Machine Learning at the Network Edge: A Survey

M. G. SARWAR MURSHED, Clarkson University, U.S.A.
CHRISTOPHER MURPHY, SRC, Inc., U.S.A.
DAQING HOU, Clarkson University, U.S.A.
NAZAR KHAN, Department of Computer Science, University of the Punjab, Pakistan
GANESH ANANTHANARAYANAN, Microsoft, U.S.A.
FARAZ HUSSAIN, Clarkson University, U.S.A.

Resource-constrained IoT devices, such as sensors and actuators, have become ubiquitous in recent years. This has led to the generation of large quantities of data in real-time, which is an appealing target for AI systems. However, deploying machine learning models on such end-devices is nearly impossible. A typical solution involves offloading data to external computing systems (such as cloud servers) for further processing but this worsens latency, leads to increased communication costs, and adds to privacy concerns. To address this issue, efforts have been made to place additional computing devices at the edge of the network, i.e., close to the IoT devices where the data is generated. Deploying machine learning systems on such edge computing devices alleviates the above issues by allowing computations to be performed close to the data sources. This survey describes major research efforts where machine learning systems have been deployed at the edge of computer networks, focusing on the operational aspects including compression techniques, tools, frameworks, and hardware used in successful applications of intelligent edge systems.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Distributed artificial intelligence**; **Distributed computing methodologies**; • **Applied computing**; • **Computer systems organization** → **Embedded and cyber-physical systems**; **Real-time systems**;

Additional Key Words and Phrases: Edge intelligence, mobile edge computing, machine learning, resource-constrained, IoT, low-power, deep learning, embedded, distributed computing

ACM Reference format:
M. G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. 2021. Machine Learning at the Network Edge: A Survey. *ACM Comput. Surv.* 54, 8, Article 170 (October 2021), 37 pages.
https://doi.org/10.1145/3469029

Authors' addresses: M. G. S. Murshed, D. Hou, and F. Hussain, Department of Electrical and Computer Engineering, Clarkson University, Potsdam, NY 13699, USA; emails: {murshem, dhou, fhussain}@clarkson.edu; C. Murphy, SRC., Inc., 6225 Round Pond Road, North Syracuse, NY 13212, USA; email: cmurphy@srcinc.com; N. Khan, Department of Computer Science, University of the Punjab, Allama Iqbal Campus, Shahrah-e-Quaid-e-Azam, Lahore 54000, Punjab, Pakistan; email: nazarkhan@pucit.edu.pk; G. Ananthanarayanan, Microsoft, One Microsoft Way, Redmond, WA 98052, USA; email: ga@microsoft.com.

0360-0300/2021/10-ART170 $15.00
https://doi.org/10.1145/3469029

1 INTRODUCTION

Due to the explosive growth of wireless communication technology, the number of **Internet of Things (IoT)** devices have increased dramatically in recent years. It has been estimated that by 2020, more than 25 billion devices will have been connected to the Internet [94] and the potential economic impact of the IoT will be $3.9 trillion to $11.1 trillion annually by 2025 [95]. IoT devices typically have limited computing power and small memories. Examples of such resource-constrained IoT devices include sensors, microphones, smart fridges, and smart lights. IoT devices and sensors continuously generate large amounts of data, which is of critical importance to many modern technological applications such as autonomous vehicles.

One of the best ways to extract information and make decisions from this data is to feed those data to a **machine learning (ML)** system. Unfortunately, limitations in the computational capabilities of resource-constrained devices inhibit the deployment of ML algorithms on them. So, the data is offloaded to remote computational infrastructure, most commonly cloud servers, where computations are performed. Transferring raw data to cloud servers increases communication costs, causes delayed system response, and makes any private data vulnerable to compromise. To address these issues, it is natural to consider processing data closer to its sources and transmitting only the necessary data to remote servers for further processing [28].

Edge computing refers to computations being performed as close to data sources as possible, instead of on far-off, remote locations [20, 126]. This is achieved by adding edge computing devices close to the resource-constrained devices where data is generated. Figure 1 provides an overview of the edge computing architecture. In the rest of the article, the term "edge-device" is used to refer to either an end-device or an edge-server.

These edge-devices possess both computational and communication capabilities [130]. For example, an embedded device such as an Nvidia Jetson Nano could serve as an edge-device if it takes data from a camera on a robotic arm and performs data processing tasks that are used to determine the next movement of the arm. Since edge-devices have limited compute power, energy consumption is a critical factor [93] and computations too intense for edge-devices are sent over to more powerful remote servers.

As edge computing has emerged as an important paradigm for IoT-based systems [129], attempts are being made to use devices at the network edge to do as much computation as possible, instead of only using the cloud for processing data [159]. For example, Floyer studied data management and processing costs of a remote wind-farm using a cloud-only system versus a combined edge-cloud system [43]. The wind-farm consisted of several data producing sensors and devices such as video surveillance cameras, security sensors, access sensors for all employees, and sensors on wind-turbines. The edge-cloud system turned out to be 36% less expensive, costing only $28,927 as opposed to the cloud-only system that cost $80,531. Also, the volume of data required to be transferred was observed to be 96% less, compared to the cloud-only system. Similar examples include a video analytics system for autonomous drones where an edge computing system is used to save bandwidth [147], which reduces the need to transfer video data to the cloud by up to 10 times. In another study, Murshed et al. deployed a resource-aware deep learning system using edge devices where all image data is processed at the network edge [105]. By processing all data locally, this system not only preserves the privacy of the data but also saves bandwidth. Further use cases of edge computing in deep learning systems and their benefits are discussed in a dedicated section on applications (Section 3).

Performing computations at the network edge has several advantages:

- The volume of data needed to be transferred to a central computing location is reduced because some of it is processed by edge-devices.

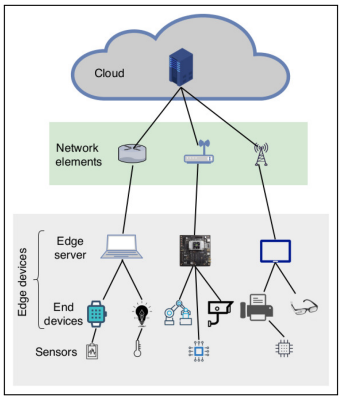

Fig. 1. A general overview of the edge computing architecture: An edge-device is either an end-device or an edge-server. An end-device is one with limited computing capability on which data can be produced and some computationally inexpensive (but useful) work can be performed. Computationally incapable sensors (last row) just produce data and are connected to either an end-device or an edge-server. The computational capability of an edge-server is also limited, but it has more computing resources than an end-device so it can be used for more complex tasks. Network elements are used to connect edge computing architecture to the cloud server.

- The physical proximity of edge-devices to the data sources makes it possible to achieve lower latency that improves real-time data processing performance.
- For the cases where data must be processed remotely, edge-devices can be used to discard **personally identifiable information (PII)** prior to data transfer, thus enhancing user privacy and security.
- Decentralization can make systems more robust by providing transient services during a network failure or cyber attack.

Major technology firms, the defense industry, and the open source community have all been at the forefront in investments in edge technology [67]:

- Researchers have developed a new architecture called Agile Condor, which uses ML algorithms to perform real-time computer vision tasks (e.g., video, image processing, and pattern recognition) [65]. With the processing capabilities of 7.5 teraflops, this architecture can run the latest deep learning-based object detection architectures such as Faster RCNN, YOLO in real-time [14]. Agile Condor can be used for **automatic target recognition (ATR)** at the network edge, near the data sources. A successful demonstration to identify targets of interest has been shown in 2018, where Agile Condor successfully identified a convoy in the real world. However, further analysis is required to validate and verify results against ground truth datasets.
- Microsoft recently introduced HoloLens 2,[1] a holographic computer. The HoloLens is built onto a headset for an augmented reality experience. It is a versatile and powerful edge-device that can work offline and also connect to the cloud. Microsoft aims to design standard computing, data analysis, medical imaging, and gaming-at-the-edge tools using the HoloLens.

[1] https://www.microsoft.com/en-us/hololens/.

- The Linux Foundation recently launched the LF Edge[2] project to facilitate applications at the edge and establish a common open source framework that is independent of the operating systems and hardware. EdgeX Foundry[3] is another Linux Foundation project that involves developing a framework for industrial IoT edge applications [44].
- GE,[4] IBM [133], Cisco [113], VMWare, and Dell[5] have all committed to investing in edge computing; VMware is developing a framework for boosting enterprise IoT efforts at the edge [13].

In the past few years, edge computing is being increasingly used for the *deployment of ML -based intelligent systems in resource-constrained environments* [25, 145, 150, 165, 166], which is the motivation for this survey. A note on terminology: Fog computing is a related term that describes an architecture where the "cloud is extended" to be closer to the IoT end-devices, thereby improving latency and security by performing computations near the network edge [64]. The main difference between fog and edge computing pertains to where the data is processed: In edge computing, data is processed either directly on the devices to which the sensors are attached or on gateway devices physically very close to the sensors; in the fog model, data is processed further away from the edge, on devices connected using a **local area network (LAN)** [66].

The use of ML models in edge environments creates a distributed intelligence architecture. In this respect, this survey describes how the following problems are being addressed by the research community:

- What are the practical applications for edge intelligence?
- How have existing ML algorithms been adapted for deployment on the edge?
- What is the state-of-the-art in distributed training (and inference)?
- What are the common hardware devices used to enable edge intelligence?
- What is the nature of the emerging software ecosystem that supports this new end-edge-cloud architecture for real-time intelligent systems?

As shown in Table 1, unlike ours, prior surveys about ML using edge computing (except Reference [117]) do not cover classical ML techniques like random forests, **Support-vector machine (SVM)**, and so on, which have been used very frequently in edge computing-based AI applications. Instead, existing surveys have focused on **deep learning (DL)** and its applications, as shown in the table. Also note that Park et al. [117] focused on wireless communication and did not discuss hardware/software platforms for edge ML. To summarize, the contributions of this article include:

- A survey of both classical machine learning and deep learning from a resource-constrained edge computing perspective,
- Common software/hardware platforms used in DL and edge computing architectures, and
- A detailed discussion of a wide range of applications pertaining to ML and edge computing from an operational perspective.

1.1 Survey Structure and Methodology

This survey focuses on machine learning systems deployed on edge-devices and also covers efforts made to train ML models on edge-devices. As shown in Figure 2, we begin by summarizing

[2] https://www.lfedge.org/.
[3] https://www.edgexfoundry.org/.
[4] https://www.ge.com/digital/iiot-platform/predix-edge.
[5] https://www.dellemc.com/en-us/service-providers/edge-computing.htm.

Table 1. A Summary of Related Surveys

Paper	Summary	Scope				
		Traditional ML	DL	Appl-ications	Software	Hardware
Park et al. [117]	Summarized the techniques that help enable ML at network edge	✓	✓	✓	✗	✗
Zhou et al. [165]	Reviewed the techniques/frameworks for training and inference of DL on edge devices	✗	✓	✓	✓	✗
Chen and Ran [25]	Conducted a survey on techniques that help to speed up DL training and inference on edge devices	✗	✓	✓	✗	✗
Lim et al. [85]	Provided a comprehensive analysis of federated learning and how this technique helps deploy DL on edge settings	✗	✓	✓	✓	✗
Wang et al. [149]	Reviewed the development of DL through edge computing architecture from a network & communication perspective	✗	✓	✓	✓	✓
This survey	Analyzes ML techniques (both deep learning & other classical methods) for resource-constrained edge settings	✓	✓	✓	✓	✓

the ML algorithms that have been used in edge computing (Section 2), discuss ML systems that exploit edge-devices (Section 3), and then briefly describe the frameworks, software, and hardware used to build and deploy intelligent systems at the edge (Section 4). The challenges hindering more widespread adoption of mobile edge computing and possible directions of future research (Section 5) are also discussed.

1.2 Paper Collection Methodology

We conducted exact keywords searches on Google Scholar, Microsoft Academic, DBLP, IEEE, ACM digital library, and arXiv to collect papers related to ML and edge computing, resulting in 124 papers. The following search terms were used:

- (machine | deep) learning + edge computing,
- (machine | deep) learning + (resource-constrained devices | IoT),
- modified learning algorithms + edge computing,
- (SVM | k-means | decision trees | convolutional neural networks | recurrent neural networks) + edge computing,
- (compressed | sparse | acceleration | deep learning | machine learning) + edge computing,
- resource-efficient deep neural networks + edge computing.

The following questions were used to determine whether to include a paper in the survey:

- Was edge computing used to improve ML efficiency in a networked ML system?
- Were ML algorithms specialized for resource-constrained devices designed or applied in a new setting?
- Were new results describing the deployment of ML systems on the edge reported?

If any of the questions above could be answered in the affirmative, then the paper was included. After careful analysis regarding the three questions mentioned above, we considered 88 out of the 124 papers collected for this survey. Papers published before May 2020 were considered for this survey.

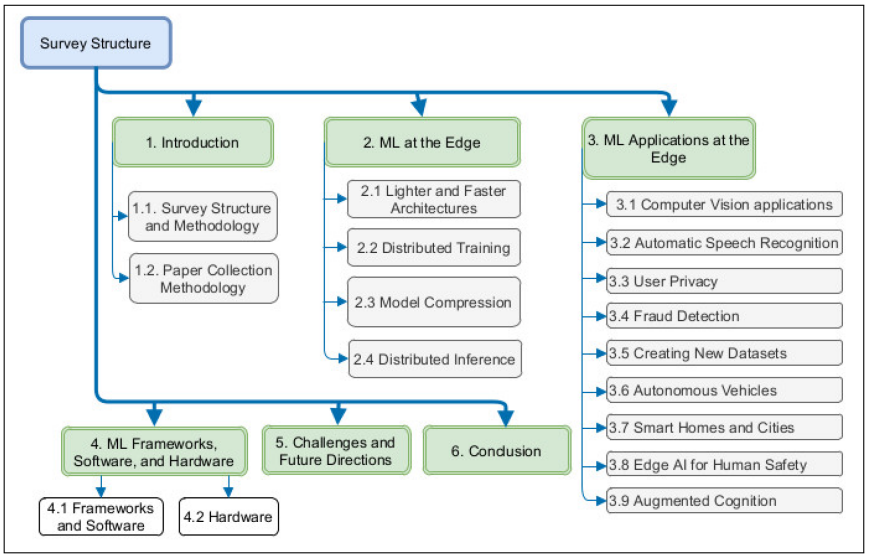

Fig. 2. An overview of the structure of the survey.

2 MACHINE LEARNING AT THE EDGE: TRAINING, COMPRESSION, AND DEPLOYMENT

Modern real-world applications, such as autonomous cars and mobile robots, increasingly require fast and accurate automated decision-making capabilities. Large computing devices can provide the needed real-time computational power, but in many situations, it is nearly impossible to deploy them without significantly affecting performance, e.g., on a robot. Instead, ML models must be adapted in a way that makes them suitable for deployment on (small) edge-devices that have limited computational power and storage capacity. This section describes classical ML and deep learning algorithms that have been used in resource-constrained settings.

With the increasing amount of information being generated by various kinds of devices placed at the network edge, the demand for ML models that can be deployed on such resource-constrained devices have also increased. Training machine learning models, especially deep learning models, typically requires a lot of computational power and a large number of training examples. Low-power IoT devices, such as on-board cameras, are continuous sources of data but their limited storage and compute capabilities make them unsuitable for training and inference of DL models. In systems with an end-edge-cloud hierarchy, centralized training is a problem due to limited bandwidth available. To address all these issues, edge-servers are increasingly being placed near these IoT end-devices and used for deploying DL models that operate on IoT-generated data. Efforts have been made for distributed training by using the edge-devices only (Section 2.2).

Due to the limited memory and compute power of edge-devices, it is important to tailor the model so it can fit and execute efficiently on those devices. Therefore, techniques have also been developed for compression ML models to make the models lighter and faster and to aid their deployment on the edge (Section 2.3.1). Table 2 presents a summary of the major contribution in ML algorithms at the network edge.

2.1 Lighter and Faster Architectures

To enable the deployment of deep learning models on resource-constrained edge-devices, the following goals need to be met while ensuring minimal loss in accuracy:

Table 2. Major Contributions in Machine Learning at the Edge

Challenges in Edge DL	Paper	Key contribution
Lighter and Faster Architectures (Section 2.1)	Zhang et al. [163]	ShuffleNet: Efficient CNN models for mobile and embedded devices
	Howard et al. [59]	MobileNets: Small, efficient, low-power, low-computation, and low-latency CNN models
	Tan and Le [135]	Efficient scaling of CNN architecture while retaining accuracy
	Tang et al. [137]	Addressed overparameterization during training to improve model quality
	Kusupati et al. [75]	Addressed unstable training & inefficient prediction while keeping maintaining accuracy and small model size
Distributed Training and High Network Communication Cost (Section 2.2)	Wang et al. [148]	Distributed training for gradient-descent-based algorithms
	Nishio and Yonetani [109]	FedCS: Faster learning federated learning for model training
	Lui et al. [89]	Intermediate edge aggregation to reduce communication cost during federated learning
	Wang et al. [92]	Local & global update comparison technique in federate learning to reduce communication cost
	Tao and Li [138]	Developed eSGD that is capable of reducing the gradient size of a CNN model by up to 90%
	Lin et al. [86]	Deep gradient compression to reduce gradient up to 600 times maintaining accuracy
Weight Quantization and Model Compression (Section 2.3)	Gupta et al. [50]	ProtoNN, an algorithmic technique for classification on resource-constrained devices.
	Kumar et al. [73]	Bonsai, a tree-based algorithm for ML prediction on resource-constrained IoT devices.
	Pradeep et al. [120]	Quantization for CNN deployment on embedded FPGAs
	Gupta et al. [51]	Reduced memory footprint using 16-bit fixed-point for weights
	Ogden and Guo [111]	Used 8-bit integers to represent CNN weights and reduced model size by 75%
	Iandola et al. [63]	SqueezeNet: An architecture with 50 times fewer parameters than AlexNet
Distributed Deep Learning Inference (Section 2.4)	Ogden and Guo [111]	Developed a distributed platform for mobile deep inference
	Teerapittayanon et al. [141]	Mapped DNNs across distributed computing hierarchies for faster inference
	Mao et al. [96]	MoDNN: A distributed mobile computing system for deploying DNNs
	Hadidi et al. [52]	Partitioning of DNN computations across devices and for reduced latency
	Hadidi et al. [53]	Dynamic distribution of DNN computations across devices
	Huai et al. [61]	Faster DNN inference by partitioning the model across multiple devices
	Stahl et al. [134]	Layer fusion scheme to reduce communication time during DNN inference

- The model size needs to be kept small by using as few trainable parameters as possible, and
- The inference time and energy need to be kept low by minimizing the number of computations.

2.1.1 Convolutional Neural Networks. It is challenging to deploy CNNs on edge-devices, owing to their limited memory and compute power. *Depthwise separable convolutions* provide a

lightweight CNN architecture (see Reference [59, Section 3.1]) by replacing the sum of multiple convolutional operations by a weighted sum of results obtained from only one convolutional operation. A standard CNN model uses each convolutional layer to generate a new set of outputs by filtering and summing the input channels. In contrast, depthwise separable convolutions divide each convolutional layer into two separate layers, which serve the same purpose as a single convolutional layer while also greatly reducing the model size and computational cost during inference. For a convolutional filter of size $K \times K$ operating on an input volume of M channels and producing an output of N channels, a depthwise separable convolution results in the computational cost being reduced by a factor of $1/N + 1/K^2$ and the number of parameters being reduced by a factor of $1/N + (M/(K^2 * M + 1))$, compared to the standard convolution. For example, if $K = 5$, $M = 64$, and $N = 128$, then depthwise separable convolutions will result in computations as well as the number of parameters being reduced by approximately 21 times.

Depthwise separable convolutions were exploited in MobileNets [59] to reduce the number of parameters from 29.3 million to 4.2 million and the number of computations by a factor of 8 while containing the accuracy loss to about only 1%. ShuffleNet [163] is another architecture that uses two other types of operations (viz., group convolutions and channel shuffling) along with depthwise separable convolutions to produce models that are lighter and faster than MobileNet while also being more accurate.

EfficientNet provides a method for systematically scaling up CNN models in terms of network depth, width, and input image resolution [135]. On ImageNet, it produced models yielding state-of-the-art accuracy using 8 times lesser memory and having 6 times faster inference. It also achieved state-of-the-art transfer learning performance using an order-of-magnitude fewer parameters. However, the inference speed of EfficientNet on resource-constrained edge-devices does not scale well. This is addressed by EfficientNetLite,[6] a suite of EfficientNet models tailored for edge-devices. Users can choose from a range of small, low latency models to relatively larger, high accuracy models. For ImageNet, even their largest model performed real-time image classification with state-of-the-art on a smart-phone CPU.

Other techniques for model size reduction include knowledge distillation, which refers to training a small model to mimic the outputs of a larger, more complex trained model [128, 137]; and low-rank factorization, which allows the decomposition of convolutional operations to reduce parameters and speed up computations during CNN operations [29].

2.1.2 Recurrent Neural Networks. **Recurrent Neural Networks (RNNs)** are powerful neural networks used for processing sequential data, but suffer from unstable training and inefficient prediction. Techniques to address these issues, such as unitary RNNs and gated RNNs, increase the model size as they add extra parameters. FastRNN and FastGRNN [75] are new architectures that provide stable training and good accuracy while keeping the model size small. In FastRNN, an additional residual connection is used that has only two additional parameters to stabilize the training by generating well-conditioned gradients.

Gated RNNs are typically more expressive than ungated RNNs but require additional parameters. FastGRNN is a gated version of FastRNN that does not increase the number of parameters. Model size was reduced by quantizing weights to be integer valued only. Prediction time was reduced by using piecewise-linear approximations to ensure that all computations use integer arithmetic only.

FastGRNN has 2–4 times fewer parameters than other leading gated RNN models such as LSTM, GRU, but provides the same or sometimes better accuracy than gated RNN models [75]. It is possible to fit FastGRNN in 1–6 kilobytes, which makes this algorithm suitable for IoT devices,

[6] https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite.

such as Arduino Uno. Since competing RNN models cannot fit on an Arduino Uno, the authors compared FastRNN and FastGRNN with other models on an Arduino MKR1000 and reported 18–42 times faster prediction. FastRNN and FastGRNN are open-source [74] and have been benchmarked on a number of problems including utterance detection, human activity recognition, and language modeling.

2.2 Distributed Training

Distributed training of DL models on multiple edge-devices and/or the cloud has two significant benefits:

- Data acquired from end-devices can be used for training/re-training at the edge-servers, thereby reducing the communication overhead of transmitting to and training on a central cloud server.
- It also leads to greater privacy, since no central location has access to data produced by end-devices.

In the next two sections, collaborative and privacy-aware learning on the end-edge-cloud architecture are discussed.

2.2.1 Distributed Training for Algorithms Based on Gradient Descent. Wang et al. introduced a technique to *train ML models at the edge without the help of external computation systems* such as cloud servers [148]. They focused on algorithms that use gradient-based approaches for training including SVMs, K-means, linear regression and **convolutional neural networks (CNNs)**. Their technique minimizes the loss function of a learning model by using the computational power of edge-devices only.

Local gradient descent is performed on multiple edge-devices based on data received at each device (i.e., on local datasets). This produces locally updated copies of the ML model. These copies are sent to another edge-device, known as the aggregator, that computes a weighted average of all the locally updated models. This averaged model is sent back to all the edge-devices so each device now carries a model averaged over training data from all edge-devices. The next iteration performs local updates once again and the process repeats until resource consumption reaches a prespecified budget.

To demonstrate the effectiveness of this technique, three Raspberry Pi devices and a laptop computer were used as edge-devices. They were connected via WiFi to an additional laptop computer that was used as the aggregator. Three datasets (viz., MNIST, Facebook metrics, and User Knowledge Modeling) were used to evaluate the ML models. The technique's performance was close to the optimum for training different ML models on different datasets.

2.2.2 Federated Learning. **Federated Learning (FL)** is a family of distributed ML methods that involve collaborative training of shared prediction DNN models on devices such as mobile phones [101]. All training data is kept on the end-devices and model training occurs in powerful local or cloud computing infrastructure. In general, there are two steps in the FL training process, viz., (*i*) local training and (*ii*) global aggregation. In local training, the end-device downloads the model from a central cloud server, computes an updated model using that local data to improve model performance. After that, an encrypted communication service is used to send a summary of all updates made by the end-device to the server. The server aggregates these updated models (typically by averaging) to construct an improved global model, as illustrated in Figure 3. This decentralized ML approach ensures the maximum use of available end-devices and does not share any data among end-devices, which helps to enhance the security and privacy of the local data.

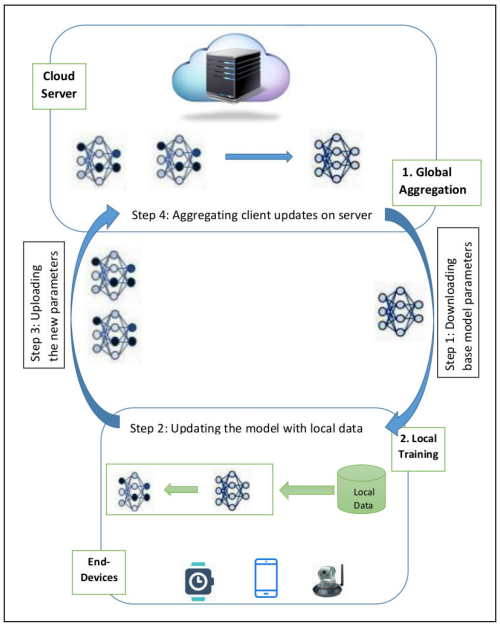

Fig. 3. Federated learning allows training on end-devices where the data is produced. First, end-devices download parameters of a trainable ML model from the cloud server. Then, those devices update the model locally with their own data. After that, all end-devices upload the updated model parameters. Finally, the cloud server aggregates multiple client updates to improve the model.

However, federated learning faces challenges that include communication overhead, interoperability of heterogeneous devices, and resource allocation [83, 85].

Federated learning with heterogeneous edge clients can lead to slow learning when some edge clients have very low-resources. To address this problem, Nishio and Yonetani introduce FedCS, which performs federated learning with an intelligent client selection mechanism to ensure faster learning of high-performing ML models [109].

Exchanging model parameters and other data between edge-devices and cloud servers is mandatory for training an edge-cloud-based DL model. However, as the size of the training model increases, more data needs to be exchanged between edge-devices and cloud servers. The high network communication cost is a bottleneck in training a model. Various techniques have been proposed to reduce communication costs during training:

- Lui et al. introduced intermediate edge aggregation before FL server aggregation [89].
- Wang et al. compare each client's local model update with the global update of the model at a central server. If the local update differs significantly from the global update, then it is not communicated to the central server, since it will become less relevant when updates are aggregated [92].
- Tao and Li introduced a new method called **Edge Stochastic Gradient Descent (eSGD)** [138] that is capable of reducing the gradient size of a CNN model by up to 90% by communicating only the most important gradients. On MNIST, they obtained an accuracy of 91.22% with a 50% gradient drop.
- Lin et al. studied the effect of gradient exchange in **distributed stochastic gradient descent (DSGD)** and found 99.9% of gradient exchanges to be redundant [86]. This observation

inspired them to propose *deep gradient compression*, which compresses the gradient from 270 to 600 times without losing much accuracy. They applied momentum correction, local gradient clipping, momentum factor masking, and warm-up training methods to reduce the size of gradient updates for training ResNet-50 from 97 MB to 0.35 MB, and for training DeepSpeech [55] from 488 MB to 0.74 MB.

2.2.3 Privacy in Learning. Privacy of sensitive training data becomes an issue when training is performed in a distributed fashion. Techniques derived from cryptography can be applied for hiding updates of local devices from a central aggregation server [18] and also for hiding the aggregated update from the local devices [47]. While ensuring such local and global privacy mechanisms, it is important to (i) contain any drop in accuracy (ii) ensure low overheads of computation and communication and (iii) introduce robustness to communication errors and delays.

Mao et al. presented a privacy-aware DNN training architecture that uses differential privacy to protect user data during the training [97]. This deep learning scheme is capable of training a model using multiple mobile devices and the cloud server collaboratively, with minimal additional cost. First, this algorithm selects one convolutional layer to partition the neural network into two parts. One part runs on edge-devices, while another part runs on the server (AWS cloud server). The first part takes raw data with sensitive user information as input and uses a differentially private activation algorithm to generate the volume of activations as output. These output activations contain Gaussian noise, which prevents external cloud servers from reproducing original input information using reversing activations. These noisy output activations are then transmitted to cloud servers for further processes. The servers take output activations as input and run the subsequent training process. This model was evaluated using a Nexus 6P phone and AWS-based servers. The authors report that they have achieved good accuracy by deploying this model on **Labeled Faces in the Wild dataset (LFW)**.

To avoid centralized aggregation of potentially sensitive information, blockchains have been explored as an alternative [72, 121]. However, the compute power and energy requirements of mining on edge-devices remains an important challenge [149].

2.3 Model Compression

In this section, we discuss compressed ML techniques designed for memory-scarce IoT devices. In the literature, we found some compression techniques for classical ML used in resource-constrained settings, which may also open up future research in model compression area. A popular technique for classical ML compression is to project data to a lower dimensional space and learn in the projected space. However, learning in a low-dimensional data space leads to poor accuracy. Techniques for regaining accuracy are covered during the discussion on model compression (Section 2.3.1).

To run on low-memory devices, deep learning models need to be reduced in size while containing the loss in accuracy. There are two broad approaches for doing this (*a*) *quantization*, which reduces the precision with which parameter values are stored, which in turn lowers the memory footprint and therefore potentially makes computations faster, and, (*b*) *model pruning*, which reduces the number of model parameters and therefore improves storage and compute time. Note that both methods can be applied on a given model one after the other. This section includes a discussion of commonly used DL model quantization and pruning techniques (Section 2.3.2). The surveys by Cheng et al. [29] and Choudhary et al. [31] provide detailed descriptions of model compression techniques.

2.3.1 Overcoming Storage Limitations with Compression. ProtoNN is a new technique, designed by Gupta et al., for *training an ML model on an edge-device* that can perform real-time prediction

tasks accurately [50]. It is a **k-nearest neighbor (k-NN)**-based algorithm with low storage requirements. The following issues arise when trying to implement a standard k-NN on a resource-constrained edge-device:

- It requires the entire training dataset to be stored, which is difficult on resource-constrained devices;
- The distance of a given sample from each training example needs to be computed, which inhibits real-time prediction on resource-constrained devices due to their limited computational power;
- The results are sensitive to the choice of the distance metric.

To address these issues, ProtoNN projects data to a lower dimensional space using a sparse-projection matrix and selects only a few prototype vectors/samples from this low-dimensional data. The algorithm learns the projection matrix and the prototypes and their labels in a joint optimization step, which helps to compensate for the reduction in accuracy caused due to the projection into lower dimensional space and selecting only a few representative prototypes.

For settings needing extremely small models (<2 kB), ProtoNN was shown to outperform the baseline compressed models such as GBDT, RBF-SVM, 1-hidden layer NN, and so on, in character recognition dataset, while in settings allowing 16–32 kB memory, it matched the performance of the baseline compressed models. This was true for binary, multiclass as well as multilabel datasets. Compared to the best uncompressed models, ProtoNN was only 1%–2% less accurate while consuming 1–2 orders of magnitude less memory (in multiclass and multilabel settings). On severely resource-constrained IoT devices, energy usage and time to make predictions are important considerations [157]. On an Arduino Uno (2 kB RAM), ProtoNN was shown to be more efficient in both aspects compared with existing methods. ProtoNN is available as part of Microsoft's EdgeML library [74] and Embedded Learning Library [102].

Tree-based ML algorithms are commonly used for classification, regression, and ranking problems. Even though the time-complexity of tree-based algorithms is logarithmic with respect to the size of the training data, their space complexity is linear, so they are not easily amenable to deployment on resource-constrained devices. Aggressively pruning or learning shallow trees are ways to shrink the model but lead to poor prediction results. *Bonsai* is a novel tree-based algorithm developed by Kumar et al. that significantly outperforms state-of-the-art techniques in terms of model size, accuracy, speed, and energy consumption [73].

To reduce the model size, Bonsai learns a single decision tree that is shallow as well as sparse. These three decisions should naturally decrease overall accuracy. Bonsai manages to retain high accuracy using two strategies:

- It performs a low-dimensional linear projection of the input. The linear projection can be performed in a streaming fashion, i.e., without storing the whole input in RAM. This is important for solving large input size problems in low-memory environments. This low-dimensional projection is learned jointly with all the other parameters of the model. Such joint learning leads to higher accuracy.
- By making a shallow decision tree more powerful (non-linear) by: (*i*) using non-axis-aligned branching, (*ii*) enabling all internal nodes to become classifiers as well, and (*iii*) making the final classification result in a sum of all results along the path traversed by the input vector.

Bonsai has been tested with a number of binary and multi-class datasets. When deployed on an Arduino Uno, Bonsai required only 70 bytes for a binary classification model and 500 bytes a 62-class classification model. Its prediction accuracy was up to 30% higher than other resource-constrained models and even comparable with unconstrained models. Prediction times and energy

usage were also measured and shown to be better for Bonsai models. Bonsai is available as part of the EdgeML library [74].

2.3.2 Model Quantization and Pruning. SqueezeNet [63] is a parameter-efficient neural network used in resource-constrained settings. This small CNN-like architecture has 50 times fewer parameters than AlexNet while preserving AlexNet-level accuracy on the ImageNet dataset. It reduced model size by (i) replacing 3×3 convolutions with 1×1 convolutions, (ii) using weighted sums of input channels via 1×1 convolutions to reduce the number of input channels, (iii) converting input channels to 3×3 filters only, and (iv) using delayed down-sampling to achieve higher classification accuracy. By using Deep Compression (weight pruning + quantization + Huffman encoding) [54], SqueezeNet was further compressed to 0.47 MB, which is 510 times smaller than 32-bit AlexNet. These techniques decrease the number of parameters at the cost of accuracy. To compensate, the authors down-sample later in the network to have larger activation maps that lead to higher accuracy. The authors report that SqueezeNet matches or exceeds the top-1 and top-5 accuracy of AlexNet while using a 50 times smaller model.

Pradeep et al. deployed a CNN model on an embedded FPGA platform [120]. They use low bit floating-point representation[7] to reduce the computational resources required for running a CNN model on FPGA. Their architecture was tested with SqueezeNet on the ImageNet dataset. The authors reported having achieved 51% top-1 accuracy on a DE-10 board at 100 MHz that only consumed 2W power. Gupta et al. used 16-bit fixed-point representation in stochastic rounding-based CNN training [51]. This lower bit representation significantly reduced memory usage with little loss in classification accuracy. Ogden and Guo use grouped weight rounding and 8-bit quantization to reduce the model size by 75% and increase inference speed while containing the accuracy drop to approximately 6% [111].

A few sensitive regions of a feature map have a greater impact on deep learning inference compared to other regions [131]. Conventional quantization techniques are applied to entire network layers or on kernel weights without considering the dynamic properties of the feature map. Such quantization sometimes hurts the overall performance of a deep learning model despite large reductions of energy usage. Song et al. developed *DRQ*, a dynamic region-based quantization that can detect the sensitive regions in the feature map dynamically [131]. To preserve the accuracy, DRQ implies high-fidelity quantization on the sensitive regions and low-fidelity quantization on the insensitive regions. This system achieved an improvement of 3% in the prediction accuracy improvement compared to the state-of-the-art mixed-precision quantization accelerator *OLAccel* [116].

The **Sparse CNN (SCNN)** accelerator compresses convolutional neural networks by detecting and pruning the zero-valued activations from the ReLU operator and the zero-valued weights of the network during training [114]. However, traditional weight pruning techniques sometimes incur additional storage overhead and reduce performance accuracy [158]. *Scalpel*, a customizing DNN pruning technique, overcomes these weaknesses by introducing two new methods: SIMD-aware weight pruning and node pruning. These pruning techniques were tested on microcontrollers and CPUs and achieved a reduction in model size of 88% and 82% (respectively) and a geometric mean performance speed-up of 3.54 times and 2.61 times (respectively) [158].

Eager Pruning is a model pruning technique that observes the rank of the significant weights of the DNN and prunes insignificant weights during training [160]. This method was shown to reduce the training computation of a network by up to 40% while still maintaining the original accuracy. Eager pruning applied on eight different deep learning architectures achieved an average speedup of 1.91 times over a state-of-the-art hardware accelerator.

[7] 8-bit for storage and 12-bit during computations.

Yang et al. developed a network compression method for the deployment of DNNs on memristor devices such as **resistive random access memory (ReRAM)** [155]. They built a *Sparse ReRAM Engine* that exploits activation and weight sparsity and prunes zero weights and zero activations. When classifying objects on the ImageNet dataset using the VGG-16 CNN model, this pruning helps to compress the model and deploy it on ReRAMs achieving a speedup up to 42.3 times and energy savings of up to 95.4% over the state-of-the-art that does not exploit sparsity.

2.4 Distributed Deep Learning Inference

Distributed deep neural network architectures allow the distribution of **deep neural networks (DNNs)** on the edge-cloud infrastructure to facilitate local and fast inference on edge-devices wherever possible. The success of a distributed neural network model depends on keeping inter-device communication costs as low as possible. Several attempts have been made to split and distribute a model between edge-devices, resulting in faster inference. DNN inference can be distributed in two ways: (*a*) vertically, along the end-edge-cloud architecture, and (*b*) horizontally, along multiple devices at the same level in the architecture.

2.4.1 Vertically-distributed Inference. We now describe two orthogonal approaches for exploiting the vertical direction of the end-edge-cloud architecture. In the first approach, **Mobile Deep Inference (MODI)**, the dynamic status of the architecture is considered to make optimal decisions about which model to run and where to run it. In the second approach, **Early Exit of Inference (EEoI)**, resources along the vertical direction are exploited to exit as early as possible to make optimal inferences while balancing the tradeoff between accuracy on the one hand and computational effort and latency on the other.

Mobile Deep Inference (MODI) Platform. Due to variable network conditions, inference on the end-edge-cloud architecture is a dynamic problem. Static approaches such as on-device inference only or remote inference only are both sub-optimal. Ogden and Guo [111] propose a dynamic solution to this problem by designing a distributed platform for mobile deep inference. They propose to store multiple DL models (compressed and uncompressed) in a centralized model manager and dynamically decide which model to run on which device based on the inference environment (memory, bandwidth, and power). If the inference environment is resource-constrained, then one of the compressed models is used, otherwise an uncompressed model gets deployed, providing higher accuracy. They show that on-device inference can provide up to 2.4 times speedup. They also propose offloading remote inference to edge-servers when networks are slow. MODI runs jointly on end-devices and servers to provide the most suitable model for mobile devices based on device resources and installed applications.

Early exit of inference (EEoI). Teerapittayanon et al. introduced distributed **deep neural networks (DDNN**[8]**)**, where sections of a deep neural network are mapped across distributed computing hierarchies [141]. Figure 4 shows the general structure of a distributed deep neural network. Their method leverages cloud servers, resource-constrained edge-server, and end-devices such as surveillance cameras for inference. In such as model, a big portion of the raw data generated by sensors is processed on edge-devices and then sent to the cloud for further processing. This reduces their data communication cost. The DNN receives input from end-devices and produces final output in the cloud. That is, the DNN is distributed over the entire end-edge-cloud architecture.

[8]DDNN is open-source: https://github.com/kunglab/ddnn.

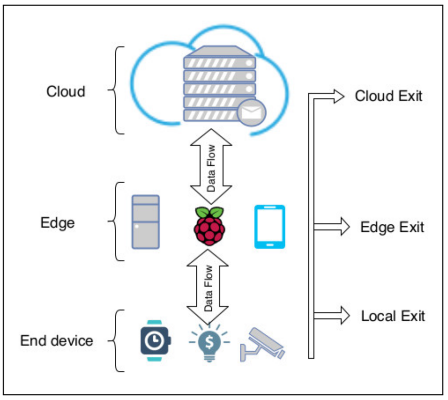

Fig. 4. A high-level view of the distributed deep neural network (DDNN) approach designed by Teerapittayanon et al. [141]. End-devices send summary information to a local aggregator that serves as a DNN layer. The DDNN is jointly trained with all resource-constrained end-devices and exit points, so the network is capable of automatically collecting and combining input data from different end-devices. If the information collected from a particular end-device is sufficient to classify a sample, then classification is done locally (i.e., on the end-device itself). If that is not possible, then the information is sent to the edge-servers for further processing. If edge-servers can complete the classification, then they send the result back to the end-devices. Otherwise, edge-servers send the information to the cloud, where the classification process is completed, and the results arereturned to the end-devices.

Additional classifiers are added to this network at the levels of end-devices and edge-servers, which allows a strategy for early exit at these classifiers to be used. This allows for fast, early inference if a sample can be classified accurately by the lower classifiers alone. Otherwise, the features extracted so far are propagated further up the network. The whole DNN is trained jointly in an end-to-end fashion so lower-level classifiers learn to produce features that are also useful for subsequent classifiers. The joint training takes place on a single, powerful server or on the cloud. They also use binary neural networks [62, 100] to reduce the model size and increase inference speed. Aggregation methods are used to fuse information from different end-devices. This method of aggregation is extended to the edge-servers as well. Using this technique reduced their data communication cost by a factor of more than 20 for a classification task on a multi-view multi-camera dataset.

2.4.2 Horizontally-distributed Inference. Mao et al. proposed a local **distributed mobile computing system (MoDNN)** to deploy DNNs on resource-constrained devices [96]. MoDNN uses a pre-trained DNN model and scans each layer of a DNN model to identify layer types. If a convolutional layer is detected, then the layer input is partitioned by a method called **Biased One-Dimensional Partition (BODP)**. BODP helps reduce computing cost by reducing the input size of convolutional layers. If a fully connected layer is detected, then the layer input is assigned to different work nodes (mobile devices) to achieve the minimum total execution time. They used a VGG-16 model, pretrained on ImageNet. MoDNN was implemented on the LG Nexus 5 (2 GB memory, 2.28 GHz processor, Android 4.4.2). They reported having successfully accelerated DNN computations by 2.17–4.28 times with 2 to 4 mobile devices.

Stahl et al. perform distributed execution of all layers of a DNN while jointly minimizing computation, memory usage, and communication overhead [134]. They report even memory distribution across six edge-devices with 100 Mbit connections running YOLOv2. They propose a communication-aware layer fusion scheme that reduces communication cost by 14.8% and speeds up inference by 1.15 times.

Distributed computations across edge-devices can lead to undesirable latency. Hadidi et al. perform partitioning of DNN computations across devices in a robust way using **coded distributed computing (CDC)** [52]. They also demonstrate low-latency recovery from errors due to delays in distributed computations. In other work, Hadidi et al. present a method for dynamic distribution of DNN computations across available devices to achieve real-time performance [53].

Huai et al. have developed a method in which DNN inference required by a robot leverages idle robots in its proximity by partitioning the model across the robots using latency prediction and optimal partition selection [61]. They report an average speedup of 6 times compared to local execution on the robot or remote execution on the cloud.

3 MACHINE LEARNING APPLICATIONS AT THE EDGE

The previous two sections covered techniques that have been developed for training and inference of ML models on the network edge. This section focuses on the actual *applications* of edge-based ML methods for deploying intelligent systems. Table 3 summarizes existing research for the development of intelligent systems using deep learning and edge computing.

3.1 Computer Vision Applications

3.1.1 Real-time Video Analytics. Real-time video analytics systems are an integral part of a wide range of applications, e.g., self-driving cars, traffic safety and planning, surveillance, and augmented reality [9]. Until recently, video analytics systems using ML algorithms could only process about 3 **frames per second (FPS)**, whereas most real-time video cameras stream data at 30 FPS [69]. Edge computing with IoT cameras has been used to address this problem and provide improved real-time video analytics services.

Ananthanarayanan et al. developed a video analytics system called Rocket [9] that produces high-accuracy outputs with low resource costs. Rocket collects video from different cameras and uses vision processing modules for decoding it. Each module uses predefined interfaces and application-level optimizations to process video data. A resource manager is used to execute data processing tasks on different resource-constrained edge-devices and cloud servers. A traffic analytics system based on the Rocket software stack has been deployed in Bellevue, WA, to track cars, pedestrians, and bikes. After processing the data in real-time, it raises an alert if anomalous traffic patterns are detected. Rocket has been shown to be effective in a variety of applications [10], which include (*a*) a smart crosswalk for pedestrians in a wheelchair, (*b*) a connected kitchen to pre-make certain food to reduce customer wait-times, (*c*) traffic dashboard for raising an alarm in abnormal traffic volumes, and (*d*) retail intelligence for product placement. Wang et al. introduced a bandwidth-efficient video analytics architecture based on edge computing that enables real-time video analytics on small autonomous drones [147]. DNNs on the drones select interesting frames to be sent to edge-devices. Periodically, the edge-devices train an SVM to improve the results on the drone DNNs. These SVMs are transmitted back to the drones where they are used to predict if the DNN output is correct. In this way, their models are being continuously learned. Ali et al. designed an edge-based video analytics system using deep learning to recognize an object in a large-scale IoT video stream [3]. There are four stages in this video analytics system: (i) frame loading/decoding and motion detection, (ii) preprocessing, (iii) object detection and decomposition, and (iv) object recognition. The first three actions are performed on the edge infrastructure and the fourth one in the cloud. To improve accuracy, this model uses a filter that finds important frames from the video stream and forwards them to the cloud for recognizing objects. Their edge-cloud-based model was 71% more efficient than the cloud-based model in terms of throughput on an object recognition task.

Table 3. Summary of Major Edge Machine Learning Applications

Application	System	Short Summary
Video Analytics	Rocket [9]	Real-time video analytics for traffic control, surveillance, and security
	JITL [147]	Real-time video analytics on small autonomous drones
	Edge Enhanced Deep Learning [3]	Recognize an object in a large-scale IoT video stream
	Traffic Estimation [71]	Real-time traffic estimation using vehicle as edge nodes
	Dietary Assessment System [88]	Real-time food recognition to assess weight-loss intervention
	Precog [37]	Reduces image recognition latency for mobile applications
	EdgeLens [143]	Real-time object detection
Speech Recognition	EdgeSpeechNets [87]	Human voice recognition
	Deep KWS [24]	Small-footprint keyword spotting technique
Privacy	Hybrid edge-cloud privacy system [112]	Protects user privacy when uploading data to the cloud
	Notify data collection [34]	Notifies users about what data is collected from the IoT device
Fraud Detection	Forgery detection [48]	Authenticates medical image data validity
Creating New Datasets	Eureka [41]	Labeling training datasets
Autonomous Vehicles	Cognitive internet [26]	Used to build vehicle networks
	DeepCrash [22]	Internet of Vehicles (IoV) system
	Pedestrian detection [106]	Pedestrian detection method for autonomous vehicles
	STTR [154]	Smart surveillance system for tracking vehicles
	Tesla auto-pilot [11]	Detects multiple kinds of objects on the road
Smart Homes/Cities	Fall detection [60]	A system that generates an alert message when an object falls
	City Data analytic [136]	Detects the threat by analyzing big data
	Energy management [23]	A framework for smart homes that improves the use of renewable energy
	Fault detection [115]	Detects faults in a smart factory
Human Safety	Person identification [104]	Identifies a person in crowded scenarios
	Threat detection [90]	Detects attacks and potentially harmful events in ride-sharing services
	MISS [35]	Processes sensitive data at the network edge and enhances security
Augmented Cognition	Augmenting cognition [127]	Augmenting human cognition to create adaptive human-machine collaboration

A CNN-based video analytics system to count vehicles on a road and estimate traffic conditions without the help of surveillance cameras was designed by Kar et al. [71]. They considered a vehicle as an edge-device and deployed their model on a dashboard camera on-board the vehicle to detect other vehicles on the road. They used 8,000 car images to train the model and then deployed the trained model to identify a moving vehicle and achieved an accuracy of 90%.

3.1.2 Image Recognition. Image recognition refers to the process of extracting meaningful information from a given image, e.g., to identify objects in that image. Deep learning techniques such as CNNs can be used to detect people, places, handwriting, and so on, in an image. The prevalence of IoT cameras and mobile devices has increased the importance of improved image recognition techniques, as they are increasingly being used in areas such as wildlife monitoring [39].

Until recently, data would almost always be transferred to the cloud where the images captured by IOT or mobile phones would be processed. Researchers have increasingly begun to use edge computing techniques to process images close to where they are captured. For example, there are currently more than 2.5 billion social media users in the world and millions of photographs and videos are posted daily on social media [132]. Mobile phones and other devices can capture high-resolution video, the uploading of which may require high bandwidth. By processing this data on the edge, the photos and videos are adjusted to a suitable resolution before being uploaded to the Internet [129]. Caffe2Go[9] is a lightweight framework that allows deploying DL systems on a mobile device and helps to reduce the size of the input layer of a DL model. Personally identifiable information in videos and images can be removed at the edge before they are uploaded to an external server, thereby enhancing user privacy. Liu et al. propose a real-time food recognition system [88] that assesses dietary intake by employing deep learning within the edge computing paradigm. By pre-processing images on edge-devices and classifying them using a CNN on the cloud, their system achieved highest accuracy, fastest results and minimum energy consumption among all techniques compared. The main function of the edge-device is to identify a blurry image taken by the user. After processing the food image on the edge-device, a clear image is sent to the cloud server for further processing.

Precog is a novel technique designed by Drolia et al. for prefetching and caching to reduce image recognition latency for mobile applications [37]. On a dataset of mobility traces of museum visitors augmented by paining recognition requests, Precog was shown to (*a*) reduce latency by 5 times, (*b*) reduce edge-server usage by 5 times, and (*c*) increase recall by 15%. Precog employs a method of predicting the next classification request and caching relevant parts of a trained classifier on end-devices. This reduces offloads to the cloud. Edge-servers estimate probabilities for the next request from connected end-devices. This estimation is done via a Markov model based on the requests that the edge-server has already received from the devices. Based on these predictions, Precog prefetches parts of trained classifiers and caches them on the end-devices. These pre-fetched parts serve as smaller models that speed-up inference on the end-devices and reduce network communication as well as cloud processing.

Tuli et al. introduced a deep learning-based real-time object detection system using IoT, fog, and cloud computing [143]. They used the YOLOv3 architecture [122] by training on the COCO dataset to evaluate their system. The authors have developed an open-source fog-cloud deployment system called EdgeLens[10] and demonstrated the capability of this system by deploying object detection YOLO software on multiple Raspberry Pi devices and cloud VMs.

3.2 Automatic Speech Recognition

There is immense community interest in developing an offline speech recognition system that supports a digital voice-assistant without the help of the cloud. Limited-vocabulary speech recognition, also known as keyword spotting, is one method to achieve offline speech recognition [151]. A typical keyword spotting system has two components: (*a*) a feature extractor to gather the

[9] https://code.fb.com/android/delivering-real-time-ai-in-the-palm-of-your-hand/.
[10] https://github.com/Cloudslab/EdgeLens.

necessary features from the human voice and (*b*) a neural network-based classifier that takes voice features as input and generates a probability for each keyword as output. DNN-based keyword spotting systems are not easily deployable on resource-constrained devices. EdgeSpeechNets is a highly efficient DNN for deploying DL models on mobile phones or other consumer devices for human voice recognition [87]. It achieved higher accuracy (approximately 97%) than state-of-the-art DNNs with a memory footprint of about 1 MB on the Google Speech Commands dataset. When deployed on the Motorola Moto E phone with a 1.4 GHz Cortex-A53 mobile processor as the edge-device, EdgeSpeechNets used 36 times fewer mathematical operations resulting in 10 times lower prediction latency, a 7.8 times smaller network size, and a 16 times smaller memory footprint than state-of-the-art DNNs. Chen et al. introduced a small-footprint keyword spotting technique based on DNNs called Deep KWS, which is suitable for mobile edge-devices [24]. Deep KWS has three components: a feature extractor, a deep neural network, and a posterior handling module. A feature vector extracted from audio input is fed to the DNN to generate frame-level posterior probabilities scores. Finally, the posterior handling module uses these scores to generate the final output score of every audio frame to recognize the audio. With respect to traditional HMM-based solutions, their model achieved a relative improvement of 45%.

3.3 User Privacy

The widespread use of IoT devices has boosted personal data production. However, inadequate security protections on these devices have simultaneously increased the potential for misuse of user data. Osia et al. designed a hybrid architecture that works with edge and cloud servers collaboratively to protect user privacy. All personal data is collected and processed on personal edge-devices to remove sensitive information [112]. Only data that is free of sensitive information is sent to the cloud server for further processing. They developed a privacy preserving, hybrid edge-cloud system that achieved 93% accuracy on a gender classification task. Das et al. have developed a distributed privacy infrastructure that can automatically discover nearby IoT devices and notify users about what data is collected about them and how it is used [34]. Users can then configure privacy preferences (e.g., opt-in/out of any service that uses sensitive data) that can then be deployed on the IoT devices.

3.4 Fraud Detection

With the increase in data being generated by IoT and smart devices, incidents of data fraud and theft are also increasing. Machine learning is being used to prevent data falsification and to authenticate data validity. Ghoneim et al. developed a new medical image forgery detection framework that can identify altered or corrupted medical images [48]. They used a multi-resolution regression filter on a noise map generated from a noisy medical image and then used SVM and other classifiers to identify corrupted images. The first part of their algorithm (which creates a noisy map from a medical image) is done on an edge computing device, and the rest is done on a cloud server. This distributed approach decreases the time for data processing as well as bandwidth consumption.

3.5 Creating New Datasets

Feng et al. designed an edge-based architecture that produces a labeled training dataset that can be used to train an ML model [41]. Based on the amount of training data collected at any stage, they train progressively more complex ML models at the edge to perform early discard of irrelevant data. This reduces the amount of labeling that a human has to perform by two orders of magnitude compared to a brute-force approach.

3.6 Autonomous Vehicles

An autonomous vehicle on average generates more than 50 GB of data every minute.[11] This data must be processed in real-time to generate driving decisions. The bandwidth of an autonomous vehicle is not large enough for transferring this enormous amount of data to remote servers. Therefore, edge computing is becoming an integral part of autonomous driving systems. A novel design of a cognitive internet of vehicles has been proposed by Chen et al. [26], which they discuss from three perspectives: intravehicle, intervehicle, and beyond-vehicle networks. Liang et al. use reinforcement learning for managing network resources by learning the dynamics of vehicular networks [84]. DeepCrash [22] is cloud-based deep learning **Internet of Vehicles (IoV)** system that can detect and report a collision event so timely emergency notifications can be generated. They report a 96% collision detection accuracy and almost seven seconds emergency notification latency. Hochstetler et al. have shown that it is possible to process real-time video and detect objects using deep learning on a Raspberry Pi combined with an Intel Movidius Neural Compute Stick [58]. They reported that their embedded system can independently process feeds from multiple sensors in an autonomous vehicle. Navarro et al. designed a pedestrian detection method for autonomous vehicles [106]. A LIDAR sensor gathers data to detect pedestrians and features are extracted from this data. These features include stereoscopic information, the movement of the object, and the appearance of a pedestrian (local features like Histogram of Oriented Gradients). Using the features obtained from raw LIDAR data, an n-dimensional feature vector is generated to represent an object on the road. This feature vector is used as input for the ML model to detect pedestrians. They use a Nuvo-1300S/DIO computer to run the ML model inside an autonomous vehicle and report 96.8% accuracy in identifying pedestrians. STTR is a smart surveillance system for tracking vehicles in real-time [154]. It processes camera streams at the edge and stores space-time trajectories of the vehicles (instead of raw video data), which reduces the size of the stored data. Vehicle information can be found by querying these space-time trajectories, which can be used to help identify suspicious vehicles after traffic accidents. Tesla has deployed auto-pilot and smart-summon technologies on its autonomous vehicles [11]. Since vehicles need to detect multiple kinds of objects on the road, they use shared backbone deep networks to reduce computation. Different object detectors are then trained on top of the backbone networks. To enable both training at scale as well as real-time inference, they use customized hardware that can execute 144 trillion int8 operations per second operating at less than 100 W.

3.7 Smart Homes and Cities

Homes equipped with intelligent systems built using numerous resource-constrained devices are increasingly being designed [56]. An important goal in the design of such smart homes is ensuring the safety of children and the elderly. A fall detection system that generates an alert message when an object falls has been developed by Hsu et al. [60]. Their approach has three steps, (*a*) a skeleton extraction performed followed by an ML prediction model to detect falls, (*b*) video and image compression, for which a Raspberry Pi is used, and (*c*) fall detection using ML on the cloud, following which users are notified in appropriate cases. Tang et al. designed a hierarchical distributed computing architecture for a smart city to analyze big data at the edge of the network, where millions of sensors are connected [136]. They used SVMs in a smart pipeline monitoring system to detect hazardous events to avoid potential damages. For example, if a house in the smart city experiences a leakage or a fire in the gas delivery system, then edge-devices will detect the threat and quickly

[11] https://datafloq.com/read/self-driving-cars-create-2-petabytes-data-annually/172.

shut down the gas supply without the need for any centralized data processing. An edge-based energy management framework for smart homes that improves the use of renewable energy to meet the requirements of IoT applications has been developed by Chang et al. [23]. Since sunlight is the main source of renewable energy, they used a weather prediction model to predict weather impacting solar energy generation. After obtaining forecast information, an energy scheduling module generates a cost-effective energy utilization schedule for the smart home. A Raspberry Pi was used to run their framework at the location of its users, helping to protect privacy-sensitive data. Park et al. developed an edge-based fault detection system for smart factory settings [115]. An LSTM recurrent neural network was deployed on an edge-device (with 1 GB memory and 16 GB flash storage) to detect faults in the working sequence of a robotic arm.

3.8 Edge AI for Human Safety

To improve pedestrian security, Miraftabzadeh et al. introduced an ANN-based technique that uses deep network embeddings as feature vectors for person re-identification in crowded scenarios [104]. Edge-devices first try to match the embedding of a detected face with locally stored embeddings. If no match is found, then the embedding is sent for matching and storage in the cloud.

Liu at al. propose an edge-based system that combines audio, video, and driving behavior data to detect attacks and potentially harmful events in ride-sharing services [90]. Driver and passenger smartphones act as edge-devices that can trigger video data to be sent to the cloud for subsequent analysis by a trained CNN model. The video data can be compressed at the edge to save cloud upload bandwidth.

A framework for an intelligent surveillance system using IoT, cloud computing, and edge computing has been designed by Dautov et al. [35]. Their prototype **Metropolitan Intelligent Surveillance System (MISS)**, which has been tested on a single-camera testbed, a cloud testbed, and in an edge-cluster setup, processes sensitive data at the edge of the network to enhance data security and reduces the amount of data transferred through the network.

3.9 Augmented Cognition

Researchers are now exploring how deep learning and edge computing can be used for augmenting human cognition to create adaptive human-machine collaboration by quickly giving expert guidance to the human for unfamiliar tasks and for amplifying the human's memory capabilities [127]. Such techniques promise to transform the way humans with low cognitive abilities can perform both routine and complex tasks, but questions pertaining to security, privacy, and ethics need to be addressed before such systems are deployed.

4 MACHINE LEARNING FRAMEWORKS, SOFTWARE, AND HARDWARE

The adoption of machine learning models as de facto solutions in an increasingly expanding list of applications has been accompanied by the development and evolution of many ML frameworks. In the resource constrained end-edge-cloud architecture, these developments have been mirrored by customized ML frameworks that facilitate the building of lightweight models as well as distributed learning and inference. We first discuss (Section 4.1) the most widely used frameworks and the software ecosystem used to build and deploy ML models on the device-edge-cloud architecture and provide a summary in Table 4. The hardware used in intelligent edge applications is described after that (Section 4.2) and summarized in Table 5.

Table 4. Machine Learning Frameworks that Have Been Used on Edge-devices

Framework	Core development language	Interface	Part running on the edge	Applications
TensorFlow Lite (Google)	C++, Java	Android, iOS Linux	TensorFlow Lite NN API	computer vision [147], speech recognition [2, 57]
Caffe2, Caffe2Go (Facebook)	C++	Android iOS	NNPack	image analysis, video analysis [70]
Apache MXNet	C++, R Python	Linux MacOS Windows	Full Model	object detection recognition [108]
Core ML2 (Apple)	Python	iOS	CoreML	image analysis [19]
ML Kit (Google)	C++ Java	Android iOS	Full Model	image recognition, text recognition, bar-code scanning [33]
AI2GO	C, Python Java, Swift	Linux MacOS	Full Model	object detection classification [6]
DeepThings	C/C++	Linux	Full Model	object detection [164]
DeepIoT	Python	Ubilinux	Full Model	human activity recognition user identification [156]
DeepCham	C++ Java	Linux Android	Full Model	object recognition [80]
SparseSep	-	Linux Android	Full Model	mobile object recognition audio classification [16]
DeepX	C++, Java, Lua	Linux Android	Full Model	mobile object recognition audio classification [78]
Edgent	-	Ubuntu	Major part of the DNN	image recognition [81]
daBNN	Arm Assembly, C++, Java	Mobile OS (Android and others)	Full model	computer vision [161]
TensorFlow Federated (Google)	Python	iOS, Android	Distributed training	computer vision [86]
EdgeML Library (Microsoft Research)	C++	-	-	image recognition [50]
CONDENSA	Python	Linux, Windows, MacOS	Full Model	language modeling and computer vision [68]

4.1 Frameworks and Software

TensorFlow is a popular machine learning framework, developed by Google. TensorFlow Lite[12] is a lightweight implementation of TensorFlow for edge-devices and embedded systems. It has been used for classification and regression on mobile devices. It supports DL without the help of a cloud server and has some neural network APIs to support hardware acceleration.[13] It can be run on multiple CPUs and GPUs and is therefore well-suited for distributed ML algorithms. The main programming languages for this framework are Java, Swift, Objective-C, C, and Python. A performance evaluation study of TensorFlow Lite by Zhang et al. showed that it occupied only 84 MB memory and took 0.26 second to execute an inference task using MobileNets on a Nexus 6p mobile device [162].

Caffe2[14] is a fast and flexible deep learning framework developed by Facebook. Caffe2Go is a lightweight and modular framework built on top of Caffe2. Both frameworks provide a straightforward way to implement deep learning models on mobile devices and can be used to analyze images in real-time.[15] Caffe2Go[16] can run on the Android and iOS platforms with the same code.

[12] https://www.tensorflow.org/lite.
[13] https://www.tensorflow.org/lite/performance/gpu_advanced.
[14] Caffe2 is now a part of Pytorch.
[15] https://engineering.fb.com/2016/11/08/android/delivering-real-time-ai-in-the-palm-of-your-hand/.
[16] https://caffe2.ai/blog/.

Table 5. Computing Devices that Have Been Used for Machine Learning at the Edge

Device type	Name	GPU	CPU	RAM	Flash memory	Power	Applications
ASICs (Section 4.2.1)	Google Coral Dev Board	GC7000 Lite Graphics + Edge TPU coprocessor	Quad Cortex-A53, Cortex-M4F	1 GB LPDDR4	8 GB LPDDR4	5 V DC	image processing [21]
	SparkFun Edge	-	32-bit ARM Cortex-M4F 48 MHz	384 KB	1 MB	6 uA/MHz	speech recognition [38]
	Intel Movidius NCS	High-performance VPU	Myriad 2 VPU	1 GB	4 GB	2 trillion 16-bit ops/s within 500 mW	classification [98] computer vision [15, 58]
	BeagleBone AI	-	Cortex-A15 Sitara AM5729 SoC with 4 EVEs	1 GB	16 GB	-	computer vision [32]
	Eta Compute EMC3531	-	ARM Cortex-M3 NXP Coolflux DSP	-	-	-	audio, video analysis[32]
FPGAs (Section 4.2.2)	ARM ML	-	ARM ML processor	1 GB	-	4 TOPs/W (Tera Operations)	image, voice recognition [146]
Embedded GPUs (Section 4.2.3)	Raspberry Pi	400 MHz VideoCore IV	Quad Cortex A53 @ 1.2 GHz	1 GB SDRAM	32 GB	2.5 Amp	video analysis [108, 153]
	NVIDIA Jetson TX1	Nvidia Maxwell 256 CUDA cores	Quad ARM A57/2 MB L2	4 GB 64-bit LPDDR4 25.6 GB/s	16 GB eMMC, SDIO, SATA	10 W	video, image analysis [79, 91], robotics [40]
	NVIDIA Jetson TX2	Nvidia Pascal 256 CUDA cores	HMP Dual Denver 2/2 MB L2 + Quad ARM A57/2 MB L2	8 GB 128-bit LPDDR4 59.7 GB/s	32 GB eMMC, SDIO, SATA	7.5 W	video, image analysis [91, 125], robotics [30]
	NVIDIA Jetson Nano	Nvidia Maxwell 128 CUDA cores	Quad ARM A57 MPCore	4 GB 64-bit LPDDR4 25.6 GB/s	16 GB eMMC, 5.1 Flash	5–10 W	computer vision [99, 144], audio analysis [46]
	OpenMV Cam	-	ARM 32-bit Cortex-M7	512 KB	2 MB	200 mA @ 3.3 V	image processing [4]
Open ISAs (Section 4.2.4)	RISC-V GAP8	-	nona-core 32-bit RISC-V @250 MHz	16 MiB SDRAM	-	1 GOPs/mW	image, audio processing [42]

It implements debugging tools by abstracting the neural network math. It uses fewer convolutional layers than traditional neural networks and optimizes the width of each layer to reduce model size.

Apache MXNet[17] is a lean, scalable, open-source framework for training deep neural networks and deploying them on resource-constrained edge-devices. MXNet supports distributed ecosystems and public cloud interaction to accelerate DNN training and deployment. It comes with tools that help to track, debug, save checkpoints, and modify hyperparameters of DNN models.

CoreML[18] is an iOS-based framework developed by Apple for building ML models and integrating them with Apple mobile applications. It allows an application developer to create an ML model to perform regression and image classification. This framework allows ML to run on edge-devices without a dedicated server. A trained DNN model is translated into CoreML format, and this translated model can be deployed using CoreML APIs to make an image classifier inside a mobile phone.

[17] https://mxnet.apache.org/.
[18] https://developer.apple.com/machine-learning/core-ml/.

ML Kit[19] is a mobile SDK framework introduced by Google. It uses Google's cloud vision APIs, mobile vision APIs, and TensorFlow Lite to perform tasks like text recognition, image labeling, and smart reply. Curukogluall et al. tested ML Kit APIs for image recognition, bar-code scanning, and text recognition on an Android device and reported that these APIs recognize different types of test objects such as tea cup, water glass, remote controller, and computer mouse successfully [33].

Introduced by Xnor, *AI2GO* helps tune deep learning models for popular use cases on resource-constrained devices. More than 100 custom ML models have been built with this framework for on-device AI inferencing. These custom models have the ability to detect objects, classify foods, and many other AI applications [6]. This platform targets specialized hardware that includes the Raspberry Pi, Ambarella S5L, Linux and macOS-based laptops, and Toradex Apalis iMX6. Allan conducted tests to benchmark the AI2GO platform on a Raspberry Pi and reported this platform to be 2 times faster than TensorFlow Lite in ML inferencing using a MobileNets v1 SSD 0.75 depth model [6].

DeepThings is a framework for adapting CNN-based inference applications on resource-constrained devices [164]. It provides a low memory footprint of convolutional layers by using **Fused Tile Partitioning (FTP)**. FTP divides the CNN model into multiple parts and generates partitioning parameters. These partitioning parameters with model weights are then distributed to edge-devices. When all edge-devices complete their computational tasks, a gateway device collects the processed data and generates results. The authors deployed YOLOv2 using DeepThings on Raspberry Pi 3 devices to demonstrate the deployment capability of this framework on IoT devices.

DeepIoT is a framework that shrinks a neural network into smaller dense matrices but keeps the performance of the algorithm almost the same [156]. This framework finds the minimum number of filters and dimensions required by each layer and reduces the redundancy of that layer. Developed by Yao et al., DeepIoT can compress a deep neural network by more than 90%, shorten execution time by more than 71%, and decrease energy consumption by 72.2% to 95.7%.

Li et al. introduced a framework called *DeepCham* that allows developers to deploy DL models on mobile environments with the help of edge computing devices [80]. DeepCham is developed for recognizing objects captured by mobile cameras, specifically targeting Android devices.

SparseSep, developed by Bhattacharya and Lane, is a framework for optimizing large-scale DL models for resource-constrained devices such as wearable hardware [16]. It runs large-scale DNNs and CNNs on devices that have ARM Cortex processors with very little impact on inference accuracy. It can also run on the NVidia Tegra K1 and the Qualcomm Snapdragon processors and was reported to run inference 13.3 times faster with 11.3 times less memory than conventional neural networks.

Lane et al. designed a software accelerator for low-power deep learning inference called *DeepX*, which allows developers to easily deploy DL models on mobile and wearable devices [78]. DeepX dramatically lowers resource overhead by decomposing a large deep model network into unit-blocks. These unit-blocks are generated using two resource control algorithms, namely, Runtime Layer Compression and Deep Architecture Decomposition, and executed by heterogeneous processors (e.g., GPUs, LPUs) of mobile phones.

Li et al. developed *Edgent*, a framework for deploying deep neural networks on small devices [81]. This framework adaptively partitions DNN computations between a small mobile device (e.g., Raspberry Pi) and the edge computing devices (e.g., laptops) and uses early-exit at an intermediate DNN layer to accelerate DNN inference.

[19] https://firebase.google.com/products/ml-kit.

daBNN is an open-source fast inference framework developed by Zhang et al., which can implement Binary Neural Networks on ARM devices [161]. An upgraded bit-packing scheme and binary direct convolution have been used in this framework to shrink the cost of convolution and speed up inference. This framework is written in C++ and ARM assembly and has Java support for the Android package. This fast framework can be 6 times faster than BMXNet[20] on Bi-Real Net 18.[21]

Joseph et al. developed a programmable system called *CONDENSA* that allows developers to design strategies for compressing DL models, resulting significant reduction in both the memory footprint and execution time [68]. It uses a Bayesian optimization-based method to find compression hyperparameters automatically by exploiting an L-C optimizer to recover the accuracy loss during compression. They reported a 2.22 times runtime improvement over an uncompressed VGG-19 on the CIFAR-10 dataset with up to 65 times memory reduction.

In addition to frameworks covered earlier, the edge ML ecosystem now also includes operating systems, domain specific programming languages, and software to facilitate the development of cloud-edge services.

SeeDot is a programming language, developed by Microsoft Research, to express ML inference algorithms and to control them at a mathematical-level [49]. Typically, most learning models are expressed in floating-point arithmetic, which are often expensive. Most resource-constrained devices do not support floating-point operations. To overcome this, SeeDot generates fixed-point code with only integer operations, which can be executed with just a few kilobytes of RAM. This helps SeeDot run a CNN on a resource-constrained micro-controller with no floating-point support. SeeDot-generated code for ML classification that uses Bonsai and ProtoNN (see Section 2) is 2.4 to 11.9 times faster than floating-point microcontroller-based code. Also, SeeDot-generated code was 5.2–9.8 times faster than code generated by high-level synthesis tools on an FPGA-based implementation.

AWS IoT Greengrass[22] is software that helps an edge-device to run serverless AWS Lambda functions. It can be used to run ML inference on an edge-device, filter device data, sync data, and only transmit important information back to the cloud. The *Azure IoT Edge*[23] is a platform that can be used to offload a large amount of work from the cloud to the edge-devices. It has been used to deploy ML models on edge-devices and cloud servers. Such workload migration reduces data communication latency and operates reliably even in offline periods.

Zephyr[24] is a real-time operating system with a small-footprint kernel, specially designed for resource-constrained devices. This OS supports multiple architectures, such as Intel x86, ARM Cortex-M, RISC-V 32, NIOS II, ARC, and Tensilica Xtensa. Zephyr is a collaborative project, which is hosted by the Linux Foundation under the Apache 2.0 license.

TVM is a compiler that has the capability of optimizing code by searching and detecting optimized tensor operators [27]. This compiler provides end-to-end compilation and optimization stacks that allow the deployment of deep learning on mobile GPUs, FPGA-based devices. This open-sourced compiler was evaluated on different edge devices such as embedded GPU (ARM Mali-T860MP4 GPU), FPGA-based accelerator, and reported a speedup of 1.2 to 3.8 times over existing hand-optimized libraries-based frameworks such as Tensorflow Lite.

[20] https://github.com/hpi-xnor/BMXNet.
[21] https://github.com/liuzechun/Bi-Real-net.
[22] https://aws.amazon.com/greengrass/.
[23] https://azure.microsoft.com/en-gb/services/iot-edge/.
[24] https://www.zephyrproject.org/.

4.2 Hardware

This section describes the low-power hardware that have been used in edge machine learning systems. High performance from a deep learning application is only achievable when an ML model is trained with a large amount of data, often on the order of terabytes. Only computationally rich GPUs and central CPUs have the ability to handle such a large amount of data in a reasonable period of time. This makes deep learning applications mostly GPU-centric. However, efforts have been taken to make resource-constrained devices compatible with deep learning, and it has been noticed that different types of small devices are being used to deploy ML at the network edge, including ASICs, FPGAs, RISC-V, and embedded devices. Table 5 lists devices commonly used to deploy ML systems at the edge.

4.2.1 Application-specific Integrated Circuits (ASICs). The ***Edge Tensor Processing Unit***[25] **(TPU)** is an ASIC chip designed by Google for accelerated ML inference on edge-devices. It is capable of running CNNs such as MobileNets V1/V2, MobileNets SSD V1/V2, and Inception V1-4 as well as TensorFlow Lite models. In the power-efficient mode, the Edge TPU can execute state-of-the-art mobile vision models at 100+ FPS.[26] The *Coral Dev Board*, also by Google, uses the Edge TPU as a co-processor to run ML applications. It has two parts, a baseboard and a **system-on-module (SOM)**. The baseboard has a 40-pin GPIO header to integrate with various sensors or IoT devices and the SOM has a Cortex-A53 processor with an additional Cortex-M4 core, 1 GB of RAM and 8 GB of flash memory that helps to run Linux OS on Edge TPU.

The *Coral USB accelerator*[27] is a device that helps to run ML inference on small devices like the Raspberry Pi. The accelerator is a co-processor for an existing system, which can connect to any Linux system using a USB-C port. Allan deployed ML inference on different edge-devices, including the Coral Dev Board and conducted tests to benchmark the inference performance [5, 8]. The Coral Dev Board performed 10 times faster than Movidius NCS, 3.5 times faster than Nvidia Jetson Nano (TF-TRT), and 31 times faster than Raspberry Pi for MobileNetV2 SSD model.

The *SparkFun Edge* is a real-time audio analysis device that runs ML inference to detect a keyword, for example, "yes," and responds accordingly.[28] Developed by Google, Ambiq, and SparkFun collaboratively, it is used for voice and gesture recognition at the edge[29] without the help of remote services. It has a 32-bit ARM Cortex-M4F 48 MHz processor with 96 MHz burst mode, extremely low-power usage, 384 KB SRAM, 1 MB flash memory, and a dedicated BLE 5 Bluetooth processor. It also has two built-in microphones, a 3-axis accelerometer, a camera connector, and other input/output connectors. This device can run for 10 days with a CR2032 coin cell battery. Ambiq Apollo3 is a Software Development Kit is available for building AI applications with the SparkFun Edge.

Intel's *Movidius*[30] is a vision processing unit that can accelerate deep neural network inferences in resource-constrained devices such as intelligent security cameras or drones. This chip can run custom vision, imaging, and deep neural network workloads on edge-devices without a connection to the network or any cloud backend. This chip can be deployed on a robot placed in rescue operations in disaster-affected areas. The rescue robot can make some life-saving decisions without human help. It can run real-time deep neural networks by performing 100 gigaflops within a 1 W power envelope. Movidius Myriad 2 is the second generation **vision processing unit (VPU)**

[25] https://cloud.google.com/edge-tpu/.
[26] https://coral.withgoogle.com/docs/edgetpu/faq/.
[27] https://coral.withgoogle.com/products/accelerator/.
[28] https://learn.sparkfun.com/tutorials/sparkfun-edge-hookup-guide/all.
[29] https://www.sparkfun.com/products/15170.
[30] https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html.

and Myriad X VPU is the most advanced VPU from Movidius to provide artificial intelligence solutions from drones and robotics to smart cameras and virtual reality.[31] Intel also provides a **Myriad Development Kit (MDK)**, which includes tools and APIs needed to implement ML on the chip.

The *Movidius Neural Compute Stick* by Intel is a USB-like stick that extends the same technology of Intel Myriad (SoC) board. This plug-and-play device can be easily attached to edge-devices running by Ubuntu 16.04.3 LTS (64 bit), CentOS* 7.4 (64 bit), Windows 10 (64 bit), Raspbian (target only), including Raspberry Pi, Intel NUC, and so on. This device has an Intel Movidius Myriad X VPU processor, which supports TensorFlow, Caffe, Apache MXNet, **Open Neural Network Exchange (ONNX)**, PyTorch, and PaddlePaddle via an ONNX conversion.

The *BeagleBone AI* is a high-end board for developers building machine-learning and computer-vision applications [32]. This device is powered by an SoC – TI AM5729 dual core Cortex-A15 processor featuring four programmable real-time units, a dual core C66x digital-signal-processor, and four embedded-vision-engines core supported through the **TIDL (Texas Instruments Deep Learning)** ML API. It can perform image classification, object detection, and semantic segmentation using TIDL.

ECM3531[32] is a high-efficiency ASIC based on the ARM Cortex-M3 and NXP Coolflux DSP processors for machine learning applications. The name of the processor of this ASIC is Tensai, which can run TensorFlow or Caffe framework. This processor offers 30-fold power reduction in a specific CNN-based image classification.

The *SmartEdge Agile*[33] device along with the accompanying *Brianium* software help build artificial intelligence models and deploy them on resource-constrained devices. SmartEdge Agile sits uses Brainium's zero-coding platform to deploy a trained intelligent model on the edge [7].

4.2.2 Field-programmable Gate Arrays (FPGAs). The *ARM ML processor* [12] allows developers to accelerate the performance of ML algorithms and deploys inference on edge-devices. The ecosystem consists of (*a*) ARM NN[34], an inference engine that provides a translation layer that bridges the gap between existing Neural Network frameworks and ARM ML processor, and (*b*) ARM Compute Library,[35] an open source library containing functions optimized for ARM processors. The ARM ML processor can run high-level neural network frameworks such as TensorFlow Lite, Caffe, and ONNX. The ARM NN SDK has all the necessary tools to run neural networks on edge-devices. This processor is designed for mobile phones, AR/VR, robotics, and medical instruments. Lai and Suda designed a set of efficient neural network kernels called CMSIS-NN[36] to maximize the performance of a neural network using limited memory and compute resources of an ARM Cortex-M processor [77].

Microsoft's *Brainwave* is an effort to use FPGA technology to solve the challenges of real-time AI and to run deep learning models in the Azure cloud and on the edge in real-time [103]. To meet the computational demands required of deep learning, Brainwave uses Intel Stratix 10 FPGAs as the heart of the system providing 39.5 TFLOPs of effective performance. Most popular deep learning models, including ResNet 50, ResNet 152, VGG-16, SSD-VGG, DenseNet-121, and SSD-VGG, are supported Brainwave FPGAs on Azure to accomplish image classification and object detection task at the network edge. Azure can parallelize pre-trained DNNs across FPGAs to scale out any service.

[31] https://www.intel.com/content/www/us/en/products/docs/processors/movidius-vpu/myriad-x-product-brief.html.
[32] https://etacompute.com/products/.
[33] https://www.avnet.com/wps/portal/integrated/solutions/iot/building-blocks/smartedge-agile/.
[34] https://mlplatform.org/.
[35] https://www.arm.com/why-arm/technologies/compute-library.
[36] https://github.com/ARM-software/CMSIS_5.

4.2.3 Embedded GPUs. The *Raspberry Pi*, a single-board computer developed by the Raspberry Pi Foundation, is one of the most common devices used for edge computing. It has been used to run ML inference without any extra hardware. The Raspberry Pi 3 Model B has a Quad Cortex A53 @ 1.2 GHz CPU, 400 MHz VideoCore IV GPU, 1 GB SDRAM. Xu et al. used Raspberry Pi 3 as edge-devices to develop a real-time human surveillance system [108, 153]. Their system is able to distinguish between human and nonhuman objects in real-time. It has a micro-SD card slot to support flash memory up to 32 GB. Xnor.ai has developed a new AI platform to run deep learning models efficiently on edge-devices such as embedded CPUs (e.g., Raspberry Pi), phones, IoT devices, and drones without using a GPU or TPU [76, 152].

The *Nvidia Jetson* is an embedded computing board that can process complex data in real-time. The Jetson AGX Xavier can operate with a 30 W power supply and perform like a GPU workstation for edge AI applications. *Jetson TX1*, *Jetson TX2,* and *Jetson Nano* are embedded AI computing devices powered by the Jetson. These three small, but powerful, computers are ideal for implementing an intelligent system on edge-devices such as smart security cameras, drones, robots, and portable medical devices. JetPack is an SDK for building AI applications with the Jetson. This SDK includes TensorRT, cuDNN, Nvidia DIGITS Workflow, ISP Support, Camera imaging, Video CODEC, Nvidia VisionWorks, OpenCV, Nvidia CUDA, and CUDA Library tools for supporting ML. It is also compatible with the Robot Operating System (ROS[37]).

OpenMV Cam[38] is a small, low-powered camera board. This board is built using an ARM Cortex-M7 processor to execute machine vision algorithms at 30 FPS. This processor can run at 216 MHz and has 512 KB of RAM, 2 MB of flash, and 10 I/O pins. The main applications of this device are face detection, eye tracking, QR code detection/decoding, frame differencing, AprilTag tracking, and line detection [4].

4.2.4 Open Instruction Set Architecture (ISA). The GAP8 [139] is a microprocessor conforming to the RISC-V open set architecture [118]. With nine cores capable of running 10 GOPS at the order of tens of mW, this 250 MHz processor is designed to accelerate CNNs for the edge computing and IoT market. The TF2GAP8 is a tool that can automatically translate TensorFlow CNN applications to GAP8 source [140].

5 CHALLENGES AND FUTURE DIRECTIONS

To fully utilize the benefits offered by edge intelligence, a number of issues that significantly inhibit more widespread adoption of edge-based machine learning applications need to be addressed.

5.1 Building New Datasets

An important characteristic of deep learning algorithms is their ability to train using labeled as well as unlabeled input data. The availability of large amounts of unlabeled data generated by edge-servers and end-devices provides a faster way of building new datasets. For low-data scenarios, data augmentation can be employed [110]. Data augmentation uses a small amount of data (transferred from sensors to edge-servers or from edge-servers to cloud servers) to generate new data. Augmentation helps ML models avoid overfitting issues by generating enough new training data [110]. However, the performance of data augmentation by edge-devices or cloud servers needs to be evaluated before its use in a learning model, especially for small datasets.

The use of various types of IoT sensors creates heterogeneous environments in edge-based intelligent systems. To deal with the diversity of data in edge environments, ML algorithms need

[37] https://www.ros.org.
[38] https://openmv.io/collections/cams/products/openmv-cam-m7.

to learn using types of data that have different features such as image, text, sound, and motion. Multimodal deep learning is adapted to learn features over multiple modalities such as audio and video [107] for the high-velocity heterogeneous data generation that is common in intelligent edge settings. An additional challenge arises when the data keep changing over time, and the joint probability between classes and the generated data changes because of seasonality, periodicity effects or hardware/software failure. An important area of research is to create novel machine learning algorithms for handling the non-stationary data streams generated by IoT sensors [36].

5.2 Centralized vs. Distributed Training

To handle the enormous amount of data produced by IoT sensors, researchers have designed an edge-based distributed learning algorithm (Section 2.4) over distributed computing hierarchies. The architecture consists of the cloud servers, the edge-servers, and end-devices like IoT sensors. Such algorithms provide acceptable accuracy with datasets that are naturally distributed; for example, fraud detection and market analysis. However, the influence of the heterogeneity of data on the accuracy of a distributed model is an open research issue [119].

Since training a deep learning model on edge-devices is difficult (due to their limited memory and computational capabilities), most existing ML systems use the cloud for training. Some attempts have been made to train models on edge-devices (such as by using model pruning and model quantization) but pruned edge-trained models often have lower accuracy, and therefore designing power-efficient algorithms for training neural networks on edge-devices is an active research area. There remains continued interest in developing new methods and frameworks that map sections of a deep learning model onto the distributed computing architecture and also in exploring the use of specialized hardware (such as ARM ML processors) to speed up deep learning training and inference.

Computation results and data-sharing across different edge-devices are key components to establish an effective ML-edge distributed system. Novel networking paradigms that are "computation-aware" are highly desirable for building such data-sharing distributed systems. 5G networks, which provide the **ultra-reliable low-latency communication (URLLC)** services, are a promising area to integrate with edge computing. 5G should help to establish more control over the network resources for supporting on-demand interconnections across different edge-devices. The adaptation of the software-defined network and network function virtualization into 5G networks to control distributed ML settings will be an appealing research area for future ML researchers.

5.3 Trust and Explainability

A major concern in the ML and wider community is of the accountability and transparency of ML systems. The current black-box-like decision-making process of ML-based AI systems has raised questions about inherent algorithmic bias [123]. This has led to the development of ***Explainable AI* (XAI)** and *interpretable AI*, which seek to provide the reasons why a particular decision was reached by the AI system. An overview of issues in XAI can be found in the survey by Adadi and Berrada [1].

Sensitive domains such as health-care, law-enforcement, and jurisprudence require high accuracy as well as explainability. However, deep learning models deployed on edge-devices are inherently less accurate given the resource-constrained environment. Therefore, a primary challenge facing the community is to explore how to ensure that no permanently harmful decision is made using an intelligent edge system.

Due to limited sizes and inherent biases in datasets, ML models can treat different subsets of the population differently and even unfairly [142]. This has led to the development of

fairness-enhanced ML models. However, this aspect of ML remains under-explored and different methods of introducing fairness are still affected by dataset bias [45]. Introducing fairness as a criterion during ML model training raises some important issues. For example, Biswas and Rajan discuss how fairness with respect to one attribute can lead to lesser fairness with respect to another attribute and also that optimizing for fairness can lead to a reduction in performance [17]. Another important focus increasingly is incorporating fairness criteria and their optimization in ML software libraries and frameworks.

5.4 Novel Topics

With the increase in the dedicated hardware for ML, an important direction of future work is the development of compilers, such as Glow [124], that optimize neural network graphs for heterogeneous hardware. Li et al. provide a detailed survey of the architectures of deep learning compilers [82]. Data produced by resource-constrained end-devices in a decentralized distributed setting is always vulnerable to security threats. Edge-based blockchain technology has great potential to prevent those threats and transfer sensitive data over a decentralized edge infrastructure. However, deploying blockchains in resource-constrained settings is challenging due to the huge computing power and energy requirements in blockchain mining [149].

6 CONCLUSION

Edge-based ML is a fast-growing research area with numerous challenges and opportunities. Using edge-devices for ML has been shown to improve not only the privacy and security of user data but also system response times. This article provides a comprehensive overview of techniques pertaining to the training and deployment of ML systems at the network edge. It highlights several new ML architectures that have been designed specifically for resource-constrained edge computing devices and reviews important applications of edge-based ML systems. Widely adopted frameworks essential for developing edge-based deep learning architectures as well as the resource-constrained devices that have been used to deploy these models are described. Finally, challenges in edge-based ML are listed and several directions for future work are outlined.

REFERENCES

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

[2] Adafruit. 2019. Micro Speech Demo. Retrieved from https://learn.adafruit.com/tensorflow-lite-for-edgebadge-kit-quickstart/micro-speech-demo.

[3] M. Ali, A. Anjum, M. U. Yaseen, A. R. Zamani, D. Balouek-Thomert, O. Rana, and M. Parashar. 2018. Edge enhanced deep learning system for large-scale video stream analytics. In *IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. 1–10. DOI: https://doi.org/10.1109/CFEC.2018.8358733

[4] Alasdair Allan. 2018. Deep Learning at the Edge on an Arm Cortex-powered Camera Board. Retrieved from https://blog.hackster.io/deep-learning-at-the-edge-on-an-arm-cortex-powered-camera-board-3ca16eb60ef7.

[5] Alasdair Allan. 2019. Benchmarking Edge Computing. Retrieved from https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245.

[6] Alasdair Allan. 2019. Benchmarking the Xnor AI2GO Platform on the Raspberry Pi. Retrieved from https://blog.hackster.io/benchmarking-the-xnor-ai2go-platform-on-the-raspberry-pi-628a82af8aea.

[7] Alasdair Allan. 2019. Hands-on with the SmartEdge Agile. Retrieved from https://blog.hackster.io/hands-on-with-the-smartedge-agile-b7b7f02b5d4b.

[8] Alasdair Allan. 2019. Measuring Machine Learning. Retrieved from https://towardsdatascience.com/measuring-machine-learning-945a47bd3750.

[9] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-time video analytics: The killer app for edge computing. *Computer* 50, 10 (2017), 58–67.

[10] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. 2019. Video analytics—Killer app for edge computing. In *17th International Conference on Mobile Systems, Applications, and Services (MobiSys'19)*. ACM, New York, NY, 695–696. DOI: https://doi.org/10.1145/3307334.3328589

[11] Andrej Karpathy. 2019. PyTorch at Tesla. Retrieved from https://www.youtube.com/watch?v=oBklltKXtDE.
[12] ARM Limited. Machine Learning ARM ML Processor. Retrieved on July 25, 2021 from https://developer.arm.com/ip-products/processors/machine-learning.
[13] Asha Barbaschow. 2018. VMware looking towards IoT and the edge. Retrieved from https://www.zdnet.com/article/vmware-looking-towards-iot-and-the-edge/.
[14] M. Barnell, C. Raymond, C. Capraro, D. Isereau, C. Cicotta, and N. Stokes. 2018. High-performance computing (HPC) and machine learning demonstrated in flight using Agile Condor. In *IEEE High Performance Extreme Computing Conference (HPEC)*. 1–4.
[15] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma. 2015. Always-on vision processing unit for mobile applications. *IEEE Micro* 35, 2 (Mar. 2015), 56–66. DOI: https://doi.org/10.1109/MM.2015.10
[16] Sourav Bhattacharya and Nicholas D. Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *14th ACM Conference on Embedded Network Sensor Systems CD-ROM (SenSys'16)*. ACM, New York, NY, 176–189.
[17] Sumon Biswas and Hridesh Rajan. 2020. Do the machine learning models on a crowd sourced platform exhibit bias? An empirical study on model fairness. In *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.
[18] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.
[19] Andrew A. Borkowski, Catherine P. Wilson, Steven A. Borkowski, Lauren A. Deland, and Stephen M. Mastorides. 2019. Using Apple machine learning algorithms to detect and subclassify non-small cell lung cancer. *Arxiv E-prints* 1808.08230 (January 2019).
[20] Brandon Butler. 2017. What is edge computing and how it's changing the network. *Network World* (2017). DOI: https://www.networkworld.com/article/3224893/what-is-edge-computing-and-how-it-s-changing-the-network.html.
[21] S. Cass. 2019. Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectrum* 56, 5 (May 2019), 16–17. DOI: https://doi.org/10.1109/MSPEC.2019.8701189
[22] W. Chang, L. Chen, and K. Su. 2019. DeepCrash: A deep learning-based internet of vehicles system for head-on and single-vehicle accident detection with emergency notification. *IEEE Access* 7 (2019), 148163–148175.
[23] X. Chang, W. Li, C. Xia, J. Ma, J. Cao, S. U. Khan, and A. Y. Zomaya. 2018. From insight to impact: Building a sustainable edge computing platform for smart homes. In *IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 928–936. DOI: https://doi.org/10.1109/PADSW.2018.8644647
[24] G. Chen, C. Parada, and G. Heigold. 2014. Small-footprint keyword spotting using deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4087–4091. DOI: https://doi.org/10.1109/ICASSP.2014.6854370
[25] J. Chen and X. Ran. 2019. Deep learning with edge computing: A review. *Proc. IEEE* 107, 8 (Aug. 2019), 1655–1674.
[26] Min Chen, Yuanwen Tian, Giancarlo Fortino, Jing Zhang, and Iztok Humar. 2018. Cognitive internet of vehicles. *Comput. Commun.* 120 (2018), 58–70.
[27] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen.
[28] Y. Chen, A. Wu, M. A. Bayoumi, and F. Koushanfar. 2013. Editorial low-power, intelligent, and secure solutions for realization of internet of things. *IEEE J. Emerg. Select. Topics Circ. Syst.* 3, 1 (Mar. 2013), 1–4. DOI: https://doi.org/10.1109/JETCAS.2013.2244771
[29] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *Arxiv E-prints* 1710.09282 (2017).
[30] Sandeep Chinchali, Apoorva Sharma, James Harrison, Amine Elhafsi, Daniel Kang, Evgenya Pergament, Eyal Cidon, Sachin Katti, and Marco Pavone. 2021. Network offloading policies for cloud robotics: a learning-based approach. *Autonomous Robots* (2021), 1–16. https://doi.org/10.1007/s10514-021-09987-4
[31] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey 1109 on model compression and acceleration. *Artif. Intell. Rev.* (2020), 1–43.
[32] Christine Long. 2019. BeagleBone AI Makes a Sneak Preview. Retrieved from https://beagleboard.org/blog/2019-05-16-beaglebone-ai-preview.
[33] N. Curukogle and B. M. Ozyildirim. 2018. Deep learning on mobile systems. In *Innovations in Intelligent Systems and Applications Conference (ASYU)*. 1–4. DOI: https://doi.org/10.1109/ASYU.2018.8554039

[34] A. Das, M. Degeling, X. Wang, J. Wang, N. Sadeh, and M. Satyanarayanan. 2017. Assisting users in a world full of cameras: A privacy-aware infrastructure for computer vision applications. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 1387–1396. DOI: https://doi.org/10.1109/CVPRW.2017.181

[35] Rustem Dautov, Salvatore Distefano, Dario Bruneo, Francesco Longo, Giovanni Merlino, Antonio Puliafito, and Rajkumar Buyya. 2018. Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms. *Softw., Pract. Exper.* 48 (2018), 1475–1492.

[36] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. 2015. Learning in nonstationary environments: A survey. *IEEE Comput. Intell. Mag.* 10, 4 (2015), 12–25.

[37] Utsav Drolia, Katherine Guo, and Priya Narasimhan. 2017. Precog: Prefetching for image recognition applications at the edge. In *2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3132211.3134456

[38] SparkFun Electronics. SparkFun Edge Hookup Guide. Retreived July 25, 2021 from https://learn.sparkfun.com/tutorials/sparkfun-edge-hookup-guide/all.

[39] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski. 2017. Where's the bear?—Automating wildlife image processing using IoT and edge cloud systems. In *IEEE/ACM 2nd International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 247–258.

[40] E. Ezra Tsur, E. Madar, and N. Danan. 2018. Code generation of graph-based vision processing for multiple CUDA cores SoC Jetson TX. In *IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 1–7. DOI: https://doi.org/10.1109/MCSoC2018.2018.00013

[41] Z. Feng, S. George, J. Harkes, P. Pillai, R. Klatzky, and M. Satyanarayanan. 2018. Edge-based discovery of training data for machine learning. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 145–158.

[42] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini. 2018. GAP-8: A RISC-V SoC for AI at the edge of the IoT. In *IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 1–4. DOI: https://doi.org/10.1109/ASAP.2018.8445101

[43] David Floyer. 2015. The Vital Role of Edge Computing in the Internet of Things. Retrieved from https://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things.

[44] The Linux Foundation. Accessed:. The Open Platform for the IoT Edge. Retrieved on July 25, 2021 from https://www.edgexfoundry.org.

[45] Sorelle A. Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P. Hamilton, and Derek Roth. 2019. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*'19)*. Association for Computing Machinery, New York, NY, 329–338.

[46] C. Gao, Antonio Rios-Navarro, Xi Chen, T. Delbrück, and Shih-Chii Liu. 2020. EdgeDRNN: Enabling low-latency recurrent neural network edge inference. In *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 41–45.

[47] Robin C. Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *CoRR* abs/1712.07557 (2017).

[48] A. Ghoneim, G. Muhammad, S. U. Amin, and B. Gupta. 2018. Medical image forgery detection for smart healthcare. *IEEE Commun. Mag.* 56, 4 (Apr. 2018), 33–37. DOI: https://doi.org/10.1109/MCOM.2018.1700817

[49] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. 2019. Compiling KB-sized machine learning models to tiny IoT devices. In *40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'19)*. ACM, New York, NY, 79–95. DOI: https://doi.org/10.1145/3314221.3314597

[50] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. ProtoNN: Compressed and accurate kNN for resource-scarce devices. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 1331–1340. Retrieved from http://proceedings.mlr.press/v70/gupta17a.html.

[51] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 1737–1746. Retrieved from http://dl.acm.org/citation.cfm?id=3045118.3045303.

[52] Ramyad Hadidi, Jiashen Cao, Michael S. Ryoo, and Hyesoon Kim. 2019. Robustly executing DNNs in IoT systems using coded distributed computing. In *56th Design Automation Conference 2019 (DAC'19)*. Association for Computing Machinery, New York, NY.

[53] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim. 2018. Distributed perception by collaborative robots. *IEEE Robot. Autom. Lett.* 3, 4 (2018), 3709–3716.

[54] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:https://arxiv.org/abs/1510.00149.

[55] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. arXiv:https://arxiv.org/abs/1412.5567.
[56] Richard Harper. 2003. *Inside the Smart House*. Springer-Verlag, Berlin.
[57] Evan Hennis, Mark Deoust, and Billy Lamberta. 2019. TensorFlow Lite Speech Command Recognition Android Demo. Retrieved from https://github.com/tensorflow/examples/tree/master/lite/examples/speech_commands/android.
[58] Jacob Hochstetler, Rahul Padidela, Qing Chen, Qiang Yang, and Songnian Fu. 2018. Embedded deep learning for vehicular edge computing. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 341–343.
[59] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:https://arxiv.org/abs/1704.04861.
[60] C. C.-H. Hsu, M. Y.-C. Wang, H. C. H. Shen, R. H. Chiang, and C. H. P. Wen. 2017. FallCare+: An IoT surveillance system for fall detection. In *International Conference on Applied System Innovation (ICASI)*. 921–922.
[61] Z. Huai, B. Ding, H. Wang, M. Geng, and L. Zhang. 2019. Towards deep learning on resource-constrained robots: A crowdsourcing approach with model partition. In *IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 989–994.
[62] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *30th International Conference on Neural Information Processing Systems*. 4114–4122.
[63] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv:https://arxiv.org/abs/1602.07360.
[64] Michaela Iorga, Larry B. Feldman, Robert Barton, Michael Martin, Nedim S. Goren, and Charif Mahmoudi. 2018. Fog Computing Conceptual Model. DOI: https://doi.org/10.6028/NIST.SP.500-325
[65] D. Isereau, C. Capraro, E. Cote, M. Barnell, and C. Raymond. 2017. Utilizing high-performance embedded computing, Agile Condor, for intelligent processing: An artificial intelligence platform for remotely piloted aircraft. In *Intelligent Systems Conference (IntelliSys)*. 1155–1159. DOI: https://doi.org/10.1109/IntelliSys.2017.8324277
[66] Kaya Ismail. 2018. Edge Computing vs. Fog Computing: What's the Difference? Retrieved from https://www.cmswire.com/information-management/edge-computing-vs-fog-computing-whats-the-difference/.
[67] R. Colin Johnson. 2019. Neural Learning on the Edge. Retrieved from https://cacm.acm.org/news/234063-neural-learning-on-the-edge/fulltext.
[68] Vinu Joseph, Ganesh L. Gopalakrishnan, Saurav Muralidharan, Michael Garland, and Animesh Garg. 2020. A programmable approach to neural network compression. *IEEE Micro* 40, 5 (Sep. 2020), 17–25.
[69] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and opportunities in DNN-based video analytics: A demonstration of the BlazeIt video query engine. In *9th Biennial Conference on Innovative Data Systems Research*. Retrieved from http://cidrdb.org/cidr2019/papers/p141-kang-cidr19.pdf.
[70] Duseok Kang, Euiseok Kim, Inpyo Bae, Bernhard Egger, and Soonhoi Ha. 2018. C-GOOD: C-code generation framework for optimized on-device deep learning. In *International Conference on Computer-Aided Design (ICCAD'18)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3240765.3240786
[71] Gorkem Kar, Shubham Jain, Marco Gruteser, Fan Bai, and Ramesh Govindan. 2017. Real-time traffic estimation at vehicular edge nodes. In *2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3132211.3134461
[72] Jae-Yun Kim and Soo-Mook Moon. 2018. Blockchain-based edge computing for deep neural network applications. In *Workshop on INTelligent Embedded Systems Architectures and Applications (INTESA'18)*. Association for Computing Machinery, New York, NY, 53–55.
[73] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 1935–1944. Retrieved from http://proceedings.mlr.press/v70/kumar17a.html.
[74] Aditya Kusupati, Don Dennis, Chirag Gupta, Ashish Kumar, Shishir Patil, and Harsha Simhadri. 2021. The EdgeML Library: An ML library for machine learning on the Edge. Retrieved on July 25, 2021 from https://github.com/Microsoft/EdgeML.
[75] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 9017–9028. Retrieved from http://papers.nips.cc/paper/8116-fastgrnn-a-fast-accurate-stable-and-tiny-kilobyte-sized-gated-recurrent-neural-network.pdf.
[76] Gant Laborde. 2019. Perf Machine Learning on Rasp Pi. Retrieved from https://medium.com/free-code-camp/perf-machine-learning-on-rasp-pi-51101d03dba2.

[77] Liangzhen Lai and Naveen Suda. 2018. Enabling deep learning at the IoT edge. In *International Conference on Computer-Aided Design (ICCAD'18)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3240765.3243473

[78] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 1–12. DOI: https://doi.org/10.1109/IPSN.2016.7460664

[79] S. Lee, K. Son, H. Kim, and J. Park. 2017. Car plate recognition based on CNN using embedded system with GPU. In *10th International Conference on Human System Interactions (HSI)*. 239–241. DOI: https://doi.org/10.1109/HSI.2017.8005037

[80] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah. 2016. DeepCham: Collaborative edge-mediated adaptive deep learning for mobile object recognition. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 64–76. DOI: https://doi.org/10.1109/SEC.2016.38

[81] En Li, Zhi Zhou, and Xu Chen. 2018. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Workshop on Mobile Edge Communications (MECOMM'18)*. ACM, New York, NY, 31–36.

[82] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. 2020. The Deep Learning Compiler: A Comprehensive Survey. arXiv:https://arxiv.org/abs/2002.03794.

[83] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2019. Federated learning: Challenges, methods, and future directions. *Arxiv Eprints* abs/1908.07873 (2019).

[84] Le Liang, Hao Ye, and Geoffrey Ye Li. 2018. Toward intelligent vehicular networks: A machine learning framework. *IEEE Internet Things J.* 6, 1 (2018), 124–135.

[85] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. 2020. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* 22, 3 (2020), 2031–2063.

[86] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2020. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. arXiv:https://arxiv.org/abs/1712.01887.

[87] Zhong Qiu Lin, Audrey G. Chung, and Alexander Wong. 2018. EdgeSpeechNets: Highly Efficient Deep Neural Networks for Speech Recognition on the Edge. arXiv:https://arxiv.org/abs/1810.08559.

[88] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, M. Yunsheng, S. Chen, and P. Hou. 2018. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Trans. Serv.Comput.* 11, 2 (Mar. 2018), 249–261. DOI: https://doi.org/10.1109/TSC.2017.2662008

[89] Lumin Liu, Jun Zhang, S. H. Song, and Khaled B. Letaief. 2019. Client-Edge-Cloud Hierarchical Federated Learning. arXiv:https://arxiv.org/abs/1905.06641

[90] L. Liu, X. Zhang, M. Qiao, and W. Shi. 2018. SafeShareRide: Edge-based attack detection in ridesharing services. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 17–29. DOI: https://doi.org/10.1109/SEC.2018.00009

[91] Qiang Liu, Siqi Huang, and Tao Han. 2017. Fast and accurate object analysis at the edge for mobile augmented reality: Demo. In *2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*. ACM, New York, NY. DOI: https://doi.org/10.1145/3132211.3132458

[92] Wang Luping, Wang Wei, and Li Bo. 2019. CMFL: Mitigating communication overhead for federated learning. https://doi.org/10.1109/ICDCS.2019.00099

[93] Salma Abdel Magid, Francesco Petrini, and Behnam Dezfouli. 2020. Image classification on IoT edge devices: Profiling and modeling. *Clust. Comput.* 23, 2 (2020), 1025–1043.

[94] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. 2018. Machine learning for internet of things data analysis: A survey. *Dig. Commun. Netw.* 4, 3 (2018), 161–175. DOI: https://doi.org/10.1016/j.dcan.2017.10.002

[95] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. 2015. *The Internet of Things: Mapping the Value Behind the Hype.* Technical Report. McKinsey and Company.

[96] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen. 2017. MoDNN: Local distributed mobile computing system for deep neural network. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. 1396–1401. DOI: https://doi.org/10.23919/DATE.2017.7927211

[97] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong. 2018. Learning from differentially private neural activations with edge computing. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 90–102. DOI: https://doi.org/10.1109/SEC.2018.00014

[98] C. Marantos, N. Karavalakis, V. Leon, V. Tsoutsouras, K. Pekmestzi, and D. Soudris. 2018. Efficient support vector machines implementation on Intel/Movidius Myriad 2. In *7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. 1–4. DOI: https://doi.org/10.1109/MOCAST.2018.8376630

[99] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge. 2020. Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 954–964.

[100] Bradley McDanel, Surat Teerapittayanon, and H. T. Kung. 2017. Embedded Binarized Neural Networks. arXiv:https://arxiv.org/abs/1709.02260.

[101] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Vol. 54. PMLR, 1273–1282. Retrieved from http://proceedings.mlr.press/v54/mcmahan17a.html.

[102] Microsoft. 2018. Embedded Learning Library. https://microsoft.github.io/ELL/.

[103] Microsoft. 2019. Project Brainwave. Retrieved from https://www.microsoft.com/en-us/research/project/project-brainwave/.

[104] S. A. Miraftabzadeh, P. Rad, K. R. Choo, and M. Jamshidi. 2018. A privacy-aware architecture at the edge for autonomous real-time identity reidentification in crowds. *IEEE Internet Things J.* 5, 4 (Aug. 2018), 2936–2946. DOI: https://doi.org/10.1109/JIOT.2017.2761801

[105] M. G. S. Murshed, J. J. Carroll, N. Khan, and F. Hussain. 2020. Resource-aware on-device deep learning for supermarket hazard detection. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 871–876. DOI: https://doi.org/10.1109/ICMLA51294.2020.00142

[106] Pedro Navarro Lorente, Carlos Fernandez, Raul Borraz, and Diego Alonso. 2016. A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3D range data. *Sensors* 17 (12 2016), 18. DOI: https://doi.org/10.3390/s17010018

[107] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. 2011. Multimodal deep learning. In *28th International Conference on International Conference on Machine Learning (ICML'11)*. Omnipress, 689–696. Retrieved from http://dl.acm.org/citation.cfm?id=3104482.3104569.

[108] Seyed Yahya Nikouei, Yu Chen, Sejun Song, Ronghua Xu, Baek-Young Choi, and Timothy R. Faughnan. 2018. Intelligent surveillance as an edge network service: from Harr-Cascade, SVM to a lightweight CNN. arXiv:1805.00331.

[109] Takayuki Nishio and Ryo Yonetani. 2018. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE International Conference on Communications (ICC)*. 1–7.

[110] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-garadi, and Uzoma Rita Alo. 2018. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Exp. Syst. Applic.* 105 (2018), 233–261. DOI: https://doi.org/10.1016/j.eswa.2018.03.056

[111] Samuel S. Ogden and Tian Guo. 2018. MODI: Mobile deep inference made efficient by edge computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*. USENIX Association. Retrieved from https://www.usenix.org/conference/hotedge18/presentation/ogden.

[112] S. A. Osia, A. S. Shamsabadi, A. Taheri, H. R. Rabiee, and H. Haddadi. 2018. Private and scalable personal data analytics using hybrid edge-to-cloud deep learning. *Computer* 51, 5 (May 2018), 42–49.

[113] Anand Oswal. 2018. Time to Get Serious About Edge Computing. Retrieved from https://blogs.cisco.com/enterprise/time-to-get-serious-about-edge-computing.

[114] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 27–40. DOI: https://doi.org/10.1145/3140659.3080254

[115] Donghyun Park, Seulgi Kim, Yelin An, and Jae-Yoon Jung. 2018. LiReD: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks. *Sensors* 18, 7 (2018). DOI: https://doi.org/10.3390/s18072110

[116] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. 2018. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *45th International Symposium on Computer Architecture (ISCA'18)*. IEEE Press, 688–698.

[117] J. Park, S. Samarakoon, M. Bennis, and M. Debbah. 2019. Wireless network intelligence at the edge. *Proc. IEEE* 107, 11 (2019), 2204–2239.

[118] David Patterson and Andrew Waterman. 2017. *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon LLC.

[119] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. 2013. A survey of methods for distributed machine learning. *Prog. Artif. Intell.* 2, 1 (01 Mar. 2013), 1–11.

[120] K. Pradeep, K. Kamalavasan, R. Natheesan, and A. Pasqual. 2018. EdgeNet: SqueezeNet like convolution neural network on embedded FPGA. In *25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 81–84. DOI: https://doi.org/10.1109/ICECS.2018.8617876

[121] MD Abdur Rahman, M. Shamim Hossain, George Loukas, Elham Hassanain, Syed Sadiqur Rahman, Mohammed F. Alhamid, and Mohsen Guizani. 2018. Blockchain-based mobile edge computing framework for secure therapy applications. *IEEE Access* 6 (2018), 72469–72478.

[122] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. arXiv:https://arxiv.org/abs/1804.02767.

[123] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. Association for Computing Machinery, New York, NY, 1135–1144. DOI: https://doi.org/10.1145/2939672.2939778

[124] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, and Man Wang. 2019. Glow: Graph Lowering Compiler Techniques for Neural Networks. arXiv:https://arxiv.org/abs/1805.00907.

[125] K. Rungsuptaweekoon, V. Visoottiviseth, and R. Takano. 2017. Evaluating the power efficiency of deep learning inference on embedded GPU systems. In *2nd International Conference on Information Technology (INCIT)*. 1–5. DOI: https://doi.org/10.1109/INCIT.2017.8257866

[126] M. Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (Jan. 2017), 30–39. DOI: https://doi.org/10.1109/MC.2017.9

[127] M. Satyanarayanan and N. Davies. 2019. Augmenting cognition through edge computing. *Computer* 52, 7 (July 2019), 37–46.

[128] Ragini Sharma, Saman Biookaghazadeh, and Ming Zhao. 2018. Are existing knowledge transfer techniques effective for deep learning on edge devices? In *27th International Symposium on High-performance Parallel and Distributed Computing (HPDC'18)*. ACM, New York, NY, 15–16.

[129] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet Things J.* 3 (2016), 637–646.

[130] W. Shi and S. Dustdar. 2016. The promise of edge computing. *Computer* 49, 5 (May 2016), 78–81. DOI: https://doi.org/10.1109/MC.2016.145

[131] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang. 2020. DRQ: Dynamic region-based quantization for deep neural network acceleration. In *ACM/IEEE 47th International Symposium on Computer Architecture (ISCA)*. 1010–1021.

[132] Flávio Souza, Diego de Las Casas, Vinícius Flores, SunBum Youn, Meeyoung Cha, Daniele Quercia, and Virgílio Almeida. 2015. Dawn of the Selfie Era: The Whos, Wheres, and Hows of Selfies on Instagram. arXiv:https://arxiv.org/abs/1510.05700.

[133] IBM Research Editorial Staff. 2017. IBM scientists team with The Weather Company to bring edge computing to life. Retrieved from https://www.ibm.com/blogs/research/2017/02/bringing-edge-computing-to-life/.

[134] Rafael Stahl, Zhuoran Zhao, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. 2019. Fully distributed deep learning inference on resource-constrained edge devices. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Dionisios N. Pnevmatikatos, Maxime Pelcat, and Matthias Jung (Eds.). Springer International Publishing, Cham, 77–90.

[135] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. *Arxiv Eprints* 1905.11946 (2019).

[136] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang. 2017. Incorporating intelligence in fog computing for big data analysis in smart cities. *IEEE Trans. Industr. Inform.* 13, 5 (Oct. 2017), 2140–2150.

[137] Jiaxi Tang, Rakesh Shivanna, Zhe Zhao, Dong Lin, Anima Singh, Ed H. Chi, and Sagar Jain. 2020. Understanding and improving knowledge distillation. *Arxiv Eprints* 2002.03532 (2020).

[138] Zeyi Tao and Qun Li. 2018. eSGD: Communication efficient distributed deep learning on the edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*. USENIX Association. https://www.usenix.org/conference/hotedge18/presentation/tao.

[139] GreenWaves Technologies. 2018. GAP8 - GreenWaves. Retrieved from https://en.wikichip.org/wiki/greenwaves/gap8.

[140] GreenWaves Technologies. 2019. TF2GAP8. Retrieved from https://github.com/GreenWaves-Technologies/tf2gap8.

[141] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 328–339. DOI: https://doi.org/10.1109/ICDCS.2017.226

[142] Tom Simonite. 2019. The best algorithms struggle to recognize black faces equally. Retrieved from https://www.wired.com/story/best-algorithms-struggle-recognize-black-faces-equally/.

[143] Shreshth Tuli, Nipam Basumatary, and Rajkumar Buyya. 2019. EdgeLens: Deep Learning based Object Detection in Integrated IoT, Fog and Cloud Computing Environments. arXiv:https://arxiv.org/abs/1906.11056.

[144] S. Ullah and D. Kim. 2020. Benchmarking Jetson platform for 3D point-cloud and hyper-spectral image classification. In *IEEE International Conference on Big Data and Smart Computing (BigComp)*. 477–482.

[145] Sahar Voghoei, Navid Hashemi Tonekaboni, Jason G. Wallace, and Hamid Reza Arabnia. 2018. Deep learning at the edge. In *International Conference on Computational Science and Computational Intelligence (CSCI)*. 895–901.

[146] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. 2017. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 36, 3 (Mar. 2017), 513–517.

[147] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan. 2018. Bandwidth-efficient live video analytics for drones via edge computing. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 159–173. DOI: https://doi.org/10.1109/SEC.2018.00019

[148] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin S. Chan. 2018. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE Conference on Computer Communications*. 63–71. DOI: https://doi.org/10.1109/INFOCOM.2018.8486403

[149] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* 22, 2 (2020), 869–904.

[150] Sally Ward-Foxton. 2019. AI at the Very, Very Edge. Retrieved from https://www.eetimes.com/document.asp?doc_id=1334918.

[151] Pete Warden. 2018. Speech Commands: A Dataset for Limited-vocabulary Speech Recognition. arXiv:https://arxiv.org/abs/1804.03209.

[152] Matt Welsh. 2019. True AI on a Raspberry Pi, with no extra hardware. Retrieved from https://medium.com/@mdwdotla/true-ai-on-a-raspberry-pi-with-no-extra-hardware-dcdbff12d068.

[153] R. Xu, S. Y. Nikouei, Y. Chen, A. Polunchenko, S. Song, C. Deng, and T. R. Faughnan. 2018. Real-time human objects tracking for smart surveillance at the edge. In *IEEE International Conference on Communications (ICC)*. 1–6. DOI: https://doi.org/10.1109/ICC.2018.8422970

[154] Zhuangdi Xu, Harshit Gupta, and Umakishore Ramachandran. 2018. STTR: A system for tracking all vehicles all the time at the edge of the network. In *12th ACM International Conference on Distributed and Event-based Systems (DEBS'18)*. ACM, New York, NY, 124–135.

[155] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. 2019. Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks. In *46th International Symposium on Computer Architecture (ISCA'19)*. Association for Computing Machinery, New York, NY, 236–249. DOI: https://doi.org/10.1145/3307650.3322271

[156] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek F. Abdelzaher. 2017. Compressing deep neural network structures for sensing systems with a compressor-critic framework. arXiv:1706.01215.

[157] Mahmut Taha Yazici, Shadi Basurra, and Mohamed Medhat Gaber. 2018. Edge machine learning: Enabling smart internet of things applications. *Big Data Cog. Comput.* 2, 3 (2018), 26.

[158] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *44th International Symposium on Computer Architecture (ISCA'17)*. Association for Computing Machinery, New York, NY, 548–560.

[159] Qunsong Zeng, Yuqing Du, Kin K. Leung, and Kaibin Huang. 2019. Energy-efficient Radio Resource Allocation for Federated Edge Learning. arXiv:https://arxiv.org/abs/1907.06040.

[160] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. 2019. Eager pruning: Algorithm and architecture support for fast training of deep neural networks. In *46th International Symposium on Computer Architecture (ISCA'19)*. Association for Computing Machinery, New York, NY, 292–303. DOI: https://doi.org/10.1145/3307650.3322263

[161] Jianhao Zhang, Yingwei Pan, Ting Yao, He Zhao, and Tao Mei. 2019. daBNN: A super fast inference framework for binary neural networks on arm devices. In *27th ACM International Conference on Multimedia*. 2272–2275. DOI: https://doi.org/10.1145/3343031.3350534

[162] Xingzhou Zhang, Yifan Wang, and Weisong Shi. 2018. pCAMP: Performance comparison of machine learning packages on the edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*. USENIX Association, Boston, MA. Retrieved from https://www.usenix.org/conference/hotedge18/presentation/zhang.

[163] X. Zhang, X. Zhou, M. Lin, and J. Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6848–6856. DOI: https://doi.org/10.1109/CVPR.2018.00716

[164] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. 2018. DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* PP (10 2018), 1–1. DOI: https://doi.org/10.1109/TCAD.2018.2858384

[165] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (Aug. 2019), 1738–1762.

[166] Guangxu Zhu, Dongzhu Liu, Yuqing Du, Changsheng You, Jun Zhang, and Kaibin Huang. 2018. Towards an Intelligent Edge: Wireless Communication Meets Machine Learning. arXiv:https://arxiv.org/abs/1809.00343.

Received October 2020; revised April 2021; accepted May 2021