

Convergence of Edge Computing and Deep Learning: A Comprehensive Survey

Yiwen Han, *Student Member, IEEE*, Xiaofei Wang, *Senior Member, IEEE*, Victor C.M. Leung, *Fellow, IEEE*, Dusit Niyato, *Fellow, IEEE*, Xueqiang Yan, Xu Chen, *Senior Member, IEEE*

Abstract—Ubiquitous sensors and smart devices from factories and communities guarantee massive amounts of data and ever-increasing computing power is driving the core of computation and services from the cloud to the edge of the network. As an important enabler broadly changing people's lives, from face recognition to ambitious smart factories and cities, artificial intelligence (especially deep learning) applications and services have experienced a thriving development process. However, due to efficiency and latency issues, the current cloud computing service architecture hinders the vision of “providing artificial intelligence for every person and every organization at everywhere”. Thus, recently, a better solution is unleashing deep learning services from the cloud to the edge near to data sources. Therefore, *edge intelligence*, aiming to facilitate the deployment of deep learning services by edge computing, has received great attention. In addition, deep learning, as the main representative of artificial intelligence, can be integrated into edge computing frameworks to build *intelligent edge* for dynamic, adaptive edge maintenance and management. With regard to mutually benefited *edge intelligence* and *intelligent edge*, this paper introduces and discusses: 1) the application scenarios of both; 2) the practical implementation methods and enabling technologies, namely deep learning training and inference in the customized edge computing framework; 3) existing challenges and future trends of more pervasive and fine-grained intelligence. We believe that this survey can help readers to garner information scattered across the communication, networking, and deep learning, understand the connections between enabling technologies, and promotes further discussions on the fusion of *edge intelligence* and *intelligent edge*.

Index Terms—Edge computing, end-edge-cloud computing, computation offloading, artificial intelligence, deep learning

I. INTRODUCTION

With the gradual popularization of computing and storage devices, from server clusters in cloud data centers (the cloud) to personal computers and smartphones, further, to wearable

devices and other IoT devices, the computing power is experiencing the overflowing from the cloud to the edge, e.g., 50 billion IoT (Internet of Things) devices will be connected to the Internet by 2020 [1]. On the other hand, Cisco estimates that nearly 850 Zettabytes (ZB) of data will be generated each year outside the cloud by 2021, while global data center traffic is only 20.6 ZB [2]. This indicates that data sources for big data are also undergoing a transformation: from large-scale cloud data centers to an increasingly wide range of edge devices. However, existing cloud computing is gradually unable to manage and analyze these massively distributed computing power and data: 1) a large number of computation tasks need to be delivered to the cloud for processing [3], which undoubtedly poses serious challenges on network transmission capabilities and the computing power of cloud computing infrastructures; 2) various applications with new forms have strict delay requirements, such as automatic driving, while the cloud cannot provide fast response that matches them since it is far away from the user [4].

Therefore, edge computing [5], [6] emerges, and it is hoped that computation tasks will be processed as close as possible to the data source. Certainly, edge computing and cloud computing are not mutually exclusive [7], [8]. Instead, the edge is a supplement or extension of the cloud. Compared with independent cloud computing, the advantages of edge computing combined with cloud computing are mainly reflected in three aspects: 1) **backbone network alleviation**, distributed edge computing nodes can handle a large number of computation tasks without transmitting the required data to the cloud, thus alleviating the transmission pressure of the network; 2) **agile service response**, services are responded by the edge, eliminating the delay of data transmission and improving the response speed; 3) **powerful cloud backup**, the cloud can provide powerful processing capabilities and massive storage when the edge cannot afford.

As the most typical and more widely used new form of applications [9], various deep learning-based intelligent services and applications have changed all aspects of people's lives due to the great advantages of Deep Learning (DL) in the fields of Computer Vision (CV) and Natural Language Processing (NLP) [10]. These achievements are not only derived from the evolution of DL but also inextricably linked to increasing data and computing power. Nevertheless, for a wider range of application scenarios, such as smart cities, Internet of Vehicles (IoVs), etc., a wider range of intelligent services has been still limited due to the following factors.

- **Cost**: training and inference of DL models in the cloud

Yiwen Han and Xiaofei Wang are with the College of Intelligence and Computing, Tianjin University, Tianjin, China. E-mails: hanyiwen@tju.edu.cn, xiaofeiwang@tju.edu.cn.

Victor C.M. Leung is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, and also with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, Canada. E-mail: vleung@iee.org.

Dusit Niyato is with School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: dniyato@ntu.edu.sg.

Xueqiang Yan is with 2012 Lab of Huawei Technologies, Shenzhen, China. Xu Chen is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. E-mail: chenxu35@mail.sysu.edu.cn.

Corresponding author: Xiaofei Wang (xiaofeiwang@tju.edu.cn)

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

requires devices or users to transmit massive amounts of data to the cloud, thus consuming a large amount of network bandwidth;

- **Latency:** using the services provided by the cloud may cause higher transmission delays. For example, self-driving cars cannot allow millisecond delays that may be caused by cloud processing [11];
- **Reliability:** cloud computing relies entirely on wireless communications and backbone networks, but for industrial manufacturing scenarios (to name a few), intelligent services must be highly reliable, even when network connections are lost;
- **Privacy:** the data required for DL involves a lot of private information, and privacy issues are critical to areas such as smart home and cities.

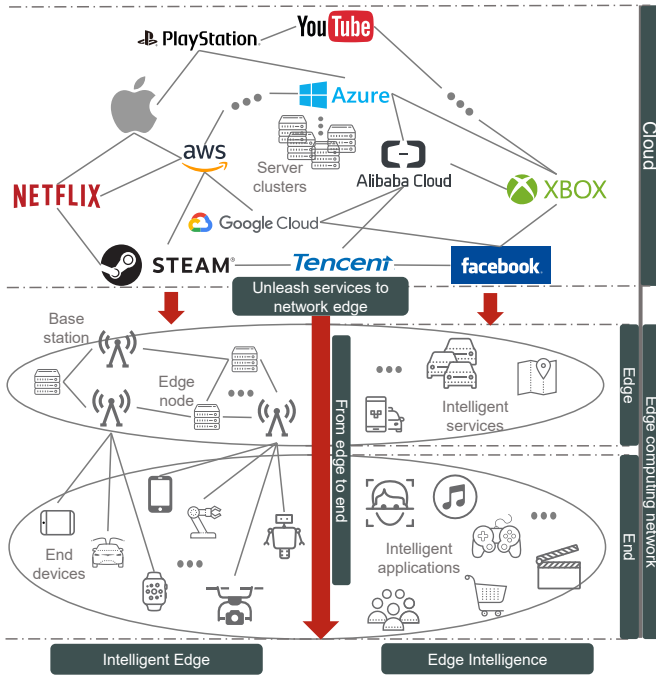


Fig. 1. Edge intelligence and intelligent edge.

Since the edge is closer to users than the cloud, edge computing is expected to solve these issues. In fact, edge computing is gradually being combined with Artificial Intelligence (AI), benefiting each other in terms of the realization of *edge intelligence* and *intelligent edge* as depicted in Fig. 1. Edge intelligence and intelligent edge are not independent of each other. Edge intelligence is the goal, and the DL services in intelligent edge are also a part of edge intelligence. In turn, intelligent edge can provide higher service throughput and resource utilization for edge intelligence.

To be specific, on one hand, edge intelligence expects to push DL computation from the cloud to the edge as much as possible, thus providing various distributed, low-latency and reliable intelligent services. As shown in Fig. 2, the advantages include: 1) DL services are deployed close to service requests, and the cloud only participates when additional processing is required [12], hence significantly reducing the latency and cost of sending data to the cloud for processing; 2) since the

raw data required for DL services is stored locally on the edge or user devices themselves instead of the cloud, privacy data is thus protected; 3) the hierarchical architecture provides more reliable DL computation; 4) with richer data and application scenarios, edge computing can promote the pervasive application of DL and realize the prospect of “providing AI for every person and every organization at everywhere” [13]; 5) diversified and valuable DL services can broaden the commercial value of edge computing and accelerate its deployment and growth.

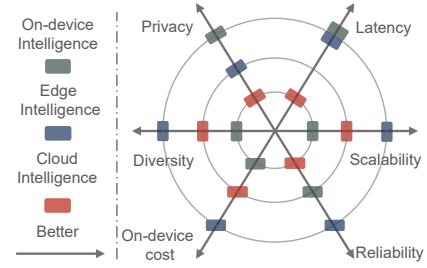


Fig. 2. Capabilities comparison of cloud, on-device and edge intelligence.

On the other hand, intelligent edge aims to incorporate DL into the edge for dynamic, adaptive edge maintenance and management. With the development of communication technology, network access methods are becoming more diverse. At the same time, the edge computing infrastructure acts as an intermediate medium, making the connection between ubiquitous end devices and the cloud more reliable and persistent [14]. This feature makes the end devices, the edge, and the cloud gradually become a community of shared resources. However, the maintenance and management of such a large and complex overall architecture (community) involving, but not limited to, wireless communication, networking, computing, storage, etc., is a major challenge [15]. Typical network optimization methodologies rely on fixed mathematical models, however, it is difficult to accurately model rapidly changing edge network environments and systems. DL is expected to deal with this problem: when faced with complex and cumbersome network information, DL can rely on its powerful learning and reasoning ability to extract valuable information from data and make adaptive decisions, achieving intelligent maintenance and management accordingly.

Therefore, considering both of them face some same challenges and practical issues in multiple aspects, as illustrated in Fig. 3, we consider five enabling technologies with respect to building edge intelligence and intelligent edge:

- 1) *DL on Edge* (DL applications on edge), technical frameworks for systematically organizing edge computing and DL to provide intelligent services;
- 2) *DL in Edge* (DL inference in edge), focusing on the practical deployment and inference of DL in the edge computing architecture to fulfill different requirements, such as accuracy and latency;
- 3) *Edge for DL* (edge computing for DL), which adapts the edge computing platform in terms of network architecture, hardware and software to cater for DL computation;
- 4) *DL at Edge* (DL training at edge), namely, training DL

models for edge intelligence at distributed edge devices under resource and privacy constraints;

- 5) *DL for Edge* (DL for optimizing edge), which resorts to DL for maintaining and managing multiple aspects [16] [17] of edge computing networks (systems), e.g., edge caching, computation offloading, communication, networking, security protection, etc.

For these enablers, “DL on Edge” and “DL for Edge” correspond to the theoretical goals of edge intelligence and the intelligent edge, respectively, while “DL in Edge”, “Edge for DL” and “DL at Edge” are their important supports involving DL inference, edge computing architecture and DL training.

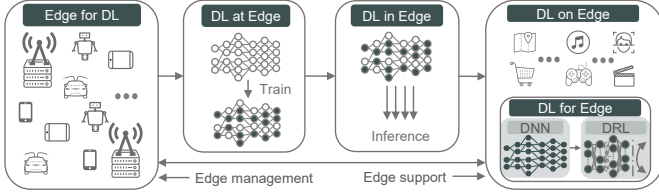


Fig. 3. Relationship between five enablers, i.e., DL on edge, DL in edge, DL at edge, edge for DL and DL for edge.

To the best of our knowledge, the most related work is [18] and [19], nonetheless, [18] concerns facilitating edge intelligence on wireless communication perspective, i.e., on one hand, exploiting edge machine learning for improving communication, and on the other hand, training machine learning at the network edge over wireless networks. Besides, discussions about DL inference and training are the main contribution of [19]. Different from [18], [19], this survey contributes on these respects: 1) comprehensively considering deployment issues of DL by edge computing, spanning networking, communication, and computation; 2) investigating the holistic technical spectrum about the convergence of DL and edge computing in terms of five enablers; 3) pointing out that DL and edge computing are mutually reinforcing and considering only deploying DL on the edge is incomplete.

This paper is organized as follows (as abstracted in Fig. 4). We present the background and motivation of this survey in Section I. Then, we give fundamentals related to edge computing and DL in Section II and Section III, respectively. Next, we introduce five enabling technologies, i.e., DL applications on edge (Section IV), DL inference in edge (Section V), edge computing for DL services (Section VI), DL training at edge (Section VII), and DL for optimizing edge (Section VIII). Finally, we discuss existing challenges and future trends in Section IX and conclude this paper in Section X. All related acronyms are listed in Table I.

II. FUNDAMENTALS OF EDGE COMPUTING

Edge computing has become an important solution to break the bottleneck of emerging technologies by virtue of its advantages of reducing data transmission, improving service latency and easing cloud computing pressure. The edge computing architecture will become an important complement to the cloud, even replacing the role of the cloud in some scenarios. More detailed information can be found in [8], [20], [21].

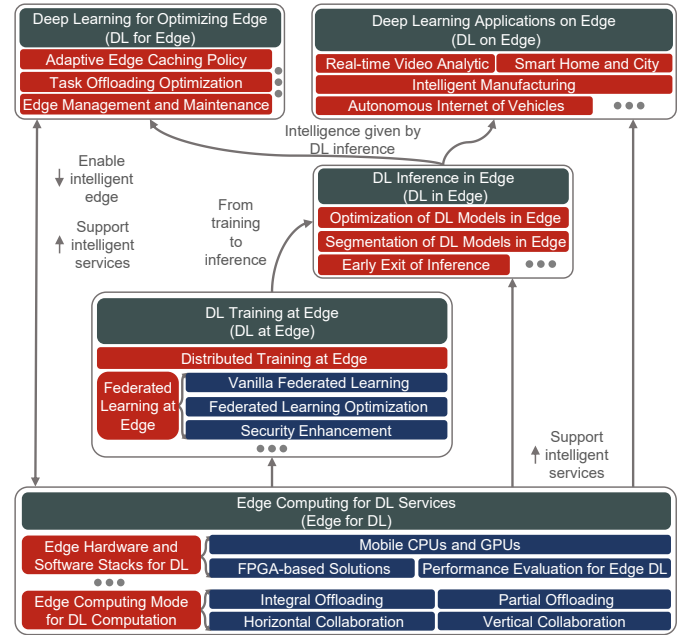


Fig. 4. Conceptual relationships of edge intelligence and intelligent edge.

A. Paradigms of Edge Computing

In the development of edge computing, there have been various new technologies aimed at working at the edge of the network, with the same principles but different focuses, such as Cloudlet [22], Micro Data Centers (MDCs) [23], Fog Computing [24] [25] and Mobile Edge Computing [5] (viz., Multi-access Edge Computing [26] now). However, the edge computing community has not yet reached a consensus on the standardized definitions, architectures and protocols of edge computing [21]. We use a common term “edge computing” for this set of emerging technologies. In this section, different edge computing concepts along the timeline are introduced and differentiated.

1) *Cloudlet and Micro Data Centers*: Cloudlets is a network architecture element that combines mobile computing and cloud computing. It represents the middle layer of the three-tier architecture, i.e., mobile devices, the micro cloud, and the cloud. Its highlights are efforts to 1) define the system and create algorithms that support low-latency edge cloud computing, and 2) implement related functionality in open source code as an extension of Open Stack cloud management software [22]. Similar to Cloudlets, MDCs [23] are also designed to complement the cloud. The idea is to package all the computing, storage, and networking equipment needed to run customer applications in one enclosure, as a stand-alone secure computing environment, for applications that require lower latency or end devices with limited battery life or computing abilities.

2) *Fog Computing*: One of the highlights of fog computing is that it assumes a fully distributed multi-tier cloud computing architecture with billions of devices and large-scale cloud data centers [24] [25]. While cloud and fog paradigms share a similar set of services, such as computing, storage, and networking, the deployment of fog is targeted to specific

TABLE I
LIST OF ABBREVIATIONS IN ALPHABETICAL ORDER

Abbr.	Definition	Abbr.	Definition	Abbr.	Definition
AC	Actor-Critic	DVFS	Dynamic Voltage and Frequency Scaling	NN	Neural Network
A3C	Asynchronous Advantage Actor-Critic	ECSP	Edge Computing Service Provider	NPU	Neural Processing Unit
AE	Auto-Encoder	EEoI	Early Exit of Inference	PPO	Proximate Policy Optimization
AI	Artificial Intelligence	EH	Energy Harvesting	QoE	Quality of Experience
APU	AI Processing Unit	ETSI	European Telecommunications Standards Institute	QoS	Quality of Service
AR	Augmented Reality	FAP	Fog radio Access Point	RNN	Recurrent Neural Network
ASIC	Application-Specific Integrated Circuit	FCNN	Fully Connected Neural Network	RoI	Region-of-Interest
BER	Bit Error Rate	FL	Federated Learning	RRH	Remote Radio Head
BS	Base Station	FPGA	Field Programmable Gate Array	RSU	Road-Side Unit
C-RAN	Cloud-Radio Access Networks	FTP	Fused Tile Partitioning	SDN	Software-Defined Network
CDN	Content Delivery Network	GAN	Generative Adversarial Network	SGD	Stochastic Gradient Descent
CNN	Convolutional Neural Network	GNNs	Graph Neural Networks	SINR	Signal-to-Interference-plus-Noise Ratio
CV	Computer Vision	IID	Independent and Identically Distributed	SNPE	Snapdragon Neural Processing Engine
D2D	Device-to-Device	IoT	Internet of Things	SRAM	Static Random Access Memory
DDoS	Distributed Denial of Service	IoV	Internet of Vehicles	TL	Transfer Learning
DDPG	Deep Deterministic Policy Gradient	KD	Knowledge Distillation	VM	Virtual Machine
DGC	Deep Gradient Compression	KNN	K-Nearest Neighbor	VR	Virtual Reality
DL	Deep Learning	MAB	Multi-Armed Bandit	V2V	Vehicle-to-Vehicle
DNNs	Deep Neural Networks	MDCs	Micro Data Centers	WLAN	Wireless Local Area Network
DQL	Deep Q-Learning	MDP	Markov Decision Process	ZB	Zettabytes
DRAM	Dynamic RAM	MLP	Multi-Layer Perceptron		
DRL	Deep Reinforcement Learning	NLP	Natural Language Processing		

geographic areas. In addition, fog is designed for applications that require real-time responding with less latency, such as interactive and IoT applications. Unlike Cloudlet, MDCs and MEC, fog computing is more focused on the IoTs.

3) *Mobile (Multi-access) Edge Computing (MEC)*: Mobile Edge Computing places computing capabilities and service environments at the edge of cellular networks [5]. It is designed to provide lower latency, context and location awareness, and higher bandwidth. Deploying edge servers on cellular Base Stations (BSs) allows users to deploy new applications and services flexibly and quickly. Later, ETSI (European Telecommunications Standards Institute) extends the terminology of MEC from Mobile Edge Computing to Multi-access Edge Computing by accommodating more wireless communication technologies, such as Wi-Fi [26].

4) *Definition of Edge Computing Terminologies*: The definition and division of edge devices are ambiguous in most literature (the boundary between edge nodes and end devices is not clear). For this reason, as depicted in Fig. 1, we further divide common edge devices into end devices and edge nodes: the “end devices” (end level) is used to refer to mobile edge devices (including smartphones, smart vehicles, etc.) and various IoT devices, and the “edge nodes” (edge level) stands for “edge nodes”, including but not limited to Cloudlets, Road-Side Units (RSUs), Fog nodes, edge servers, MEC servers and so on, namely servers deployed at the edge of the network.

5) *Collaborative End-Edge-Cloud Computing*: While cloud computing is born for processing computation-intensive tasks,

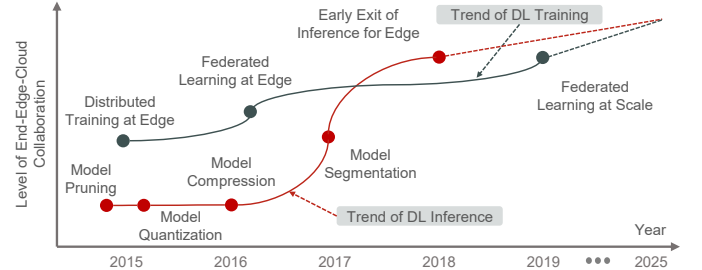


Fig. 5. Computation collaboration is becoming more important for DL with respect to both training and inference.

such as DL, it is limited as the amount of data generated by end devices explodes. Therefore, collaborative end-edge-cloud computing for DL is emerging as depicted in Fig. 5. In this novel computing paradigm, as abstracted in Fig. 6, computation tasks with lower computational intensities, generated by end devices, can be executed directly at the end or be offloaded to the edge, thus avoiding the delay caused by sending data to the cloud. For a computation-intensive task, it will be reasonably segmented and dispatched separately to the end, edge and cloud for execution, reducing the execution delay of the task while ensuring the accuracy of the results [12], [46], [47]. The focus of this collaborative paradigm is not only the successful completion of tasks but also achieving the optimal balance of equipment energy consumption, server loads, transmission and execution delays.

TABLE II
SUMMARY OF EDGE COMPUTING AI HARDWARES AND SYSTEMS

	Owner	Production	Feature
Integrated Commodities	Microsoft	Data Box Edge [27]	Competitive in data preprocessing and data transmission
	Intel	Movidius Neural Compute Stick [28]	Prototype on any platform with plug-and-play simplicity
	NVIDIA	Jetson [29]	Easy-to-use platforms that runs in as little as 5 Watts
	Huawei	Atlas Series [30]	An all-scenario AI infrastructure solution that bridges “device, edge, and cloud”
AI Chips for Edge Devices	Qualcomm	Snapdragon 8 Series [31]	Powerful adaptability to major DL frameworks
	HiSilicon	Kirin 600/900 Series [32]	Independent NPU for DL computation
	HiSilicon	Ascend Series [33]	Full coverage from the ultimate low energy consumption scenario to high computing power scenario
	MediaTek	Helio P60 [34]	Simultaneous use of GPU and NPU to accelerate neural network computing
	NVIDIA	Turing GPUs [35]	Powerful capabilities and compatibility but with high energy consumption
	Google	TPU [36]	Stable in terms of performance and power consumption
	Intel	Xeon D-2100 [37]	Optimized for power- and space-constrained cloud-edge solutions
	Samsung	Exynos 9820 [38]	Mobile NPU for accelerating AI tasks
Edge AI Computing Systems	Huawei	KubeEdge [39]	Native support for edge-cloud collaboration
	Baidu	OpenEdge [40]	Computing framework shielding and application production simplification
	Microsoft	Azure IoT Edge [41]	Remotely edge management with zero-touch device provisioning
	Linux Foundation	EdgeX [42]	IoT edge across the industrial and enterprise use cases
	Linux Foundation	Akraino Edge Stack [43]	Integrated distributed cloud edge platform
	NVIDIA	NVIDIA EGX [44]	Real-time perception, understanding, and processing at the edge
	Amazon	AWS IoT Greengrass [45]	Tolerance to edge devices even with intermittent connectivity

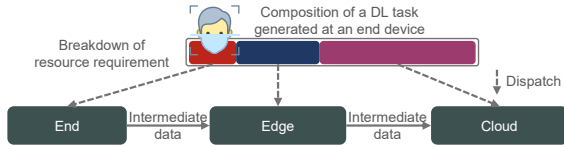


Fig. 6. A sketch of collaborative end-edge-cloud DL computing.

B. Edge Computing Hardware and Systems

In this section, we discuss potential enabling hardware of edge intelligence, i.e., customized AI chips and commodities for both end devices and edge nodes. Besides, edge-cloud systems for DL are introduced as well (listed in Table II).

1) *AI Chips for Edge Devices*: Emerged edge AI chips can be classified into three categories according to their technical architecture: 1) GPU-based chips, which tend to have good compatibility and performance, but generally consume more energy, e.g., NVIDIA’s GPUs based on Turing architecture [35]; 2) Customized chips based on FPGAs [48], [49], which are energy-saving and require less computation resources, but with worse compatibility and limited programming capability compared to GPUs; 3) ASIC (Application-Specific Integrated Circuit) chips such as Google’s TPU [36] and HiSilicon’s Ascend series [33], usually with a custom design that is more stable in terms of performance and power consumption.

As the most prosperous edge devices, the chips on smartphones have developed rapidly, and their capabilities have been extended to the acceleration of AI computing. To name a few, Qualcomm first applies AI hardware acceleration [31] in Snapdragon and releases Snapdragon Neural Processing

Engine (SNPE) SDK [50], which supports almost all major DL frameworks. Compared to Qualcomm, HiSilicon’s 600 series and 900 series chips [32] do not depend on GPUs. However, it additionally introduces a Neural Processing Unit (NPU) to achieve fast calculation of vectors and matrices, which greatly improves the efficiency of DL. Compared to HiSilicon and Qualcomm, MediaTek’s Helio P60 not only uses GPUs but also introduces an AI Processing Unit (APU) to further accelerate neural network computing [34]. Performance comparison of most commodity chips with respect to DL can be found in [51], and more customized chips of edge devices will be discussed in detail later.

2) *Integrated Commodities Potentially for Edge Nodes*: Edge nodes are expected to have computing and caching capabilities and to provide high-quality network connection and computing services near end devices. Compared to most end devices, edge nodes have more powerful computing capability to process tasks. On the other side, edge nodes can respond to end devices more quickly than the cloud. Therefore, by deploying edge nodes to perform the computation task, the task processing can be accelerated while ensuring accuracy. In addition, edge nodes also have the ability to cache, which can improve the response time by caching popular contents. For example, practical solutions including Huawei’s Atlas modules [30] and Microsoft’s Data Box Edge [27] can carry out preliminary DL inference and then transfer to the cloud for further improvement.

3) *Edge Computing Systems*: Solutions for edge computing systems are blooming. For DL services with complex configuration and intensive resource requirements, edge computing systems with advanced and excellent microservice architecture

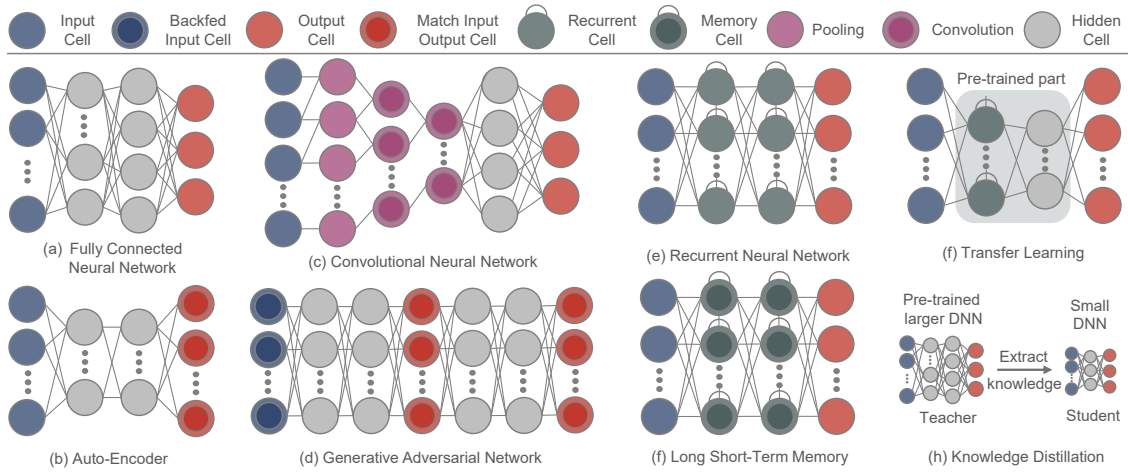


Fig. 7. Basic structures and functions of typical DNNs and DL.

are the future development direction. Currently, Kubernetes is as a mainstream container-centric system for the deployment, maintenance, and scaling of applications in cloud computing [52]. Based on Kubernetes, Huawei develops its edge computing solution “KubeEdge” [39] for networking, application deployment and metadata synchronization between the cloud and the edge (also supported in Akraino Edge Stack [43]). “OpenEdge” [40] focus on shielding computing framework and simplifying application production. For IoT, Azure IoT Edge [41] and EdgeX [42] are devised for delivering cloud intelligence to the edge by deploying and running AI on cross-platform IoT devices.

III. FUNDAMENTALS OF DEEP LEARNING

With respect to CV, NLP, and AI, DL is adopted in a myriad of applications and corroborates its superior performance [65]. Currently, a large number of GPUs, TPUs, or FPGAs are required to be deployed in the cloud to process DL service requests. Nonetheless, the edge computing architecture, on account of it covers a large number of distributed edge devices, can be utilized to better serve DL. Certainly, edge devices typically have limited computing power or power consumption compared to the cloud. Therefore, the combination of DL and edge computing is not straightforward and requires a comprehensive understanding of DL models and edge computing features for design and deployment. In this section, we compendiously introduce DL and related technical terms, paving the way for discussing the integration of DL and edge computing (more details can be found in [66]).

A. Neural Networks in Deep Learning

DL models consist of various types of Deep Neural Networks (DNNs) [66]. Fundamentals of DNNs in terms of basic structures and functions are introduced as follows.

1) *Fully Connected Neural Network (FCNN)*: The output of each layer of FCNN, i.e., Multi-Layer Perceptron (MLP), is fed forward to the next layer, as in Fig. 7(a). Between contiguous FCNN layers, the output of a neuron (cell), either the input or hidden cell, is directly passed to and activated by

neurons belong to the next layer [67]. FCNN can be used for feature extraction and function approximation, however with high complexity, modest performance, and slow convergence.

2) *Auto-Encoder (AE)*: AE, as in Fig. 7(b), is actually a stack of two NNs that replicate input to its output in an unsupervised learning style. The first NN learns the representative characteristics of the input (encoding). The second NN takes these features as input and restores the approximation of the original input at the match input output cell, used to converge on the identity function from input to output, as the final output (decoding). Since AEs are able to learn the low-dimensional useful features of input data to recover input data, it is often used to classify and store high-dimensional data [68].

3) *Convolutional Neural Network (CNN)*: By employing pooling operations and a set of distinct moving filters, CNNs seize correlations between adjacent data pieces, and then generate a successively higher level abstraction of the input data, as in Fig. 7(c). Compared to FCNNs, CNNs can extract features while reducing the model complexity, which mitigates the risk of overfitting [69]. These characteristics make CNNs achieve remarkable performance in image processing and also useful in processing structural data similar to images.

4) *Generative Adversarial Network (GAN)*: GAN originates from game theory. As illustrated in Fig. 7(d), GAN is composed of *generator* and *discriminator*. The goal of the *generator* is to learn about the true data distribution as much as possible by deliberately introducing feedback at the backfed input cell, while the *discriminator* is to correctly determine whether the input data is coming from the true data or the *generator*. These two participants need to constantly optimize their ability to generate and distinguish in the adversarial process until finding a Nash equilibrium [70]. According to the features learned from the real information, a well-trained *generator* can thus fabricate indistinguishable information.

5) *Recurrent Neural Network (RNN)*: RNNs are designed for handling sequential data. As depicted in Fig. 7(e), each neuron in RNNs not only receives information from the upper layer but also receives information from the previous channel of its own [10]. In general, RNNs are natural choices for predicting future information or restoring missing parts of

TABLE III
POTENTIAL DL FRAMEWORKS FOR EDGE COMPUTING

Framework	Owner	Support Edge	Android	iOS	Arm-linux	FPGA	DSP	GPU	Mobile GPU	Support Training
CNTK [53]	Microsoft	×	×	×	×	×	×	✓	×	✓
Chainer [54]	Preferred Networks	×	×	×	×	×	×	✓	×	✓
TensorFlow [55]	Google	✓	×	×	✓	×	×	✓	×	✓
DL4J [56]	Skymind	✓	✓	×	✓	×	×	✓	×	✓
TensorFlow Lite [57]	Google	✓	✓	×	✓	×	×	✓	✓	×
MXNet [58]	Apache Incubator	✓	✓	✓	✓	×	×	✓	×	✓
(Py)Torch [59]	Facebook	✓	✓	✓	✓	✓	×	✓	×	✓
CoreML [60]	Apple	✓	×	✓	×	×	×	×	✓	×
SNPE [50]	Qualcomm	✓	✓	×	✓	×	✓	×	✓	×
NCNN [61]	Tencent	✓	✓	✓	✓	×	×	×	✓	×
MNN [62]	Alibaba	✓	✓	✓	✓	×	×	×	✓	×
Paddle-Mobile [63]	Baidu	✓	✓	✓	✓	✓	×	×	✓	×
MACE [64]	XiaoMi	✓	✓	✓	✓	×	×	×	✓	×

sequential data. However, a serious problem with RNNs is the gradient explosion. LSTM, as in Fig. 7(f), improving RNN with adding a gate structure and a well-defined memory cell, can overcome this issue by controlling (prohibiting or allowing) the flow of information [71].

B. Deep Reinforcement Learning (DRL)

As depicted in Fig. 8, the goal of RL is to enable an agent in the environment to take the best action in the current state to maximize long-term gains, where the interaction between the agent's action and state through the environment is modeled as a Markov Decision Process (MDP). DRL is the combination of DL and RL, but it focuses more on RL and aims to solve decision-making problems. The role of DL is to use the powerful representation ability of DNNs to fit the value function or the direct strategy to solve the explosion of state-action space or continuous state-action space problem. By virtue of these characteristics, DRL becomes a powerful solution in robotics, finance, recommendation system, wireless communication, etc [18], [72].

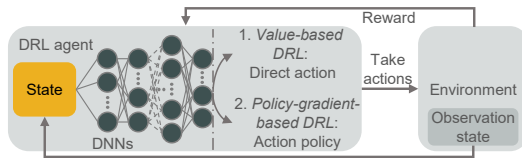


Fig. 8. Value-based and policy-gradient-based DRL approaches.

1) *Value-based DRL*: As a representative of value-based DRL, Deep Q -Learning (DQL) uses DNNs to fit action values, successfully mapping high-dimensional input data to actions [73]. In order to ensure stable convergence of training, experience replay method is adopted to break the correlation between transition information and a separate target network is set up to suppress instability. Besides, Double Deep Q -Learning (Double-DQL) can deal with that DQL generally

overestimating action values [74], and Dueling Deep Q -Learning (Dueling-DQL) [75] can learn which states are (or are not) valuable without having to learn the effect of each action at each state.

2) *Policy-gradient-based DRL*: Policy gradient is another common strategy optimization method, such as Deep Deterministic Policy Gradient (DDPG) [76], Asynchronous Advantage Actor-Critic (A3C) [77], Proximate Policy Optimization (PPO) [78], etc. It updates the policy parameters by continuously calculating the gradient of the policy expectation reward with respect to them, and finally converges to the optimal strategy [79]. Therefore, when solving the DRL problem, DNNs can be used to parameterize the policy, and then be optimized by the policy gradient method. Further, Actor-Critic (AC) framework is widely adopted in policy-gradient-based DRL, in which the policy DNN is used to update the policy, corresponding to the Actor; the value DNN is used to approximate the value function of the state action pair, and provides gradient information, corresponding to the Critic.

C. Distributed DL Training

At present, training DL models in a centralized manner consumes a lot of time and computation resources, hindering further improving the algorithm performance. Nonetheless, distributed training can facilitate the training process by taking full advantage of parallel servers. There are two common ways to perform distributed training, i.e., *data parallelism* and *model parallelism* [80]–[83] as illustrated in Fig. 9.

Model parallelism first splits a large DL model into multiple parts and then feeds data samples for training these segmented models in parallel. This not only can improve the training speed but also deal with the circumstance that the model is larger than the device memory. Training a large DL model generally requires a lot of computation resources, even thousands of CPUs are required to train a large-scale DL model. In order to solve this problem, distributed GPUs can be utilized for model parallel training [84]. Data parallelism means dividing

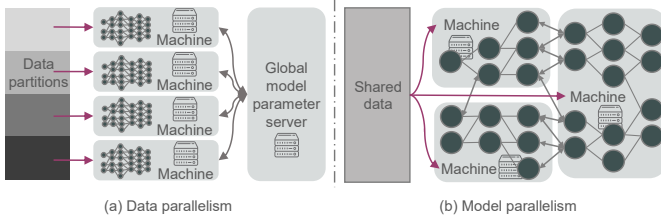


Fig. 9. Distributed training in terms of data parallelism and model parallelism.

data into multiple partitions, and then respectively training copies of the model in parallel with their own allocated data samples. By this means, the training efficiency of model training can be improved [85].

Coincidentally, a large number of end devices, edge nodes, and cloud data centers, are scattered and envisioned to be connected by virtue of edge computing networks. These distributed devices can potentially be powerful contributors once the DL training jumps out of the cloud.

D. Transfer Learning (TL)

TL can transfer knowledge, as shown in Fig. 7(f), from the source domain to the target domain so as to achieve better learning performance in the target domain [86]. By using TL, existing knowledge learned by a large number of computation resources can be transferred to a new scenario, and thus accelerating the training process and reducing model development costs. Recently, a novel form of TL emerges, viz., Knowledge Distillation (KD) [87] emerges. As indicated in Fig. 7(h), KD can extract implicit knowledge from a well-trained model (teacher), inference of which possess excellent performance but requires high overhead. Then, by designing the structure and objective function of the target DL model, the knowledge is “transferred” to a smaller DL model (student), so that the significantly reduced (pruned or quantized) target DL model achieves high performance as possible.

E. Potential DL Frameworks for the Edge

Development and deployment of DL models rely on the support of various DL frameworks. However, different DL frameworks have their own application scenarios. For deploying DL on and for the edge, efficient lightweight DL frameworks are required. Features of DL frameworks potentially supporting future edge intelligence are listed in Table III (excluding frameworks inconvenient or unavailable for edge devices, such as Theano [88]).

IV. DEEP LEARNING APPLICATIONS ON EDGE

In general, DL services are currently deployed in cloud data centers (the cloud) for handling requests, due to the fact that most DL models are complex and hard to compute their inference results on the side of resource-limited devices. However, such kind of “end-cloud” architecture cannot meet the needs of real-time DL services such as real-time analytics, smart manufacturing and etc. Thus, deploying DL applications on the edge can broaden the application scenarios of DL

especially with respect to the low latency characteristic. In the following, we present applications related to both DL and edge computing and how their elements are organized systematically, highlighting their advantages over architectures without edge computing.

A. Real-time Video Analytic

Real-time video analytic is important in various fields, such as automatic pilot, VR and Augmented Reality (AR), smart surveillance, etc. In general, applying DL for it requires high computation and storage resources. Unfortunately, executing these tasks in the cloud often incurs high bandwidth consumption, unexpected latency, and reliability issues. With the development of edge computing, those problems tend to be addressed by moving video analysis near to the data source, viz., end devices or edge nodes, as the complementary of the cloud. In this section, as depicted in Fig. 10, we summarize related works as a hybrid hierarchical architecture, which is divided into three levels: end, edge, and cloud.

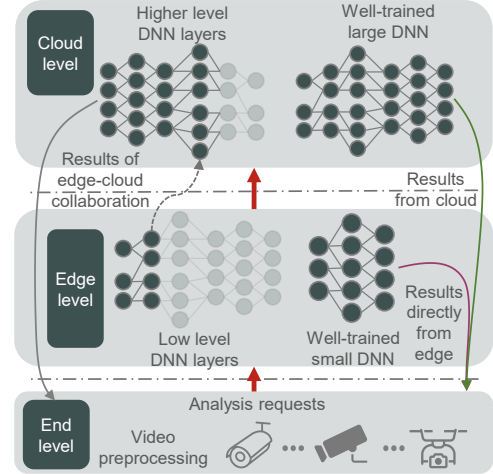


Fig. 10. The collaboration of the end, edge and cloud layer for performing real-time video analytic by deep learning.

1) *End Level*: At the end level, video capture devices, such as smartphones and surveillance cameras are responsible for video capture, media data compression [89], image pre-processing, and image segmentation [90]. By coordinating with these participated devices, collaboratively training a domain-aware adaptation model can lead to better object recognition accuracy when used together with a domain-constrained deep model [91]. Besides, in order to appropriately offload the DL computation to the end devices, the edge nodes or the cloud, end devices should comprehensively consider tradeoffs between video compression and key metrics, e.g., network condition, data usage, battery consumption, processing delay, frame rate and accuracy of analytics, and thus determine the optimal offloading strategy [89].

2) *Edge Level*: Numerous distributed edge nodes at the edge level generally cooperate with each other to provide better services. For example, LAVEA [92] attaches edge nodes to the same access point or BS as well as the end devices, which ensure that services can be as ubiquitous as Internet access.

In addition, since the cloud is more suitable for DL training while the DL inference can be benefited by executing it on the edge, *EdgeEye* [93] separates the DL model and allocates these partitions to end devices and edge nodes at different levels. Other than the edge cooperation, compressing the DL model on the edge can also improve holistic performance. As in [94], the resource consumption of the edge layer can be greatly reduced while ensuring the analysis performance, by reducing the unnecessary filters in CNN layers.

3) *Cloud Level*: At the cloud level, the cloud is responsible for the integration of DL models among the edge layer and updating parameters of distributed DL models on edge nodes [89]. Since the distributed model training performance on an edge node may be significantly impaired due to its local knowledge, the cloud needs to integrate different well-trained DL models to achieve global knowledge. When the edge is unable to provide the service confidently (e.g., detecting objects with low confidence), the cloud can use its powerful computing power and global knowledge for further processing and assist the edge nodes to update DL models.

B. Autonomous Internet of Vehicles (IoVs)

It is envisioned that vehicles can be connected to improve safety, enhance efficiency, reduce accidents, and decrease traffic congestion in transportation systems [95]. There are many information and communication technologies such as networking, caching, edge computing which can be used for facilitating the IoVs, though usually studied respectively. On one hand, edge computing provides low-latency, high-speed communication and fast-response services for vehicles, making automatic driving possible. On the other hand, DL techniques are important in various smart vehicle applications. Further, they are expected to optimize complex IoVs systems.

In [95], a framework which integrates these technologies is proposed. This integrated framework enables dynamic orchestration of networking, caching and computation resources to meet requirements of different vehicular applications [95]. Since this system involves multi-dimensional control, a DRL-based approach is first utilized to solve the optimization problem for enhancing the holistic system performance. Similarly, DRL is also used in [96] to obtain the optimal task offloading policy in vehicular edge computing. Besides, V2V (Vehicle-to-Vehicle) communication technology can be taken advantaged to further connect vehicles, either as an edge node or an end device managed by DRL-based control policies [97].

C. Intelligent Manufacturing

Two most important principles in the intelligent manufacturing era are automation and data analysis, the former one of which is the main target and the latter one is one of the most useful tools [98]. In order to follow these principles, intelligent manufacturing should first address response latency, risk control, and privacy protection, and hence requires DL and edge computing. In intelligent factories, edge computing is conducive to expand the computation resources, the network bandwidth, and the storage capacity of the cloud to the IoT edge, as well as realizing the resource scheduling and data

processing during manufacturing and production [99]. For autonomous manufacturing inspection, *aDeepIns* [98] uses DL and edge computing to guarantee performance and process delay respectively. The main idea of this system is partitioning the DL model, used for inspection, and deploying them on the end, edge and cloud layer separately for improving the inspection efficiency.

Nonetheless, with the exponential growth of IoT edge devices, 1) how to remotely manage evolving DL models and 2) how to continuously evaluate these models for them are necessary. In [100], a framework, dealing with these challenges, is developed to support complex-event learning during intelligent manufacturing, thus facilitating the development of real-time application on IoT edge devices. Besides, the power, energy efficiency, memory footprint limitation of IoT edge devices [101] should also be considered. Therefore, caching, communication with heterogeneous IoT devices, and computation offloading can be integrated [102] to break the resource bottleneck.

D. Smart Home and City

The popularity of IoTs will bring more and more intelligent applications to home life, such as intelligent lighting control systems, smart televisions, and smart air conditioners. But at the same time, smart homes need to deploy numerous wireless IoT sensors and controllers in corners, floors, and walls. For the protection of sensitive home data, the data processing of smart home systems must rely on edge computing. Like use cases in [103], [104], edge computing is deployed to optimize indoor positioning systems and home intrusion monitoring so that they can get lower latency than using cloud computing as well as the better accuracy. Further, the combination of DL and edge computing can make these intelligent services become more various and powerful. For instance, it endows robots the ability of dynamic visual servicing [105] and enables efficient music cognition system [106].

If the smart home is enlarged to a community or city, public safety, health data, public facilities, transportation, and other fields can benefit. The original intention of applying edge computing in smart cities is more due to cost and efficiency considerations. The natural characteristic of geographically distributed data sources in cities requires an edge computing-based paradigm to offer location-awareness and latency-sensitive monitoring and intelligent control. For instance, the hierarchical distributed edge computing architecture in [107] can support the integration of massive infrastructure components and services in future smart cities. This architecture can not only support latency-sensitive applications on end devices but also perform slightly latency-tolerant tasks efficiently on edge nodes, while large-scale DL models responsible for deep analysis are hosted on the cloud. Besides, DL can be utilized to orchestrate and schedule infrastructures to achieve the holistic load balancing and optimal resource utilization among a region of a city (e.g., within a campus [108]) or the whole city.

V. DEEP LEARNING INFERENCE IN EDGE

In order to further improve the accuracy, DNNs become deeper and require larger-scale dataset. By this means, dra-

matic computation costs are introduced. Certainly, the outstanding performance of DL models is inseparable from the support of high-level hardware, and it is difficult to deploy them in the edge with limited resources. Therefore, large-scale DL models are generally deployed in the cloud while end devices just send input data to the cloud and then wait for the DL inference results. However, the cloud-only inference limits the ubiquitous deployment of DL services. Specifically, it can not guarantee the delay requirement of real-time services, e.g., real-time detection with strict latency demands. Moreover, for important data sources, data safety and privacy protection should be addressed. To deal with these issues, DL services tend to resort to edge computing. Therefore, DL models should be further customized to fit in the resource-constrained edge, while carefully treating the trade-off between the inference accuracy and the execution latency of them.

A. Optimization of DL Models in Edge

DL tasks are usually computationally intensive and requires large memory footprints. But in the edge, there are not enough resources to support raw large-scale DL models. Optimizing DL models and quantize their weights can reduce resource costs. In fact, model redundancies are common in DNNs [109], [110] and can be utilized to make model optimization possible. The most important challenge is how to ensure that there is no significant loss in model accuracy after being optimized. In other words, the optimization approach should transform or re-design DL models and make them fit in edge devices, with as little loss of model performance as possible. In this section, optimization methods for different scenarios are discussed: 1) general optimization methods for edge nodes with relatively sufficient resources; 2) fine-grained optimization methods for end devices with tight resource budgets.

1) *General Methods for Model Optimization*: On one hand, increasing the depth and width of DL models with nearly constant computation overhead is one direction of optimization, such as inception [111] and deep residual networks [112] for CNNs. On the other hand, for more general neural network structures, existing optimization methods can be divided into four categories [113]: 1) parameter pruning and sharing [114], [115], including also weights quantization [116]–[118]; 2) low-rank factorization [109]; 3) transferred/compact convolution filters [94], [119], [120]; 4) knowledge distillation [121]. These approaches can be applied to different kinds of DNNs or be composed to optimize a complex DL model for the edge.

2) *Model Optimization for Edge Devices*: In addition to limited computing and memory footprint, other factors such as network bandwidth and power consumption also need to be considered. In this section, efforts for running DL on edge devices are differentiated and discussed.

- *Model Inputs*: Each application scenario has specific optimization spaces. For example, with respect to real-time human detection, by narrowing down the classifier's searching space (as depicted in Fig. 11) to focus on human objects in surveillance video frames, CNN models are hence able to be run on an edge device to detect pedestrians [122]. This kind of method, which greatly

compresses the size of model inputs without altering the structure of neural networks, can significantly reduce required computation overhead. But at the same time, it requires a deep understanding of the related application scenario to dig out the potential optimization space.

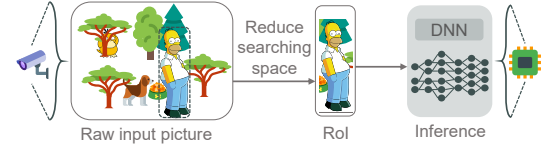


Fig. 11. Optimization for model inputs, e.g., narrowing down the searching space of DL models (pictures are with permission from [123]).

- *Model Structures*: Not paying attention to specific applications, but focusing on the widely used DNNs' structures is also feasible. By this means, point-wise group convolution and channel shuffle [124], paralleled convolution and pooling computation [125], depth-wise separable convolution [94] can greatly reduce computation cost while maintaining accuracy. Besides, as depicted in Fig. 12, parameters pruning can be applied adaptively in model structure optimization as well [126]–[128]. Furthermore, the optimization can be more efficient if across the boundary between algorithm, software and hardware. Specifically, general hardware is not ready for the irregular computation pattern introduced by model optimization. Therefore, hardware architectures, as in [126], should be designed to work directly for optimized models.

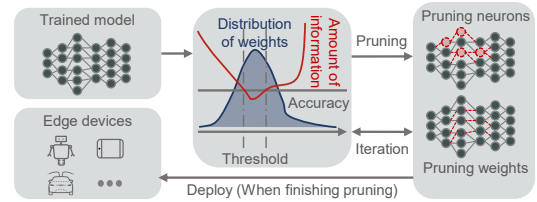


Fig. 12. Adaptive parameters pruning in model structure optimization.

- *Model Frameworks*: Given the high memory footprint and computational demands of DL, running them on edge devices requires expert-tailored software and hardware frameworks. A software framework is valuable if it 1) provides a library of optimized software kernels to enable deployment of DL [129]; 2) automatically compresses DL models into smaller dense matrices by finding the minimum number of non-redundant hidden elements [130]; 3) performs quantization and coding on all commonly used DL structures [127], [130], [131]; 4) specializes DL models to contexts and shares resources across multiple simultaneously executing DL models [131]. With respect to the hardware, running DL models on SRAM (Static Random Access Memory) achieves better energy savings compared to DRAM (Dynamic RAM) [127]. Hence, DL performance can be benefited if underlying hardware directly supports running optimized DL models [132] on the on-chip SRAM.

B. Segmentation of DL Models in Edge

In [12], the delay and power consumption of the most advanced DL models are evaluated on the cloud and edge devices, finding that uploading data to the cloud is the bottleneck of current DL servicing methods (leading to a large overhead of transmitting). Dividing the DL model and performing distributed computation can achieve better end-to-end delay performance and energy efficiency. In addition, by pushing part of DL tasks from the cloud to the edge, the throughput of the cloud can be improved. Therefore, the DL model can be segmented into multiple partitions and then allocated to 1) heterogeneous local processors (e.g., GPUs, CPUs) on the end device [133], 2) distributed edge nodes [134], [135], or 3) collaborative ‘end-edge-cloud’ architecture [12], [46], [136], [137].

Partitioning the DL model horizontally, i.e., along the end, edge and cloud, is the most common segmentation method. The challenge lies in how to intelligently select the partition points. As illustrated in Fig. 13, a general process for determining the partition point can be divided into three steps [12], [136]: 1) measuring and modeling the resource cost of different DNN layers and the size of intermediate data between layers; 2) predicting the total cost by specific layer configurations and network bandwidth; 3) choosing the best one from candidate partition points according to delay, energy requirements, etc. Another kind of model segmentation is vertically partitioning particularly for CNNs [135]. In contrast to horizontal partition, vertical partition fuses layers and partitions them vertically in a grid fashion, and thus divides CNN layers into independently distributable computation tasks.

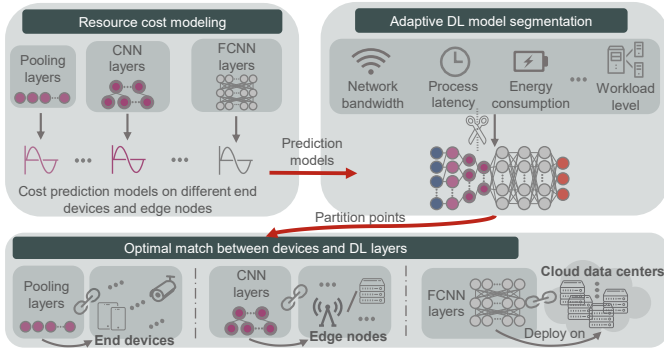


Fig. 13. Segmentation of DL models in the edge.

C. Early Exit of Inference (EEoI)

To reach the best trade-off between model accuracy and processing delay, multiple DL models with different model performance and resource cost can be maintained for each DL service. Then, by intelligently selecting the best model, the desired adaptive inference is achieved [138]. Nonetheless, this idea can be further improved by the emerged EEoI [139].

The performance improvement of additional layers in DNNs is at the expense of increased latency and energy consumption in feedforward inference. As DNNs grow larger and deeper, these costs become more prohibitive for edge devices to run

real-time and energy-sensitive DL applications. By additional side branch classifiers, for partial samples, EEoI allows inference to exit early via these branches if with high confidence. For more difficult samples, EEoI will use more or all DNN layers to provide the best predictions.

As depicted in Fig. 14, by taking advantage of EEoI, fast and localized inference using shallow portions of DL models at edge devices can be enabled. By this means, the shallow model on the edge device can quickly perform initial feature extraction and, if confident, can directly give inference results. Otherwise, the additional large DL model deployed in the cloud performs further processing and final inference. Compared to directly offloading DL computation to the cloud, this approach has lower communication costs and can achieve higher inference accuracy than those of the pruned or quantized DL models on edge devices [98], [140]. In addition, since only immediate features rather than the original data are sent to the cloud, it provides better privacy protection. Nevertheless, EEoI shall not be deemed independent to model optimization (Section V-A2) and segmentation (Section V-B). The envision of distributed DL over the end, edge and cloud should take their collaboration into consideration, e.g., developing a collaborative and on-demand co-inference framework [141] for adaptive DNN partitioning and EEoI.

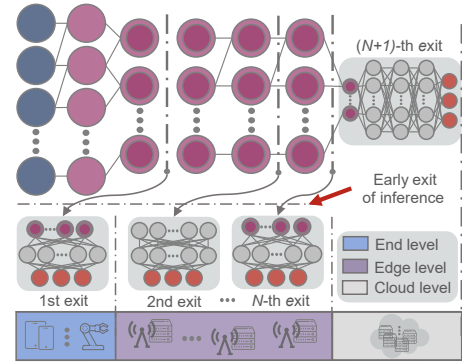


Fig. 14. Early exit of inference for DL inference in the edge.

VI. EDGE COMPUTING FOR DEEP LEARNING

Extensive deployment of DL services, especially mobile DL, requires the support of edge computing. This support is not just at the network architecture level, the design, adaptation, and optimization of edge hardware and software are equally important. Specifically, 1) customized edge hardware and corresponding optimized software frameworks and libraries can help DL execution more efficiently; 2) the edge computing architecture can enable the offloading of DL computation; 3) well-designed edge computing systems can better maintain DL services running on the edge; 4) fair platforms for evaluating DL performance on the edge help to further evolve above implementations.

A. Edge Hardware and Software Stacks for DL

1) *Mobile CPUs and GPUs*: DL applications are more valuable if directly enabled on lightweight edge devices, such

as mobile phones, wearable devices and surveillance cameras, near to the location of events. Low-power IoT devices, as a part of edge devices, can be used to undertake lightweight DL computation, and hence avoiding communication with the cloud, but it still need to face limited computation resources, memory footprint and energy consumption. To relax these bottlenecks, in [125], ARM Cortex-M micro-controllers are studied to make them potentially feasible edge hardware for DL. By the developed *CMSIS-NN*, a collection of efficient NN kernels, the memory footprint of NNs on ARM Cortex-M processor cores can be minimized, and then the DL model can be fitted into IoT devices, meantime achieving normal performance and energy efficiency.

DeepMon [142], as a suite of optimizations (on both hardware and software) for processing CNN layers on mobile GPUs, can largely reduce the inference time by taking advantage of the similarity between consecutive frames in first-person-view videos. Such a feature incurs a large amount of computation repetition in CNN layers, and thus inspires the idea of caching computation results of CNN layers. In addition, by means of matrix decomposition, high-dimensional matrix operations, particularly multiplications, in CNN layers become available in mobile GPUs and can be accelerated. In view of this work, different kinds of mobile GPUs, already deployed in edge devices, can be potentially explored with specific DL models and play a more important role in enabling edge intelligence.

Other than the inference [125], [142], important factors that affect the DL training performance on mobile CPUs and GPUs are discussed in [143]. Since commonly used DNN models, such as VGG [144], are too large for the memory size of mainstream edge devices, a relatively small Mentee network [145] is adopted to evaluate DL training. Evaluation results point out that the size of DL models is crucial for training performance and the efficient fusion of mobile CPUs and GPUs is important for accelerating the training process.

2) *FPGA-based Solutions*: Though GPU solutions are widely adopted in the cloud for DL training and inference, however, restricted by the tough power and cost budget in the edge, these solutions may not be available. Besides, edge nodes should be able to serve multiple DL computation requests at a time, and it makes simply using lightweight CPUs and GPUs impractical. Therefore, edge hardware based on FPGA (Field Programmable Gate Array) is explored to study their feasibility for edge DL.

FPGA-based edge devices can achieve CNN acceleration with arbitrarily sized convolution and reconfigurable pooling [125], and they perform faster than the state-of-the-art CPU and GPU implementations [126] with respect to RNN-based speech recognition applications while achieving higher energy efficiency. In [49], the design and setup of an FPGA-based edge computing platform are developed to support DL computation offloading from mobile devices to the edge FPGA platform. On implementing the FPGA-based edge, a wireless router and an FPGA board are combined together. Testing this preliminary system with typical vision applications, the FPGA-based edge platform shows its advantages, in terms of both energy consumption and hardware cost, over the GPU

(or CPU)-based edge.

Nevertheless, FPGAs and GPUs/CPU which are more suitable for edge computing are still pending is still pending, as shown in Table IV. Elaborated experiments are performed in [146] to investigate the advantages of adopting FPGAs for edge computing over GPUs: 1) capable of providing workload insensitive throughput; 2) guaranteeing consistently high performance for high-concurrency DL computation; 3) better energy efficiency. However, the disadvantage of FPGAs lies in that developing efficient DL algorithms on FPGA is unfamiliar to most programmers. Although tools such as Xilinx SDSoc can greatly reduce the difficulty [49], at least for now, additional works are still required to transplant the state-of-the-art DL models, programmed for GPUs, into the FPGA platform.

3) *Performance Evaluation for Edge DL*: Throughout selecting appropriate edge hardware and associated software stacks for deploying different kinds of DL services on edge, it is necessary to evaluate their performance. In addition, impartial evaluation methodologies can point out possible directions to optimize software stacks for specific edge hardware.

In [147], for the first time, the performance of DL libraries, when executing DL inference on resource-constrained edge devices, are evaluated, pertaining to metrics like latency, memory footprint, and energy. In addition, particularly for Android smartphones, as one kind of edge devices with mobile CPUs or GPUs, *AI Benchmark* [51] extensively investigates the DL computation capabilities over various device configurations. Experimental results show that no single DL library or hardware platform can entirely outperform others, and loading the DL model may take more time than executing it. These discoveries imply that there are still opportunities to further optimize the fusion of edge hardware, edge software stacks, and DL libraries.

B. Edge Computing Mode for DL Computation

DL brings intensive computation tasks that end devices cannot afford. The concept of edge computing can potentially cope with this dilemma by offloading DL computation from end devices to edge nodes. Accompanied by the edge architectures, DL-centric edge nodes can become the significant extension of cloud computing infrastructure to deal with massive DL tasks. In this section, we classify four edge computing modes for DL computation, as exhibited in Fig. 15.

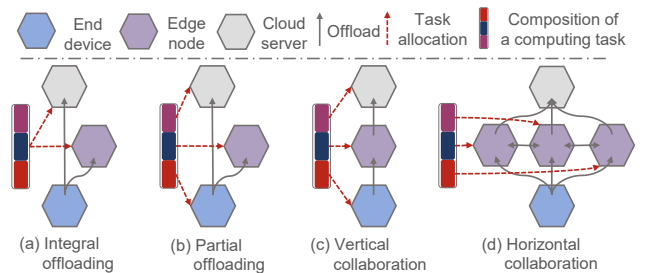


Fig. 15. Edge computing modes for DL computation.

TABLE IV
COMPARISON OF SOLUTIONS FOR EDGE NODES

Metrics	Preferred Hardware	Analysis
Resource overhead	FPGA	FPGA can be optimized by customized designs.
DL training	GPU	Floating point capabilities are better on GPU.
DL inference	FPGA	FPGA can be customized for the specific DL model.
Interface scalability	FPGA	It is more free to implement interfaces on FPGAs.
Space occupation	CPU/FPGA	Lower power consumption of FPGA leads to smaller space occupation.
Compatibility	CPU/GPU	CPUs and GPUs have more stable architecture.
Development efforts	CPU/GPU	Toolchains and software libraries facilitate the practical development.
Energy efficiency	FPGA	Customized designs can be optimized.
Concurrency support	FPGA	FPGAs are suitable for stream processing.
Timing latency	FPGA	Timing on FPGAs can be an order of magnitude faster than GPUs.

1) *Integral Offloading*: The most natural mode of DL computation offloading is similar to the existed “end-cloud” computing, i.e., the end device sends its computation requests to the cloud for DL inference results (as depicted in Fig. 15(a)). This kind of offloading is straightforward by extricating itself from DL task decomposition and combinatorial problems of resource optimization (may bring about additional computation cost and scheduling delay), and thus simple to implement.

In [148], a distributed infrastructure, that ties together powerful edge nodes with less powerful end devices, is proposed. DL inference can be performed on the end or edge, depending on the tradeoffs between the required DL accuracy, inference latency, battery level, and network conditions. With regard to each DL task, the end device will decide whether locally processing or totally offloading it to the edge node.

Further, the workload optimization among edge nodes should not be ignored in the offloading problem, since edge nodes are commonly resource-restrained compared to the cloud. In order to satisfy the delay and energy requirements of accomplishing a DL task with limited edge resources, providing DL models with different model size and performance in the edge can be adopted to fulfill one kind of tasks. Hence, multiple Virtual Machines (VMs) or containers, undertaking different DL models separately, can be deployed on the edge node to process DL requests. Specifically, when a DL model with lower complexity can meet the requirements, it will be selected as the serving model. By optimizing the workload assignment weights and computing capacities of VMs, an optimization model that aims at reducing the energy cost and delay can be developed [149], while guaranteeing the DL inference accuracy.

2) *Partial Offloading*: Partially offloading the DL task to the edge is also feasible (as depicted in Fig. 15(b)). An offloading system can be developed to enable online fine-grained partition of a DL task, and determine how to allocate these divided tasks to the end device and the edge node. As exemplified in [150], the proposed *MAUI*, capable of adaptively partitioning general computer programs, can conserve an order of magnitude energy by optimizing the task allocation strategies, under the network constraints. More

importantly, this solution can decompose the whole program at runtime instead of manually partitioning of programmers before program deploying. Though this work does not consider DL applications, it still sheds light on the potential of partial offloading of DL tasks.

It is not realistic to pre-install DL models on any edge nodes for handling every kind of requests from end devices (especially when the mobility are concerned) and it is even impractical to stuff all DL models into every edge node. Therefore, when the DL model is not pre-installed, the end device should first upload its DL model to the edge node. Unfortunately, it can seriously delay the offloaded DL computation due to long uploading time.

To deal with this challenge, an incremental offloading system *IONN* is proposed in [151]. Differ from packing up the whole DL model for uploading, *IONN* divides a DL model, prepared for uploading, into multiple partitions and uploads them to the edge node in sequential. On the other hand, the edge node, receiving the partitioned models, incrementally builds the DL model as each partitioned model arrives, while being able to execute the offloaded partial DL computation even before the entire DL model is uploaded. Therefore, the key lies in the determination of best DL partitions and the uploading order. Specifically, on one hand, DNN layers whose performance benefit is high and whose uploading overhead is low are preferred to be uploaded first, and thus making the edge node quickly build a partial DNN in order to achieve the best-expected query performance. On the other hand, unnecessary DNN layers, which cannot bring in any performance increase, will not be uploaded and hence avoiding the offloading.

3) *Vertical Collaboration*: Expected offloading strategies among “End-Edge” architecture, as discussed in Section VI-B1 and VI-B2, are feasible for supporting less computation-intensive DL services and small-scale concurrent DL queries. However, when a large number of DL queries need to be processed at one time, a single edge node is certainly insufficient.

A natural choice of collaboration is the edge performs data pre-processing and preliminary learning, when the DL tasks are offloaded. Then, the intermediate data, viz., the output

of edge architectures, are transmitted to the cloud for further DL computation [152]. Nevertheless, the hierarchical structure of DNNs can be further excavated for fitting the vertical collaboration. In [12], all layers of a DNN are profiled on the end device and the edge node in terms of the data and computation characteristics, in order to generate performance prediction models. Based on these prediction models, wireless conditions and server load levels, the proposed *Neurosurgeon* evaluates each candidate point in terms of end-to-end latency or mobile energy consumption and partition the DNN at the best one. Then, it decides the allocation of DNN partitions, i.e., which part should be deployed on the end, the edge or the cloud, while achieving best latency and energy consumption of end devices.

By taking advantages of EEOI (Section V-C), vertical collaboration can be more adapted. Partitions of a DNN can be mapped onto a distributed computing hierarchy (i.e., the end, the edge and the cloud) and can be trained with multiple early exit points [140]. Therefore, the end and the edge can perform a portion of DL inference on themselves rather than directly requesting the cloud. Using an exit point after inference, results of DL tasks, the local device is confident about, can be given without sending any information to the cloud. For providing more accurate DL inference, the intermediate DNN output will be sent to the cloud for further inference by using additional DNN layers. Nevertheless, the intermediate output, e.g., high-resolution surveillance video streams, should be carefully designed much smaller than the raw input, therefore drastically reducing the network traffic required between the end and the edge (or the edge and the cloud).

Though vertical collaboration can be considered as an evolution of cloud computing, i.e., “end-cloud” strategy. Compared to the pure “end-edge” strategy, the process of vertical collaboration may possibly be delayed, due to it requires additional communication with the cloud. However, vertical collaboration has its own advantages. One side, when edge architectures cannot afford the flood of DL queries by themselves, the cloud architectures can share partial computation tasks and hence ensure servicing these queries. On the other hand, the raw data must be preprocessed at the edge before they are transmitted to the cloud. If these operations can largely reduce the size of intermediate data and hence reduce the network traffic, the pressure of backbone networks can be alleviated.

4) *Horizontal Collaboration*: In Section VI-B3, vertical collaboration is discussed. However, devices among the edge or the end can also be united without the cloud to process resource-hungry DL applications, i.e., horizontal collaboration. By this means, the trained DNN models or the whole DL task can be partitioned and allocated to multiple end devices or edge nodes to accelerate DL computation by alleviating the resource cost of each of them. *MoDNN*, proposed in [153], executes DL in local distributed mobile computing system over a Wireless Local Area Network (WLAN). Each layer of DNNs is partitioned into slices to increase parallelism and to reduce memory footprint, and these slices are executed layer-by-layer. By the execution parallelism among multiple end devices, the DL computation can be significantly accelerated.

With regard to specific DNN structures, e.g., CNN, a finer

grid partitioning can be applied to minimize communication, synchronization, and memory overhead [115]. In [135], a Fused Tile Partitioning (FTP) method, able to divide each CNN layer into independently distributable tasks, is proposed. In contrast to only partitioning the DNN by layers as in [12], FTP can fuse layers and partitions them vertically in a grid fashion, hence minimizing the required memory footprint of participated edge devices regardless of the number of partitions and devices, while reducing communication and task migration cost as well. Besides, to support FTP, a distributed work stealing runtime system, viz., idle edge devices stealing tasks from other devices with active work items [135], can adaptively distribute FTP partitions for balancing the workload of collaborated edge devices.

VII. DEEP LEARNING TRAINING AT EDGE

Present DL training (distributed or not) in the cloud data center, namely cloud training, or cloud-edge training [47], viz., training data are preprocessed at the edge and then transmitted to cloud, are not appropriate for all kind of DL services, especially for DL models requiring locality and persistent training. Besides, a significant amount of communication resources will be consumed, and hence aggravating wireless and backbone networks if massive data are required to be continually transmitted from distributed end devices or edge nodes to the cloud. For example, with respect to surveillance applications integrated with object detection and target tracking, if end devices directly send a huge amount of real-time monitoring data to the cloud for persistent training, it will bring about high networking cost. In addition, merging all data into the cloud might violate privacy issues. All these challenges put forward the need for a novel training scheme against existing cloud training.

Naturally, the edge architecture, which consists of a large number of edge nodes with modest computing resources, can cater for alleviating the pressure of networks by processing the data or training at themselves. Training at the edge or potentially among “end-edge-cloud”, treating the edge as the core architecture of training, is called as “DL at Edge”. Such kind of DL training may require significant resources to digest distributed data and exchange updates.

A. Distributed Training at Edge

Distributed training at the edge can be traced back to the work of [154], where a decentralized SGD (Stochastic Gradient Descent) method is proposed for the edge computing network to solve a large linear regression problem. However, this proposed method is designed for seismic imaging application and can not be generalized for future DL training, since the communication cost for training large scale DL models is extremely high. In [155], two different distributed learning solutions for edge computing environments are proposed. As depicted in Fig. 16, one solution is that each end device trains a model based on local data, and then these model updates are aggregated at edge nodes. Another one is edge nodes train their own local models, and their model updates are exchanged

and refined for constructing a global model. Though large-scale distributed training at edge evades transmitting bulky raw dataset to the cloud, the communication cost for gradients exchanging between edge devices is inevitably introduced. Besides, in practical, edge devices may suffer from higher latency, lower transmission rate and intermittent connections, and therefore further hindering the gradients exchanging between DL models belong to different edge devices.

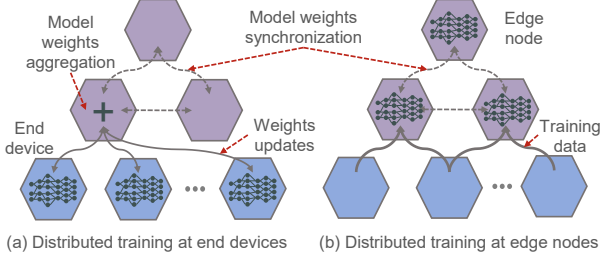


Fig. 16. Distributed DL training at edge environments.

Most of the gradient exchanges are redundant, and hence updated gradients can be compressed to cut down the communication cost while preserving the training accuracy (such as *DGC* in [156]). *First*, *DGC* stipulates that only important gradients are exchanged, i.e., only gradients larger than a heuristically given threshold are transmitted. In order to avoid the information losing, the rest of the gradients are accumulated locally until they exceed the threshold. To be noted, gradients whether being immediately transmitted or accumulated for later exchanging will be coded and compressed, and hence saving the communication cost. *Second*, considering the sparse update of gradients might harm the convergence of DL training, momentum correction and local gradient clipping are adopted to mitigate the potential risk. By momentum correction, the sparse updates can be approximately equivalent to the dense updates. Before adding the current gradient to previous accumulation on each edge device locally, gradient clipping is performed to avoid the exploding gradient problem possibly introduced by gradient accumulation. Certainly, since partial gradients are delayed for updating, it might slow down the convergence. Hence, *finally*, for preventing the stale momentum from jeopardizing the performance of training, the momentum for delayed gradients is stopped, and less aggressive learning rate and gradient sparsity are adopted at the start of training to reduce the number of extreme gradients being delayed.

With the same purpose of reducing the communication cost of synchronizing gradients and parameters during distributed training, two mechanisms can be combined together [157]. The first is transmitting only important gradients by taking advantage of sparse training gradients [158]. Hidden weights are maintained to record times of a gradient coordinate participating in gradient synchronization, and gradient coordinates with large hidden weight value are deemed as important gradients and will be more likely be selected in the next round training. On the other hand, the training convergence will be greatly harmed if residual gradient coordinates (i.e., less important gradients) are directly ignored, hence, in each training round,

small gradient values are accumulated. Then, in order to avoid that these outdated gradients only contribute little influence on the training, momentum correction, viz., setting a discount factor to correct residual gradient accumulation, is applied.

Particularly, when training a large DL model, exchanging corresponded model updates may consume more resources. Using an online version of KD can reduce such kind of communication cost [159]. In other words, the model outputs rather the updated model parameters on each device are exchanged, making the training of large-sized local models possible. Besides communication cost, privacy issues should be concerned as well. For example, in [160], personal information can be purposely obtained from training data by making use of the privacy leaking of a trained classifier. The privacy protection of training dataset at the edge is investigated in [161]. Different from [155]–[157], in the scenario of [161], training data are trained at edge nodes as well as be uploaded to the cloud for further data analysis. Hence, Laplace noises [162] are added to these possibly exposed training data for enhancing the training data privacy assurance.

B. Federated Learning at Edge

In Section VII-A, the holistic network architecture is explicitly separated, specifically, training is limited at the end devices or the edge nodes independently instead of among both of them. Certainly, by this means, it is simple to orchestrate the training process since there is no need to deal with heterogeneous computing capabilities and networking environments between the end and the edge. Nonetheless, DL training should be ubiquitous as well as DL inference. Federated Learning (FL) [163], [164] is emerged as a practical DL training mechanism among the end, the edge and the cloud. Though in the framework of native FL, modern mobile devices are taken as the clients performing local training. Naturally, these devices can be extended more widely in edge computing [165], [166]. End devices, edge nodes and servers in the cloud can be equivalently deemed as clients in FL. These clients are assumed capable of handling different levels of DL training tasks, and hence contribute their updates to the global DL model. In this section, fundamentals, improvements and practical use cases of FL are discussed.

1) *Vanilla Federated Learning*: Without requiring uploading data for central cloud training, FL [163], [164] can allow edge devices to train their local DL models with their own collected data and upload only the updated model instead. As depicted in Fig. 17, FL iteratively solicits a random set of edge devices to 1) download the global DL model from an aggregation server (use “server” in following), 2) train their local models on the downloaded global model with their own data, and 3) upload only the updated model to the server for model averaging. Privacy and security risks can be significantly reduced by restricting the training data to only the device side, and thus avoiding the privacy issues as in [160], incurred by uploading training data to the cloud. Besides, FL introduces *Federated Averaging* to combine local SGD on each device with a server performing model averaging. Experimental results corroborate *Federated Averaging* is robust

to unbalanced and non-IID data and can facilitate the training process, viz., reducing the rounds of communication needed to train a DL model.

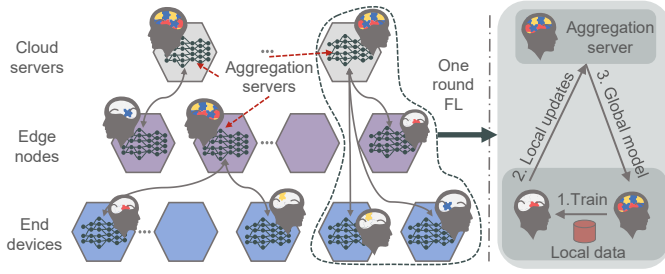


Fig. 17. Federated learning among hierarchical network architectures.

To summarize, FL can deal with several key challenges in edge computing networks: 1) **Non-IID training data**. Training data on each device is sensed and collected by itself. Hence, any individual training data of a device will not be able to represent the global one. In FL, this can be met by *Federated Averaging*; 2) **Limited communication**. Devices might potentially off-line or located in a poor communication environment. Nevertheless, performing more training computation on resource-sufficient devices can cut down communication rounds needed for global model training. In addition, FL only selects a part of devices to upload their updates in one round, therefore successfully handling the circumstance where devices are unpredictably off-line; 3) **Unbalanced contribution**. It can be tackled by *Federated Averaging*, specifically, some devices may have less free resources for FL, resulting in varying amounts of training data and training capability among devices; 4) **Privacy and security**. The data need to be uploaded in FL is only the updated DL model. Further, secure aggregation and differential privacy [162], which are useful in avoiding the disclosure of privacy-sensitive data contained in local updates, can be applied naturally.

2) *Federated Learning Optimization*: Indeed, in FL, raw training data are not required to be uploaded, thus largely reducing the communication cost. However, FL still needs to transmit locally updated models to the central server. Supposing the size of a DL model is large enough, uploading updates, such as weights of the model, from edge devices to the central server may also consume nonnegligible communication resources. To meet this challenge, [167] presents *structured update* and *sketched update* to enhance the communication efficiency when clients uploading updates to the server. *Structured update* means restricting the model updates to have a pre-specified structure, specifically, 1) low-rank matrix; or 2) sparse matrix. On the other hand, for *sketched update*, full model updates are maintained, but before sending them to the server for global model aggregation, combined operations of subsampling, probabilistic quantization, and structured random rotations will be performed to compress the full updates.

Different from only investigating on reducing communication cost on the uplink, [168] takes both server-to-device (downlink) and device-to-server (uplink) communication into consideration. For the downlink, the weights of the global DL model are reshaped into a vector, and then subsampling

and quantization are applied [167]. Naturally, such kind of model compression is lossy, and unlike on the uplink (there are multiple edge devices uploading their models for averaging), the loss cannot be mitigated by averaging on the downlink. *Kashin's representation* [169] can be utilized before subsampling as a basis transform to mitigate the error incurred by subsequent compression operations. Furthermore, for the uplink, each edge device is not required to locally train a model based on the whole global model, but only trains an update to a smaller sub-model instead. These sub-models are subsets of the global model, and by this means, reducing the amount of data in updates uploading.

Likewise, computation resources of the edge device are also scarce compared to the cloud. Additional challenges should be considered: 1) not only communication but also computation resources are limited at edge devices; 2) training dataset at different edge devices may be distributed non-uniformly [170]–[172]. To solve these challenges, the frequency of the global model aggregation under given resource budgets among all participating devices is required to be optimally determined. Based on the deduced convergence bound for distributed learning with non-IID data distributions, the aggregation frequency can be optimized with theoretical guarantees [170].

Aiming to accelerate the global aggregation in FL, [173] takes advantage of over-the-air computation [174]–[176], of which the principle is to explore the superposition property of a wireless multiple-access channel to compute the desired function by the concurrent transmission of multiple edge devices. The interferences of wireless channels can be harnessed instead of merely overcoming them. During the transmission, concurrent analog signals from edge devices can be naturally weighed by channel coefficients, and then the server only needs to superpose these reshaped weights as the aggregation results, nonetheless, without other aggregation operations.

3) *Security Enhancement*: In vanilla FL, local data samples are processed on each edge device. Such a manner can prevent the devices from revealing private data to the server. However, the server also should not trust edge devices completely, since devices with abnormal behavior can forge or poison their training data, which results in worthless model updates, and hence harming the global model. To make FL capable of tolerating a small number of devices training on the poisoned dataset, *robust federated optimization* [166] defines a trimmed mean operation. By filtering out not only the values produced by poisoned devices but also the natural outliers in the normal devices, robust aggregation protecting the global model from data poisoning is achieved.

Other than intentional attacks, passive adverse effects on the security, brought by unpredictable network conditions and computation capabilities, should be concerned as well. FL must be robust to the unexpectedly drop out of edge devices, or else once a device loses its connection, the synchronization of FL in one round will be failed. To solve this issue, *Secure Aggregation* protocol is proposed in [177] to achieve the robustness of tolerating up to one-third devices failing to timely process the local training or upload the updates.

In turn, malfunctions of the aggregation server in FL may result in inaccurate global model updates and thereby distorting

all local model updates. Besides, edge devices (with a larger number of data samples) may be less willing to participate FL with others (with less contribution). Therefore, in [178], combining Blockchain and FL as *BlockFL* is proposed to realize 1) locally global model updating at each edge device rather a specific server, ensuring device malfunction cannot affect other local updates when updating the global model; 2) appropriate reward mechanism for stimulating edge devices to participate in FL.

VIII. DEEP LEARNING FOR OPTIMIZING EDGE

DNNs (general DL models) can extract latent data features, while DRL can learn to deal with decision-making problems by interacting with the environment. Computation and storage capabilities of edge nodes, along with the collaboration of the cloud, make it possible to use DL to optimize edge computing networks and systems. With regard to various edge management issues such as edge caching, offloading, communication, security protection, etc., 1) DNNs can process user information and data metrics in the network, as well as perceiving the wireless environment and the status of edge nodes, and based on these information 2) DRL can be applied to learn the long-term optimal resource management and task scheduling strategies, so as to achieve the intelligent management of the edge, viz., intelligent edge as shown in Table V.

A. Adaptive Edge Caching Policy

From CDN (Content Delivery Network) [179] to caching contents in cellular networks, caching in the network have been investigated over the years to deal with soaring demand for multimedia services [180]. Aligned with the concept of pushing contents near to users, edge caching [181], is deemed as a promising solution for further reducing the redundant data transmission, easing the pressure of cloud data centers and improving the QoE.

Two challenges related to edge caching shall be met: 1) the content popularity distribution among the coverage of edge nodes is hard to estimate, since it may be different and change with spatio-temporal variation [182]; 2) in view of massive heterogeneous devices in edge computing environments, the hierarchical caching architecture and complex network characteristics further perplex the design of content caching strategy [183]. Specifically, the optimal edge caching strategy can only be deduced when the content popularity distribution is known. However, users' predilection for contents is actually unknown since the mobility, personal preference and connectivity of them may vary all the time. In this section, DL for determining edge caching policies, as illustrated in Fig. 18, are discussed.

1) *Use Cases of DNNs*: Traditional caching methods are generally with high computational complexity since they require a large number of online optimization iterations to determine content placement and delivery. But when using DNNs to optimize edge caching, online heavy computation iterations can be avoided by offline training, and only DL inference giving optimization strategies is required. A DNN, which consists of an encoder for data regularization and a followed hidden layer, can be trained with solutions generated by

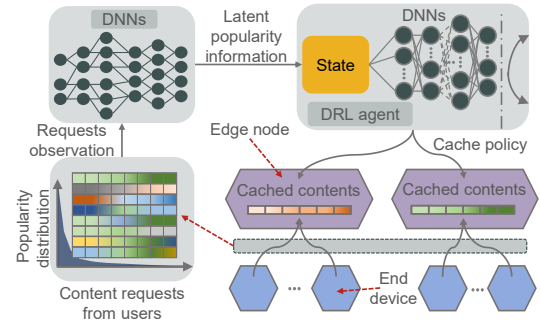


Fig. 18. DL and DRL for optimizing the edge caching policy.

optimal or heuristic algorithms and be deployed to determine the cache policy [184], hence avoiding online optimization iterations. Similarly, in [185], inspired by the fact that the output of optimization problem about partial cache refreshing has some patterns, an MLP is trained for accepting the current content popularity and the last content placement probability as input to generate the cache refresh policy.

As illustrated in [184] [185], the complexity of optimization algorithms can be transferred to the training of DNNs, and thus breaking the practical limitation of employing them. Herein, DL is used to learn input-solution relations. Apparently, DNN-based method is only available when optimization algorithms for the original caching problem exist. Therefore, the performance of DNN-based method is bounded by fixed optimization algorithms and cannot be deemed as self-adapted.

In addition, DL can be utilized for customized edge caching [186]. Specifically, on minimizing content-downloading delay in the self-driving car, an MLP is deployed in the cloud to predict the popularity of contents to be requested and then the outputs of MLP are delivered to the edge nodes (namely MEC servers at RSUs in [186]). According to these outputs, each edge node caches contents that are most likely to be requested. On self-driving cars, CNN is chosen to predict the age and gender of the owner. Once these features of owners are identified, k-means clustering [187] and binary classification algorithms are used to determine which contents, already cached in edge nodes, should be further downloaded and cached from edge nodes to the car. Moreover, with regard to taking full advantage of users' features, [188] points out that the user's willing to access the content at different environments is varying. Inspired by this, RNN can be used to predict the trajectories of users. And based on these predictions, all contents of users interest can be then prefetched and cached in advance on the edge node of each predicted location.

2) *Use Cases of DRL*: The function of DNNs described in Section VIII-A1 can be deemed as a part of the whole edge caching solution, i.e., the DNN itself does not deal with the whole optimization problem. Different from these DNNs-based edge caching, DRL can exploit the context of users and networks and take adaptive strategies for maximizing the long-term caching performance [189] as the main body of the optimization method. Traditional RL algorithms are limited by the requirement for handcrafting features and the flaw that hardly handling high-dimensional observation data and actions

[190]. Compared to traditional RL irrelevant to DL, such as *Q*-learning [191] and MAB (Multi-Armed Bandit) learning [182], the advantage of DRL lies in that DNNs can learn key features from the raw observation data. The integrated DRL agent combining RL and DL can optimize its strategies with respect to cache management in edge computing networks directly from high-dimensional observation data.

In [192], DDPG is used to train a DRL agent, in order to maximize the long-term cache hit rate, to make proper cache replacement decisions. This work considers a scenario with a single BS, in which the DRL agent decides whether to cache the requested contents or replace the cached contents. While training the DRL agent, the reward is devised as the cache hit rate. In addition, *Wolpertinger* architecture [193] is utilized to cope with the challenge of large action space. In detail, a primary action set is first set for the DRL agent and then using K-Nearest Neighbor (KNN) to map the practical action inputs to one out of this set. In this manner, the action space is narrowed deliberately without missing the optimal caching policy. Compared DQL-based algorithms searching the whole action space, the trained DRL agent with DDPG and *Wolpertinger* architecture is able to achieve competitive cache hit rates while reducing the runtime.

B. Task Offloading Optimization

Edge computing allows edge devices offload part of their computing tasks to the edge node [194], under constraints of energy, delay, computing capability, etc. As shown in Fig. 19, these constraints put forward challenges of identifying 1) which edge nodes should receive tasks, 2) what ratio of tasks edge devices should offload and 3) how many resources should be allocated to these tasks. To solve this kind of task offloading problem is NP-hard [195], since at least combination optimization of communication and computing resources along with the contention of edge devices is required. Particularly, the optimization should concern both the time-varying wireless environments (such as the varying channel quality) and requests of task offloading, hence drawing the attention of using learning methods [196]–[205]. Among all these works related to learning-based optimization methods, DL-based approaches have advantages over others when multiple edge nodes and radio channels are available for computation offloading. At this background, large state and action spaces in the whole offloading problem make the conventional learning algorithms [196] [206] [198] infeasible actually.

1) *Use Cases of DNNs*: In [199], the computation offloading problem is formulated as a multi-label classification problem. By exhaustively searching the solution in an offline way, the obtained optimal solution can be used to train a DNN with the composite state of the edge computing network as the input, and the offloading decision as the output. By this means, optimal solutions may not require to be solved online avoiding belated offloading decision making, and the computation complexity can be transferred to DL training.

Further, a particular offloading scenario with respect to Blockchain is investigated in [202]. The computing and energy resources consumption of mining tasks on edge devices may

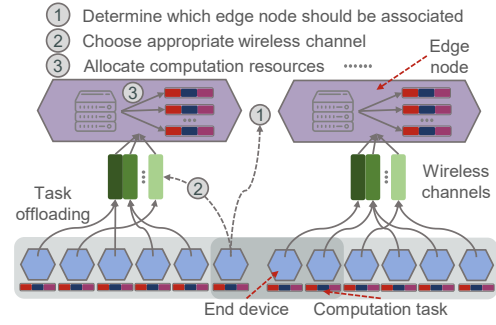


Fig. 19. Computation offloading problem in edge computing.

limit the practical application of Blockchain in the edge computing network. Naturally, these mining tasks can be offloaded from edge devices to edge nodes, but it may cause unfair edge resource allocation. Thus, all available resources are allocated in the form of auctions to maximize the revenue of the Edge Computing Service Provider (ECSP). Based on an analytical solution of the optimal auction, an MLP can be constructed [202] and trained with valuations of the miners (i.e., edge devices) for maximizing the expected revenue of ECSP.

2) *Use Cases of DRL*: Though offloading computation tasks to edge nodes can reduce the processing delays, the reliability of offloading suffers from the potentially low quality of wireless channels. For task offloading, not only the delay violation probability but also the decoding error probability should be concerned. The coding rate, by which transmitting the data, is crucial to make the offloading meet the required reliability level. Hence, in [200], effects of the coding block-length are investigated and an MDP concerning resource allocation is formulated and solved by DQL, in order to improve the average offloading reliability. Exploring further on scheduling fine-grained computing resources of the edge device, in [208], Double-DQL [74] is used to select the best Dynamic Voltage and Frequency Scaling (DVFS) algorithm. Compared to the DQL, the experiment results indicate that Double-DQL can save more energy and achieve higher training efficiency. Obviously, the action space of DQL-based approaches may increase rapidly with increasing edge devices. To narrow the size of the action space, a pre-classification step can be performed before learning [203].

IoT edge environment powered by Energy Harvesting (EH) is investigated in [201], [204]. In EH environment, the energy harvesting makes the offloading problem more complicated, since IoT edge devices can harvest energy from ambient radio-frequency signals. Hence, CNN is used to compress the state space in the learning process [204]. Further, in [201], inspired by the additive structure of the reward function, *Q*-function decomposition is applied in Double-DQL, and it improves the vanilla Double-DQL. However, value-based DRL can only deal with discrete action space. In order to perform more fine-grained power control for local execution and task offloading, policy-gradient-based DRL should be considered. For example, DDPG can be used to adaptively allocate the power of edge devices, and it possesses superiority over the discrete power control strategy based on DQL [205].

TABLE V
DL FOR OPTIMIZING EDGE APPLICATION SCENARIOS

	Related Work	DNNs or DRL	Related Methods	Objective	
Adaptive Edge Caching Policy	[184]	DL	FCNN	Estimate content popularity and deduce proactive caching strategies	
	[185]	DL	CNN	Avoid the repeated calculation of cache refreshing optimization	
	[186]	DL	FCNN	Train caching policies in the self-driving car to minimize content downloading delay	
	[188]	DL	RNN	Predict user mobility and cache contents of users interest in advance	
	[207]	DRL	AC	Schedule users and content caching to minimize the average transmission delay	
	[192]	DRL	DDPG	Maximize the long-term cache hit rate	
Task Offloading Optimization	[199]	DL	FCNN	Optimize offloading actions to minimize the computation and offloading overhead	
	[202]	DL	FCNN	Develop an optimal auction based on DL for the edge resource allocation	
	[200]	DRL	DQL	Allocate computational resource for offloaded tasks in order to improve the average end-to-end reliability	
	[208]	DRL	Double-DQL	Adaptively select dynamic voltage and frequency scaling algorithms for energy-efficient edge scheduling	
	[203]	DRL	DQL	Determine the computation offloading decision and the allocated computation resource of all edge devices	
	[205]	DRL	DDPG	Perform more fine-grained power control for local execution and task offloading	
	[204]	DRL	DQL	Choose the MEC server and determine the offloading rate in energy harvesting environment	
	[201]	DRL	Double-DQL	Optimize computation offloading policies for a virtual MEC system	
	[209]	DRL	FCNN	Reduce the computational complexity without compromising the solution quality	
Edge Management and Maintenance	Edge Communication	[210]	DL	RNN & LSTM	Assist the smooth transition of device connections and offloaded tasks between edge nodes
		[211]	DRL	DQL & TL	Control communication modes of edge devices and on-off states of their processors to minimize long-term system power consumption
	Edge Security	[212]	DRL	DQL	Provide secure offloading from the edge device to the edge node against jamming attacks
		[213]	DRL	DQL	Improve the servicing performance of the edge computing network
	Joint Edge Optimization	[207]	DRL	AC	Minimize the average end-to-end servicing delay
		[95]	DRL	Double-Dueling DQL	Improve the performance of future IoVs
		[97]	DRL	DQL	Optimize caching placement and computing resource allocation as well as to determine possible connections

Freely letting DRL agents take over the whole process of computation offloading may lead to huge computational complexity. Therefore, using DRL for partial decision making can largely reduce the complexity. In [209], maximizing the weighted sum computation rate is first formulated as the final objective of the problem. Then, the problem is decomposed into two sub-problems, viz., offloading decision and resource allocation. By only using DRL to deal with the NP-hard offloading decision problem rather both, finally, the action space of DRL agent is narrowed, and the offloading performance is not impaired by optimally resources allocating as well.

C. Edge Management and Maintenance

Edge computing services are envisioned to be deployed on BSs in cellular networks, as implemented in [214]. Therefore, edge management and maintenance require optimizations from

multiple perspectives (including communication perspective). Many works focus on applying DL in wireless communication [215]–[217]. However, there are still other perspectives should be concerned as discussed in this section.

1) *Edge Communication*: When edge nodes are serving mobile devices (users), the handover problem of edge devices in edge computing networks should be addressed. DL-based methods can be used to assist the smooth transition of connections between edge nodes [210]. Specifically, the whole handover problem is to minimize the interruptions of a mobile device moving from an edge node to the next one throughout its moving trajectory. An MLP can be used to predict available edge nodes at a given location and time. Moreover, it still needs to evaluate the cost (the latency of servicing a request) for the interaction between the mobile device and each edge node to determine the best edge node the mobile device should associate to. Nonetheless, modeling

the cost of these interactions requires a more capable learning model. Therefore, a two-layer stacked RNN with LSTM cells is implemented for modeling the cost of interaction. At last, based on the capability of predicting available edge nodes along with corresponding potential cost, the mobile device can associate with the best edge node, and hence the possibility of disruption is minimized.

To deal with the energy minimization problem in the communication scenario with multiple modes (to serve various IoT services), i.e., C-RAN (Cloud-Radio Access Networks) mode, D2D (Device-to-Device) mode, and FAP (Fog radio Access Point) mode, DQL can be used to control communication modes of edge devices and on-off states of processors throughout the communicating process [211]. Then, when the communication mode and the processors' on-off states of a given edge device are determined by the DQL agent, the whole problem is degraded into an RRH (Remote Radio Head) transmission power minimization problem and solved.

2) *Edge Security*: Due to the fact that edge devices are generally equipped with limited computation, energy and radio resources, the transmission between them and the edge node is more vulnerable to various attacks, such as jamming attacks and Distributed Denial of Service (DDoS) attacks, compared to cloud computing. Therefore, the security of edge architectures should be enhanced. Certainly, security protection generally requires additional energy consumption and the overhead of both computation and communication. Consequently, each edge device shall optimize its defense strategies, viz., choosing the transmit power, channel and time, without violating its resource limitation. The optimization is challenging since it is hard to estimate the attack model and the dynamic model of edge computing networks.

DRL-based security solutions can provide secure offloading (from the edge device to the edge node) to against jamming attacks [212] or protect user location privacy and the usage pattern privacy [218]. The edge device observes the status of edge nodes and the attack characteristics, and then determines the defense level and key parameters in security protocols. By setting the reward as the anti-jamming communication efficiency, such as the Signal-to-Interference-plus-Noise Ratio (SINR) of the signals, the Bit Error Rate (BER) of the received messages, and the protection overhead, the DQL-based security agent can be trained to cope with various types of attacks.

3) *Joint Edge Optimization*: Edge computing can cater for the rapid growth of smart devices and the advent of massive computation-intensive and data-consuming applications. Nonetheless, it also makes the operation of future networks even more complex [219]. To manage the complex networks with respect to comprehensive resource optimization [16] is challenging, particularly under the premise of considering key enablers of the future network, including Software-Defined Network (SDN) [220], IoTs, Internet of Vehicles (IoVs).

In general, SDN is designed for separating the control plane from the data plane, and thus allowing the operation over the whole network with a global view. Compared to the distributed nature of edge computing networks, SDN is a centralized approach, and it is challenging to directly apply SDN to

edge computing networks. In [213], an SDN-enabled edge computing network catering for smart cities is investigated. To improve the servicing performance of this prototype network, DQL is deployed in its control plane to orchestrate networking, caching and computing resources.

Edge computing can empower IoT with more computation-intensive and delay-sensitive services but also raises challenges for efficient management and synergy of storage, computation, and communication resources. For minimizing the average end-to-end servicing delay, policy-gradient-based DRL combined with AC architecture can deal with the assignment of edge nodes, the decision about whether to store the requesting content or not, the choice of the edge node performing the computation tasks and the allocation of computation resources [207]. IoV is a special case of IoTs and focuses on connected vehicles. Similar to the consideration of integrating networking, caching and computing as in [207], Double-Dueling DQL (i.e., combining Double DQL and Dueling DQL) with more robust performance, can be used to orchestrate available resources to improve the performance of future IoV [95].

Further, considering the mobility of vehicles in the IoVs, the hard service deadline constraint might be easily broken, and this challenge is often either neglected or tackled inadequately because of high complexities. To deal with the mobility challenge, in [97], the mobility of vehicles is first modeled as discrete random jumping, and the time dimension is split into epochs, each of which comprises several time slots. Then, a small timescale DQL model, regarding the granularity of time slot, is devised for incorporating the impact of vehicles' mobility in terms of the carefully designed immediate reward function. At last, a large timescale DQL model is proposed for every time epoch. By using such multi-timescale DRL, issues about both immediate impacts of the mobility and the unbearable large action space in the resource allocation optimization are solved.

IX. OPEN RESEARCH CHALLENGES

In order to identify unresolved issues, understand existing challenges, and circumvent potential misleading directions, we discuss issues related to the five enabling technologies separately, i.e., "DL on Edge", "DL in Edge", "Edge for DL", "DL at Edge" and "DL for Edge".

A. ("DL on Edge") More Promising Applications

With the rapid development of wireless communication techniques, the enabled pervasive connections and the high-speed uplink/downlink rates illuminate the bloom of DL services on edge, and there are many promising applications not dabbled yet, to name a few:

- *Cloud-edge gaming and VR application*: Both of them or their combination requires faster gaming interactions and video rendering [221]. With more DL techniques are universally embedded in these emerged applications, the introduced processing delay and additional computation cost make the cloud gaming architecture struggle to meet the latency requirements. Edge computing architectures,

near to users, can be leveraged with the cloud to form a hybrid gaming architecture.

- *Autonomous vehicle*: Intelligent driving involves speech recognition, image recognition, intelligent decision making, etc. Processing massive data is indispensable in autonomous vehicles, and edge computing is an important tool for real-time data processing. Various applications in intelligent driving, such as collision warning, require edge computing platforms to ensure millisecond-level interaction delay. In addition, edge computing can provide location awareness. Customized perception is more conducive to analyze the traffic environment around the vehicle, thus enhancing the driving safety.
- *Intelligent transportation*: With respect to smart cities, the challenge of building intelligent transportation systems should be met, aiming to lower the probabilities of traffic accidents and to improve traffic efficiency. For example, DRL algorithms can be deployed on the edge, e.g., RSUs, to realize adaptive traffic signal controls [222]. Therefore, it can effectively alleviate urban traffic congestion by dynamically adjusting the traffic light plan to cope with real-time traffic fluctuations, thereby facilitating people's lives.

To summarize, if DL and edge are well-integrated, they can offer great potential for the development of innovative applications. There are still many areas to be explored to provide operators, suppliers and third parties with new business opportunities and revenue streams.

B. (“DL in Edge”) General DL Model for Inference

When deploying DL in edge devices, it is necessary to accelerate DL inference by model optimization. In this section, challenges with respect to model compression, model segmentation and EEoI, used to optimize DL models, is discussed.

1) *Generalization of EEoI*: Currently, EEoI can be applied to classification problems in DL [139], but there is no generalized solution for a wider range of DL applications. Furthermore, in order to build an intelligent edge and support edge intelligence, not only DL but also the possibility of applying EEoI to DRL should be explored, since applying DRL to real-time resource management for the edge, as discussed in Section VIII, requires stringent response speed.

2) *Hybrid Model Modification*: Coordination issues with respect to model optimization, model segmentation, and EEoI should be thought over. These customized DL models are often used independently to enable “end-edge-cloud” collaboration. Model optimizations, such as model quantification and pruning, may be required on the end and edge sides, but because of the sufficient computation resources, the cloud does not need to take the risk of model accuracy to use these optimizations. Therefore, how to design a hybrid precision scheme, that is, to effectively combine the simplified DL models in the edge with the raw DL model in the cloud is important.

3) *Coordination between training and inference*: Pruning, quantizing and introducing EEoI into trained raw DL models require retraining to give them the desired inference performance. In general, customized models can be trained offline

in the cloud. However, the advantage of edge computing lies in its response speed and might be neutralized because of belated DL training. Moreover, due to a large number of heterogeneous devices in the edge and the dynamic network environment, the customization requirements of DL models are not monotonous. Then, is this continuous model training requirement reasonable, and will it affect the timeliness of model inference? How to design a mechanism to avoid these side-effects?

C. (“Edge for DL”) Complete Edge Architecture for DL

Edge intelligence and intelligent edge require a complete system framework, covering data acquisition, service deployment and task processing. In this section, we discuss the efforts that need to be made to build a complete edge computing framework for DL, as summarized in Table VI.

1) *Edge for Data Processing*: Both pervasively deployed DL services on the edge and DL algorithms for optimizing edge cannot be realized without data acquiring. Edge architecture should be able to efficiently acquire and process the original data, sensed or collected by edge devices, and then feed them to DL models.

Adaptively acquiring data at the edge and then transmitting them to cloud (as done in [7]) is a natural way to alleviate the workload of edge devices and to reduce the potential resource overhead. In addition, it is better to further compress the data, which can alleviate the bandwidth pressure of the network, while the transmission delay can be reduced to provide better QoS. Most existed works focus only on vision applications [89]. However, the heterogeneous data structures and characteristics of a wide variety of DL-based services are not addressed well yet. Therefore, developing a heterogeneous, parallel and collaborative architecture for edge data processing for various DL services will be helpful.

2) *Reduction of DL Computation*: Users within the same area might request recognition results of similar interesting objects, and it may introduce redundant DL inference tasks. By caching requesting results and necessary DL models [223] or sharing intermediate DL inference results among different DL models [224], repetitive or similar requests can be directly disposed at the edge. In this manner, the processing delay can be largely reduced without bringing too much extra computation. Nevertheless, similar works [225] [226] concerns DL computation without dabbling DRL applications, which might be essential for the near future.

3) *Microservice for DL Services*: Edge and cloud services have recently started undergoing a major shift from monolithic entities to graphs of hundreds of loosely-coupled microservices [227]. Executing DL computations may need a series of software dependencies, and it calls for a solution for isolating different DL services on the shared resources. Naturally, Virtual Machine (VM) and Docker (noted Unikernels [228] and Docker complement each other) are both feasible for this purpose. However, compared to VM, Docker can provide faster deployment, smaller footprint, better performance and finer-grained resource management, which make it potentially more appropriate for edge computing [229].

TABLE VI
RESEARCH DIRECTIONS AND CHALLENGES OF “EDGE FOR DL”

Research Direction	Expected Objective	Challenge or Potential Solution	Metric
Edge for data processing	<ul style="list-style-type: none"> Efficiently acquire and process massive sensed or collected data by edge infrastructures 	<ul style="list-style-type: none"> Adaptive data acquirement Adaptive data compression Universal or customized architecture 	<ul style="list-style-type: none"> Overhead, Processing speed, Throughput
Reduction of DL computation	<ul style="list-style-type: none"> Dispose DL requests at the edge without redundant computation 	<ul style="list-style-type: none"> DRL support Cache inference results Deploy multiple DL models Sharing intermediate DL inference results 	<ul style="list-style-type: none"> Overhead, Inference latency, Model accuracy
Microservice for DL services	<ul style="list-style-type: none"> Host DL services on microservice frameworks deployed on the edge 	<ul style="list-style-type: none"> Microservice framework for edge computing Live migration of microservices Coordination between the cloud and the edge 	<ul style="list-style-type: none"> Overhead, Throughput, Stability
Incentive and trusty offloading	<ul style="list-style-type: none"> Provide edge nodes with incentives to take over DL computations Guarantee the security to avoid the risks from anonymous edge nodes 	<ul style="list-style-type: none"> Apply blockchain in the edge architecture Existing blockchain solutions do not support the execution of DL 	<ul style="list-style-type: none"> Overhead, Throughput, Security, Inference latency
Edge architecture and testbed for DL	<ul style="list-style-type: none"> Automate phases of the DL development process Practically supporting ideas in Section VI-B Testify and optimize the performance of DL services on the edge 	<ul style="list-style-type: none"> Exploit underlying edge hardware in terms of performance and power Develop a standard and comprehensive testbed 	<ul style="list-style-type: none"> Overhead, Inference latency, Throughput, Stability

At present, microservice frameworks deployed on the edge for hosting DL services are still not investigated, due to several important challenges: 1) microservice frameworks should be flexible enough to handle DL deployment and management; 2) the live migration of microservices should be achieved to cater for the user mobilities, in order to reduce migration times and unavailability of DL services; 3) the microservice framework should be able to orchestrate the cloud and distributed edge infrastructures, leveraging computing resources of both sides to achieve better performance as illustrated in Section VI-B3.

4) Incentive and trusty offloading mechanism for DL:

Heavy DL computations on resource-limited end devices can be offloaded to nearby edge nodes (Section VI-B). However, there are still several issues, 1) an incentive mechanism should be established for stimulating edge nodes to take over DL computations; 2) the security should be guaranteed to avoid the risks from anonymous edge nodes [230].

Blockchain, as a decentralized public database storing transaction records across participated devices, can avoid the risk of tampering the records [231]. By taking advantage of these characteristics, incentive and trust problems with respect to computation offloading can potentially be tackled. To be specific, all end devices and edge nodes have to first put down deposits to the blockchain to participate. The end device request the help of edge nodes for DL computation, and meantime send a “require” transaction to the blockchain with a bounty. Once an edge nodes complete the computation, it returns results to the end device with sending a “complete” transaction to the blockchain. After a while, other participated edge nodes also execute the offloaded task and validate the former recorded result. At last, for incentives, firstly recorded edge nodes win the game and be awarded [232]. However, this idea about blockchained edge is still in its infancy. Existing blockchains such as Ethereum [233] do not support the execution of complex DL computations, which raises the challenge of adjusting blockchain structure and protocol in order to break this limitation.

5) *Edge Architecture and Testbed for DL*: Based on edge hardware, additional software architecture shall also be developed for efficiently 1) exploiting underlying hardware in terms of performance and power; 2) configuring DL models and maintaining DL services automatically. In [234], to meet these requirements, *ALOHA* is proposed as an integrated framework to automate most phases of the DL development process. By means of the architecture-awareness of *ALOHA*, the development process of DL inference on the edge is automated. Similarly, [235] explores an approach of DL development and deployment targeting to the edge, and [236] focus on the DL deployment among BSs as the edge platform. In regard to services composition and deployment, a system architecture *Zoo* is proposed to enable easy and type-safe composition of different data analytics services [237]. However, these pioneer works cannot support valuable and also challenging features mentioned in Section VI-B, such as computation offloading and collaboration.

Besides, a standard testbed is missing, which hinders the study of edge architectures for DL. To testify the end-to-end performance of DL services, not only the edge computing architecture but also its combination with end devices and the cloud shall be established. End-to-end emulators for MEC, such as *openLEON* proposed in [238] and *CAVBench* [239] particular for vehicular scenario, can be used as the experimental platforms. Nevertheless, simulations of the control panel of managing DL services are still not dabbled. An integrated testbed, consisting of wireless links and networking models, service requesting simulation, edge computing platforms, cloud architectures, etc., will be ponderable in facilitating the evolution of “Edge for DL”.

D. (“DL at Edge”) Practical Training Principles

Compared with DL inference in the edge, DL training at the edge is currently mainly limited by the weak performance of edge devices and the fact that DL frameworks or libraries (available for edge devices) still do not support training. At

present, most studies are at the theoretical level, i.e., simulating the process of DL training at the edge. In this section, we point out the challenges faced by “DL at Edge”.

1) *Data Parallelism versus Model Parallelism*: DL models are both computation and memory intensive. When they become deeper and larger, it is not feasible to acquire their inference results or complete their training by a single device. Therefore, large DL models are trained in distributed manners, in terms of data parallelism, model parallelism or their combination (Section III-C), over thousands of CPU or GPU cores during training. However, differing from parallel training over bus- or switch-connected CPUs or GPUs in the cloud, perform model training at distributed edge devices should further consider wireless environments, device configurations, data privacies, etc.

At present, FL only copies the whole DL model to every participated edge devices, namely in the manner of data parallelism. Hence, taking the limited computing capabilities of edge devices (at least for now) into consideration, partitioning a large-scale DL model and allocating these segments to different edge devices for training may be a more feasible and practical solution. Certainly, this does not mean abandoning the native data parallelism of FL, instead, posing the challenge of blending data parallelism and model parallelism particularly for training DL models at the edge, as illustrated in Fig. 20.

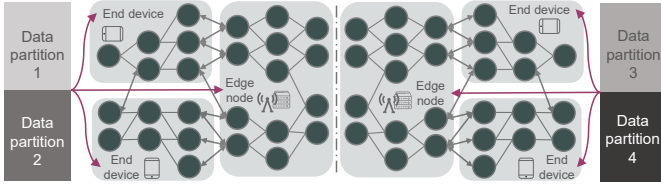


Fig. 20. DL training at the edge by both data parallelism and parallelism.

2) *Improving FL at Edge*: Existing FL methods [163], [164] focus on synchronous training, and can only process hundreds of devices in parallel. However, this synchronous updating mode potentially cannot scale well, and is inefficient and inflexible in view of two key properties of FL, specifically, 1) infrequent training tasks, since edge devices typically have weaker computing power and limited endurance and thus cannot afford intensive training tasks; 2) limited and uncertain communication between edge devices, compared to typical distributed training in the cloud.

Thus, whenever the global model is updating, the server is limited to selecting from a subset of available edge devices to trigger a training task. In addition, due to limited computing power and battery endurance, task scheduling varies from device to device, making it difficult to synchronize selected devices at the end of each epoch. Some devices may no longer be available when they should be synchronized, and hence the server must determine the timeout threshold to discard the laggard. If the number of surviving devices is too small, the server has to discard the entire epoch including all received updates. These bottlenecks in FL can potentially be addressed by asynchronous training mechanisms [240], [241].

In addition, the aim of existing FL works is to train a single global model. Nonetheless, as edge intelligence continues to

bloom, there shall be more different DL models that need to be trained and deployed on different edge nodes or devices. Adjusting FL to train separate and different kinds of models still remains as a challenge [242], [243].

3) *Transfer Learning-based Training*: Due to resource constraints, training and deploying computation-intensive DL models on edge devices such as mobile phones is challenging. In order to facilitate learning on such resource-constrained edge devices, TL can be utilized. For instance, in order to reduce the amount of training data and speeding up the training process, using unlabeled data to transfer knowledge between edge devices can be adopted [244]. By using the cross-modal transfer in the learning of edge devices across different sensing modalities, required labeled data and the training process can be largely reduced and accelerated, respectively.

Besides, KD, as a method of TL, can also be exploited thanks to several advantages [121]: 1) using information from well-trained large DL models (teachers) to help lightweight DL models (students), expected to be deployed on edge devices, converge faster; 2) improving the accuracy of students; 3) helping students become more general instead of being overfitted by a certain set of data. Although results of [121], [244] show some prospects, further research is needed to extend the TL-based training method to DL applications with different types of perceptual data.

E. (“DL for Edge”) Deployment and Improvement of Intelligent Edge

At present, there have been many attempts to use DL to optimize and schedule resources in edge computing networks. In this regard, there are many potential areas where DL can be applied, including online content streaming [245], routing and traffic control [246] [247], etc. However, because DL solutions do not rely entirely on accurate modeling of networks and devices, finding a scenario where DL can be applied is not the most important concern. Besides, if DL is applied to the optimization of real-time edge computing networks, the training and inference of DL models or DRL algorithms may bring certain side effects, such as the additional bandwidth consumed by training data transmission and the latency of DL inference.

Existing works mainly concern about solutions of “DL for Edge” at the high level, but overlook the practical feasibility at the low level. Though DL show its performance in these works, as illustrated in Fig. 21, the deployment issues of DL models and DRL algorithms should also be carefully considered:

- Where DL and DRL should be deployed, in view of the resource overhead of them and the requirement of managing edge computing networks in real time?
- When using DL to determine caching policies or optimize task offloading, will the benefits of DL be neutralized by the bandwidth consumption and the processing delay brought by DL itself?
- How to explore and improve edge computing architectures in Section VI to support “DL for Edge”?
- Are the ideas of customized DL models, introduced in Section V, can help to facilitate the practical deployment?

- How to modify the training principles in Section VII to enhance the performance of DL training, in order to meet the timeliness of edge management?

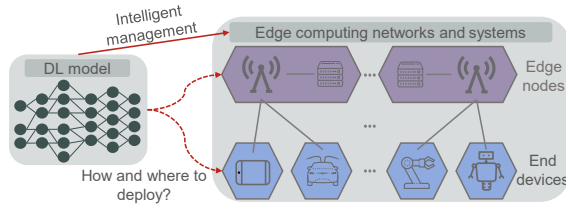


Fig. 21. Deployment issues of intelligent edge, i.e., how and where to deploy DL models for optimizing edge computing networks (systems).

Besides, the abilities of the state-of-the-art DL or DRL, such as Multi-Agent Deep Reinforcement Learning [248]–[250], Graph Neural Networks (GNNs) [251], [252], can also be exploited to facilitate this process. For example, end devices, edge nodes, and the cloud can be deemed as individual agents. By this means, each agent trains its own strategy according to its local imperfect observations, and all participated agents work together for optimizing edge computing networks. In addition, the structure of edge computing networks across the end, the edge, and the cloud is actually an immense graph, which comprises massive latent structure information, e.g., the connection and bandwidth between devices. For better understanding edge computing networks, GNNs, which focuses on extracting features from graph structures instead of two-dimensional meshes and one-dimensional sequences, might be a promising method.

X. CONCLUSIONS

Deep learning, as a representative of artificial intelligence, and edge computing is expected to benefit each other. This survey comprehensively introduces and discusses various applicable scenarios and fundamental enabling techniques for *edge intelligence* and *intelligent edge*. In summary, the key issue of extending deep learning from the cloud to the edge of the network is: under the multiple constraints of networking, communication, computing power, and energy consumption, devising and developing edge computing architecture to achieve the best performance of DL training and inference. As the computing power of the edge increases, edge intelligence will become common, and the intelligent edge will play an important supporting role to improve the performance and efficiency of edge intelligence. We hope the integration of both two can cause widespread concerns and discussions, bringing evolutions to more applications and scenarios.

REFERENCES

- [1] “Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are.” [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf
- [2] “Cisco Global Cloud Index: Forecast and Methodology.” [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html>
- [3] M. V. Barbera, S. Kosta, A. Mei *et al.*, “To offload or not to offload? The bandwidth and energy costs of mobile cloud computing,” in *2013 IEEE Conference on Computer Communications (INFOCOM 2013)*, 2013, pp. 1285–1293.
- [4] W. Hu, Y. Gao, K. Ha *et al.*, “Quantifying the Impact of Edge Computing on Mobile Applications,” in *Proc. 7th ACM SIGOPS Asia-Pacific Workshop Syst. (APSys 2016)*, 2016, pp. 1–8.
- [5] “Mobile-Edge Computing Introductory Technical White Paper,” ETSI. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf
- [6] W. Shi, J. Cao *et al.*, “Edge Computing: Vision and Challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
- [7] B. A. Mudassar, J. H. Ko, and S. Mukhopadhyay, “Edge-cloud collaborative processing for intelligent internet of things,” in *Proc. the 55th Annual Design Automation Conference (DAC 2018)*, 2018, pp. 1–6.
- [8] A. Yousefpour, C. Fung, T. Nguyen *et al.*, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, 2019.
- [9] J. Redmon, S. Divvala *et al.*, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 2016, pp. 779–788.
- [10] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.
- [11] H. Khelifi, S. Luo, B. Nour *et al.*, “Bringing deep learning at the edge of information-centric internet of things,” *IEEE Commun. Lett.*, vol. 23, no. 1, pp. 52–55, Jan. 2019.
- [12] Y. Kang, J. Hauswald, C. Gao *et al.*, “Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge,” in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS 2017)*, 2017, pp. 615–629.
- [13] “Democratizing AI.” [Online]. Available: <https://news.microsoft.com/features/democratizing-ai/>
- [14] Y. Yang, “Multi-tier computing networks for intelligent IoT,” *Nature Electronics*, vol. 2, no. 1, pp. 4–5, Jan. 2019.
- [15] C. Li, Y. Xue, J. Wang *et al.*, “Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management,” *ACM Comput. Surv.*, vol. 51, no. 2, pp. 1–34, Apr. 2018.
- [16] S. Wang, X. Zhang, Y. Zhang *et al.*, “A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications,” *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
- [17] T. X. Tran, A. Hajisami *et al.*, “Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges,” *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.
- [18] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, “Wireless Network Intelligence at the Edge,” *arXiv preprint arXiv:1812.02858 (to be published in IEEE Proceedings)*, 2018.
- [19] Z. Zhou, X. Chen *et al.*, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing,” *arXiv preprint arXiv:1905.10083 (submitted to IEEE Proceedings)*, 2019.
- [20] C. Mouradian, D. Naboulsi, S. Yangui *et al.*, “A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges,” *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 2018.
- [21] K. Bilal, A. Khalid, A. Erbad, and S. U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers,” *Comput. Networks*, vol. 130, no. 2018, pp. 94–120, 2018.
- [22] M. Satyanarayanan, P. Bahl, R. Cceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, 2009.
- [23] M. Aazam and E. Huh, “Fog computing micro datacenter based dynamic resource estimation and pricing model for iot,” in *Proc. IEEE 29th International Conference on Advanced Information Networking and Applications (AINA 2019)*, Mar. 2015, pp. 687–694.
- [24] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proc. the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.
- [25] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Cham: Springer International Publishing, 2014, pp. 169–186.
- [26] “Multi-access Edge Computing.” [Online]. Available: <http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing>
- [27] “What is Azure Data Box Edge?” [Online]. Available: <https://docs.microsoft.com/zh-cn/azure/databox-online/data-box-edge-overview>
- [28] “Intel Movidius Neural Compute Stick.” [Online]. Available: <https://software.intel.com/en-us/movidius-ncs>
- [29] “Latest Jetson Products.” [Online]. Available: <https://developer.nvidia.com/buy-jetson>
- [30] “An all-scenario AI infrastructure solution that bridges ‘device, edge, and cloud’ and delivers unrivaled compute power to lead you towards an AI-fueled future.” [Online]. Available: <https://e.huawei.com/en/solutions/business-needs/data-center/atlas>

- [31] “Snapdragon 8 Series Mobile Platforms.” [Online]. Available: <https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms>
- [32] “Kirin.” [Online]. Available: <http://www.hisilicon.com/en/Products/ProductList/Kirin>
- [33] “The World’s First Full-Stack All-Scenario AI Chip.” [Online]. Available: <http://www.hisilicon.com/en/Products/ProductList/Ascend>
- [34] “MediaTek Helio P60.” [Online]. Available: <https://www.mediatek.com/products/smartphones/mediatek-helio-p60>
- [35] “NVIDIA Turing GPU Architecture.” [Online]. Available: <https://www.nvidia.com/en-us/geforce/turing/>
- [36] N. P. Jouppi, A. Borchers, R. Boyle, P. L. Cantin, and B. Nan, “In-Datcenter Performance Analysis of a Tensor Processing Unit,” in *Proc. 44th Int. Symp. Comput. Archit. (ISCA 2017)*, 2017, pp. 1–12.
- [37] “Intel Xeon Processor D-2100 Product Brief: Advanced Intelligence for High-Density Edge Solutions.” [Online]. Available: <https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/d-2100-brief.html>
- [38] “Mobile Processor: Exynos 9820.” [Online]. Available: <https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/>
- [39] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, “Extend Cloud to Edge with KubeEdge,” in *Proc. 2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)*, 2018, pp. 373–377.
- [40] “OpenEdge, extend cloud computing, data and service seamlessly to edge devices.” [Online]. Available: <https://github.com/baidu/openedge>
- [41] “Azure IoT Edge, extend cloud intelligence and analytics to edge devices.” [Online]. Available: <https://github.com/Azure/iotedge>
- [42] “EdgeX, the Open Platform for the IoT Edge.” [Online]. Available: <https://www.edgexfoundry.org/>
- [43] “Akraio Edge Stack.” [Online]. Available: <https://www.lfedge.org/projects/akraio/>
- [44] “NVIDIA EGX Edge Computing Platform: Real-Time AI at the Edge.” [Online]. Available: <https://www.nvidia.com/en-us/data-center/products/egx-edge-computing/>
- [45] “AWS IoT Greengrass: Bring local compute, messaging, data caching, sync, and ML inference capabilities to edge devices.” [Online]. Available: <https://aws.amazon.com/greengrass/>
- [46] G. Li, L. Liu, X. Wang *et al.*, “Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge,” in *Proc. International Conference on Artificial Neural Networks (ICANN 2018)*, 2018, pp. 402–411.
- [47] Y. Huang, Y. Zhu, X. Fan *et al.*, “Task Scheduling with Optimized Transmission Time in Collaborative Cloud-Edge Learning,” in *Proc. 27th International Conference on Computer Communication and Networks (ICCCN 2018)*, 2018, pp. 1–9.
- [48] E. Nurvitadhi, G. Venkatesh, J. Sim *et al.*, “Can fpgas beat gpus in accelerating next-generation deep neural networks?” in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017)*, 2017, pp. 5–14.
- [49] S. Jiang, D. He, C. Yang *et al.*, “Accelerating Mobile Applications at the Network Edge with Software-Programmable FPGAs,” in *2018 IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018, pp. 55–62.
- [50] “Qualcomm Neural Processing SDK for AI.” [Online]. Available: <https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk>
- [51] A. Ignatov, R. Timofte, W. Chou *et al.*, “AI Benchmark: Running Deep Neural Networks on Android Smartphones,” *arXiv preprint arXiv:1810.01109*.
- [52] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sep. 2014.
- [53] “Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit.” [Online]. Available: <https://github.com/microsoft/CNTK>
- [54] S. Tokui, K. Oono *et al.*, “Chainer: a next-generation open source framework for deep learning,” in *Proc. workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NeurIPS 2015)*, 2015, pp. 1–6.
- [55] M. Abadi, P. Barham *et al.*, “TensorFlow: A System for Large-Scale Machine Learning,” in *Proc. the 12th USENIX conference on Operating Systems Design and Implementation (OSDI 2016)*, 2016, pp. 265–283.
- [56] “Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0.” [Online]. Available: <https://deeplearning4j.org>
- [57] “Deploy machine learning models on mobile and IoT devices.” [Online]. Available: <https://www.tensorflow.org/lite>
- [58] T. Chen, M. Li, Y. Li *et al.*, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,” *arXiv preprint arXiv:1512.01274*, 2015.
- [59] “PyTorch: tensors and dynamic neural networks in Python with strong GPU acceleration.” [Online]. Available: <https://github.com/pytorch/>
- [60] “Core ML: Integrate machine learning models into your app.” [Online]. Available: <https://developer.apple.com/documentation/coreml?language=objc>
- [61] “NCNN is a high-performance neural network inference framework optimized for the mobile platform.” [Online]. Available: <https://github.com/Tencent/ncnn>
- [62] “MNN is a lightweight deep neural network inference engine.” [Online]. Available: <https://github.com/alibaba/MNN>
- [63] “Multi-platform embedded deep learning framework.” [Online]. Available: <https://github.com/PaddlePaddle/paddle-mobile>
- [64] “MACE is a deep learning inference framework optimized for mobile heterogeneous computing platforms.” [Online]. Available: <https://github.com/XiaoMi/mace>
- [65] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
- [66] S. S. Haykin and K. Elektroingenieur, *Neural networks and learning machines*. Pearson Prentice Hall, 2009.
- [67] R. Collobert and S. Bengio, “Links between perceptrons, MLPs and SVMs,” in *Proc. the Twenty-first international conference on Machine learning (ICML 2004)*, 2004, p. 23.
- [68] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.
- [69] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *2014 European Conference on Computer Vision (ECCV 2014)*, 2014, pp. 818–833.
- [70] I. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)*, 2014, pp. 2672–2680.
- [71] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [72] S. S. Mousavi, M. Schukat, and E. Howley, “Deep Reinforcement Learning: An Overview,” in *Proc. the 2016 SAI Intelligent Systems Conference (IntelliSys 2016)*, 2016, pp. 426–440.
- [73] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [74] H. Van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-Learning,” in *Proc. the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 2016, pp. 2094–2100.
- [75] Z. Wang, T. Schaul, M. Hessel *et al.*, “Dueling network architectures for deep reinforcement learning,” in *Proc. the 33rd International Conference on Machine Learning (ICML 2016)*, 2016, pp. 1995–2003.
- [76] T. P. Lillicrap, J. J. Hunt, A. Pritzel *et al.*, “Continuous control with deep reinforcement learning,” in *Proc. the 6th International Conference on Learning Representations (ICLR 2016)*, 2016.
- [77] V. Mnih, A. P. Badia, M. Mirza *et al.*, “Asynchronous Methods for Deep Reinforcement Learning,” in *Proc. the 33rd International Conference on Machine Learning (ICML 2016)*, 2016, pp. 1928–1937.
- [78] J. Schulman, F. Wolski, P. Dhariwal *et al.*, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [79] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proc. the 12th International Conference on Neural Information Processing Systems (NeurIPS 1999)*, 1999, pp. 1057–1063.
- [80] Monin and Yaglom, “Large Scale Distributed Deep Networks,” in *Proc. Advances in Neural Information Processing Systems 25 (NeurIPS 2012)*, 2012, pp. 1223–1231.
- [81] Y. Zou, X. Jin, Y. Li *et al.*, “Mariana: Tencent deep learning platform and its applications,” in *Proc. VLDB Endow.*, vol. 7, no. 13, 2014, pp. 1772–1777.
- [82] X. Chen, A. Eversole, G. Li *et al.*, “Pipelined Back-Propagation for Context-Dependent Deep Neural Networks,” in *13th Annual Conference of the International Speech Communication Association (INTERSPEECH 2012)*, 2012, pp. 26–29.
- [83] M. Stevenson, R. Winter *et al.*, “1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs,” in *15th Annual Conference of the International Speech Communication Association (INTERSPEECH 2014)*, 2014, pp. 1058–1062.
- [84] A. Coates, B. Huval, T. Wang *et al.*, “Deep learning with cots hpc systems,” in *Proc. the 30th International Conference on Machine Learning (PMLR 2013)*, 2013, pp. 1337–1345.
- [85] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, “SparkNet: Training Deep Networks in Spark,” *arXiv preprint arXiv:1511.06051*, 2015.
- [86] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

- [87] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
- [88] "Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently." [Online]. Available: <https://github.com/Theano/Theano>
- [89] J. Ren, Y. Guo, D. Zhang *et al.*, "Distributed and Efficient Object Detection in Edge Computing: Challenges and Solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, Nov. 2018.
- [90] C. Liu, Y. Cao, Y. Luo *et al.*, "A New Deep Learning-Based Food Recognition System for Dietary Assessment on An Edge Computing Service Infrastructure," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 249–261, Mar. 2018.
- [91] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition," in *Proc. the First ACM/IEEE Symposium on Edge Computing (SEC 2016)*, 2016, pp. 64–76.
- [92] S. Yi, Z. Hao, Q. Zhang *et al.*, "LAVEA: Latency-aware Video Analytics on Edge Computing Platform," in *Proc. the Second ACM/IEEE Symposium on Edge Computing (SEC 2017)*, 2017, pp. 1–13.
- [93] P. Liu, B. Qi, and S. Banerjee, "EdgeEye - An Edge Service Framework for Real-time Intelligent Video Analytics," in *Proc. the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2018)*, 2018, pp. 1–6.
- [94] S. Y. Nikouei, Y. Chen, S. Song *et al.*, "Smart surveillance as an edge network service: From harr-cascade, svm to a lightweight cnn," in *IEEE 4th International Conference on Collaboration and Internet Computing (CIC 2018)*, 2018, pp. 256–265.
- [95] Y. He, N. Zhao *et al.*, "Integrated Networking, Caching, and Computing for Connected Vehicles: A Deep Reinforcement Learning Approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.
- [96] Q. Qi and Z. Ma, "Vehicular Edge Computing via Deep Reinforcement Learning," *arXiv preprint arXiv:1901.04290*, 2018.
- [97] L. T. Tan and R. Q. Hu, "Mobility-Aware Edge Caching and Computing in Vehicle Networks: A Deep Reinforcement Learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10 190–10 203, Nov. 2018.
- [98] L. Li, K. Ota, and M. Dong, "Deep Learning for Smart Industry: Efficient Manufacture Inspection System with Fog Computing," *IEEE Trans. Ind. Inf.*, vol. 14, no. 10, pp. 4665–4673, 2018.
- [99] L. Hu, Y. Miao, G. Wu *et al.*, "iRobot-Factory: An intelligent robot factory based on cognitive manufacturing and edge computing," *Future Gener. Comput. Syst.*, vol. 90, pp. 569–577, Jan. 2019.
- [100] J. A. C. Soto, M. Jentsch *et al.*, "CEML: Mixing and moving complex event processing and machine learning to the edge of the network for IoT applications," in *Proc. the 6th International Conference on the Internet of Things (IoT 2016)*, 2016, pp. 103–110.
- [101] G. Plastiras, M. Terzi, C. Kyrkou, and T. Theodoridis, "Edge Intelligence: Challenges and Opportunities of Near-Sensor Machine Learning Applications," in *Proc. IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2018)*, Jul. 2018, pp. 1–7.
- [102] Y. Hao, Y. Miao, Y. Tian *et al.*, "Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT," *arXiv preprint arXiv:1901.02126*, 2019.
- [103] S. Liu, P. Si, M. Xu *et al.*, "Edge Big Data-Enabled Low-Cost Indoor Localization Based on Bayesian Analysis of RSS," in *Proc. 2017 IEEE Wireless Communications and Networking Conference (WCNC 2017)*, 2017, pp. 1–6.
- [104] A. Dhakal *et al.*, "Machine learning at the network edge for automated home intrusion monitoring," in *Proc. IEEE 25th International Conference on Network Protocols (ICNP 2017)*, 2017, pp. 1–6.
- [105] N. Tian, J. Chen, M. Ma *et al.*, "A Fog Robotic System for Dynamic Visual Servoing," *arXiv preprint arXiv:1809.06716*, 2018.
- [106] L. Lu, L. Xu, B. Xu *et al.*, "Fog Computing Approach for Music Cognition System Based on Machine Learning Algorithm," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 4, pp. 1142–1151, Dec. 2018.
- [107] B. Tang, Z. Chen, G. Heffernan *et al.*, "Incorporating Intelligence in Fog Computing for Big Data Analysis in Smart Cities," *IEEE Trans. Ind. Inf.*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017.
- [108] Y.-C. Chang and Y.-H. Lai, "Campus Edge Computing Network Based on IoT Street Lighting Nodes," *IEEE Syst. J. (Early Access)*, 2018.
- [109] E. Denton *et al.*, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation," in *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)*, 2014, pp. 1269–1277.
- [110] W. Chen, J. Wilson, S. Tyree *et al.*, "Compressing Neural Networks with the Hashing Trick," in *Proc. the 32nd International Conference on International Conference on Machine Learning (ICML 2015)*, 2015, pp. 2285–2294.
- [111] C. Szegedy, Wei Liu, Yangqing Jia *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, 2015, pp. 1–9.
- [112] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 2016, pp. 770–778.
- [113] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *arXiv preprint arXiv:1710.09282*, 2017.
- [114] S. Han, J. Pool, J. Tran *et al.*, "Learning both Weights and Connections for Efficient Neural Networks," in *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, 2015, pp. 1135–1143.
- [115] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2016)*, 2016, pp. 1–12.
- [116] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, 2015, pp. 3123–3131.
- [117] M. Rastegari, V. Ordonez *et al.*, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *European Conference on Computer Vision (ECCV 2016)*, 2016, pp. 525–542.
- [118] B. McDaniel, "Embedded Binarized Neural Networks," in *Proc. the 2017 International Conference on Embedded Wireless Systems and Networks (EWSN 2017)*, 2017, pp. 168–173.
- [119] F. N. Iandola, S. Han, M. W. Moskewicz *et al.*, "Squeezenet: Alexnet-level Accuracy with 50x Fewer Parameters and < 0.5 MB Model Size," *arXiv preprint arXiv:1602.07360*, 2016.
- [120] A. G. Howard, M. Zhu, B. Chen *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [121] R. Sharma, S. Biookaghazadeh *et al.*, "Are Existing Knowledge Transfer Techniques Effective For Deep Learning on Edge Devices?" in *Proc. the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2018)*, 2018, pp. 15–16.
- [122] S. Y. Nikouei *et al.*, "Real-time human detection as an edge service enabled by a lightweight cnn," in *2018 IEEE International Conference on Edge Computing (IEEE EDGE 2018)*, 2018, pp. 125–129.
- [123] Fox, "Homer simpson." [Online]. Available: https://simpsons.fandom.com/wiki/File:Homer_Simpson.svg
- [124] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, 2018, pp. 6848–6856.
- [125] L. Du *et al.*, "A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 65, no. 1, pp. 198–208, Jan. 2018.
- [126] S. Han, Y. Wang, H. Yang *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *Proc. the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017)*, 2017, pp. 75–84.
- [127] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *Proc. the 6th International Conference on Learning Representations (ICLR 2016)*, 2016.
- [128] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proc. the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM (SenSys 2016)*, 2016, pp. 176–189.
- [129] L. Lai and N. Suda, "Enabling deep learning at the IoT edge," in *Proc. the International Conference on Computer-Aided Design (ICCAD 2018)*, 2018, pp. 1–6.
- [130] S. Yao, Y. Zhao, A. Zhang *et al.*, "DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework," in *Proc. the 15th ACM Conference on Embedded Network Sensor Systems (SenSys 2017)*, 2017, pp. 1–14.
- [131] S. Han, H. Shen, M. Philipose *et al.*, "MCDNN: An Execution Framework for Deep Neural Networks on Resource-Constrained Devices," in *Proc. the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2016)*, 2016, pp. 123–136.
- [132] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA 2016)*, 2016, pp. 243–254.
- [133] N. D. Lane, S. Bhattacharya, P. Georgiev *et al.*, "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," in *15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2016)*, 2016, pp. 1–12.

- [134] J. Zhang *et al.*, “A Locally Distributed Mobile Computing Framework for DNN based Android Applications,” in *Proc. the Tenth Asia-Pacific Symposium on Internetwork (Internetwork 2018)*, 2018, pp. 1–6.
- [135] Z. Zhao, K. M. Barijough, and A. Gerstlauer, “DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.
- [136] Z. Zhao, Z. Jiang, N. Ling *et al.*, “ECRT: An Edge Computing System for Real-Time Image-based Object Tracking,” in *Proc. the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys 2018)*, 2018, pp. 394–395.
- [137] H. Li, K. Ota, and M. Dong, “Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing,” *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018.
- [138] S. S. Ogden and T. Guo, “MODI: Mobile Deep Inference Made Efficient by Edge Computing,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2018)*, 2018.
- [139] S. Teerapittayanon *et al.*, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *Proc. the 23rd International Conference on Pattern Recognition (ICPR 2016)*, 2016, pp. 2464–2469.
- [140] S. Teerapittayanon, B. McDanel, and H. T. Kung, “Distributed Deep Neural Networks over the Cloud, the Edge and End Devices,” in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)*, 2017, pp. 328–339.
- [141] E. Li, Z. Zhou, and X. Chen, “Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy,” in *Proc. the 2018 Workshop on Mobile Edge Communications (MECOMM 2018)*, 2018, pp. 31–36.
- [142] L. N. Huynh, Y. Lee, and R. K. Balan, “DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications,” in *Proc. the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2017)*, 2017, pp. 82–95.
- [143] Y. Chen, S. Biookaghazadeh, and M. Zhao, “Exploring the Capabilities of Mobile Devices Supporting Deep Learning,” in *Proc. the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2018)*, 2018, pp. 17–18.
- [144] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [145] R. Venkatesan and B. Li, “Diving deeper into mentee networks,” *arXiv preprint arXiv:1604.08220*, 2016.
- [146] S. Biookaghazadeh, F. Ren, and M. Zhao, “Are FPGAs Suitable for Edge Computing?” *arXiv preprint arXiv:1804.06404*, 2018.
- [147] 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.
- [148] X. Ran, H. Chen, Z. Liu *et al.*, “Delivering Deep Learning to Mobile Devices via Offloading,” in *Proc. the Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network 2017)*, 2017, pp. 42–47.
- [149] W. Zhang, Z. Zhang, S. Zeadally *et al.*, “MASM: A Multiple-algorithm Service Model for Energy-delay Optimization in Edge Artificial Intelligence,” *IEEE Trans. Ind. Inf. (Early Access)*, 2019.
- [150] E. Cuervo, A. Balasubramanian, D.-k. Cho *et al.*, “MAUI: Making Smartphones Last Longer with Code Offload,” in *Proc. the 8th international conference on mobile systems, applications, and services (MobiSys 2010)*, 2010, pp. 49–62.
- [151] 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 *Proc. the ACM Symposium on Cloud Computing (SoCC 2018)*, 2018, pp. 401–411.
- [152] Y. Huang, X. Ma, X. Fan *et al.*, “When deep learning meets edge computing,” in *IEEE 25th International Conference on Network Protocols (ICNP 2017)*, 2017, pp. 1–2.
- [153] J. Mao, X. Chen, K. W. Nixon *et al.*, “MoDNN: Local distributed mobile computing system for Deep Neural Network,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, 2017, pp. 1396–1401.
- [154] G. Kamath, P. Agnihotri, M. Valero *et al.*, “Pushing Analytics to the Edge,” in *2016 IEEE Global Communications Conference (GLOBECOM 2016)*, 2016, pp. 1–6.
- [155] L. Valerio, A. Passarella, and M. Conti, “A communication efficient distributed learning framework for smart environments,” *Pervasive Mob. Comput.*, vol. 41, pp. 46–68, Oct. 2017.
- [156] Y. Lin, S. Han, H. Mao *et al.*, “Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training,” *eprint arXiv:1712.01887*, 2017.
- [157] Z. Tao and C. William, “eSGD : Communication Efficient Distributed Deep Learning on the Edge,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018, pp. 1–6.
- [158] N. Strom, “Scalable distributed DNN training using commodity GPU cloud computing,” in *Sixteenth Annual Conference of the International Speech Communication Association (INTERSPEECH 2015)*, 2015, pp. 1488–1492.
- [159] E. Jeong, S. Oh, H. Kim *et al.*, “Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data,” *arXiv preprint arXiv:1811.11479*, 2018.
- [160] M. Fredrikson, S. Jha, and T. Ristenpart, “Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures,” in *Proc. the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, 2015, pp. 1322–1333.
- [161] M. Du, K. Wang, Z. Xia, and Y. Zhang, “Differential Privacy Preserving of Training Model in Wireless Big Data with Edge Computing,” *IEEE Trans. Big Data (Early Access)*, 2018.
- [162] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Theory of Cryptography*. Springer Berlin Heidelberg, 2006, pp. 265–284.
- [163] H. B. McMahan, E. Moore, D. Ramage *et al.*, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, 2017, pp. 1273–1282.
- [164] K. Bonawitz, H. Eichner *et al.*, “Towards Federated Learning at Scale: System Design,” *arXiv preprint arXiv:1902.01046*, 2019.
- [165] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, “Distributed Federated Learning for Ultra-Reliable Low-Latency Vehicular Communications,” *arXiv preprint arXiv:1807.08127*, 2018.
- [166] C. Xie, S. Koyejo, and I. Gupta, “Practical Distributed Learning: Secure Machine Learning with Communication-Efficient Local Updates,” *arXiv preprint arXiv:1903.06996*, 2019.
- [167] J. Konečný, H. B. McMahan, F. X. Yu *et al.*, “Federated Learning: Strategies for Improving Communication Efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
- [168] S. Caldas, J. Konečný, H. B. McMahan, and A. Talwalkar, “Expanding the Reach of Federated Learning by Reducing Client Resource Requirements,” *arXiv preprint arXiv:1812.07210*, 2018.
- [169] B. S. Kashin, “Diameters of some finite-dimensional sets and classes of smooth functions,” *Izv. Akad. Nauk SSSR Ser. Mat.*, vol. 41, pp. 334–351, 1977.
- [170] S. Wang, T. Tuor, T. Saloniidis *et al.*, “When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning,” in *IEEE Conference on Computer Communications (INFOCOM 2018)*, Apr. 2018, pp. 63–71.
- [171] S. Wang, T. Tuor, T. Saloniidis *et al.*, “Adaptive federated learning in resource constrained edge computing systems,” *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
- [172] T. Tuor, S. Wang, T. Saloniidis *et al.*, “Demo abstract: Distributed machine learning at resource-limited edge nodes,” in *2018 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 2018)*, 2018, pp. 1–2.
- [173] K. Yang, T. Jiang, Y. Shi, and Z. Ding, “Federated Learning via Over-the-Air Computation,” *arXiv preprint arXiv:1812.11750*, 2018.
- [174] B. Nazer *et al.*, “Computation over multiple-access channels,” *IEEE Trans. Inf. Theory*, vol. 53, no. 10, pp. 3498–3516, Oct. 2007.
- [175] L. Chen, N. Zhao, Y. Chen *et al.*, “Over-the-Air Computation for IoT Networks: Computing Multiple Functions With Antenna Arrays,” *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5296–5306, Dec. 2018.
- [176] G. Zhu, Y. Wang, and K. Huang, “Broadband Analog Aggregation for Low-Latency Federated Edge Learning (Extended Version),” *arXiv preprint arXiv:1812.11494*, 2018.
- [177] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, “Practical Secure Aggregation for Privacy-Preserving Machine Learning,” in *Proc. the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, 2017, pp. 1175–1191.
- [178] H. Kim, J. Park, M. Bennis, and S.-L. Kim, “On-Device Federated Learning via Blockchain and its Latency Analysis,” *arXiv preprint arXiv:1808.03949*, 2018.
- [179] M. Hofmann and L. Beaumont, “Chapter 3 - caching techniques for web content,” in *Content Networking*, 2005, pp. 53–79.
- [180] X. Wang, M. Chen, T. Taleb *et al.*, “Cache in the air: Exploiting content caching and delivery techniques for 5G systems,” *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.
- [181] E. Zeydan, E. Bastug, M. Bennis *et al.*, “Big data caching for networking: moving from cloud to edge,” *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sep. 2016.

- [182] J. Song, M. Sheng, T. Q. S. Quek *et al.*, "Learning-based content caching and sharing for wireless networks," *IEEE Trans. Commun.*, vol. 65, no. 10, pp. 4309–4324, Oct. 2017.
- [183] X. Li, X. Wang, P.-J. Wan *et al.*, "Hierarchical Edge Caching in Device-to-Device Aided Mobile Networks: Modeling, Optimization, and Design," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1768–1785, Aug. 2018.
- [184] Z. Chang, L. Lei, Z. Zhou *et al.*, "Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018.
- [185] J. Yang, J. Zhang, C. Ma *et al.*, "Deep learning-based edge caching for multi-cluster heterogeneous networks," *Neural Computing and Applications*, Feb. 2019.
- [186] A. Ndikumana, N. H. Tran, and C. S. Hong, "Deep Learning Based Caching for Self-Driving Car in Multi-access Edge Computing," *arXiv preprint arXiv:1810.01548*, 2018.
- [187] T. Kanungo, D. M. Mount *et al.*, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.
- [188] Y. Tang, K. Guo *et al.*, "A smart caching mechanism for mobile multimedia in information centric networking with edge computing," *Future Gener. Comput. Syst.*, vol. 91, pp. 590–600, Feb. 2019.
- [189] D. Adelman and A. J. Mersereau, "Relaxations of weakly coupled stochastic dynamic programs," *Operations Research*, vol. 56, no. 3, pp. 712–727, 2008.
- [190] H. Zhu, Y. Cao, W. Wang *et al.*, "Deep Reinforcement Learning for Mobile Edge Caching: Review, New Features, and Open Issues," *IEEE Network*, vol. 32, no. 6, pp. 50–57, Nov. 2018.
- [191] K. Guo, C. Yang, and T. Liu, "Caching in Base Station with Recommendation via Q-Learning," in *2017 IEEE Wireless Communications and Networking Conference (WCNC 2017)*, 2017, pp. 1–6.
- [192] C. Zhong, M. C. Gursoy *et al.*, "A deep reinforcement learning-based framework for content caching," in *52nd Annual Conference on Information Sciences and Systems (CISS 2018)*, 2018, pp. 1–6.
- [193] G. Dulac-Arnold, R. Evans, H. van Hasselt *et al.*, "Deep Reinforcement Learning in Large Discrete Action Spaces," *arXiv preprint arXiv:1512.07679*, 2015.
- [194] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Thirdquarter 2017.
- [195] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
- [196] J. Xu, L. Chen *et al.*, "Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.
- [197] T. Q. Dinh, Q. D. La, T. Q. S. Quek, and H. Shin, "Distributed Learning for Computation Offloading in Mobile Edge Computing," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6353–6367, Dec. 2018.
- [198] T. Chen and G. B. Giannakis, "Bandit convex optimization for scalable and dynamic iot management," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 1276–1286, Feb. 2019.
- [199] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2017)*, 2017, pp. 1–6.
- [200] T. Yang, Y. Hu, M. C. Gursoy *et al.*, "Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks," in *15th International Symposium on Wireless Communication Systems (ISWCS 2018)*, 2018, pp. 1–5.
- [201] X. Chen, H. Zhang, C. Wu *et al.*, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," *IEEE Internet Things J. (Early Access)*, 2018.
- [202] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal Auction for Edge Computing Resource Management in Mobile Blockchain Networks: A Deep Learning Approach," in *2018 IEEE International Conference on Communications (ICC 2018)*, 2018, pp. 1–6.
- [203] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *2018 IEEE Wireless Communications and Networking Conference (WCNC 2018)*, Apr. 2018, pp. 1–6.
- [204] M. Min, L. Xiao, Y. Chen *et al.*, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.
- [205] Z. Chen and X. Wang, "Decentralized Computation Offloading for Multi-User Mobile Edge Computing: A Deep Reinforcement Learning Approach," *arXiv preprint arXiv:1812.07394*, 2018.
- [206] T. Chen *et al.*, "Harnessing Bandit Online Learning to Low-Latency Fog Computing," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*, 2018, pp. 6418–6422.
- [207] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor-critic deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019.
- [208] Q. Zhang, M. Lin, L. T. Yang *et al.*, "A Double Deep Q-learning Model for Energy-efficient Edge Scheduling," *IEEE Trans. Serv. Comput. (Early Access)*, 2018.
- [209] L. Huang, S. Bi, and Y.-j. A. Zhang, "Deep Reinforcement Learning for Online Offloading in Wireless Powered Mobile-Edge Computing Networks," *arXiv preprint arXiv:1808.01977*, 2018.
- [210] S. Memon *et al.*, "Using machine learning for handover optimization in vehicular fog computing," in *Proc. the 34th ACM/SIGAPP Symposium on Applied Computing (SAC 2019)*, 2019, pp. 182–190.
- [211] Y. Sun *et al.*, "Deep reinforcement learning-based mode selection and resource management for green fog radio access networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1960–1971, Apr. 2019.
- [212] L. Xiao, X. Wan, C. Dai *et al.*, "Security in mobile edge caching with reinforcement learning," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 116–122, Jun. 2018.
- [213] Y. He, F. R. Yu, N. Zhao *et al.*, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.
- [214] C.-y. Li, H.-y. Liu *et al.*, "Mobile Edge Computing Platform Deployment in 4G LTE Networks: A Middlebox Approach," in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
- [215] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2595–2621, Fourthquarter 2018.
- [216] R. Li, Z. Zhao, X. Zhou *et al.*, "Intelligent 5g: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 175–183, Oct. 2017.
- [217] X. Chen, J. Wu, Y. Cai *et al.*, "Energy-efficiency oriented traffic offloading in wireless networks: A brief survey and a learning approach for heterogeneous cellular networks," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 4, pp. 627–640, Apr. 2015.
- [218] M. Min, X. Wan, L. Xiao *et al.*, "Learning-Based Privacy-Aware Offloading for Healthcare IoT with Energy Harvesting," *IEEE Internet Things J. (Early Access)*, 2018.
- [219] T. E. Bogale, X. Wang, and L. B. Le, "Machine Intelligence Techniques for Next-Generation Context-Aware Wireless Networks," *arXiv preprint arXiv:1801.04223*, 2018.
- [220] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
- [221] W. Zhang, J. Chen, Y. Zhang *et al.*, "Towards efficient edge cloud augmentation for virtual reality MMOGs," in *Proc. the Second ACM/IEEE Symposium on Edge Computing (SEC 2017)*, 2017, pp. 1–14.
- [222] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers (MARLIN-ATSC): Methodology and Large-Scale Application on Downtown Toronto," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1140–1150, Sep. 2013.
- [223] U. Drolia, K. Guo, J. Tan *et al.*, "Cachier: Edge-Caching for Recognition Applications," in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)*, 2017, pp. 276–286.
- [224] Z. Lai, Y. Cui, Z. Wang, and X. Hu, "Immersion on the Edge: A Co-operative Framework for Mobile Immersive Computing," in *Proc. the ACM SIGCOMM 2018 Conference on Posters and Demos (SIGCOMM 2018)*, 2018, pp. 39–41.
- [225] P. Guo, B. Hu *et al.*, "FoggyCache: Cross-Device Approximate Computation Reuse," in *Proc. the 24th Annual International Conference on Mobile Computing and Networking (MobiCom 2018)*, 2018, pp. 19–34.
- [226] M. Xu, M. Zhu *et al.*, "DeepCache: Principled Cache for Mobile Deep Vision," in *Proc. the 24th Annual International Conference on Mobile Computing and Networking (MobiCom 2018)*, 2018, pp. 129–144.
- [227] Y. Gan, Y. Zhang, D. Cheng *et al.*, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proc. the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)*, 2019.
- [228] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," *Queue*, vol. 11, no. 11, p. 30, 2013.

- [229] B. I. Ismail, E. M. Goortani, M. Bazli *et al.*, "Evaluation of Docker as Edge Computing Platform," *2015 IEEE Conference on Open Systems (ICOS 2015)*, pp. 130–135, 2015.
- [230] J. Xu, S. Wang, B. Bhargava, and F. Yang, "A Blockchain-enabled Trustless Crowd-Intelligence Ecosystem on Mobile Edge Computing," *IEEE Trans. Ind. Inf. (Early Access)*, 2019.
- [231] Z. Zheng, S. Xie, H. Dai *et al.*, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress 2017)*, 2017, pp. 557–564.
- [232] J.-y. Kim and S.-M. Moon, "Blockchain-based edge computing for deep neural network applications," in *Proc. the Workshop on INTElligent Embedded Systems Architectures and Applications (INTESA 2018)*, 2018, pp. 53–55.
- [233] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: <http://gavwood.com/Paper.pdf>
- [234] P. Meloni, O. Ripolles, D. Solans *et al.*, "ALOHA: an architectural-aware framework for deep learning at the edge," in *Proc. the Workshop on INTElligent Embedded Systems Architectures and Applications (INTESA 2018)*, 2018, pp. 19–26.
- [235] L. Lai *et al.*, "Rethinking Machine Learning Development and Deployment for Edge Devices," *arXiv preprint arXiv:1806.07846*, 2018.
- [236] M. Polese, R. Jana, V. Kounev *et al.*, "Machine Learning at the Edge: A Data-Driven Architecture with Applications to 5G Cellular Networks," *arXiv preprint arXiv:1808.07647*, 2018.
- [237] J. Zhao, T. Tiplea, R. Mortier *et al.*, "Data Analytics Service Composition and Deployment on IoT Devices," in *Proc. the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2018)*, 2018, pp. 502–504.
- [238] C. Andrés Ramiro, C. Fiandrino, A. Blanco Pizarro *et al.*, "openLEON: An End-to-End Emulator from the Edge Data Center to the Mobile Users Carlos," in *Proc. the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH 2018)*, 2018, pp. 19–27.
- [239] Y. Wang, S. Liu, X. Wu, and W. Shi, "CAVBench: A Benchmark Suite for Connected and Autonomous Vehicles," in *2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)*, 2018, pp. 30–42.
- [240] S. Zheng, Q. Meng, T. Wang *et al.*, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. the 34th International Conference on Machine Learning (ICML 2017)*, 2017, pp. 4120–4129.
- [241] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," *arXiv preprint arXiv:1903.03934*, 2019.
- [242] H. H. Zhuo, W. Feng, Q. Xu *et al.*, "Federated Reinforcement Learning," *arXiv preprint arXiv:1901.08277*, 2019.
- [243] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017, pp. 4424–4434.
- [244] T. Xing, S. S. Sandha, B. Balaji *et al.*, "Enabling Edge Devices that Learn from Each Other: Cross Modal Training for Activity Recognition," in *Proc. the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2018)*, 2018, pp. 37–42.
- [245] J. Yoon, P. Liu, and S. Banerjee, "Low-Cost Video Transcoding at the Wireless Edge," in *2016 IEEE/ACM Symposium on Edge Computing (SEC 2016)*, 2016, pp. 129–141.
- [246] N. Kato *et al.*, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 146–153, Jun. 2017.
- [247] Z. M. Fadlullah, F. Tang, B. Mao *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, Fourthquarter 2017.
- [248] J. Foerster, I. A. Assael *et al.*, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)*, 2016, pp. 2137–2145.
- [249] S. Omidshafiei, J. Papis, C. Amato *et al.*, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proc. the 34th International Conference on Machine Learning (ICML 2017)*, 2017, pp. 2681–2690.
- [250] R. Lowe, Y. WU *et al.*, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017, pp. 6379–6390.
- [251] J. Zhou, G. Cui *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.
- [252] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

Yiwen Han [S18] received his B.S. degree from Nanchang University, China, and M.S. degree from Tianjin University in 2015 and 2018, respectively, both in communication engineering. He is currently pursuing a Ph.D. degree with Tianjin University. His current research interests include edge computing, reinforcement learning, and deep learning.

Xiaofei Wang [S06, M13, SM18] is currently a Professor with the Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, China. He was a Post-Doctoral Fellow with The University of British Columbia from 2014 to 2016. Focusing on the research of social-aware cloud computing, cooperative cell caching, and mobile traffic offloading, he has authored over 80 technical papers in the IEEE JSAC, the IEEE TWC, the IEEE WIRELESS COMMUNICATIONS, the IEEE COMMUNICATIONS, the IEEE TMM, the IEEE INFOCOM, and the IEEE SECON. He was a recipient of the National Thousand Talents Plan (Youth) of China. He received the "Scholarship for Excellent Foreign Students in IT Field" by NIPA of South Korea from 2008 to 2011, the "Global Outstanding Chinese Ph.D. Student Award" by the Ministry of Education of China in 2012, and the Peiyang Scholarship from Tianjin University. In 2017, he received the "Fred W. Ellersick Prize" from the IEEE Communication Society.

Victor C. M. Leung [S75, M89, SM97, F03] received the B.A.Sc. (Hons.) degree in electrical engineering from the University of British Columbia (UBC) in 1977, and was awarded the APEBC Gold Medal as the head of the graduating class in the Faculty of Applied Science. He attended graduate school at UBC on a Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship and received the Ph.D. degree in electrical engineering in 1982. From 1981 to 1987, Dr. Leung was a Senior Member of Technical Staff and satellite system specialist at MPR Teltech Ltd., Canada. He is also serving as the Overseas Dean of the School of Information and Electronic Engineering at Zhejiang Gongshang University, China. Dr. Leung has co-authored more than 1000 journal/conference papers, 37 book chapters, and co-edited 12 book titles. Dr. Leung is a registered Professional Engineer in the Province of British Columbia, Canada.

Dusit Niyato [M09, SM15, F17] is currently a Professor in the School of Computer Science and Engineering, at Nanyang Technological University, Singapore. He received B.Eng. from King Mongkut's Institute of Technology Ladkrabang (KMUTL), Thailand in 1999 and Ph.D. in Electrical and Computer Engineering from the University of Manitoba, Canada in 2008. His research interests are in the area of Internet of Things (IoT) and network resource pricing.

Xueqiang Yan is currently a senior research engineer at Huawei Technologies, China. His main research direction is future mobile communication technologies, network convergence and evolution.

Xu Chen received the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2012. He was a post-doctoral research associate with Arizona State University, Tempe, USA, from 2012 to 2014, and a Humboldt Fellow with the University of Goettingen, Germany, from 2014 to 2016. He is currently a professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He received the 2017 IEEE ComSoc Pacific-Asia Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the 2014 Hong Kong Young Scientist Runner-Up Award, the Best Paper Award at IEEE ICC 2017, the Best Paper Runner-Up Award at IEEE INFOCOM 2014, and the Honorable Mention Award at IEEE ISI 2010.