Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing

Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, Junshan Zhang

Abstract—**With the breakthroughs in deep learning, the recent years have witnessed a booming of artificial intelligence (AI) applications and services, spanning from personal assistant to recommendation systems to video/audio surveillance. More recently, with the proliferation of mobile computing and Internet-of-Things (IoT), billions of mobile and IoT devices are connected to the Internet, generating zillions Bytes of data at the network edge. Driving by this trend, there is an urgent need to push the AI frontiers to the network edge so as to fully unleash the potential of the edge big data. To meet this demand, edge computing, an emerging paradigm that pushes computing tasks and services from the network core to the network edge, has been widely recognized as a promising solution. The resulted new inter-discipline, edge AI or edge intelligence, is beginning to receive a tremendous amount of interest. However, research on edge intelligence is still in its infancy stage, and a dedicated venue for exchanging the recent advances of edge intelligence is highly desired by both the computer system and artificial intelligence communities. To this end, we conduct a comprehensive survey of the recent research efforts on edge intelligence. Specifically, we first review the background and motivation for artificial intelligence running at the network edge. We then provide an overview of the overarching architectures, frameworks and emerging key technologies for deep learning model towards training/inference at the network edge. Finally, we discuss future research opportunities on edge intelligence. We believe that this survey will elicit escalating attentions, stimulate fruitful discussions and inspire further research ideas on edge intelligence.**

I. Introduction

We are living in an unprecedented booming era of artificial intelligence (AI). Driving by the recent advancements of algorithm, computing power and big data, deep learning [1] — the most dazzling sector of AI — has made substantial breakthroughs in a wide spectrum of fields, ranging from computer vision, speech recognition, natural language processing to chess playing (e.g., AlphaGo) and robotics [2]. Benefited from these breakthroughs, a set of intelligent applications, as exemplified by intelligent personal assistants, personalized shopping recommendation, video surveillance and smart home appliances have quickly ascended to the spotlight and gained enormous popularity. It is widely recognized that these intelligent applications are significantly enriching people's lifestyle, improving human productivity, and enhancing social efficiency.

As a key driver that boosts AI development, big data has recently gone through a radical shift of data source from the mega-scale cloud datacenters to the increasingly widespread end devices, e.g., mobile devices and IoT devices. Traditionally, big data, such as online shopping records, social media contents and business informatics, were mainly born and stored at mega-scale datacenters. However, with the proliferation of mobile computing and IoT, the trend is reversing now. Specifically, Cisco estimates that nearly 850 ZB will be generated by all people, machines, and things at the network edge by 2021 [3]. In sharp contrast, the global datacenter traffic will only reach 20.6 Zettabytes by 2021. Clearly, via bringing the huge volumes of data to AI, the edge ecosystem will present many novel application scenarios for AI and fuel the continuous booming of AI.

Pushing the AI frontier to the edge ecosystem that resides at the last mile of the Internet, however, is highly non-trivial, due to the concerns on performance, cost and privacy. Towards this goal, the conventional wisdom is to transport the data bulks from the IoT devices to the cloud datacenters for analytics [4]. However, when moving a tremendous amount of data across the wide-area-network (WAN), both monetary cost and transmission delay can be prohibitively high, and the privacy leakage can also be a major concern [5]. An alternative is on-device analytics that run AI applications on the device to process the IoT data locally, which, however, may suffer from poor performance and energy efficiency. This is because many AI applications require high computational power that greatly outweighs the capacity of resource- and energy-constrained IoT devices.

To address the above challenges, edge computing [6] has recently been proposed, which pushes cloud services from the network core to the network edges that are in closer proximity to IoT devices and data sources. As shown in Fig. 1, here an edge node can be nearby end-device connectable by device-to-device (D2D) communications [7], a server attached to an access point (e.g., WiFi, router, base station), a network gateway, or even a micro-datacenter available for use by nearby devices. While edge nodes can be varied in size: ranging from a credit-card-sized computer to a micro-datacenter with several server racks, physical proximity to the information-generation sources is the most crucial characteristic emphasized by edge computing. Essentially, the physical proximity between the computing and information-generation sources promises several benefits compared to the traditional cloud-based computing paradigm, including low-latency, energy-efficiency, privacy protection, reduced bandwidth consumption, on-premises and context-awareness [6], [8].

Indeed, the marriage of edge computing and AI has given rise to a new research area, namely "edge intelligence" or "edge AI" [9], [10]. Instead of entirely relying on the cloud, edge intelligence makes the most of the widespread edge

Z. Zhou, X. Chen, E. Li, L. Zeng and K. Luo are with the School of Data and Computer Science, Sun Yat-sen University (SYSU), Guangzhou 510006, China. E-mail: {zhouzhi9, chenxu35}@mail.sysu.edu.cn, {lien5, luok7, zenglk3}@mail2.sysu.edu.

J. Zhang is with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287-7206, USA. E-mail: Junshan.Zhang@asu.edu.

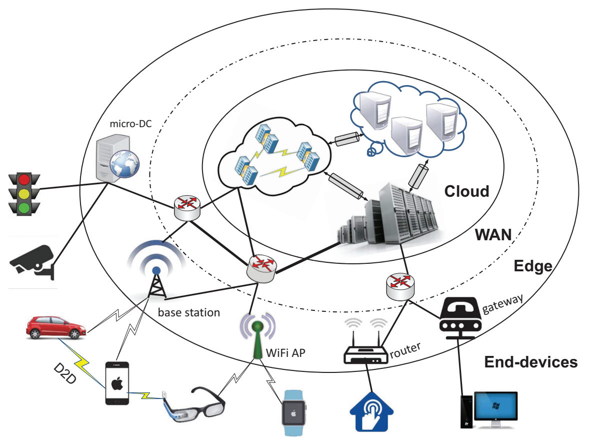


Fig. 1: An illustration of edge computing

resources to gain AI insight. Notably, edge intelligence has garnered much attention from both the industry and academia. For example, the celebrated Gartner hype cycle has incorporated edge intelligence as an emerging technology that will reach a plateau of productivity in the following 5 to 10 years [11]. Major enterprises, including Google, Microsoft, Intel and IBM, have put forth pilot projects to demonstrate the advantages of edge computing in paving the last mile of AI. These effort have boosted a wide spectrum of AI applications, spanning from live video analytics [12], cognitive assistance [13] to precision agriculture, smart home [14] and industrial internet-of-things (IIoT) [15].

Notably, research and practice on this emerging interdiscipline — edge intelligence, is still in a very early stage. There is,, in general a lack of venue dedicated for summarizing, discussing, and disseminating the recent advances of edge intelligence, in both industrial and academia. To bridge this gap, in this paper we conduct a comprehensive and concrete survey of the recent research efforts on edge intelligence. Specifically, we will first review the background of artificial intelligence. We will then discuss the motivation, definition and rating of edge intelligence. Next, we will further review and taxonomically summarize the emerging computing architectures and enabling technologies for edge intelligence model training and inference. Finally, we will discuss some open research challenges and opportunities on edge intelligence. The paper is organized as follows:

- Sec. II gives an overview of the basic concepts of artificial intelligence, with a focus on deep learning — the most popular sector of AI.
- Sec. III discusses the motivation, definition, and rating of edge intelligence.
- Sec. IV reviews the architectures, enabling techniques, systems and frameworks for training edge intelligence models.
- Sec. V reviews the architectures, enabling techniques, systems and frameworks for edge intelligence model inference.
- Sec. VI discusses future directions and challenges of edge intelligence.

For this survey, we hope it can elicit escalating attentions, stimulate fruitful discussions and inspire further research ideas on edge intelligence.

II. A Primer on Artificial Intelligence

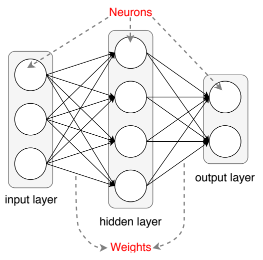
In this section, we review the concepts, models and methods for artificial intelligence, with a particular focus on deep learning — the most popular sector of artificial intelligence.

A. *Artificial Intelligence*

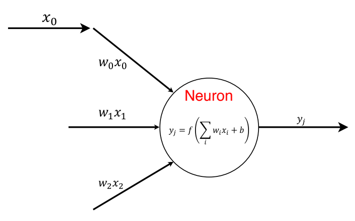
While AI has recently ascended to the spotlight and gained tremendous attention, it is not a new term and it was first coined in 1956. Simply put, AI is an approach to building intelligent machines capable of carrying out tasks as humans do. This is obviously a very broad definition, and it can refer from Apple Siri to Google AlphaGo and too powerful technologies yet to be invented. In simulating human intelligence, AI systems typically demonstrate at least some of the following behaviors associated with human intelligence: planning, learning, reasoning, problem-solving, knowledge representation, perception, motion, and manipulation and, to a lesser extent, social intelligence and creativity. During the past 60 year's development, AI has experienced rise, fall and again rise and fall. The latest rise of AI after 2010's was partially due to the breakthroughs made by deep learning, a method that has achieved human-level accuracy in some interesting areas.

B. *Deep Learning and Deep Neural Networks*

Machine learning (ML) is an effective method to achieve the goal of AI. Many machine learning methodologies as exemplified by decision tree, K-means clustering and Bayesian network, etc. have been developed to train the machine to make classifications and predictions, based on the data obtained from the real world. Among the existing machine learning methods, deep learning, by leveraging artificial neural network (ANN) [16] to learn the deep representation of the data, have resulted in an amazing performance in multiple tasks, including image classification, face recognition, etc. Since the ANN adopted by deep learning model typically consists of a series of layers, the model is called deep neural network (DNN). As shown in Fig. 2, each layer of a DNN is composed of neurons that are able to generate the non-linear outputs based on the data from the input of the neuron.



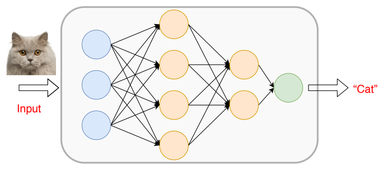
(a) The layers in a DL model



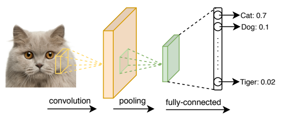
(b) The architecture of a neuron

Fig. 2: A standard composition of DL model

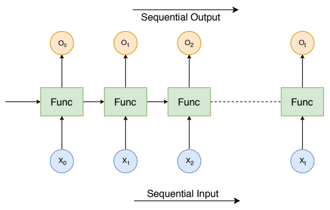
The neurons in the input layer receive the data and propagate them to the middle layer (a.k.a the hidden layer). Then the neurons in the middle layer generate the weighted sums of the input data and output the weighted sums using the specific activation functions (e.g., tanh), and the outputs are then propagated to the output layer. The final results are presented at the output layer. With more complex and abstract layers than a typical model, DNNs are able to learn the high-level features, enabling high precision inference in tasks. Fig. 3 presents three popular structures of DNNs: Multilayer Perceptrons (MLPs), Convolution Neural Network (CNN) and Recurrent Neural Network (RNN).



(a) Multilayer Perceptrons



(b) Convolution Neural Network



(c) Recurrent Neural Network

Fig. 3: Three typical structures of DL models

MLPs models are the most basic deep neural network, which is composed of a series of fully-connected layers [17]. Different from fully-connected layers in MLPs, in CNN models, the convolution layers extract the simple features from input by executing convolution operations. Applying various convolutional filters, CNN models can capture the high-level representation of the input data, making it most popular for computer vision tasks, e.g., image classification (e.g., AlexNet [18], VGG network [19], ResNet [20], MobileNet [21]) and object detection (e.g., Fast R-CNN [22], YOLO [23], SSD [24]). RNN models are another type of DNNs, which use sequential data feeding. Shown in Fig. 3(c), the basic unit of RNN is called cell; and further, each cell consists of layers and a series of cells enables the sequential processing of RNN models. RNN models are widely used in the task of natural language processing, e.g., language modeling, machine translation, question answering and document classification.

Deep learning represents the state-of-the-art AI technology as well as a highly resource-demanding workload that naturally suits for edge computing. Therefore, due to space limitation, in the remaining of this paper, we will focus on the interaction between deep learning and edge computing. We believe that the techniques discussed can also have meaningful implications for other AI models and methods, i.e., stochastic gradient descent is a popular training method for many AI/ML algorithms (e.g., k-means, support vector machine, lasso regression) [25], the optimization techniques of stochastic gradient descent training introduced in this paper can be also deployed on other AI models training process.

C. *From Deep Learning to Model Training and Inference*

For each neuron in a DNN layer, it has a vector of weights associated with the input data size of the layer. Needless to say, the weights in a deep learning model need to be optimized through a training process.

In a training process for a deep learning model, the values of weights in the model are often randomly assigned initially. Then the output of the last layer represents the task result, and a loss function is set to evaluate the correctness of the results by calculating the error rate (e.g., root mean squared error) between the results and the true label. To adjust the weights of each neuron in the model, an optimization algorithm, such as Stochastic Gradient Descent (SGD) [25], is used and the gradient of the loss function is calculated. Leveraging the Back Propagation mechanism [26], [27], the error rate is propagated back across the whole neural network and the weights are updated based on the gradient and the learning rate. By feeding a large number of training samples and repeating this process until the error rate is below a predefined threshold, a deep learning model with high precision is obtained.

DNN model inference happens after training. For instance, for an image classification task, with the feeding of a large amount of training samples, the deep neural network is trained to learn how to recognize an image, and then inference takes real-world images as inputs and quickly draws the predictions/classifications of them. The training procedure consists of the feed-forward process and the backpropagation process. Note that the inference involves the feed-forward process only, i.e., the input from real-world is passed through the whole neural network and the model outputs the prediction.

D. *Popular Deep Learning Models*

For a better understanding of the deep learning and their applications, in this subsection we give an overview of various popular deep learning models.

Convolution Neural Network (CNN): For image classification, as the first CNN to win the ImageNet Challenge in 2012, AlexNet [18] consists of 5 convolution layers and 3 fully-connected layers. AlexNet requires 61 million weights and 724 million MACs (Multiply-Add Computation) to classify the image with a size of 227*227. To achieve higher accuracy, VGG-16 [19] is trained to a deeper structure of 16 layers consisting of 13 convolution layers and 3 fully-connected layers, requiring 138 million weights and 15.5G MACs to classify the image with a size of 224*224. To improve accuracy while reducing the computation of DNN inference, GoogleNet [28] introduces an inception module composed of different sized filters. GoogleNet achieves a better accuracy performance than VGG-16, while only requiring 7 million weights and 1.43G MACs to process the image with the same size. ResNet [20] , the state-of-the-art effort, uses the "shortcut" structure to reach a human-level accuracy with a top-5 error rate below 5%. The "shortcut" module is used to solve the gradient vanishing problem during the training process, making it possible to train a DNN model with deeper structure. Convolution neural network typically employed in computer vision. Given a series of images or video from real-world, with the utilization of CNN, the AI system learns to automatically extract the features of these inputs to complete a specific task, e.g., image classification, face authentication, image semantic segmentation.

Recurrent Neural Network (RNN): For sequential input data, recurrent neural networks (RNNs) have been developed to address the time-series problem. The input of RNN consists of the current input and the previous samples. Each neuron in a RNN owns an internal memory that keeps the information of the computation from the previous samples. The training of RNN is based on Backpropagation Through Time (BPTT) [29]. Long Short Term Memory (LSTM) [30] is an extended version of RNNs. In LSTM, the gate is used to represents the basic unit of a neuron. As shown in Fig. 4, each neuron in LSTM is called memory cell and includes a multiplicative forget gate, input gate, and output gate. These gates are used to control the access to memory cells and to prevent them from perturbation by irrelevant inputs. Information is added or removed through the gate to the memory cell. Gates are different neural networks that determine what information is allowed on the memory cell. The forget gate can learn what information is kept or forgotten during training. Recurrent neural network has been widely used in natural language processing due to the superior of processing the data with an input length that is not fixed. The task of the AI here is to build a system that can comprehend natural language spoken by humans, e.g., natural language modeling, word embedding, machine translation.

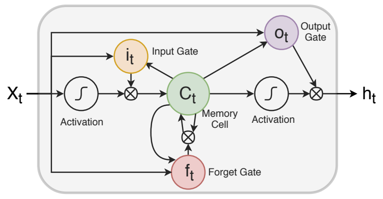


Fig. 4: Structure of a LSTM memory cell

Generative Adversarial Network (GAN): Shown in Fig. 5, generative adversarial networks (GANs) [31] consists of two main components, namely the generative and discriminator network (i.e., generator and discriminator). The generator is responsible for generating new data after it learns the data distribution from a training dataset of real data. The discriminator is in charge of classifying the real data from the fake data generated by the generator. GAN is often deployed in image generation, image transformation, image synthesis, image super-resolution and other applications.

Deep Reinforcement Learning (DRL): Deep Reinforcement Learning (DRL) is composed of DNNs and reinforcement learning (RL). The goal of DRL is to create an intelligent agent that can perform efficient policies to maximize the rewards of long-term tasks with controllable actions. The typical application of DRL is to solve various scheduling problems, such as decision problems in games, rate selection of video transmission, etc.

In the DRL approach, the reinforcement learning searches for the optimal policy of actions over states from the environment, and the DNN is in charge of representing a large number of states and approximating the action values to estimate the quality of the action in the given states. The reward is a function to represent the distance between the predefined requirement and the performance of an action. Through continuous learning, the agent of DRL model can be used for various tasks, e.g., gaming [32].

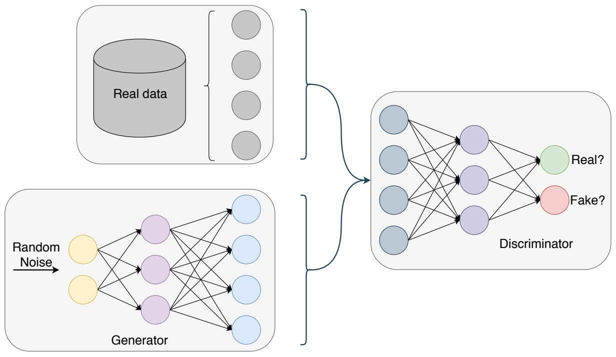


Fig. 5: Composition of a generative adversarial network

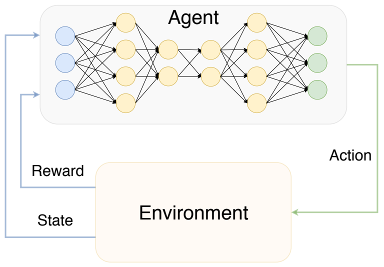


Fig. 6: Concept of a deep reinforcement learning model

III. Edge Intelligence

The marriage of edge computing and artificial intelligence gives the birth of edge intelligence. In this section, we discuss the motivation, benefits and definition of edge intelligence.

A. Motivation and Benefits of Edge Intelligence

The fusion of AI and edge computing is natural, since there is a clear intersection between them. Specifically, edge computing aims at coordinating a multitude of collaborative edge devices and servers to process the generated data in proximity; and AI strives for simulating intelligent human behavior in devices/machines by learning from data. Besides enjoying the general benefits of edge computing (e.g., low-latency, reduced bandwidth consumption), pushing AI to the edge further benefits each other in the following aspects.

On one hand, data generated at the network edge need AI to fully unlock their potential. As a result of the proliferation of the skyrocketing number and types of mobile and IoT devices, large volumes of multi-modal data (e.g., audio, picture and video) of physical surroundings are continuously sensed at the device side. In this context, AI will be functionally necessary due to its ability to quickly analyze those huge data volumes and extract insights from them for high-quality decision making. As one of the most popular AI techniques, deep learning brings the ability to automatically identify patterns and detect anomalies in the data sensed by the edge device, as exemplified by population distribution, traffic flow, humidity, temperature, pressure and air quality. The insights extracted from the sensed data are then fed to the real-time predictive decision-making (e.g., public transportation planning, traffic control and driving alert) in response to the fast-changing environments, increasing the operational efficiency. As forecasted by Gartner [33], more than 80 percent of enterprise IoT projects will include an AI component by 2022, up from only 10 percent today.

On the other hand, edge computing is able to prosper AI with richer data and application scenarios. It is widely recognized that the driving force behind the recent booming of deep learning is four-folds: algorithm, hardware, data and application scenarios. While the effect of algorithm and hardware on the development of deep learning is intuitive, the role of data and application scenarios have been mostly overlooked. Specifically, to improve the performance of a deep learning algorithm, the most commonly adopted approach is to refine the DNN with more layers of neurons. By doing this, we need to learn more parameters in the DNN, and so does the data required for training increase. This definitely demonstrates the importance of data on the development of AI. Having recognized the importance of data, the next problem is, where is the data from. Traditionally, data is mostly born and stored in the mega-scale datacenters. Nevertheless, with the rapid development of IoT, the trend is reversing now. According to Cisco's report [3], in the near future, massive IoT data will be generated at the edge side. If these data are processed by AI algorithms at the cloud data center, it will consume a lot of bandwidth resources and bring great pressure to the cloud data center. To address these challenges, edge computing is proposed to achieve low latency data processing by sinking the computing capability from the cloud data center to the edge side, i.e., data generation source, which may enable AI processing with high performance.

While edge computing and AI complement each other from a technical perspective, their application and popularization are also mutually beneficial.

On one hand, AI democratization requires edge computing as a key infrastructure. AI technologies have witnessed great success in many digital products or services in our daily life, e.g., online shopping, service recommendation, video surveillance, smart home devices, etc. AI is also a key driving force behind emerging innovative frontiers, such as self-driving cars, intelligent finance, cancer diagnosis and medicine discovery. Beyond the above examples, to enable a richer set of applications and push the boundaries of what's possible, AI democratization or ubiquitous AI [34] has been declared by major IT companies, with the vision of "making AI for every person and every organization at everywhere". To this end, AI should go 'closer' to the people, data and end devices. Clearly, edge computing is more competent than cloud computing in achieving this goal. Firstly, compared to the cloud datacenter, edge servers are in closer proximity to people, data source and devices. Secondly, compared to cloud computing, edge computing is also more affordable and accessible. Finally, edge computing has the potential to provide more diverse application scenarios of AI than cloud computing. Due to these advantages, edge computing is naturally a key enabler for ubiquitous AI.

On the other hand, edge computing can be popularized with AI applications. During the early development of edge computing, there has always been the concern in the cloud computing community with which high-demand applications edge computing could take to the next level that cloud computing could not, and what are the killer applications of edge computing. To clear up the doubt, Microsoft has conducted continuous exploration on what kinds should be moved from the cloud to the edge since 2009 [35], ranging from voice command recognition, AR/VR and interactive cloud gaming [36] to real-time video analytics. By comparison, real-time video analytics is envisioned to be a killer application for edge computing [12], [37], [38]. As an emerging application built on top of computer vision, real-time video analytics continuously pulls high-definition videos from surveillance cameras and requires high computation, high bandwidth, high privacy and low-latency to analyze the videos. The one viable approach that can meet these strict requirements is edge computing. Looking back to the above evolution of edge computing, it can be foreseen that novel AI applications emerged from the sectors such as industrial IoT, intelligent robots, smart cities and smart home will play a crucial role in the popularization of edge computing. This is mainly due to the fact that many mobile and IoT related AI applications represent a family of practical applications that are computation- and energy-intensive, privacy- and delay- sensitive, and thus naturally align well with edge computing.

Due to the superiority and necessity of running AI application on the edge, edge AI has recently received great attention. In December 2017, in a white paper, "A Berkeley View of Systems Challenges for AI" [39] published by UC Berkeley, the cloud-edge AI system is envisioned as an important research direction to achieve the goal of mission-critical and personalized AI. In August 2018, edge AI emerges in the Gartner Hype Cycle for the first time [40]. According to Gartners prediction, edge AI is still in the innovation trigger phase, and it will reach a plateau of productivity in the following 5 to 10 years. In the industry, many pilot projects have also been carried out towards edge AI. Specifically, on the edge AI service platform, the traditional cloud providers, such as Google, Amazon and Microsoft, have launched service platforms to bring the intelligence to the edge, through enabling end devices to run ML inferences with pre-trained models locally. On edge AI chips, various high-end chips designated for running ML models have been made commercially available on the market, as exemplified by Google Edge TPU, Intel Nervana NNP, Huawei Ascend 910 & Ascend 310.

B. Scope and Rating of Edge Intelligence

While the term edge AI or edge intelligence is brand new, explorations and practices in this direction have begun early. As aforementioned, in 2009, to demonstrate the benefits of edge computing, Microsoft has built an edge-based prototype

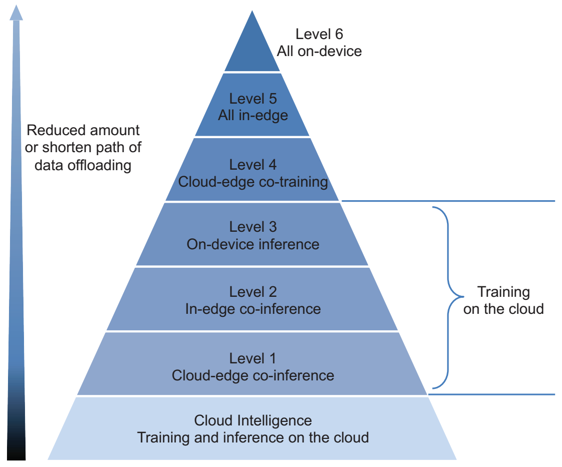


Fig. 7: A 6-level rating for edge intelligence

to support mobile voice command recognition, an AI application. Albeit the early begin of exploration, there is still not a formal definition for edge intelligence.

Currently, most organizations [41], [42] and presses [43] refer to edge intelligence as the paradigm of running AI algorithms locally on an end device, with data (sensor data or signals) that are created on the device. While this represents the current most common approach (e.g., with high-end AI chips) towards edge intelligence in the real world, it is crucial to note that this definition greatly narrows down the solution scope of edge intelligence. Running computation intensive algorithms as exemplified by DNN models locally is very resource-intensive, requiring high-end processors to be equipped in the device. Such stringent requirement not only increases the cost of edge intelligence but is also incompatible and unfriendly to existing legacy end devices that have limited computing capacities.

In this paper, we submit that the scope of edge intelligence should not be restricted to running AI models solely on the edge server or device. In fact, as demonstrated by a dozen recent studies, for DNN models, running them with edge-cloud synergy can reduce both the end-to-end latency and energy consumption, when compared to the local execution approach. Due to these practical advantages, we believe that such collaborative hierarchy should be integrated into the design of efficient edge intelligence solutions. Further, existing thoughts on edge intelligence mainly focus on the inference phase (i.e., running the AI model), assuming that the training of the AI model is performed in the power cloud datacenters, since the resource consumption of the training phase significantly overweights the inference phase. However, this means that the enormous amount of training data should be shipped from devices or edges to the cloud, incurring prohibitive communication overhead as well as the concern on data privacy.

Instead, we believe that edge intelligence should be the paradigm that fully exploits the available data and resources across the hierarchy of end devices, edge nodes and cloud datacenters to optimize the overall performance of training and inferencing a DNN model. This indicates that edge intelligence does not necessarily mean that the DNN model is fully trained or inferenced at the edge, but can work in a cloud-edge-device coordination manner via data offloading. Specifically, according to the amount and path length of data offloading, we rate edge intelligence into 6 levels, as shown in Fig. 7. Specifically, the definition of various levels of edge intelligence is given as follows:

- Cloud Intelligence: training and inferencing the DNN model fully in the cloud.
- Level-1 – Cloud-Edge Co-Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in an edge-cloud cooperation manner. Here edge-cloud cooperation means that data is partially offloaded to the cloud.
- Level-2 – In-Edge Co-Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in an in-edge manner. Here in-edge means that the model inference is carried out within the network edge, which can be realized by fully or partially offloading the data to the edge nodes or nearby devices (via D2D communication).
- Level-3 – On-Device Inference and Cloud Training: training the DNN model in the cloud, but inferencing the DNN model in a fully local on-device manner. Here on-device means that no data would be offloaded.
- Level-4 – Cloud-Edge Co-Training & Inference: training and inferencing the DNN model both in the edge-cloud cooperation manner.
- Level-5 – All In-Edge: training and inferencing the DNN model both in the in-edge manner.
- Level-6 – All On-Device: training and inferencing the DNN model both in the on-device manner.

As the level of edge intelligence goes higher, the amount and path length of data offloading reduce. As a result, the transmission latency of data offloading decreases, the data privacy increases and the WAN bandwidth cost reduces. However, this is achieved at the cost of increased computational latency and energy consumption. This conflict indicates that there is no "best-level" in general; instead, the "best-level" edge intelligence is application-dependent and it should be determined by jointly considering multi-criteria such as latency, energy efficiency, privacy and WAN bandwidth cost. In the later sections, we will review enabling techniques as well as existing solutions for different levels of edge intelligence.

IV. Edge Intelligence Model Training

With the proliferation of mobile and IoT devices, data which is essential for AI model training is increasingly generated at the network edge. In this section, we focus on distributed training of DNN at the edge, including the architectures, key performance indicators, enabling techniques and existing systems & frameworks.

A. Architectures

The architectures of distributed DNN training at the edge can be divided into three modes, Centralized, Decentralized, Hybrid (Cloud-Edge-Device). Fig. 8 shows the three architectures, illustrated by subfigures (a), (b) and (c), respectively.

The cloud refers to the central datacenter whereas the end devices are represented by mobile phones, cars and surveillance cameras, which are also data sources. For the edge server, we use base stations as the legend.

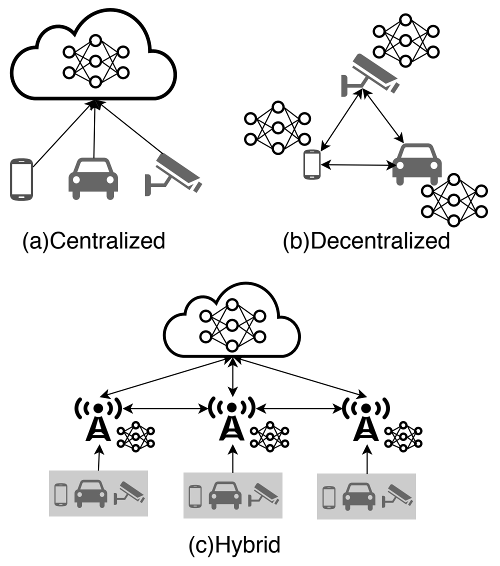


Fig. 8: The architecture modes of distributed training

1) Centralized: Fig 8(a) describes a centralized DNN training, where the DNN model is trained in the cloud datacenter. The data for training is generated and gathered from distributed end devices such as mobile phones, cars and surveillance cameras. Once the data arrived, the cloud datacenter will perform DNN training using these data. Therefore, the system based on the centralized architecture can be identified in Cloud Intelligence, Level-1, Level-2 or Level-3 in Fig. 7 according to the specific inference mode that the system employs.

2) Decentralized: Under the decentralized mode as shown in 8(b), each computing node trains its own DNN model locally with local data, which preserves private information locally. To obtain the global DNN model by sharing local training improvement, nodes in the network will communicate with each other to exchange the local model updates. In this mode, the global DNN model can be trained without the intervention of the cloud datacenter, corresponding to the Level-5 edge intelligence defined in Fig. 7.

3) Hybrid: The hybrid mode combines the centralized mode and the decentralized mode. As shown in Fig. 8(c), as the hub of the architecture, the edge servers may train the DNN model by either decentralized updates with each other or centralized training with the cloud datacenter, thus the hybrid architecture covers Level-4 and Level-5 in Fig. 7. The hybrid architecture is also called as Cloud-Edge-Device training due to the involved roles.

B. Key Performance Indicators

To better assess a distributed training method, there are six key performance indicators.

1) Training Loss: Essentially, the DNN training process solves an optimization problem that seeks to minimize the training loss. Since the training loss captures the gap between the learned (e.g., predicted) value and the labeled data, it indicates how well the trained DNN model fits the training data. Therefore, it is expected that the training loss can be minimized. Training loss is mainly affected by training samples and training methods.

2) Convergence: The convergence indicator is specialized for the decentralized methods. Intuitively, a decentralized method works only if the distributed training processes converge to a consensus, which is the training result of the method. The term convergence measures whether and how fast a decentralized method converges to such a consensus. Under the decentralized training mode, the convergence value depends on the way the gradient is synchronized and updated.

3) Privacy: When training the DNN model by using the data originated at a massive of end devices, the raw data or intermediate data should be transferred out of the end devices. Obviously, it is inevitable to deal with privacy issues in this scenario. To preserve privacy, it is expected that less privacy-sensitive data is transferred out of the end-devices. Whether privacy protection is implemented depends on whether the raw data is offloaded to the edge.

4) Communication Cost: Training the DNN model is data-intensive, since the raw data or intermediate data should be transferred across the nodes. Intuitively, this communication overhead increases the training latency, energy and bandwidth consumption. Communication overhead is affected by the size of the original input data, the way of transmission and the available bandwidth.

5) Latency: Arguably, latency is one of the most fundamental performance indicators of distributed DNN model training, since it directly influences when the trained model is available for use. The latency of the distributed training process typically consists of the computation latency and the communication latency. The computation latency is tightly dependent on the capability of the edge nodes. The communication latency may vary from the size of transmitted raw or intermediate data, and the bandwidth of network connection.

6) Energy Efficiency: When training the DNN model in a decentralized manner, both the computation and communication process consume enormous energy. However, for most end-devices, they are energy-constrained. As a result, it is highly desirable that the DNN model training is energy-efficient. Energy efficiency is mainly affected by the size of the target training model and resources of the used devices.

It is worth noting that the performance indicators training loss and convergence are common objectives, thus they may not be explicitly claimed by some literature on DNN training.

C. Enabling Technologies

In this subsection, we review the enabling technologies for improving one or more of the aforementioned key performance indicators when training edge intelligence model. Table I summarizes the highlights of each enabling technology.

1) Federated Learning: Federated learning is dedicated to optimizing privacy issue in the above key performance indicators. Federated learning is an emerging yet promising approach to preserve privacy when training the DNN model based on

TABLE I: Technologies for distributed DNN training at the edge

Technology	Highlights	Related Work
Federated Learning	• Leave the training data distributed on the end devices • Train the shared model on the server by aggregating locally-computed updates • Preserve privacy	[44]–[49]
Aggregation Frequency Control	• Determine the best trade-off between local update and global parameter aggregation under a given resource budget • Intelligent communication control	[50]–[52]
Gradient Compression	• Gradient quantization by quantizing each element of gradient vectors to a finite-bit low precision value • Gradient sparsification by transmitting only some values of the gradient vectors	[53]–[57]
DNN Splitting	• Select a splitting point to reduce latency as much as possible • Preserve privacy	[58]–[61]
Knowledge Transfer Learning	• First train a base network (teacher network) on a base dataset and task and then transfer the learned features to a second target network (student network) to be trained on a target dataset and task • The transition from generality to specificity	[59], [60], [62]
Gossip Training	• Random gossip communication among devices • Full asynchronization and total decentralization • Preserve privacy	[63]–[66]

data originated by multiple clients. Rather than aggregating the raw data to a centralized datacenter for training, federated learning [45] leaves the raw data distributed on the clients (e.g., mobile devices), and trains a shared model on the server by aggregating locally-computed updates. The main challenges of federated learning are optimization and communication.

For the optimization problem, the challenge is to optimize the gradient of a shared model by the distributed gradient updates on mobile devices. On this issue, federated learning adopts stochastic gradient descent (SGD). SGD updates the gradient over extremely small subsets (mini-batch) of the whole dataset, which is a simple but widely-used gradient descent method. Shokri et al. [46] design a selective stochastic gradient descent (SSGD) protocol, allowing the clients to train independently on their own datasets and selectively share small subsets of their models key parameters to the centralized aggregator. Since SGD is easy to be parallelized as well as asynchronously executed, SSGD targets both privacy and training loss. Specifically, while preserving clients own privacy, the training loss can be reduced by sharing the models among clients, comparing to training solely on their own inputs. A flaw of [46] is that it does not consider unbalanced and non-IID (none Independent Identical Distribution) data. As an extension, McMahan et al. [45] advocate a decentralized approach, termed as federated learning, and present FedAvg method for federated learning with the deep neural network based on iterative model averaging. Here iterative model averaging means that the clients update the model locally with one-step SGD and then the server averages the resulting models with weights. The optimization on [45] emphasizes the properties of unbalanced and non-IID since the distributed data may come from various sources.

For the communication problem, it is the unreliable and unpredictable network that poses the challenge of communication efficiency. In federated learning, each client sends a full model or a full model update back to the server in a typical round. ~~For large models, th~~is step is likely to be the bottleneck due to the unreliable network connections. To decrease the number of rounds for training, Mcmahan et al. [45] propose to increase the computation of local updates on clients. However, it is impractical when the clients are under ~~severe computation resources constraint.~~ In response to this issue, Konečný et al. [47] propose to reduce communication cost with two new update schemes, namely structured update and sketched update. In a structured update, the model directly learns an update from a restricted space parametrized using a smaller number of variables, e.g. either low-rank or a random mask. If using a sketched update, the model first learns a full model update and then compressed the update using a combination of quantization, random rotations, and subsampling before sending it to the server.

Though Federated Learning technique exploits a new decentralized deep learning architecture, it is built upon a central server for aggregating local updates. Considering the scenario of training a DNN model over a fully decentralized network, i.e., a network without a central server, Lalitha et al. [48] propose a Bayesian-based distributed algorithm, in which each device updates its belief by aggregating information from its one-hop neighbors to train a model that best fits the observations over the entire network. Furthermore, with the emerging blockchain technique, Kim et al. [49] propose Blockchain Federated Learning (BlockFL) with the devices model update exchanged and verified by leveraging blockchain. BlockFL also works for a fully decentralized network, where machine learning model can be trained without any central coordination, even when some devices lack their own training data samples.

2) Aggregation Frequency Control: This method focuses on the optimization of communication overhead during the DNN model training. On training deep learning model in edge computing environment, a commonly adopted idea (e.g., federated learning) is to train distributed models locally first, and then aggregate updates centrally. In this case, the control of updates aggregation frequency significantly influences the communication overhead. Thus, the aggregation process, including aggregation content as well as aggregation frequency, should be controlled carefully.

Based on the above insight, Hsieh et al. [50] develop Gaia system and the Approximate Synchronous Parallel (ASP) model for geo-distributed DNN model training. The basic idea of Gaia is to decouple the communication within a datacenter from the communication between datacenters, enabling different communication and consistency models for each. To this end, the ASP model is developed to dynamically eliminate insignificant communication between datacenters, where the aggregation frequency is controlled by the preset significance threshold. However, Gaia focuses on geo-distributed datacenters that are capacity-unlimited, making it is not generally applicable to edge computing nodes whose capacity is highly constrained.

To incorporate the capacity constraint of edge nodes, Wang et al. [51] propose a control algorithm that determines the best trade-off between local update and global parameter aggregation under a given resource budget. The algorithm is based on the convergence analysis of distributed gradient descent and can be applied to federated learning in edge computing with provable convergence. To implement federated learning in the capacity-limited edge computing environment, Nishio et al. [52] study the client selection problem with resource constraints. In particular, an update aggregation protocol named FedCS is developed to allow the centralized server to aggregate as many client updates as possible and to accelerate performance improvement in machine learning models. An illustration of FedCS is shown in Fig. 9.

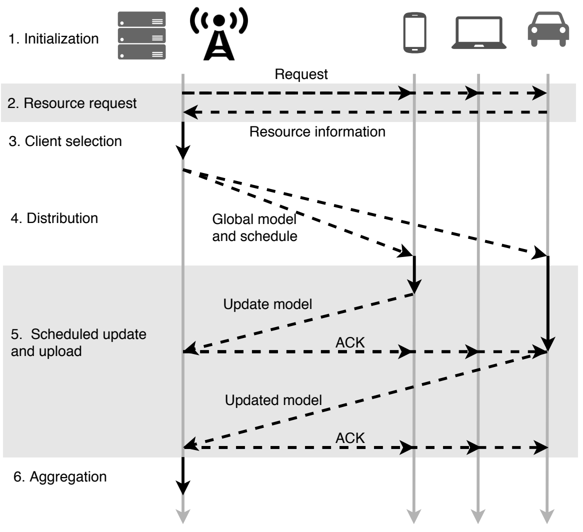


Fig. 9: Overview of FedCS protocol

3) Gradient Compression: To reduce ~~the communication overhead~~ incurred by decentralized training, gradient compression is another intuitive approach to compress the model update (i.e., gradient information). To this end, gradient quantization and gradient sparsification have been advocated. Specifically, gradient quantization performs lossy compression of the gradient vectors by quantizing each of their elements to a finite-bit low precision value. Gradient sparsification reduces the communication overhead by transmitting part of the gradient vectors.

Lin et al. [53] observe that 99.9% of the gradient exchange in distributed SGD are redundant, which demonstrates the power of gradient compression. Based on this observation, Lin et al. propose Deep Gradient Compression (DGC), which compresses the gradient by 270-600$\times$ for a wide range of CNNs and RNNs. To preserve accuracy during this compression, DGC employs four methods: momentum correction, local gradient clipping, momentum factor masking, and warm-up training.

Inspired by the above work [53], Tao et al. [54] propose Edge Stochastic Gradient Descent (eSGD), a family of sparse schemes with both convergence and practical performance guarantees. To improve the first order gradient-based optimization of stochastic objective functions in edge computing, eSGD includes two mechanisms: (1) determine which gradient coordinates are important and only transmits these coordinates; (2) design momentum residual accumulation for tracking out-of-date residual gradient coordinates in order to avoid low convergence rate caused by sparse updates. A concise convergence analysis of sparsified SGD is given in [55], where SGD are analyzed with k-sparsification or compression (e.g., top-k or random-k). The analysis shows that this scheme converges at the same rate as vanilla SGD when equipped with error compensation (keeping track of accumulated errors in memory). In other words, communication can be reduced by a factor of the dimension of the problem (sometimes even more) whilst still converging at the same rate.

Quantizing the gradients to low-precision values can also reduce the communication bandwidth. In this regard, Tang et al. [56] develop a framework of compressed, decentralized training and proposes two different algorithms, called extrapolation compression and difference compression respectively. The analysis on the two algorithms proves that both converge at the rate of $O(1/\sqrt{nT})$ where n is the number of clients and T is the number of iterations, matching the convergence rate for full precision, centralized training. Amiri et al. [57] implement distributed stochastic gradient descent (DSGD) at the wireless edge with the help of a remote parameter server. Besides, Amiri et al. further develop DSGD in digital and analog scheme respectively. Digital DSGD (D-DSGD) assumes that the clients operate on the boundary of the Multiple Access Channel (MAC) capacity region at each iteration of DSGD algorithm, and employs gradient quantization and error accumulation to transmit their gradient estimates within the bit budget allowed by the employed power allocation. In Analog DSGD (A-DSGD), the clients first sparsify their gradient estimates with error accumulation and then project them to a lower dimensional space imposed by the available channel bandwidth. These projections are transmitted directly over the MAC without employing any digital code.

4) DNN Splitting: The aim of DNN splitting is to protect privacy. DNN splitting protects user privacy by transmitting partially processed data rather than transmitting raw data. To enable a privacy-preserving edge-based training of DNN models, DNN splitting is conducted between the end devices and the edge server. This bases on the important observation that a DNN model can be split inside between two successive layers with two partitions deployed on different locations without losing accuracy.

An inevitable problem on DNN splitting is how to select the splitting point such that distributed DNN training is still under the latency requirement. On this problem, Mao et al. [58] utilize the differentially private mechanism and partitions DNN after the first convolutional layer to minimize the cost of mobile devices. The proof in [58] guarantees that applying differentially private mechanism on activations is feasible for outsourcing training tasks to untrusted edge servers. Wang et al. [59] consider this problem across mobile devices and cloud datacenters. To benefit from the computation power of cloud datacenters without privacy risks, Wang et al. design Arden (privAte infeRence framework based on Deep nEural Networks), a framework which partitions the DNN model with a lightweight privacy-preserving mechanism. By arbitrary data nullification and random noise addition, Arden achieves privacy protection. Considering the negative impact of private perturbation to the original data, Wang et al. use a noisy training method to enhance the cloud-side network robustness to perturbed data.

Osia et al. [60] introduce a hybrid user-cloud framework on the privacy issue, which utilizes a private-feature extractor as its core component and breaking down large, complex deep models for cooperative, privacy-preserving analytics. In this framework, the feature extractor module is properly designed to output the private feature constrained to keeping the primary information while discarding all the other sensitive information. Three different techniques are employed to make sensitive measures unpredictable: dimensionality reduction, noise addition, and Siamese fine-tuning.

When applying DNN splitting for privacy-preserving, it is remarkable that this technique also works for dealing with the tremendous computation of DNN. Exploiting the fact that edge computing usually involves a large number of devices, parallelization approaches is usually employed to manage DNN computation. DNN training in parallel includes two kinds of parallelism, data parallelism and model parallelism. However, data parallelism may bring heavy overhead of communication while model parallelism usually leads to severe under-utilization of computation resources. To address these problems, Harlap et al. [61] propose pipeline parallelism, an enhancement to model-parallelism, where multiple mini-batches are injected into the system at once to ensure efficient and concurrent use of computation resources. Based on pipeline parallelism, Harlap et al. design PipeDream, a system which supports pipelined training, and automatically determine how to systematically split a given model across the available computing nodes. PipeDream shows the advantage of reducing communication overhead and utilizing computing resource efficiently. The overview of PipeDream's automated mechanism is in Fig. 10.

5) Knowledge Transfer Learning: Knowledge transfer learning, or transfer learning for simplicity, is closely connected with DNN splitting technique. In transfer learning, for the purpose of reducing DNN model training energy cost on edge devices, we first train a base network (teacher network)

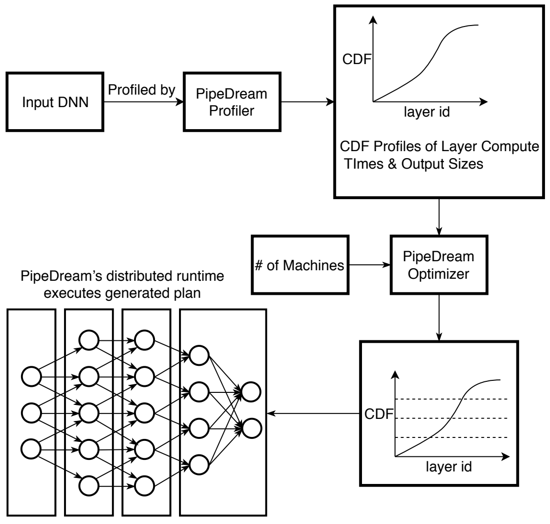

Input DNN
Profiled by
PipeDream Profiler
CDF
layer id
CDF Profiles of Layer Compute TImes & Output Sizes
# of Machines
PipeDream Optimizer
PipeDream's distributed runtime executes generated plan
CDF
layer id

Fig. 10: PipeDreams automated mechanism

on a base dataset, and then we repurpose the learned features, i.e., transfer them to a second target network (student network) to be trained on a target dataset. This process will tend to work if the features are general (i.e., suitable to both base and target tasks) instead of specific to the base task. The transition involves a process from generality to specificity.

The approach of transfer learning seems to be promising for learning on edge devices since it has greatly reduced resource demand, but a thorough investigation on its effectiveness is lacking. To bridge this gap, Sharma et al. [62] and Chen et al. [44] provide extensive studies on the performance (in both accuracy and convergence speed) of Transfer Learning, considering different student network architectures and different techniques for transferring knowledge from teacher to student. The result varies with architectures and transfer techniques. A good performance improvement is obtained by transferring knowledge from both the intermediate layers and last layer of the teacher to a shallower student while other architectures and transfer techniques do not fare so well and some of them even lead to negative performance impact.

Transfer Learning technique regards the shallow layers of a pre-trained DNN on one dataset as a generic feature extractor that can be applied to other target tasks or datasets. With this feature, Transfer Learning is employed in many pieces of research and inspires the design of some frameworks. Osia et al. [60], which we have mentioned on Sec. IV-C4 , use Transfer Learning to determine the degree of generality and particularity of a private feature. Arden, proposed in [59], partitions a deep neural network across the mobile device and the cloud data center, where the raw data is transformed by the shallow portions of the DNN on the mobile device side. As [59] referred, the design of DNN splitting in Arden is inspired by Transfer Learning.

6) Gossip Training: Aiming at shortening the training latency, gossip training is a new decentralized training method, which is built on randomized gossip algorithms. The early work on random gossip algorithms is gossip averaging [63], which can fast converge towards a consensus among nodes by exchanging information peer-to-peer. The gossip distributed algorithms enjoy the advantage of full synchronization and total decentralization as they have no requirement on centralized nodes or variables. Inspired by this, GoSGD (Gossip Stochastic Gradient Descent) [64] is proposed to train DNN models in an asynchronous and decentralized way. GoSGD manages a group of independent nodes, where each of them hosts a DNN model and iteratively proceeds two steps: gradient update and mixing update. Specifically, each node updates its hosted DNN model locally in gradient update step and then shares its information with another randomly selected node in mixing update step, as shown in Fig. 11. The steps repeat until all the DNN converge on a consensus.

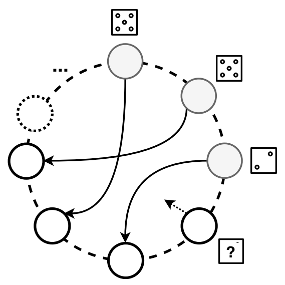


Fig. 11: Communication with randomly selected partner in gossip manner

The aim of GoSGD is to address the issue of speeding up the training of convolutional networks. Instead, another gossip-based algorithm, gossiping SGD [65], is designed to retain the positive features of both synchronous and asynchronous SGD methods. Gossiping SGD replaces the all-reduce collective operation of synchronous training with a gossip aggregation algorithm, achieving an asynchronous manner.

Both [64] and [65] apply gossip algorithms on the updates of SGD, but neither of them s performance convergence degradation at large scale. By deployment on large-scale systems, Daily et al. [66] show that the trivial gossip-based algorithms at scale lead to a communication imbalance, poor convergence and heavy communication overhead. To mitigate these issues, Daily et al. introduce GossipGraD, a gossip communication protocol based SGD algorithm which is practical for scaling deep learning algorithms on large scale systems. GossipGrad reduces the overall communication complexity from $\Theta(\log(p))$ to $O(1)$ and considers diffusion such that computing nodes exchange their updates (gradients) indirectly after every $\log(p)$ steps. It also considers the rotation of communication partners for facilitating direct diffusion of gradients and asynchronous distributed sample shuffling during the feedforward phase in SGD to prevent over-fitting.

D. Summary of Existing Systems and Frameworks

In this subsection, we summary the systems and framework for distributed EI model training on the edge. An overview of the above-mentioned existing systems and frameworks is given in Table. II, including the architecture, EI level, objectives, employed technologies and effectiveness.

In general, a key challenge for distributed EI model training is the data privacy issue. It is because the distributed data sources may originate from individual persons and different organizations. ~~For users, they may be sensitive to their own private data, not allowing any private information to be shared.~~ For companies, they have to consider the privacy policy to avoid legal subpoenas and extra-judicial surveillance. Therefore, the design of distributed training systems needs to carefully consider privacy preservation. Systems considering privacy issue in Table. II include FedAvg, BlockFL, GossipGraD and so on. The decentralized architecture is naturally friendly to users privacy, for which the systems that are based on the decentralized architecture such as BlockFL and GossipGraD typically preserve privacy better. As a contrast, the centralized architecture involves a centralized data collection operation, and the hybrid architecture requires a data transmission operation. For this reason, the systems based on these two architectures would implement more extra efforts in data privacy protection.

Compared with the DNN training under cloud-based framework, the DNN training under edge-based framework pays more attention to protecting users' privacy and training an available deep learning model faster. Under cloud-based training, a large amount of raw data generated at the client side is directly transmitted to the cloud data center through the long WAN, which not only causes hidden dangers of user privacy leakage but also consumes huge bandwidth resources. Moreover, in some scenarios such as military and disaster applications when the access to the cloud center is impossible, the edge-based training will be highly desirable. On the other hand, the cloud data center can collect a larger amount of data and train an AI model with more powerful resources, and hence the advantage of cloud intelligence is that it can train a much larger-scale and more accurate model.

V. Edge Intelligence Model Inference

After the distributed training of deep learning model, then the efficient implementation of model inference at the edge will be critical for enabling high-quality edge intelligence service deployment. In this section, we discuss the DNN model inference at the edge, including the architectures, key performance indicators, enabling techniques and existing systems & frameworks.

A. Architectures

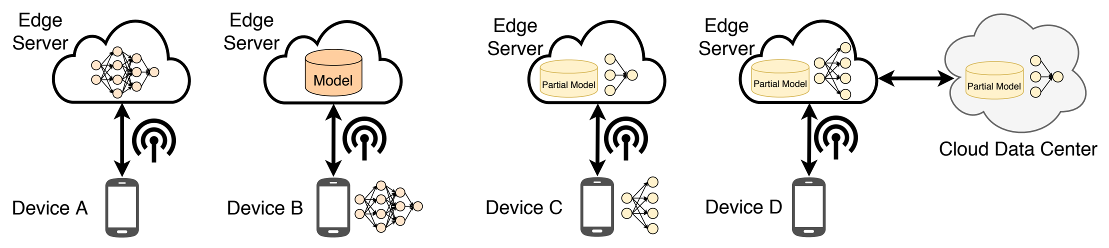
Besides the common cloud-based and device-cloud inference architectures, we further define several major edge-centric inference architectures and classify them into four modes, namely edge-based, device-based, edge-device and edge-cloud, which are illustrated in Fig. 12.

In Fig. 12, we represent different four DNN model inference modes respectively. We describe the main workflow of each mode as follows.

1) Edge-based: In Fig. 12(a), Device A is in the edge-based mode, which means that the device receives the input data then send them to the edge server. When the DNN model inference is done at the edge server, the prediction results will be returned to the device. Under this inference mode, since the DNN model is on the edge server, it is easy to implement

TABLE II: A Overview of Systems and Frameworks on EI Model Training

System or Framework	Architecture	EI Level	Objectives	Employed Technology	Effectiveness
FedAvg [45]	Hybrid	Level-4	• Robustness to non-IID and unbalanced optimization • Low communication cost • Privacy preservation	• Federated Learning • Iterative model averaging	• Reduce communication rounds by 10-100 × as compared to synchronized stochastic gradient descent
SSGD [46]	Hybrid	Level-4	• Jointly training an DNN model among clients • Privacy preservation	• Federated Learning • Selective SGD	• Clients' privacy is preserved while the model accuracy beyond training solely
Zoo [67]	Hybrid	Level-4	• Reducing communication cost • Privacy preservation	• Federated Learning • Composable services	• Processes each image within constant time despite the size difference of images
BlockFL [49]	Decentralized	Level-6	• Federated Learning in decentralized manner • Low latency • Privacy preservation	• Federated Learning • Blockchain	• Latency increase up to 1.5% to achieve the optimal block generation than the simulated minimum latency
Gaia [50]	Centralized	Cloud Intelligence	• Geo-distributed scalability • Intelligent communication mechanism over WANs • Generic and flexible for most machine learning algorithms	• Aggregation frequency control • ASP model	• Speedup 1.8-53.5× over distributed machine learning systems • Within 0.94-1.40× of the speed of running the same machine learning algorithm on machines on a local area network (LAN).
DGC [53]	N/A	N/A	• Reducing the communication bandwidth • High compression rate without losing model accuracy • Fast Convergence	• Gradient Compression • Momentum correction • Local gradient clipping • Momentum factor Masking • Warm-up training	• Achieve a gradient compression ratio from 270× to 600× without losing accuracy • Cut the gradient size of ResNet-50 from 97MB to 0.35MB and for DeepSpeech from 488MB to 0.74MB.
eSGD [54]	Hybrid	Level-4	• Scaling up edge training of CNN • Reducing communication cost	• Selective transmit important gradient coordinates • Momentum residual accumulation	• Reach 91.2%, 86.7%, 81.5% accuracy on MNIST data set with gradient drop ratio 50%, 75%, 87.5% respectively
INCEPTIONN [68]	Hybrid	Level-5	• Maximizing the opportunities for Compression • Avoiding the bottleneck at aggregators	• Lossy gradient compression NIC-integrated compression accelerator • Gradient-centric aggregator-free training	• Reduce the communication time by 70.9%-80.7% • Offer 2.2-3.1 × speedup over the conventional training system while achieving the same level of accuracy.
Arden [59]	Centralized	Cloud Intelligence	• Maximize utilization of computing resources • Low latency • Fast convergence	• DNN splitting • Arbitrary data nullification • Random noise addition	• The average reductions compared with the other four DNNs in terms of time, memory, and energy are 60.10%, 92.07%, and 77.05%, respectively
PipeDream [61]	Hybrid	Level-5	• Maximizing utilization of computing resources • Low latency • Fast convergence	• DNN splitting • Pipeline parallelism	• Using 4 machines to train the > 100 million parameter VGG16 on the ImageNet1K dataset, PipeDream converges 2.5× faster than using a single machine and 3× faster than data parallel training
GoSGD [64]	Decentralied	Level-6	• Speeding up DNN training • Fast convergence	• Gossip Training	• Do a better use of the exchanges comparing to EASGD • Converge a lot faster comparing to EASGD
Gossiping SGD [65]	Decentralied	Level-6	• Speeding up DNN training • Scaling up DNN training • Asynchronous training	• Gossip Training • Model partition	• One iteration of gossiping SGD is faster than one iteration of all-reduce SGD • Work quickly at the initial step size.
GossipGraD [66]	Decentralied	Level-6	• Reducing communication complexity • Fast convergence • Privacy preservation	• Gossip Training • Model partition	• Achieve about 100% compute efficiency for ResNet50 using 128 NVIDIA Pascal P100 GPUs while matching the top-1 classification accuracy published in literature.



(a) Edge-based Mode (b) Device-based Mode (c) Edge-Device Mode (d) Edge-Cloud Mode

Fig. 12: Major edge-centric inference modes: edge-based, device-based, edge-device and edge-cloud

the application on different mobile platforms. But the main disadvantage is that the inference performance depends on network bandwidth between the device and the edge server.

2) Device-based: In Fig. 12(b), Device B is in the device-based mode. The mobile device obtains the DNN model from the edge server and performs the model inference locally. During the inference process, the mobile device does not communicate with the edge server. So the inference is reliable, but it requires a large amount of resources such as CPU, GPU, and RAM on the mobile device. The performance depends on the local device itself.

3) Edge-device: In Fig. 12(c), Device C is in the edge-device mode. Under the edge-device mode, the device first partitions the DNN model into multiple parts according to the current system environmental factors such as network bandwidth, device resource and edge server workload. Then, the device will execute the DNN model up to a specific layer and send the intermediate data to the edge server. The edge server will execute the remain layers and sends the prediction results to the device. Compared to the edge-based mode and the device-based mode, the edge-device mode is more reliable and flexible. It may also require huge resource on the mobile device because the convolution layers at the front position of a DNN model is computational-intensive generally.

4) Edge-cloud: In Fig. 12(d), Device D is in the edge-cloud mode. It is similar to the edge-device mode and is suitable for the case that the device is highly resource constrained. In this mode, the device is responsible for input data collection and the DNN model is executed through edge-cloud synergy. The performance of this model heavily depends on the network connection quality.

We should emphasize that the four edge-centric inference modes above can be adopted in a system simultaneously to carry out complex AI model inference tasks (e.g., Cloud-Edge-Device hierarchy), by efficiently pooing heterogeneous resources across a multitude of end devices, edge nodes and clouds.

B. Key Performance Indicators

To describe the service quality of edge intelligence model inference, we introduce the following five metrics.

1) Latency: Latency refers to the time spent in the whole inference process, including pre-processing, model inference, data transmission, and post-processing. For some real-time intelligent mobile applications (e.g., AR/VR mobile gaming and intelligent robots), they usually have stringent deadline requirement such as 100ms latency. Latency indicator is affected by many factors, including the resources on edge devices, the way of data transmission and the way to execute the DNN model.

2) Accuracy: Accuracy refers to the ratio of the number of the input samples that get the correct predictions from inference to the total number of input samples, reflecting the performance of the DNN models. For some mobile applications requiring a high level of reliability, such as self-driving car and face authentication, they demand the ultrahigh accuracy on the DNN model inference. Besides the DNN model's own inference capability, the inference accuracy depends on the speed of feeding the input data to the DNN model. For a video analytics application, under a fast feeding rate, some input sample may be skipped due to the edge device's constraint resources, causing a drop in accuracy.

3) Energy: To execute a DNN model, compared with the edge server and the cloud data center, the end devices are usually battery-limited. The computation and communication overheads of DNN model inference bring a large amount of energy consumption. For an edge intelligence application, energy efficiency is of great importance and is affected by the size of DNN model and the resources on edge devices.

4) Privacy: The IoT and mobile devices generate a huge amount of data, which could be privacy sensitive. Thus, it is also important to protect privacy and data security near the data source for an edge intelligence application during the model inference stage. Privacy protection depends on the way of processing the original data.

5) Communication overhead: Except for the device-based mode, the communication overhead affects the inference performance of the other modes greatly. It is necessary to minimize the overhead during the DNN model inference in an edge intelligence application, particularly the expensive wide-area network bandwidth usage for the cloud. Communication overhead here mainly depends on the mode of DNN inference and the available bandwidth.

6) Memory Footprint: Optimizing the memory footprint of performing deep neural network model inference on mobile devices is very necessary. On the one hand, typically, a high-precision deep neural network model is accompanied by millions of parameters, which is very hungry for the hardware resources of mobile devices. On the other hand, unlike high-performance discrete GPUs on the cloud data center, there is no dedicated high-bandwidth memory for mobile GPUs on mobile devices [69]. Moreover, mobile CPUs and GPUs typically compete for shared and scarce memory bandwidth. For the optimization of the DNN inference at the edge side, memory footprint is a non-negligible indicator. Memory footprint is mainly affected by the size of the original DNN model and the way of loading the tremendous DNN parameters.

C. Enabling Technologies

In this subsection, we review the enabling technologies for improving one or more of the aforementioned key performance indicators for edge intelligence model inference. Table III summarizes the highlights of each enabling technology.

1) Model Compression: To alleviate the tension between resource hungry DNNs and resource-poor end devices, DNN compression has been commonly adopted to reduce the model complexity and resource requirement, enabling local, on-device inference which in turn reduces the response latency and has fewer privacy concerns. That is, model compression method optimizes the above four indicators, latency, energy, privacy and memory footprint. Various DNN compression techniques have been proposed, including weight pruning, data quantization, and compact architecture design.

Weight pruning represents the most widely adopted technique of model compression. This technique removes redundant weights (i.e., connections between neurons) from a trained DNN. Specifically, it first ranks the neurons in the DNN according to how much the neuron contributes, and then removes the low-ranking neurons to reduce the model size. Since removing neurons damages the accuracy of the DNN, then how to reduce the network size meanwhile preserving the accuracy is the key challenge. For modern large-scale DNNs, a pilot research [70] in 2015 tackled this challenge by applying a magnitude-based weight pruning method. The basic idea of this method is as follows: first remove small weights whose magnitudes are below a threshold (e.g., 0.001) and then fine-tune the model to restore the accuracy. For AlexNet and VGG-16, this method can reduce the number of weights by 9x and 13x with no loss of accuracy on ImageNet. The follow-up work Deep Compression [71] which blends the advantages of pruning, weight sharing and Huffman coding to compress DNNs, further pushes the compression ratio to 35-49x.

However, for energy-constrained end devices, the above magnitude-based weight pruning method may not be directly applicable, since empirical measurements show that the reduction of the number of weights does not necessarily translate into significant energy saving [72]. This is because for DNNs as exemplified by AlexNet, the energy of the convolutional layers dominates the total energy cost, while the number in the fully-connected layers contributes most of the total number of

TABLE III: Technologies for distributed DNN inference at the edge

Technology	Highlights	Related Work
Model Compression	• Weight pruning and quantization to reduce storage and computation	[70]–[77]
Model Partition	• Computation offloading to the edge server or mobile devices • Latency- and energy-oriented optimization	[10], [78]–[86]
Model Early-Exit	• Partial DNNs model inference • Accuracy-aware	[10], [15], [78], [87]–[91]
Edge Caching	• Fast response towards reusing the previous results of the same task	[92]–[96]
Input Filtering	• Detecting difference between inputs, avoiding abundant computation	[97]–[101]
Model Selection	• Inputs-oriented optimization • Accuracy-aware	[102]–[106]
Support for Multi-Tenancy	• Scheduling multiple DNN-based task • Resource-efficient	[38], [104], [107]–[111]
Application-specific Optimization	• Optimizations for the specific DNN-based application • Resource-efficient	[104], [112]

weights in the DNN. This suggests that the number of weights may not be a good indicator for energy, and the weight pruning should be directly energy-aware for end devices. As the first step towards this end, an online DNN energy estimation tool (https://energyestimation.mit.edu/) has been developed by MIT to enable fast and easy DNN energy estimation. This fine-grained tool profiles the energy for the data movement from different levels of the memory hierarchy, the number of MACs, and the data sparsity at the granularity of DNN layer. Based on this energy estimation tool, an energy-aware pruning method called EAP [73] is proposed.

Another mainstream technique for model compression is data quantization. Instead of adopting the 32-bit floating point format, this technique uses a more compact format to represent layer inputs, weights, or both. Since representing a number with fewer bits reduces memory footprint and accelerates computation, data quantization improves overall computation and energy efficiency. Most prior proposals for quantization tune the bit-width only for a fixed number type in an ad hoc manner, which may lead to a suboptimal result. To address this issue, the recent work [76] investigated the problem of optimal number representations at the layer granularity, in terms of finding the optimal bit-width for the canonical format based on IEEE 754 Standard. This problem is challenging due to the combinatorial explosion of feasible number formats. In response, the authors developed a portable API called Number abstract data type (ADT). it enables users to declare the data to be quantized in a layer (e.g., inputs, weights, or both) as Number type. By doing so, ADT encapsulates the internal representation of a number, thus separating the concern for developing an effective DNN from the concern of optimizing the number representation at a bit level.

While most existing efforts use a single compression technique, they may not suffice to meet the diverse requirements and constraints on accuracy, latency, storage, and energy imposed by some IoT devices. Emerging studies have shown how different compression techniques can be coordinated to maximally compress DNN models. For example, both Deep Compression [71] and Minerva [74] combine weight pruning and data quantization to enable fast, low-power and highly-accurate DNN inference. More recently, researchers argue that for a given DNN, the combination of compression techniques should be selected on demand, i.e., adapting to the application driven system performance (e.g. accuracy, latency and energy) and the varying resource availability across platforms (e.g. storage and processing capability). To this end, the proposed automatic optimization framework AdaDeep [75] systematically formulates the goals and constraints on accuracy, latency, storage and energy into a unified optimization problem, and leverages deep reinforcement learning (DRL) to effectively find a good combination of compression techniques.

2) Model Partition: To alleviate the pressure of the edge intelligence application execution on end devices, as shown in Fig. 13, one intuitive idea is the model partition, offloading the computational-intensive part to the edge server or the nearby mobile devices, obtaining a better model inference performance. Model partition mainly cares about the issues of latency, energy and privacy.

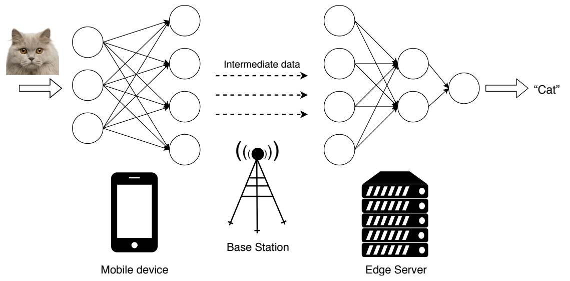


Fig. 13: An illustration for model partition between devices and edge server

The model partition can be divided into two types, partition between server and device and partition between devices. For the model partition between server and device, Neurosurgeon [79] represents an iconic effort. In Neurosurgeon, DNN model is partitioned between the device and the server, the key challenge is to figure out one suitable partition point to get the optimal model inference performance. Considering from latency aspect and energy efficiency aspect respectively, the authors propose a regression-based method to estimate the latency of each layer in the DNN model and return an optimal partition point which makes the model inference meet latency requirement or energy requirement.

Hereafter, Ko et al. propose an edge-host partitioning method [83], which combines model partition with lossy feature encoding. That is, the intermediate data after model partition will be compressed using lossy feature encoding before transmission. Also jointly leverages model partition and lossy feature encoding, the JALAD [81] framework formulates the model partition as an integer linear programming (ILP) problem to minimize the model inference latency under a guaranteed accuracy constraint. For DNNs those are characterized by a directed acyclic graph (DAG) rather than a chain, optimizing the model partition to minimize the latency is proven to be NP-hard in general. In response, Hu *et al.* [80] propose an approximation algorithm that provides worst-case performance guarantee, based on the graph min-cut method. The above frameworks all have an assumption that the server has the DNN model of the edge intelligence application. IONN [82] propose an incremental offloading technique for edge intelligence application. IONN partitions the DNN layers and incrementally uploads them to allow collaborative DNN model inference by mobile devices and the edge server. Compared to the approach that uploads the entire model, IONN significantly improves query performance and energy consumption during DNN model uploading.

Another type of model partition is the partition between devices. As the pioneering effort of model partition between devices, MoDNN [85] introduces WiFi Direct technique to build a micro-scale computing cluster in WLAN with multiple authorized WiFi-enabled mobile devices for partitioned DNN model inference. The mobile device that carries the DNN task will be the group owner and the others act as the worker nodes. Two partition schemes are proposed in MoDNN to accelerate DNN layer execution. The experiment shows that with 2 to 4 worker nodes, MoDNN accelerates DNN model inference by 2.17-4.28x. In the follow-up work MeDNN [84], greedy two-dimensional partition is proposed to adaptively partition DNN model onto multiple mobile devices and utilize a structured sparsity pruning technique to compress DNN model. MeDNN improves DNN model inference by 1.86-2.44x with 2-4 worker nodes and saves 26.5% of additional computing time and 14.2% of extra communication time. Note that DNN layers are partitioned horizontally in MoDNN and MeDNN, in contrast, DeepThings [86] employs a fused tile partitioning method that partitions the DNN layers vertically to reduce the memory footprint.

DeepX [77] also tries to partition DNN models but it only partitions the DNN model into several sub-models and distributes them on local processors. DeepX proposes two schemes: Runtime Layer Compression (RLC) and Deep Architecture Decomposition (DAD). The layer after compression will be executed by specific local processors (CPU, GPU, and DSP). An additional note is that when we have multiple tasks of model partition, we need to make optimization for the scheduler. LEO [113] is a novel sensing algorithm scheduler that maximizes the performance for multiple continuous mobile sensor applications by partitioning the sensing algorithm execution and distributing tasks on CPU, co-processor, GPU and the cloud.

3) Model Early-Exit: A DNN model with a high accuracy usually has a deep structure. It consumes a large number of resources to execute such a DNN model on the end device. To accelerate model inference, model early-exit method leverages output data of early layer to get the classification result, which means that the inference process is finished by using partial DNN model. Latency is the optimization target of model early-exit.

BranchyNet [87] is a programming framework that implements the model early-exit mechanism. With BranchyNet, the standard DNN model structure is modified by adding exit branches at certain layer locations. Each exit branch is an exit point and shares part of DNN layers with the standard DNN model. Fig. 14 shows a CNN model with three five points. The input data can be classified at these diverse early exit point.

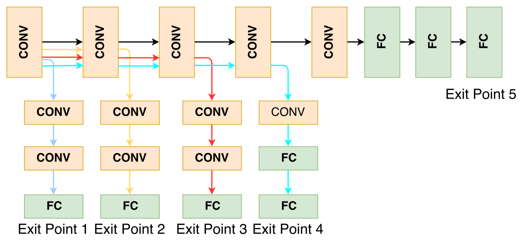


Fig. 14: A CNN model with five exit points

Based on BranchyNet, a framework named DDNNs [88] for distributed deep neural networks across the cloud, edge and devices is proposed. DDNNs has a three-layer structure framework, including device layer, edge server layer and cloud layer. Each layer represents an exit point of BranchyNet. Three aggregation methods including max pooling (MP), average pooling (AP) and concatenation (CC) are proposed. The aggregation methods work when multiple mobile devices send intermediate to an edge server or when multiple edge servers send intermediate data to the cloud data center. MP aggregates the data vectors by taking the max of each component. AP aggregates the data vectors by taking the average of each component. CC just simply concatenates the data vectors as one vector. Also built on top of BranchyNet, Edgent [10] is proposed to navigate the accuracy-latency tradeoff when jointly applying model early-exit and model partition. The basic idea of Edgent is to maximize the accuracy under a given latency requirement, via the regression-based layer latency prediction model.

In addition to BranchyNet, there are different methods to implement model early-exit. For example, Cascading network [91] simply adds max pooling layer and fully-connected layer to the standard DNN model and achieves a speedup of 20%. DeepIns [15] proposes a manufacture inspection system for the smart industry using DNN model early-exit. In DeepIns, edge devices are responsible for data collection, the edge server acts as the first exit point and the cloud data center acts as the second exit point. Then Lo et al. [90] proposes adding an authentic operation (AO) unit to the basic BranchyNet model. The AO unit determines whether an input has to be transferred to the edge server or cloud data center for further execution by setting different threshold criteria of confidence level for different DNN model output classes. And Bolukbasi et al. [89] trains a policy that determines whether the current samples should proceed to the next layer by adding regularization to the evaluation latency of the DNN model.

4) Edge Caching: Edge caching is a new kind of method used to accelerate DNN model inference, i.e., optimizing the latency issue, by caching the DNN inference results. The core idea of edge caching is to cache and reuse the task results such as the prediction of image classification at the network edge, reducing the querying latency of edge intelligence application. Fig. 15 shows the basic process of semantic cache technique, if the request from mobile devices hit the cached results stored in the edge server, the edge server will return the result, otherwise, the request will be transferred to the cloud data center for inference with the model of full precision.

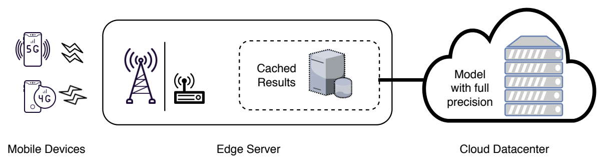


Fig. 15: The process of semantic cache technique

Glimpse [92] is a pioneering effort to introduce cache technique to DNN inference task. For an object detection application, Glimpse proposes to reusing the stale detection result to detect the object on current frames. The results of the detected object of the stale frames are cached on the mobile devices, then Glimpse extracts a subset of these cached results and computes the optical flow of features between processed frames and the current frame. And the computing results of optical flow will guide us to move the bounding box to the right location in the current frame. Glimpse achieves an acceleration of 1.6-5.5x.

But caching results locally does not scale beyond tens of images, then Cachier [93] is proposed to achieve recognition of thousands of objects. In Cachier, results of edge intelligence application are cached in the edge server, storing the features of input (e.g., image) and the corresponding task results. Then Cachier uses the least frequently used (LFS) as the cache replacement strategy. If the input can not hit the cache, the edge server will transfer the input to the cloud data center. Cachier can increase responsiveness by 3x or more. Precog [94] is the extension of Cachier. In Precog, the cached data is not only stored in the edge server but also in the mobile device. Precog uses predictions of Markov chains to prefetch data onto the mobile device and reach a speedup of 5x. In addition, Precog also proposes to dynamically adjust the cached feature extraction model on the mobile device according to the environment information. Shadow Puppets is another improved version of Cachier. Cachier extracts features from input using the standard feature extraction like locality sensitive hashing (LSH), but these features may not reflect the similarity as precise as the human dose. Then in Shadow Puppets [96], it uses a small-footprint DNN to generate hash codes to represent the input data and get a remarkable latency improvement of 5-10x.

Considering the application scenario that the same application runs on multiple devices in close proximity and the DNN model often processes similar input data, then FoggyCache [95] are proposed to minimize these redundant computations. There are two challenges in FoggyCache: one is that the input data distribution is unknown so the problem is how to index the input data with a constant lookup quality, and the other is how to represent the similarity of the input data. To address these two challenges, FoggyCache proposes adaptive locality sensitive hashing (A-LSH) and homogenized kNN (H-kNN) schemes, respectively. FoggyCache reduces computation latency and energy consumption by a factor of 3x to 10x.

5) Input Filtering: Input filtering is an efficient method to accelerate DNN model inference, especially for the video analytics. As shown in Fig. 16, the key idea of input filtering is to remove the non-target-object frames of input data, avoiding redundant computation of DNN model inference, so that improving inference accuracy, shortening inference latency and reducing energy cost.

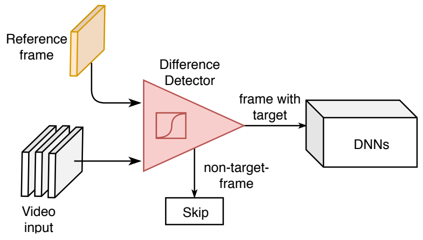


Fig. 16: The workflow of input filtering

NoScope [97] is proposed to accelerate video analysis by skipping the frames that have little change. To this end, NoScope implements a difference detector that highlights temporal differences across frames, for example, the detector monitors the frames to check whether cars appear in the frames and the frame with cars will be processed in DNN model inference. The difference is detected by using lightweight binary classifiers. Under a scenario of continuous video transmission from a swarm of drones, Wang et al. [98] optimize for the first hop wireless bandwidth of DNN inference. In particular, four strategies are proposed to reduce total transmission: EarlyDiscard, Just-in-Time-Learning (JITL), Reachback and Context-Aware.

FFS-VA [100] is a pipelined system for multi-stage video analytic. There are three stages to build the filtering system of FFS-VA. The first is a stream-specialized different detector (SDD) which is used to remove the frames only containing a background. The second is a stream-specialized network model (SNM) to identify target-object frames. And the third is a Tiny-YOLO-Voc (T-YOLO) model to remove the frames whose target objects are fewer than a threshold. Canel et al. [101] proposes a two-stage filtering system for video analytics. It first extracts the semantic content of the frames by outputting the intermediate data of DNN, then these output features are accumulated in a frame buffer. The buffer is viewed as a directed acyclic graph and the filtering system uses Euclidean distance as the similarity metric to figure out top-k interesting frames.

The above frameworks focus on filtering uninteresting frames of a video stream for a single camera. ReXCam [99] accelerates DNN model inference on cross-camera analytics. The ReXCam leverages a learned spatiotemporal model to filter video frames. ReXCam reduces computation workload by 4.6x and improves DNN model inference accuracy by 27%.

6) Model Selection: Model selection method is proposed to optimize the DNN inference issue of latency, accuracy and energy. The main idea of model selection is that we can first train a set of DNN models for the same task with various model size offline, and then adaptively select the model for inference online. Model selection is similar to the model early-exit, and the exit point of model early-exit mechanism can be viewed as a DNN model. But the key difference is that the exit point shares part of DNN layers with the main branch model and the models in the model selection mechanism are independent.

Park et al. [102] proposes a big/little DNN model selection framework. That is, a little and fast model is used to try to classify the input data and the big model is only used when the confidence of the little model is less than a predefined threshold. Taylor et al. [103] points out that different DNN models (e.g., MobileNet, ResNet, Inception) reach lowest inference latency or highest accuracy on different evaluation metrics (top-1 or top-5) for different images. Then they propose a framework for selecting the best DNN in terms of latency and accuracy. In this framework, a model selector is trained to select the best DNN for different input images. Similarly, IF-CNN [105] also trains a model selector called recognition predictor (RP) to change the model used in the task. RP is a DNN model of multi-task, meaning that RP has multiple outputs. The output of RP represents the probability of top-1 label of each candidate DNN model. The input of RP is the image and if the output of RP is over the predefined threshold, the corresponding DNN model will be selected.

Besides the optimization for DNN model inference latency, aiming at energy saving, Stamoulis et al. [106] cast the adaptive DNN model selection issue as a hyper-parameter optimization problem by taking into account the accuracy and communication constraints imposed by the devices. Then a Bayesian Optimization (BO) is adopted to solve this problem, achieving an improvement by up to 6x in terms of minimum energy per image under accuracy constraints.

7) Support for Multi-Tenancy: In practice, an end or edge device typically runs more than one DNN applications concurrently. For example, the advanced driver assistance system (ADAS) for internet vehicles simultaneously runs DNN programs for vehicle detection, pedestrian detection, traffic sign recognition, and lane line detection. In this case, multiple DNN applications would compete for the limited resource. Without careful support for multi-tenancy, i.e., resource allocation and task scheduling for those concurrent applications, the global efficiency would be greatly deteriorated. The support for multi-tenancy focuses on the optimization of energy and memory footprint.

Taking the dynamics of runtime resources into consideration, NestDNN [107] is proposed to offering flexible resource-accuracy trade-offs for each DNN model. NestDNN implements a new model pruning and recovery scheme, transforming the DNN model into a single compact multi-capacity model which consists of a set of descendent models. Each descendent model offers a unique resource-accuracy trade-off. For each concurrent descendent model, NestDNN encodes its accuracy and latency into a cost function, then NestDNN builds a resource-accuracy runtime scheduler to make the optimal trade-off for each concurrent descendent model. Also addressing the challenge of enabling flexible trade-offs, Mainstream [110] uses the popular transfer-learning DNN training method to train multiple DNN models with different degrees of accuracy and implements a greedy approach to find the optimal scheduler that fits the cost budget. For multiple DNN model executions on one single device, HiveMind [111] is proposed to improve the GPU utilization for these concurrent workloads. HiveMind consists of two key components: a compiler and a runtime module. The compiler optimizes the data transmission, data preprocessing and computation across the workloads, and then the runtime module transforms the optimized models into an execution DAG, which will be executed on the GPU while trying to extract as much concurrency as possible.

At a finer granularity, DeepEye [108] is proposed to optimizing the inference of multi-task on the mobile device by scheduling the executions of heterogeneous DNN layers. DeepEye first segregates DNN layers of all task into two pools: convolution layers and fully-connected layers. For the convolution layers, a FIFO queue based execution strategy is employed. For the fully-connected layers, DeepEye adopts a greedy approach for caching the parameters of the fully-connected layers to maximize memory utilization.

8) Application-Specific Optimization: While the above optimizing techniques are generally applicable to EI applications, application-specific optimization can be exploited to further optimize the performance of EI applications, i.e., accuracy, latency, energy and memory footprint. For example, for video-based applications, two knobs, i.e., frame rate and resolution can be flexibly adjusted to reduce resource demand. However, since such resource-sensitive knobs also deteriorate the inference accuracy, they naturally incur a cost-accuracy tradeoff. This requires us to strike a nice balance between the resource cost and inference accuracy when tuning the video frame rate and resolution.

Towards the above goal, Chameleon [104] adjusts the knobs for each video analytic task by sharing the best top-k configuration between each task. In Chameleon, the video tasks are grouped according to the spatial correlation, then leader of the group will search for the best top-k configurations and share them with the followers. DeepDecision [112] formulates the knob-tunning problem as a multiple-choice multiple-constraint knapsack program and solves it with an improved brute-force search method.

It is also worth noting that, in the computer architecture community, hardware acceleration for efficient DNN inference has been a very hot topic and gathered extensive research efforts. Interested readers are encouraged to refer to the recent monograph [114] for more comprehensive discussions about recent advancements on hardware acceleration for DNN processing.

D. Summary of Existing Systems and Frameworks

To showcase the application of the above enabling techniques for edge intelligence model inference, the relevant systems and frameworks are summarized in Table IV, including the perspectives of target applications, architecture and EI level, optimization objectives and adopted techniques, as well as effectiveness.

Clearly, existing systems and frameworks have adopted different subsets of enabling techniques tailored to specific edge intelligence applications and requirements. To maximize the overall performance of a generic edge intelligence system, comprehensive enabling techniques and various optimization methods should work in a cooperative manner to provide rich design flexibility. Nevertheless, we would face a high dimensional configuration problem that is required to determine a large number of performance-critical configuration parameters in real time. Taking video analytics, for example, the high dimensional configuration parameters can include video frame rate, resolution, model selection and model early-exit, etc. Due to the combinatorial nature, high dimensional configuration problem involves a huge search space of parameters and is very challenging to tackle.

VI. Future Research Directions

Based on the comprehensive discussions above on existing efforts, we now articulate the key open challenges and future research directions for edge intelligence (EI).

A. Programming and Software Platforms

Currently many companies around the world focus on the AI cloud computing service provisioning. Some leading companies are also starting to provide programming/software platforms to deliver edge computing services, such as Amazon's Greengrass, Microsoft's Azure IoT Edge and Google's Cloud IoT Edge. Nevertheless, currently, most of these platforms mainly serve as relays for connecting to the powerful cloud data centers.

As more and more AI-driven computation-intensive mobile and IoT applications are emerging, edge intelligence as a service (EIaaS) can become a pervasive paradigm and EI platforms with powerful edge AI functionalities will be developed and deployed. This is substantially different from machine learning as a service (MLaaS) provided by public clouds. Essentially, MLaaS belongs to cloud intelligence and it focuses on selecting the proper server configuration and machine learning framework to train model in the cloud in a cost-efficient manner. While in a sharp contrast, EIaaS concerns more about how to perform model training and inference in resource-constrained and privacy-sensitive edge computing environments. To fully realize the potential of EI services, there are several key challenges to overcome. First of all, the EI platforms should be heterogeneity-compatible. In the future, there are many dispersive EI service providers/vendors, and the common open standard should be set such that users can enjoy seamless and smooth services across heterogeneous EI platforms anywhere and anytime. Second, there are many AI programming frameworks available (e.g., Tensorflow, Torch and Caffe). In the future, the portability of the edge AI models trained by different programming frameworks across heterogeneously distributed edge nodes should be supported. Third, there are many programming frameworks designed specifically for edge devices (e.g., TensorFlow Lite, Caffe2, CoreML and MXNet), however, empirical measurements [115] show that there is no single winner that can outperform other frameworks in all metrics. A framework that performs efficiently on more metrics can be expected in the future. Last but not least, lightweight virtualization and computing techniques such as container and function computing should be further explored to enable efficient EI service placement and migration over resource-constrained edge environments.

B. Resource-friendly Edge AI Model Design

Many existing AI models such as CNN and LSTM were originally designed for applications such as computer vision and natural language processing. Most of deep learning based AI models are highly resource-intensive, which means that powerful computing capability supported by abundant hardware resources (e.g., GPU, FPGA, TPU) is an important boost the performance of these AI models. Therefore, as mentioned above there are many studies to exploit model compression techniques (e.g., weight pruning) to resize the AI models, making them more resource-friendly for edge deployment.

Along with a different line, we can promote a resource-aware edge AI model design. Instead of utilizing the existing resource-intensive AI models, we can leverage the AutoML idea [116] and the Neural Architecture Search (NAS) techniques [117] to devise resource-efficient edge AI models tailored to the hardware resource constraints of the underlying edge devices and servers. For example, methods such as reinforcement learning, genetic algorithm, and Bayesian optimization can be adopted to efficiently search over the AI model design parameter space (i.e., AI model components and their connections) by taking into account the impact of hardware resource (e.g., CPU, memory) constraints on the performance metrics such as execution latency and energy overhead.

C. Computation-aware Networking Techniques

For EI, computation-intensive AI-based applications are typically run on the distributed edge computing environment. As a result, advanced networking solutions with computation awareness is highly desirable, such that the computation results and data can be efficiently shared across different edge nodes.

For the future 5G networks, the Ultra-Reliable Low-Latency Communication (URLLC) has been defined for mission-critical application scenarios that demand low delay and high reliability. Therefore, it will be promising to integrate the 5G URLLC capability with edge computing to provide Ultra-Reliable Low-Latency EI (URLL-EI) services. Also, advanced techniques such as software-defined network and network function virtualization will be adopted in 5G. These techniques will enable flexible control over the network resources for supporting on-demand interconnections across different edge nodes for computation-intensive AI applications.

TABLE IV: An Overview of Systems and Frameworks on EI Model Infernce

System or Framework	Application	Architecture	EI Level	Objectives	Optimization Technology	Online/Offline	Effectiveness
VideoEdge [38]	Video Analytics	Cloud-Edge-Device	Level-1	• Accuracy • Resource cost	• Frame rate adaptation • Resolution adaptation • Multi-tenancy • Service placement	Online	• Accuracy improvement: 5.4-25.4×
Chameleon [104]	Video Analytics	Device-cloud	Level-1	• Accuracy • Resource cost	• Frame rate adaptation • Resolution adaptation • Model selection	Online	• Resource reduction: 2-3×
DeepX [77]	Mobile Sensing Apps	On Device	Level-2	• Accuracy • Energy	• Model compression • Model partition	Online	• Energy reduction: 7.12-26.7×
Edgent [10]	N/A	Device-Edge	Level-2	• Accuracy • Latency	• Early-exit • Model partition	Offline	• Accuracy improvement
AdaDeep [75]	N/A	On Device	Level-3	• Accuracy • Energy • Storage	• Model compression	Online	• Latency reduction: 9.8× • Energy reduction: 4.3× • Storage reduction: 38×
DeepIns [15]	IIoT	Edge-Cloud	Level-1	• Accuracy • Latency	• Early-exit	Offline	• Latency reduction: 0.98-1.21x
Neurosurgeon [79]	N/A	Device-Cloud	Level-1	• Latency • Energy	• Model partition	N/A	• Latency reduction: 3.1-40.7× • Energy reduction: 59.5%-94.7%
Minerva [74]	N/A	On Device	Level-3	• Energy	• Hardware Acceleration • Model Compression	Offline	• Energy saving: 8×
FoggyCache [95]	IIoT	Device-Edge	Level-2	• Accuracy • Latency	• Edge Caching	Online	• Latency reduction: 3-10×x • Energy reduction: 3-10×
NoScope [97]	Video Analytics	Cloud	Cloud Intelligence	• Latency	• Input filtering	N/A	• Latency reduction: 265-15500×
JALAD [81]	N/A	Device-Cloud	Level-1	• Latency	• Model compression • Model partition	Offline	• Latency reduction: 1-25.1×
DDNNs [88]	N/A	Cloud-Edge-Device	Level-1	• Latency • Accuracy	• Model selection	N/A	• Latency reduction: over 20×
FFS-VA [100]	Video Analytics	On Device	Level-3	• Latency	• Input filtering • Multi-tenancy	N/A	• Latency reduction: 3× • Throughput improvement: more than 7×
Cachier [93]	N/A	Cloud-Edge	Level-1	• Throughput	• Edge Caching	N/A	• Throughput improvement: more than 3×
Taylor et al. [103]	N/A	On Device	Level-3	• Accuracy • Latency	• Input filtering • Model selection	N/A	• Accuracy improvement: 7.52% • Latency reduction: 1.8×
DeepDecision [112]	Video Analytics	Cloud-Edge	Level-1	• Accuracy • Latency • Energy	• Application-level optimization • Model selection	N/A	• Latency reduction: 2-10×

On the other hand, autonomous networking mechanism design is important to achieve efficient EI service provisioning under dynamic heterogeneous network coexistence (e.g., LTE/5G/WiFi/LoRa), allowing newly added edge nodes and devices to self-configure in the plug and play manner. Also, the computation-aware communication techniques are starting to draw attention, such as gradient coding [118] to mitigate straggler effect in distributed learning and over-the-air computation for distributed stochastic gradient descent [119], which can be useful for edge AI model training acceleration.

D. Trade-off Design with Various DNN Performance Metrics

For an edge intelligence application with a specific mission, there is usually a series of DNN model candidates that are capable of finishing the task. However, it is difficult for software developers to choose an appropriate DNN model for the EI application because the standard performance indicators such as top-k accuracy or mean average precision fail to reflect the runtime performance of DNN model inference on edge devices. For instance, during the EI application deployment phase, beside accuracy, inference speed and resource usage are also critical metrics. We need to explore the trade-offs between these metrics and identify the factors that affect them.

In the effort [120], for the object recognition application, the authors investigate the influence of the main factors, e.g., number of proposals, input image size and the selection of feature extractor, on inference speed and accuracy. Based on their experiment results, a new combination of these factors is found to outperform the state-of-the-art method. Therefore, it is necessary to explore the trade-offs between different metrics, helping the improve the efficiency of deploying EI application.

E. Smart Service and Resource Management

By the distributed nature of edge computing, edge devices and nodes that offer EI functionality are dispersive across diverse geo-locations and regions. Different edge devices and nodes may run different AI models and deploy different specific AI tasks. Therefore, it is important to design efficient service discovery protocols such that users can identify and locate the relevant EI service providers to fulfill their need in a timely manner. Also, to fully exploit the dispersive resource across edge nodes and devices, the partition of complex edge AI models into small subtasks and efficiently offloading these tasks among the edge nodes and devices for collaborative executions are essential.

Since for many EI application scenarios (e.g., smart cities), the service environments are of high dynamics and it is hard to accurately predict future events. As a result, it would require the outstanding capability of online edge resource orchestration and provisioning to continuously accommodate massive

EI tasks. Real-time joint optimization of heterogeneous computation, communication, and cache resource allocations and the high dimensional system parameter configuration (e.g., choosing the proper model training and inference techniques) tailored to diverse task demands is critical. To tackle the algorithm design complexity, an emerging research direction is to leverage the AI techniques such as deep reinforcement learning to adapt efficient resource allocation policy in a data-driven self-learning way.

F. Security and Privacy Issues

The open nature of edge computing imposes that the decentralized trust is required such that the EI services provided by different entities are trustworthy [121]. Thus, lightweight and distributed security mechanism designs are critical to ensure user authentication and access control, model and data integrity, and mutual platform verification for EI. Also, it is important to study novel secure routing schemes and trust network topologies for EI service delivery when considering the coexistence of trusted edge nodes with malicious ones.

On the other hand, the end users and devices would generate a massive volume of data at the network edge, and these data can be privacy sensitive since they may contain users location data, health or activities records, or manufacturing information, among many others. Subject to the privacy protection requirement, e.g., EU's General Data Protection Regulation (GDPR), directly sharing the original datasets among multiple edge nodes can have a high risk of privacy leakage. Thus, federated learning can be a feasible paradigm for privacy-friendly distributed data training such that the original datasets are kept in their generated devices/nodes and the edge AI model parameters are shared. To further enhance the data privacy, more and more research efforts are devoted to utilizing the tools of differential privacy, homomorphic encryption and secure multi-party computation for designing privacy-preserving AI model parameter sharing schemes [122].

G. Incentive and Business Models

EI ecosystem will be a grand open consortium that consists of EI service providers and users, which can include but not limited to: platform providers (e.g., Amazon), AI software providers (e.g., SenseTime), edge device providers (e.g., Hikvision), network operators (e.g., AT&T), data generators (e.g., IoT and mobile device owners), and service consumers (i.e., EI users). The high-efficiency operation of EI services may require close collaboration and integration across different service providers, e.g., for implementing expanded resource sharing and smooth service handover. Thus, proper incentive mechanism and business model are essential for stimulate effective and efficient cooperation among all members of EI ecosystem. Also, for EI service, a user can be a service consumer and meanwhile a data generator as well. In this case, a novel smart pricing scheme is needed to factorize users service consumption and the value of its data contribution.

As a means for decentralized collaboration, blockchain with a smart contract may be integrated into EI service by running on decentralized edge servers. It is worthwhile to do research on how to smartly charge the price and properly distribute the revenue among the members in the EI ecosystem according to their proof of work. Also, designing resource-friendly lightweight blockchain consensus protocol for edge intelligence is highly desirable.

VII. Concluding Remarks

Driving by the flourishing of both AI and IoT, there is a stringent need to pushing the AI frontier from the cloud to the network edge. To fulfill this trend, edge computing has been widely recognized as a promising solution to support computation-intensive AI applications in resource-constrained environments. The nexus between edge computing and AI gives birth to the novel paradigm of edge intelligence.

In this paper, we conduct a comprehensive survey of the recent research efforts on edge intelligence. Specifically, we first review the background and motivation for artificial intelligence running at the network edge. We then provide an overview of the overarching architectures, frameworks and emerging key technologies for deep learning model towards training and inference at the network edge. Finally, we discuss the open challenges and future research directions on edge intelligence. We hope this survey is able to elicit escalating attentions, stimulate fruitful discussions and inspire further research ideas on edge intelligence.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[3] Cisco Global Cloud Index: Forecast and Methodology, 20162021 White Paper. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

[4] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing grouped aggregation in geo-distributed streaming analytics," in *Proc. of ACM HPDC*, 2015.

[5] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. of ACM SIGCOMM*, 2015.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive d2d collaboration for energy-efficient mobile edge computing," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 64–71, 2017.

[8] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, 2017.

[9] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *arXiv:1809.07857*, 2018.

[10] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. of the Workshop on Mobile Edge Communications (MECOMM)*, 2018, pp. 31–36.

[11] 5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018. [Online]. Available: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/

[12] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[13] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. ACM of Mobisys*, 2014.

[14] C. Jie, L. Xu, R. Abdallah, and W. Shi, "Edgeos_h: A home operating system for internet of everything," in *Proc. of IEEE ICDCS*, 2017.

[15] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, 2018.

[16] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[17] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.

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

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE CVPR*, 2016.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[22] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards real-time object detection on embedded systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 3, pp. 417–431, 2018.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

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

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[27] Y. Chauvin and D. E. Rumelhart, *Backpropagation: theory, architectures, and applications*. Psychology Press, 2013.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

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

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

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[33] 3 AI Trends for Enterprise Computing. [Online]. Available: https://www.gartner.com/smarterwithgartner/3-ai-trends-for-enterprise-computing/

[34] Democratizing AI. [Online]. Available: https://news.microsoft.com/features/democratizing-ai/

[35] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, no. 4, pp. 14–23, 2009.

[36] Microsoft Interactive Cloud Gaming. https://azure.microsoft.com/en-us/solutions/gaming/.

[37] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *Proc. of USENIX NSDI*, 2017.

[38] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[39] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for ai," *arXiv:1712.05855*, 2017.

[40] 5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018. [Online]. Available: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/

[41] IEC White Paper Edge intelligence:2017. [Online]. Available: https://www.iec.ch/whitepaper/edgeintelligence/

[42] Accelerating AI on the intelligent edge. [Online]. Available: https://azure.microsoft.com/en-us/blog/accelerating-ai-on-the-intelligent-edge-microsoft-and-qualcomm-create-vision-ai-developer-kit/

[43] Edge Intelligence for Industrial Internet of Things. [Online]. Available: https://www.comsoc.org/publications/magazines/ieee-network/cfp/edge-intelligence-industrial-internet-things

[44] Q. Chen, z. Zheng, C. Hu, D. Wang, and F. Liu, "Data-driven task allocation for multi-task transfer learning on the edge," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019.

[45] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv:1602.05629*, 2016.

[46] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[47] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv:1610.05492*, 2016.

[48] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning."

[49] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "On-device federated learning via blockchain and its latency analysis," *arXiv:1808.03949*, 2018.

[50] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds." in *NSDI*, 2017, pp. 629–647.

[51] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *learning*, vol. 8, p. 9, 2018.

[52] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," *arXiv:1804.08333*, 2018.

[53] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv:1712.01887*, 2017.

[54] Z. Tao and Q. Li, "esgd: Communication efficient distributed deep learning on the edge," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), Boston, MA*, 2018.

[55] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4452–4463.

[56] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *Advances in Neural Information Processing Systems*, 2018, pp. 7663–7673.

[57] M. M. Amiri and D. Gunduz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," *arXiv:1901.00844*, 2019.

[58] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," 2018.

[59] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2407–2416.

[60] S. A. Osia, A. S. Shamsabadi, A. Taheri, K. Katevas, S. Sajadmanesh, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *arXiv:1703.02952*, 2017.

[61] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, G. R. Ganger, and P. B. Gibbons, "Pipedream: Pipeline parallelism for dnn training."

[62] R. Sharma, S. Biookaghazadeh, B. Li, and M. Zhao, "Are existing knowledge transfer techniques effective for deep learning with edge devices?" in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 42–49.

[63] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[64] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," *arXiv:1611.09726*, 2016.

[65] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, "How to scale distributed deep learning?" *arXiv preprint arXiv:1611.04581*, 2016.

[66] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent," *arXiv:1803.05880*, 2018.

[67] J. Zhao, R. Mortier, J. Crowcroft, and L. Wang, "Privacy-preserving machine learning based data analytics on edge devices," 2018.

[68] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 175–188.

[69] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 331–344.

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

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

[72] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 367–379.

[73] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[74] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 267–278.

[75] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 389–400.

[76] Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, and J. W. Lee, "A portable, automatic data quantizer for deep neural networks," in *Proc. of ACM PACT*, 2018.

[77] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 23.

[78] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on industrial internet of things," *IEEE Network*, 2019.

[79] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.

[80] C. Huang, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *Proc. of IEEE INFOCOM*, 2019.

[81] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," *arXiv:1812.10027*, 2018.

[82] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2018, pp. 401–411.

[83] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," *arXiv:1802.03835*, 2018.

[84] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 751–756.

[85] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1396–1401.

[86] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[87] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE, 2016, pp. 2464–2469.

[88] ——, "Distributed deep neural networks over the cloud, the edge and end devices," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 328–339.

[89] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," *arXiv:1702.07811*, 2017.

[90] C. Lo, Y.-Y. Su, C.-Y. Lee, and S.-C. Chang, "A dynamic deep neural network design for efficient workload allocation in edge computing," in *Computer Design (ICCD), 2017 IEEE International Conference on*. IEEE, 2017, pp. 273–280.

[91] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "The cascading neural network: building the internet of smart things," *Knowledge and Information Systems*, vol. 52, no. 3, pp. 791–814, 2017.

[92] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. of ACM Sensys*, 2015.

[93] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. of IEEE ICDCS*, 2017.

[94] U. Drolia, K. Guo, and P. Narasimhan, "Precog: prefetching for image recognition applications at the edge," in *Proc. of the ACM/IEEE Symposium on Edge Computing (SEC)*, 2017.

[95] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Proc. of ACM Mobicom*, 2018.

[96] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, "Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[97] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.

[98] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 159–173.

[99] S. Jain, J. Jiang, Y. Shu, G. Ananthanarayanan, and J. Gonzalez, "Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale," *arXiv:1811.01268*, 2018.

[100] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao, "Ffs-va: A fast filtering system for large-scale video analytics," in *Proc. of ACM ICPP*, 2018.

[101] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Picking interesting frames in streaming video."

[102] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2015, pp. 124–132.

[103] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," in *Proc. of ACM LCTES*, 2018.

[104] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proc. of ACM SIGCOMM*, 2018.

[105] G. Shu, W. Liu, X. Zheng, and J. Li, "If-cnn: Image-aware inference framework for cnn with the collaboration of mobile devices and cloud," *IEEE Access*, vol. 6, pp. 621–633, 2018.

[106] D. Stamoulis, T.-W. R. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu, "Designing adaptive neural networks for energy-constrained image classification," in *Proc. of ACM ICCAD*, 2018.

[107] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. of ACM Mobicom*, 2018.

[108] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. of ACM Mobisys*, 2017.

[109] Z. Fang, M. Luo, T. Yu, O. J. Mengshoel, M. B. Srivastava, and R. K. Gupta, "Mitigating multi-tenant interference in continuous mobile offloading," in *International Conference on Cloud Computing*. Springer, 2018, pp. 20–36.

[110] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *Proc. of USENIX ATC*, 2018.

[111] D. Narayanan, K. Santhanam, A. Phanishayee, and M. Zaharia, "Accelerating deep learning workloads through efficient multi-model execution," in *NIPS Workshop on Systems for Machine Learning (December 2018)*.

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

[113] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 320–333.

[114] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[115] X. Zhang, Y. Wang, and W. Shi, "pcamp: Performance comparison of machine learning packages on the edges," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[116] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *European Conference on Computer Vision*. Springer, 2018, pp. 815–832.

[117] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv:1611.01578*, 2016.

[118] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[119] G. Zhu, Y. Wang, and K. Huang, "Low-latency broadband analog aggregation for federated edge learning," *arXiv:1812.11494*, 2018.

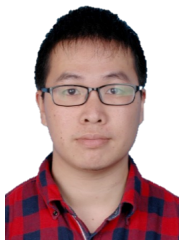
[120] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[121] D. Li, Z. Zhang, W. Liao, and Z. Xu, "KLRA: A kernel level resource auditing tool for iot operating system security," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 427–432.

[122] M. Du, K. Wang, Y. Chen, X. Wang, and Y. Sun, "Big data privacy preserving in multi-access edge computing for heterogeneous internet of things," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 62–67, 2018.

Zhi Zhou received the B.S., M.E. and Ph.D. degrees in 2012, 2014 and 2017, respectively, all from the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently a research fellow in School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. In 2016, he has been a Visiting Scholar at University of Gottingen. He was the sole recipient of 2018 ACM Wuhan & Hubei Computer Society Doctoral Dissertation Award, a recipient of the Best Paper Award of IEEE UIC 2018, and a general co-chair of 2018 International Workshop on Intelligent Cloud Computing and Networking (ICCN). His research interests include edge computing, cloud computing and distributed systems.

Xu Chen is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the vice director of National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012, and worked as a Postdoctoral Research Associate at Arizona State University, Tempe, USA from 2012 to 2014, and a Humboldt Scholar Fellow at Institute of Computer Science of University of Goettingen, Germany from 2014 to 2016. He received the prestigious Humboldt research fellowship awarded by Alexander von Humboldt Foundation of Germany, 2014 Hong Kong Young Scientist Runner-up Award, 2016 Thousand Talents Plan Award for Young Professionals of China, 2017 IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award, 2017 IEEE ComSoc Young Professional Best Paper Award, Honorable Mention Award of 2010 IEEE international conference on Intelligence and Security Informatics (ISI), Best Paper Runner-up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and Best Paper Award of 2017 IEEE International Conference on Communications (ICC). He is currently an Associate Editor of IEEE Internet of Things Journal and IEEE Journal on Selected Areas in Communications (JSAC) Series on Network Softwarization and Enablers.



En Li received the B.S. degree in communication engineering from the School of Physics & Telecommunication Engineering, South China Normal University (SCNU), Guangzhou, China in 2017. He is currently pursuing the masters degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include mobile deep computing, edge intelligence, deep learning.

Liekang Zeng received the B.S. degree in computer science from the School of Data and Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China in 2018. He is currently pursuing the masters degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include mobile edge computing, deep learning, distributed computing.

Ke Luo received the B.S. degree in computer science from the School of Data and Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China in 2018. He is currently working towards the Ph.D. degree in the School of Data and Computer Science, SYSU. His primary research interests include cloud computing, mobile edge computing, and distributed systems.

Junshan Zhang received the Ph.D. degree from the School of ECE at Purdue University in 2000. He joined the School of ECEE, Arizona State University in August 2000, where he has been the Fulton Chair Professor since 2015. His research interests are in the general field of information networks and data science, including communication networks, Internet of Things (IoT), Fog Computing, social networks, and smart grid. He is a Fellow of the IEEE, and a recipient of the ONR Young Investigator Award in 2005 and the NSF CAREER award in 2003. He received the IEEE Wireless Communication Technical Committee Recognition Award in 2016. His papers have won several awards, including the Kenneth C. Sevcik Outstanding Student Paper Award of ACM SIGMETRICS/IFIP Performance 2016, the Best Paper Runner-up Award of IEEE INFOCOM 2009 and IEEE INFOCOM 2014, and the Best Paper Award at IEEE ICC 2008 and ICC 2017. He was TPC co-chair for a number of major conferences in communication networks, including IEEE INFOCOM 2012 and ACM MOBIHOC 2015. He was the general chair for ACM/ IEEE SEC 2017, WiOPT 2016, and IEEE Communication Theory Workshop 2007. He was a Distinguished Lecturer of the IEEE Communications Society. He was an associate editor for IEEE Transactions on Wireless Communications, an editor for the Computer Networks journal, and an editor for IEEE Wireless Communication Magazine. He is currently serving as the editor-in-chief for IEEE Transactions on Wireless Communications, an editor-at-large for IEEE/ACM Transactions on Networking, and an editor for IEEE Network.