Algorithms

Although neural networks are currently widely deployed in academia and industry to conduct nonlinear statistical modeling with state-of-the-art performance on problems that were previously thought to be complex, it is difficult to deploy neural networks on edge devices with limited resources due to their intensive demands on computation and memory. To address this limitation, two main categories of solutions have been recently proposed. One category refers to the deep model compression method, which aims to aid the application of current advanced networks in devices. The other is the EI algorithm, which refers to the efficient machine learning algorithms that we developed to run on the resource-constrained edges directly.

1) Compression: Compression techniques are roughly categorized into three groups: Parameter sharing and pruning methods, low-rank approximation methods, and knowledge transfer methods [18], [19].

Parameter sharing and pruning method control the capacity and storage cost by reducing the number of parameters which are not sensitive to the performance. This method only needs to save the values of these representatives and the indexes of these parameters. Courbariaux *et al.* [20] proposed a binary neural network to quantify the weights. More specifically, it refers to restricting the value of the network weight by setting it to -1 or 1, and it simplifies the design of hardware that is dedicated to deep learning. Gong *et al.* [21] employed the k-means clustering algorithm to quantize the weights of fully connected layers, which could achieve up to 24 times the compression of the network with only 1% loss of classification accuracy for the CNN network in the ImageNet challenge. Chen *et al.* [22]presented a HashedNets weight sharing architecture that groups connection weights into hash buckets randomly by using a low-cost hash function, where all connections of each hash bucket have the same value. The values of the parameters are adjusted using the standard backpropagation method [23] during training. Han *et al.* [24] pruned redundant connections using a three-step method: First, the network learns which connections are important, and then they prune the unimportant connections. Finally, they retrain the network to fine tune the weights for the remaining connections.

Low-rank approximation refers to reconstructing the dense matrix to estimate the representative parameters. Denton *et al.* [25] use singular value decomposition to reconstruct the weight of all connected layers, and they triple the speedups of convolutional layers on both CPU and GPU, and the loss of precision is controlled within 1%. Denil *et al.* [26] employ low-rank approximation to compress the weights of different layers and reduce the number of dynamic parameters. Sainath [27] uses a low-rank matrix factorization on the final weight layer of a DNN for acoustic modeling.

Knowledge transfer is also called teacher-student training. The idea of knowledge transfer is to adopt a teacher-student strategy and use a pre-trained network to train a compact network for the same task [28]. It was first proposed by Caruana *et al.* [29]. They used a compressed network of trained network models to mark some unlabeled simulation data and reproduced the output of the original larger network. The work in [30] trained a parametric student model to estimate a Monte Carlo teacher model. Ping *et al.* [31] use the neurons in the hidden layer to generate more compact models and preserve as much of the label information as possible. Based on the idea of function-preserving transformations, the work in [32] instantaneously transfers the knowledge from a teacher network to each new deeper or wider network.

TABLE I
TYPICAL APPROACHES FOR DEEP COMPRESSION.

Method	Description	Advantages	Disadvantages
Parameter sharing and pruning	Reducing uninformative parameters that are not sensitive to the performance	Robust to various settings, support training from scratch and pre-trained model	Pruning requires manual setup of sensitivity for layers, which demands fine-tuning of the parameters and may be cumbersome for some applications.
Low-rank factorization	Using the matrix decomposition method to figure out the representative parameters	Straightforward for model compression, standardized pipeline, support pre-trained models and training from scratch	The implementation involves the decomposition operation, which is computationally expensive
Knowledge transfer	Using a pre-trained neural network to train a compact network on the same task	Make deeper models thinner, significantly reduce the computational cost	Only be applied to the classification tasks with softmax loss functions, network structure only support training from scratch

Table I concludes the above three typical compression technologies, and describes the advantages and disadvantages of each technology.

2) EI algorithm: In this paper, we define EI algorithms as the those designed for the resource-constrained edges directly. Google Inc. [9] presented efficient CNN for mobile vision applications, called MobileNets. The two hyperparameters that Google introduced allow the model builder to choose the right sized model for the specific application. It not only focuses on optimizing for latency but also builds small networks. MobileNets are generated mainly from depthwise separable convolutions, which were first introduced in the work of [33] and subsequently employed in Inception models [34]. Flattened networks [35] are designed for fast feedforward execution. They consist of a consecutive sequence of one-dimensional filters that span every direction of three-dimensional space to achieve comparable performance as conventional convolutional networks [36]. Another small network is the Xception network [37]; Chollet *et al.* proposes the dubbed Xception architecture inspired by Inception V3, where Inception modules have been replaced with depthwise separable convolutions. It shows that the architecture slightly outperforms Inception V3 on the ImageNet data set. Subsequently, Iandola *et al.* [38] developed Squeezenet, a small

CNN architecture. It achieves AlexNet-level [39] accuracy with 50 times fewer parameters on ImageNet data set (510 times smaller than AlexNet).

In 2017, Microsoft Research India proposed Bonsai [40] and ProtoNN [41]. Then, they developed EMI-RNN [42] and FastGRNN [43] in 2018. Bonsai [40] refers to a tree-based algorithm used for efficient prediction on IoT devices. More specifically, it is designed for supervised learning tasks such as regression, ranking, and multi-class classification, etc. ProtoNN [41] is inspired by k-Nearest Neighbor (KNN) and could be deployed on the edges with limited storage and computational power (e.g., an Arduino UNO with 2kB RAM) to achieve excellent prediction performance. EMI-RNN [42] requires 72 times less computation than standard Long Short-term Memory Networks (LSTM) [44] and improving accuracy by 1%.

Open Problems: Here are three main open problems that need to be addressed to employ EI algorithms on edges. First, to reduce the size of algorithms, many techniques have been proposed to reduce the number of connections and parameters in neural network models. However, the pruning process usually affects algorithm accuracy. Hence, how to reduce the model size while guaranteeing high accuracy is a research direction in the EI area. Second, collaboration between edges calls for an algorithm that runs in a distributed manner on multiple edges. It is a challenge to research how to split an algorithm based on the computing resources of the edges. Third, how to achieve collaborative learning on the cloud and edges is also a research direction.

B. Packages

In order to execute AI algorithms efficiently, many deep learning packages are specifically designed to meet the computing paradigm of AI algorithms, such as TensorFlow, Caffe, MXNet, and PyTorch. However, these packages are focused on the cloud and not suitable for the edge. On the cloud, packages use a large-scale dataset to train deep learning models. One of the main tasks of packages is to learn a number of weights in each layer of a model. They are deployed on the high-performance platforms, such as GPU, CPU, FPGA, and ASIC (TPU [45]) clusters. On the edges, due to limited resources, packages do not train models in most cases. They carry on inference tasks by leveraging the models which have been trained in the cloud. The input is small-scale real-time data and the packages are installed on the heterogeneous edges, such as edge server, mobile phone, Raspberry Pi, laptop, etc.

To support processing data and executing AI algorithms on the edges, several edge-based deep learning packages have been released by some top-leading tech-giants. Compared with cloud versions, these frameworks require significantly fewer resources, but behave almost the same in terms of inference. TensorFlow Lite [15] is TensorFlow's lightweight solution which is designed for mobile and edge devices. It leverages many optimization techniques, including optimizing the kernels for mobile apps, pre-fused activations, and quantized kernels to reduce the latency. Apple published CoreML [16], a deep learning package optimized for on-device performance to minimizes memory footprint and power consumption. Users are allowed to integrate the trained machine learning model into Apple products, such as Siri, Camera, and QuickType. Facebook developed QNNPACK (Quantized Neural Networks PACKage) [46], which is a mobile-optimized library for high-performance neural network inference. It provides an implementation of common neural network operators on quantized 8-bit tensors.

In the meantime, cloud-based packages are also starting to support edge devices, such as MXNet [14] and TensorRT [47]. MXNet is a flexible and efficient library for deep learning. It is designed to support multiple platforms (either cloud platforms or edge ones) and execute training and inference tasks. TensorRT is a platform for high-performance deep learning inference, not training and will be deployed on the cloud and edge platforms. Several techniques, including weight and activation precision calibration, layer and tensor fusion, kernel auto-tuning, and multi-stream execution are used to accelerate the inference process.

Zhang *et al.* made a comprehensive performance comparison of several state-of-the-art deep learning frameworks on the edges and evaluated the latency, memory footprint, and energy of these frameworks with two popular deep learning models on different edge devices [48]. They found that no framework could achieve the best performance in all dimensions, which indicated that there was a large space to improve the performance of AI frameworks on the edge. It is very important and urgent to develop a lightweight, efficient and highly-scalable framework to support AI algorithms on edges.

Open Problems: There are several open problems that need to be addressed to be able to build data processing frameworks on the edge. First, to execute real-time tasks on the edge, many packages sacrifice memory to reduce latency. However, memory on the edge is also limited. Thus, how to tradeoff the latency and memory? Second, having access to personalized, training on the edge is ideal while the training process usually requires huge computing resources. Therefore, how to implement a local training process with limited computing power? Last, with the support of *OpenEI*, the edge will need to handle multiple tasks which raises the problem of how to execute multiple tasks on a package in the meantime.

C. Running environments

The most typical workloads from EI are model inference and collaborative model training, so the EI running environments should be capable of handling deep learning packages, allocating computation resources and migrating computation loads. Meanwhile, they should be lightweight enough and can be deployed on heterogeneous hardware platforms. Taking the above characteristic into account, some studies like TinyOS, ROS, and OpenVDAP are recognized as potential systems to support EI.

TinyOS [49] is an application based operating system for sensor networks. The biggest challenge that TinyOS has solved is to handle concurrency intensive operations with small physical size and low power consumption [50]. TinyOS takes an event-driven design which is composed of a tiny scheduler and a components graph. The event-driven design makes TinyOS achieve great success in sensor networks. However, enabling effective computation migration is still a big challenge for TinyOS.

Robot Operating System(ROS) [51] is recognized as the typical representative of next the generation of mobile operating systems to cope with the Internet of Things. In ROS, the process that performs computations is called a node. For each service, the program or features are divided into several small pieces and distributed on several nodes, and the ROS topic is defined to share messages between ROS nodes. The communication-based design of ROS gives it high reusability for robotics software development. Meanwhile, the active community and formation of the ecosystem put ROS in a good position to be widely deployed for edge devices. However, as ROS is not fundamentally designed for resource allocation and computation migration, there are still challenges in deploying EI service directly on ROS.

OpenVDAP [52] is an edge based data analysis platform for Connected and Autonomous Vehicles(CAVs). OpenVDAP is a full stack platform which contains Driving Data Integrator(DDI), Vehicle Computing Units(VCU), edge-based vehicle operating system($EdgeOS_v$), and libraries for vehicular data analysis($libvdap$). Inside OpenVDAP, VCU supports EI by allocating hardware resources according to an application, and $libvdap$ supports EI by providing multi-versions of models to accelerate the model inference.

Open Problems: There is a crucial open problem that needs to be addressed: how to design a lightweight edge operating system with high availability. For the scenario with dynamic changes in topology and high uncertainty in wireless communication, the edge operating system calls for high availability related to the consistency, resource management, computation migration, and failure avoidance. Meanwhile, the edge operating system should be light enough to be implemented on the computing resource-constraint edge.

D. Hardware

Recently, heterogeneous hardware platforms present the potential to accelerate specific deep learning algorithms while reducing the processing time and energy consumption [53], [54]. For the hardware on EI, various heterogeneous hardware are developed for particular EI application scenario to address the resource limitation problem in the edge.

ShiDianNao [55] first proposed that the artificial intelligence processor should be deployed next to the camera sensors. The processor accesses the image data directly from the sensor instead of DRAM, which reduces the power consumption of sensor data loading and storing. ShiDianNao is 60 times more energy efficient and 30 times faster than the previous state-of-the-art AI hardware, so it will be suitable for the EI applications related to computer vision. EIE [56] is an efficient hardware design for compressed DNN inference. It leverages multiple methods to improve energy efficiency, such as exploiting DNN sparsity and sharing DNN weights, so it is deployed on mobile devices to process some embedded EI applications. In industry, many leaders have published some dedicated hardware modules to accelerate EI applications; for example, IBM TrueNorth [57] and Intel Loihi [58] are both the neuromorphic processors.

In addition to ASICs, several studies have deployed FPGAs or GPUs for EI application scenarios, such as speech recognition. ESE [59] used FPGAs to accelerate the LSTM model on mobile devices, which adopted the load-balance-aware pruning method to ensure high hardware utilization and the partitioned compressed LSTM model on multiple PEs to process LSTM data flow in parallel. The implementation of ESE on Xilinx FPGA achieved higher energy efficiency compared with the CPU and GPU. Biookaghazadeh *et al.* used a specific EI workload to evaluate FPGA and GPU performance on the edge devices. They compared some metrics like data throughput and energy efficiency between the FPGA and GPU. The evaluation results showed that the FPGA is more suitable for EI application scenarios [60]. In industry, NVIDIA published the Jetson AGX Xavier module [61], which is equipped with a 512-core Volta GPU and an 8-core ARM 64-bit CPU. It supports the CUDA and TensorRT libraries to accelerate EI applications in several scenarios, such as robot systems and autonomous vehicles.

Open Problems: There are several open problems that need to be addressed to design a hardware system for EI scenarios. First, novel hardware designed for EI has improved the processing speed and energy efficiency; hence, the question remains whether there is any relationship between the processing speed and power. For example, if the processing power is limited, we need to know how to calculate the maximum speed that the hardware reaches. Second, the EI platform may be equipped with multiple types of heterogeneous computing hardware, so managing the hardware resource and scheduling the EI application among the types of hardware to ensure high resource utilization are important questions. Third, we need to be able to evaluate how suitable the hardware system is for each specific EI application.

V. Typical Application Scenarios

With the development of EI techniques, many novel applications are quickly emerging, such as live video analytics for public safety, connected and autonomous driving, smart homes, and smart and connected health care services. As shown in Figure 4, *OpenEI* provides RESTful API to support these AI scenarios. This section will illustrate the typical application scenarios and discuss how to leverage *OpenEI* to support these applications.

A. Video Analytics in Public Safety

Video Analytics in Public Safety(VAPS) is one of the most successful applications on edge computing since it has the high real-time requirements and unavoidable communication overhead. *OpenEI* is used to deploy on cameras or edge severs to support VAPS and provide an API for the user. Third-party developers execute the widely used algorithms on public safety scenarios by calling *http://ip:port/ei_algorithms/safety/* plus the name of the algorithms. The applications of video analysis for public safety that *OpenEI* supports are divided into the following two aspects.

The first aspect is from the algorithm perspective, which is aimed at designing a lightweight model to support EI. The strength of edge devices is that the data is stored locally so there is no communication delay. However, one drawback is that edge devices are not powerful enough to implement large neural networks; the other is that the vibration in a video frame makes it more difficult to analyze. For example, criminal scene auto detection is a typical application of VAPS. The challenges are created by real-time requirements and the mobility of the criminal. Hence, the model should do preprocessing on each frame to evict the influence of mobility. In addition to criminal scene auto detection, for some applications like High-Definition Map generation, masking some private information like people's face is also a potential VAPS application. The objective is to enable the edge server to mask the private information before uploading the data. Video frame preprocessing at the edge supports EI by accelerating the model training and inference process.

The second aspect is from the system perspective, which enables edge devices like smartphones and body cameras to run machine learning models for VAPS applications. In [62], an edge based real-time video analysis system for public safety is proposed to distribute the computing workload in both the edge node and the cloud in an optimized way. A3 [63] is an edge based amber alert application which support distributed collaborative execution on the edge. SafeShareRide [64] is an edge based detection platform which enables a smartphone to conduct real-time detection including video analysis for both passengers and drivers in ridesharing services. Moreover, a reference architecture which enables the edge to support VAPS applications is also crucial for EI. In [65], Liu *et al.* proposed a reference architecture to deploy VAPS applications on police vehicles. EI will be supported through efficient data management and loading.

B. Connected and Autonomous Vehicles

Today, a vehicle is not just a mechanical device but is gradually becoming an intelligent, connected, and autonomous system. We call these advanced vehicles connected and autonomous vehicles (CAVs). CAVs are significant application scenarios for EI and many applications on CAVs are tightly integrated into EI algorithms, such as localization, object tracking, perception, and decision making. *OpenEI* also provides API for the CAVs scenarios to execute the AI algorithm on a vehicle. The input is the video data collected by on-board cameras.

The autonomous driving scenario has conducted many classic computer vision and deep learning algorithms [66], [67]. Since the algorithms will be deployed on the vehicle, which is a resource-constrained and real-time EC system, the algorithm should consider not only precision but also latency, as the end-to-end deep learning algorithm YOLOv3 [68]. To evaluate the performance of algorithms in the autonomous driving scenario, Geiger *et al.* published KITTI benchmark datasets [69], which provide a large quantity of camera and LiDAR data for various autonomous driving applications.

Lin *et al.* explored the hardware computing platform design of autonomous vehicles [70]. They chose three core applications on autonomous vehicles, which are localization, object detection, and object tracking, to run on heterogeneous hardware platform: GPUs, FPGAs, and ASICs. According to the latency and energy results, they provided the design principle of the end-to-end autonomous driving platform. From the industry, NVIDIA published the DRIVE PX2 platform for autonomous vehicles [71]. To evaluate the performance of the computing platform designed for CAVs, Wang *et al.* first proposed CAVBench [72], which takes six diverse on-vehicle applications as evaluation workloads and provides the matching factor between the workload and the computing platform.

In the real world, we still need a software framework to deploy EI algorithms on the computing platform of connected and autonomous vehicle. OpenVDAP [52], Autoware [73], and Baidu Apollo [74] are open-source software frameworks for autonomous driving, which provide interfaces for developers to build and customize autonomous driving vehicles.

C. Smart Homes

Smart homes have become popular and affordable with the development of EC and AI technologies. By leveraging different types of IoT devices (e.g., illuminate devices, temperature and humidity sensors, surveillance system, etc.), it is feasible to keep track of the internal state of a home and ensure its safety, comfort, and convenience under the guarantee of EI. The benefit of involving EI in a smart home is twofold. First, home privacy will be protected since most of the computing resources are confined to the home internal gateway and sensitive family data is prohibited from the outflow. Second, the user experience will be improved because the capability of intelligent edge devices facilitates the installation, maintenance, and operation of a smart home with less labor demanded. As an important EI scenario, *OpenEI* provides APIs to call the AI algorithms related to the smart home. *http://ip:port/ei_algorithms/home/power_monitor* is used to call to execute the power monitoring algorithms on the edge.

Intelligence in the home has been developed to some extent, and related products are available on the market. As one of the most intelligent devices in the smart home ecosystem, smart speaker such as Amazon Echo [75], Google Home [76] are quite promising models that involve in EI. They accept the user's instructions and respond accordingly by interacting with a third party service or household appliances. From timing to turning off lights, from memo to shopping, their intelligence enhances people's quality of life significantly. Despite this, the utility of the edge is not well reflected and utilized in this technology. Relying on cloud cognitive services, smart speakers need to upload data to the cloud and use deep neural networks for natural language understanding and processing, which becomes a hidden danger of family data privacy leakage and increases the burden of unnecessary network transmission. EI is the principal way to solve these problems.

Considering the privacy of the home environment and the accessibility of smart home devices, it is completely feasible and cost-effective to offload intelligent functions from the cloud to the edge, and there have been some studies demonstrating EI capabilities. Wang *et al.* found that a smart home will benefit from EI to achieve energy efficiency [77]. Zhang *et al.* developed IEHouse, a non-intrusive status recognition system, for household appliance [78] with the assistance of deep neural networks. Zhang *et al.* proposed a CNN model running on edge devices in a smart home to recognize activity with promising results [12]. In addition to indoor activity detection, surveillance systems play an important role in protecting the home security both indoor and outside. Because the video stream occupies a considerable storage space and transmission bandwidth, it is almost impossible to upload every frame recorded from a surveillance system to the cloud for further processing, especially for high resolution videos [79]. EI enables a surveillance device to have certain image processing capabilities, such as object recognition and activity detection, to extract valid information from redundant videos to save unnecessary computing and storage space.

Home entertainment systems also benefit from EI to provide a better user experience. A home audio and video system is one typical example. With EI involved, the system handles the user's personalized recommendation service by itself, without uploading any privacy data about the user's preferences to the cloud, so that the user has a smoother and safer entertainment experience. Meanwhile, with the maturity of Augmented Reality and Virtual Reality technology, users are able to have a better game immersive experience. MUVR is proposed in this scenario to boost the multi-user gaming experience with the edge caching mechanism [80]. Motion sensing games are a typical example. EI gives it the capability to detect action and behavior without equipping users with a control bar or body sense camera.

D. Smart and Connected Health

Health and biomedicine are entering a data-driven epoch [81], [82]. On the one hand, the development of medical instruments indicates health status with data. On the other hand, the popularization of health awareness has led citizens to track their physical condition with smart edge devices. Similar to the other three scenarios above, *OpenEI* also provides the API to call for the related algorithms. The EI of smart health is quite promising and is created from the following aspects.

First is pre-hospital emergency medicine, where the emergent patient is been cared for before reaching the hospital, or during emergency transfer between hospitals, emergency medical service (EMS) systems are provided in the form of basic life support (BLS) and advanced life support (ALS). Current EMS systems focus on responsiveness and transportation time, while the health care solutions are traditional and less efficient, some of which have been used since the 1990s. Although ALS is equipped with higher level care, the number of ALS units is highly constrained because of limited budgets [83]. In addition, the data transmission is greatly affected by the moving scenario and the extreme weather in the cloud computing. Considering the limitation of the status quo, EI is an alternative way to enhance EMS quality in terms of responsiveness and efficiency by building a bidirectional real time communication channel between the ambulance and the hospital, which has intelligent features like natural language processing, and image processing.

Second is smart wearable sensors. Most of the current technologies for smart wearable sensors are based on cloud computing because of the limitations of computing resources and capabilities. That is, wearable sensors are more like a data collector than a data analyst. Fortunately, EI research in this field is emerging. Rincon *et al.* deployed an artificial neural network on wearable sensors to detect emotion [84]. With the promising development of EC, there will be more light-weight intelligent algorithms running on smart wearable devices to monitor, analyze, and predict health data in a timely manner, which it will ease the pressure on caregivers and doctors, and let users have better knowledge of their physical condition.

Third is the preservation and sharing of medical data. The US government has promoted the sharing of personal health information since 2009 [85], but it turns out that although doctors' usage of electronic medical records has improved, interoperability is missing, and the large amount of medical data gathered together does not produce real value. EI improves the status quo by training the sharing system to mark and recognize medical images, promote communication, and improve treatment efficiency.

VI. Conclusion

With the development of AI and EC, EI emerges since it has the potential to reduced bandwidth and cost while maintaining the quality of service compared to processing on the cloud. In this paper, we define EI as a capability that enables edges to execute artificial intelligence algorithms. To support EI, several techniques are being developed, including algorithms, deep learning packages, running environments and hardware. This paper discussed the challenges that these techniques brings and illustrated four killer applications in the EI area.

To address the challenges for data analysis of EI, computing power limitation, data sharing and collaborating, and the mismatch between the edge platform and AI algorithms, we presented an Open Framework for Edge Intelligence (*OpenEI*) which is a lightweight software platform to equip the edge with intelligent processing and data sharing capability. We hope that *OpenEI* will be used as a model for prototyping in EI. We also hope that this paper provides helpful information to researchers and practitioners from various disciplines when designing new technologies and building new applications for EI.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] Cisco. (November 19, 2018) Cisco global cloud index: Forecast and methodology, 2016–2021 white paper. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

[3] B. L. Mearian. (2013) Self-driving cars could create 1GB of data a second. [Online]. Available: https://www.computerworld.com/article/2484219/emerging-technology/self-driving-cars-could-create-1gb-of-data-a-second.html

[4] J. MSV. (2018) 5 Artificial Intelligence Trends To Watch Out For In 2019. [Online]. Available: https://www.forbes.com/sites/janakirammsv/2018/12/09/5-artificial-intelligence-trends-to-watch-out-for-in-2019/amp/

[5] (2019) Azure iot edge. https://azure.microsoft.com/en-us/services/iot-edge/. [Online]. Available: https://azure.microsoft.com/en-us/services/iot-edge/

[6] (2019) Cloud IoT Edge: Deliver Google AI capabilities at the edge. https://cloud.google.com/iot-edge/. [Online]. Available: https://cloud.google.com/iot-edge/

[7] (2019) Aws iot greengrass. https://aws.amazon.com/greengrass/. [Online]. Available: https://aws.amazon.com/greengrass/

[8] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[9] 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,” *arXiv preprint arXiv:1704.04861*, 2017.

[10] IEC. (2018) Edge intelligence (white paper). [Online]. Available: https://basecamp.iec.ch/download/iec-white-paper-edge-intelligence-en/

[11] G. Plastiras, M. Terzi, C. Kyrkou, and T. Theocharidcs, “Edge intelligence: Challenges and opportunities of near-sensor machine learning applications,” in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–7.

[12] S. Zhang, W. Li, Y. Wu, P. Watson, and A. Zomaya, “Enabling edge intelligence for activity recognition in smart homes,” in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2018, pp. 228–236.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “TensorFlow: A System for Large-Scale Machine Learning,” in *OSDI*, vol. 16, 2016, pp. 265–283.

[14] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” in *LearningSys at NIPS*. ACM, 2015.

[15] (2018) Introduction to TensorFlow Lite. https://www.tensorflow.org/mobile/tflite/. [Online]. Available: https://www.tensorflow.org/mobile/tflite/

[16] (2019) Core ml: Integrate machine learning models into your app. https://developer.apple.com/documentation/coreml. [Online]. Available: https://developer.apple.com/documentation/coreml

[17] S. Teerapittayanon, B. McDanel, and H. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 328–339.

[18] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv preprint arXiv:1710.09282*, 2017.

[19] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[20] M. Courbariaux, Y. Bengio, and J.-P. B. David, “Training deep neural networks with binary weights during propagations. arxiv preprint,” *arXiv preprint arXiv:1511.00363*, 2015.

[21] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arXiv preprint arXiv:1412.6115*, 2014.

[22] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[23] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[24] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[25] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[26] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, “Predicting parameters in deep learning,” in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[27] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, “Low-rank matrix factorization for deep neural network training with high-dimensional output targets,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6655–6659.

[28] B. B. Sau and V. N. Balasubramanian, “Deep model compression: Distilling knowledge from noisy teachers,” *arXiv preprint arXiv:1610.09650*, 2016.

[29] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[30] A. K. Balan, V. Rathod, K. P. Murphy, and M. Welling, “Bayesian dark knowledge,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3438–3446.

[31] P. Luo, Z. Zhu, Z. Liu, X. Wang, X. Tang *et al.*, “Face model compression by distilling knowledge from neurons.” in *AAAI*, 2016, pp. 3560–3566.

[32] T. Chen, I. Goodfellow, and J. Shlens, “Net2net: Accelerating learning via knowledge transfer,” *arXiv preprint arXiv:1511.05641*, 2015.

[33] L. Sifre and S. Mallat, “Rigid-motion scattering for image classification,” Ph.D. dissertation, Citeseer, 2014.

[34] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[35] J. Jin, A. Dundar, and E. Culurciello, “Flattened convolutional neural networks for feedforward acceleration,” *arXiv preprint arXiv:1412.5474*, 2014.

[36] M. Wang, B. Liu, and H. Foroosh, “Factorized convolutional neural networks.” in *ICCV Workshops*, 2017, pp. 545–553.

[37] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *arXiv preprint*, pp. 1610–02 357, 2017.

[38] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[40] A. Kumar, S. Goyal, and M. Varma, “Resource-efficient machine learning in 2 kb ram for the internet of things,” in *International Conference on Machine Learning*, 2017, pp. 1935–1944.

[41] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, “Protonn: compressed and accurate knn for resource-scarce devices,” in *International Conference on Machine Learning*, 2017, pp. 1331–1340.

[42] D. Dennis, C. Pabbaraju, H. V. Simhadri, and P. Jain, “Multiple instance learning for efficient sequential data classification on resource-constrained devices,” in *Advances in Neural Information Processing Systems*, 2018, pp. 10 976–10 987.

[43] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, “Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9031–9042.

[44] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[46] D. Marat, W. Yiming, and L. Hao. (2018) Qnnpack: Open source library for optimized mobile deep learning. [Online]. Available: https://code.fb.com/ml-applications/qnnpack/

[47] (2018) Nvidia tensorrt: Programmable inference accelerator. https://developer.nvidia.com/tensorrt. [Online]. Available: https://developer.nvidia.com/tensorrt

[48] X. Zhang, Y. Wang, and W. Shi, "pCAMP: Performance Comparison of Machine Learning Packages on the Edges," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/zhang

[49] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, "TinyOS: An operating system for sensor networks," in *Ambient intelligence*. Springer, 2005, pp. 115–148.

[50] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGOPS operating systems review*, vol. 34, no. 5, pp. 93–104, 2000.

[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[52] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong, "OpenVDAP: An open vehicular data analytics platform for CAVs," in *Distributed Computing Systems (ICDCS), 2018 IEEE 38th International Conference on*. IEEE, 2018.

[53] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–4.

[54] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: comparison of fpga, cpu, gpu, and asic," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 77–84.

[55] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[56] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 243–254.

[57] D. S. Modha, "Introducing a brain-inspired computer," *Published online at http://www. research. ibm. com/articles/brain-chip. shtml*, 2017.

[58] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[59] 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 the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[60] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?" in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[61] NVIDIA. (2019) Jetson AGX Xavier. [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-agx-xavie

[62] Q. Zhang, Z. Yu, W. Shi, and H. Zhong, "Demo abstract: Evaps: Edge video analysis for public safety," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2016, pp. 121–122.

[63] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Distributed collaborative execution on the edges and its application to amber alerts," *IEEE Internet of Things Journal*, 2018.

[64] L. Liu, X. Zhang, M. Qiao, and W. Shi, "SafeShareRide: Edge-based attack detection in ridesharing services," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 17–29.

[65] L. Liu, X. Zhang, Q. Zhang, W. Andrew, and W. Shi, "AutoVAPS: an IoT-Enabled Public Safety Service on Vehicles," in *4th International Science of Smart City Operations and Platforms Engineering (SCOPE)*. ACM, 2019.

[66] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[67] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[68] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[69] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3354–3361.

[70] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "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*. ACM, 2018, pp. 751–766.

[71] NVIDIA Corporation. (2019) NVIDIA DRIVE PX2: Scalable AI platform for Autonomous Driving. https://www.nvidia.com/en-us/self-driving-cars/drive-platform.

[72] Y. Wang, S. Liu, X. Wu, and W. Shi, "CAVBench: A benchmark suite for connected and autonomous vehicles," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 30–42.

[73] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[74] Baidu. (2019) Apollo Open Platform. http://apollo.auto/index.html.

[75] Amazon. (2019) Amazon echo. [Online]. Available: https://developer.amazon.com/echo

[76] Google. (2019) Google assistant. [Online]. Available: https://assistant.google.com/platforms/speakers/

[77] Y. Wang, X. Zhang, L. Chao, L. Wu, and X. Peng, "PowerAnalyzer: An energy-aware power monitor system aiming at energy-saving," in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2017, pp. 1–8.

[78] X. Zhang, Y. Wang, L. Chao, C. Li, L. Wu, X. Peng, and Z. Xu, "IEHouse: A non-intrusive household appliance state recognition system," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–8.

[79] R. Abdallah, L. Xu, and W. Shi, "Lessons and experiences of a DIY smart home," in *Proceedings of the Workshop on Smart Internet of Things*. ACM, 2017, p. 4.

[80] Y. Li and W. Gao, "MUVR: Supporting multi-user mobile virtual reality with resource constrained edge cloud," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 1–16.

[81] F. Martin-Sanchez and K. Verspoor, "Big data in medicine is driving big changes," *Yearbook of medical informatics*, vol. 9, no. 1, p. 14, 2014.

[82] J. Andreu-Perez, C. C. Poon, R. D. Merrifield, S. T. Wong, and G.-Z. Yang, "Big data for health," *IEEE J Biomed Health Inform*, vol. 19, no. 4, pp. 1193–1208, 2015.

[83] X. Wu, R. Dunne, Z. Yu, and W. Shi, "STREMS: a smart real-time solution toward enhancing ems prehospital quality," in *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2017 IEEE/ACM International Conference on*. IEEE, 2017, pp. 365–372.

[84] J. A. Rincon, Â. Costa, P. Novais, V. Julian, and C. Carrascosa, "Using non-invasive wearables for detecting emotions with intelligent agents," in *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16*. Springer, 2016, pp. 73–84.

[85] (2019) Hitech (health information technology for economic and clinical health) act of 2009. https://searchhealthit.techtarget.com/definition/hitech-act. [Online]. Available: https://searchhealthit.techtarget.com/definition/HITECH-Act