Convergence of Edge Computing and Deep Learning: A Comprehensive Survey

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

***Abstract*—Ubiquitous sensors and smart devices from factories and communities are generating 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, developments of artificial intelligence (especially deep learning, DL) based applications and services are thriving. 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, unleashing DL services using resources at the network edge near the data sources has emerged as a desirable solution. Therefore, *edge intelligence*, aiming to facilitate the deployment of DL services by edge computing, has received significant attention. In addition, DL, as the representative technique 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 beneficial *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 DL training and inference in the customized edge computing framework; 3) challenges and future trends of more pervasive and fine-grained intelligence. We believe that by consolidating information scattered across the communication, networking, and DL areas, this survey can help readers to understand the connections between enabling technologies while promoting further discussions on the fusion of *edge intelligence* and *intelligent edge*, i.e., Edge DL.**

***Index Terms*—Edge computing, deep learning, wireless communication, computation offloading, artificial intelligence.**

Manuscript received July 12, 2019; revised December 22, 2019; accepted January 20, 2020. Date of publication January 30, 2020; date of current version May 28, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2101901 and Grant 2018YFC0809803, in part by the National Science Foundation of China under Grant 61702364, Grant 61972432, and Grant U1711265, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X355, in part by the Chinese National Engineering Laboratory for Big Data System Computing Technology, in part by the Canadian Natural Sciences and Engineering Research Council, in part by the Singapore NRF National Satellite of Excellence, Design Science and Technology for Secure Critical Infrastructure National Satellite of Excellence under Grant DeST-SCI2019-0007, in part by the A*STAR-NTU-SUTD Joint Research Grant Call on Artificial Intelligence for the Future of Manufacturing under Grant RGANS1906, in part by WASP/NTU under Grant M4082187 (4080), in part by the Singapore MOE Tier 1 under Grant 2017-T1-002-007 RG122/17, in part by MOE Tier 2 under Grant MOE2014-T2-2-015 ARC4/15, in part by the Singapore under Grant NRF2015-NRF-ISF001-2277, and in part by the Singapore Energy Market Authority Energy Resilience under Grant NRF2017EWT-EP003-041. *(Corresponding author: Yiwen Han.)*

Xiaofei Wang and Yiwen Han are with the Tianjin Key Laboratory of Advanced Networking, College of Intelligence and Computing, Tianjin University, Tianjin 300350, China (e-mail: xiaofeiwang@tju.edu.cn; hanyiwen@tju.edu.cn).

Victor C. M. Leung is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China, and also with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: vleung@ieee.org).

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

Xueqiang Yan is with 2012 Lab, Huawei Technologies, Shenzhen 201206, China (e-mail: yanxueqiang1@huawei.com).

Xu Chen is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China (e-mail: chenxu35@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/COMST.2020.2970550

I. Introduction

With the proliferation of computing and storage devices, from server clusters in cloud data centers (the cloud) to personal computers and smartphones, further, to wearable and other Internet of Things (IoT) devices, we are now in an information-centric era in which computing is ubiquitous and computation services are overflowing from the cloud to the edge. According to a Cisco white paper [1], 50 billion IoT devices will be connected to the Internet by 2020. 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 these massively distributed computing power and analyze their 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 capacity and the computing power of cloud computing infrastructures; 2) many new types of applications, e.g., cooperative autonomous driving, have strict or tight delay requirements that the cloud would have difficulty meeting since it may be far away from the users [4].

Therefore, edge computing [5], [6] emerges as an attractive alternative, especially to host computation tasks as close as possible to the data sources and end users. Certainly, edge computing and cloud computing are not mutually exclusive [7], [8]. Instead, the edge complements and extends the cloud. Compared with cloud computing only, the main advantages of edge computing combined with cloud computing are three folds: 1) **backbone network alleviation**, distributed

edge computing nodes can handle a large number of computation tasks without exchanging the corresponding data with the cloud, thus alleviating the traffic load of the network; 2) **agile service response**, services hosted at the edge can significantly reduce the delay of data transmissions and improve the response speed; 3) **powerful cloud backup**, the cloud can provide powerful processing capabilities and massive storage when the edge cannot afford.

As a typical and more widely used new form of applications [9], various deep learning-based intelligent services and applications have changed many 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., there are only a limited number of intelligent services offered due to the following factors.

- ***Cost:*** training and inference of DL models in the cloud requires devices or users to transmit massive amounts of data to the cloud, thus consuming a large amount of network bandwidth;
- ***Latency:*** the delay to access cloud services is generally not guaranteed and might not be short enough to satisfy the requirements of many time-critical applications such as cooperative autonomous driving [11];
- ***Reliability:*** most cloud computing applications relies on wireless communications and backbone networks for connecting users to services, but for many industrial scenarios, intelligent services must be highly reliable, even when network connections are lost;
- ***Privacy:*** the data required for DL might carry a lot of private information, and privacy issues are critical to areas such as smart home and cities.

Since the edge is closer to users than the cloud, edge computing is expected to solve many of 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 is expected to push DL computations from the cloud to the edge as much as possible, thus enabling various distributed, low-latency and reliable intelligent services. As shown in Fig. 2, the advantages include: 1) DL services are deployed close to the requesting users, 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, protection of user privacy is enhanced; 3) the hierarchical computing architecture provides more reliable DL computation; 4) with richer data and application scenarios, edge computing

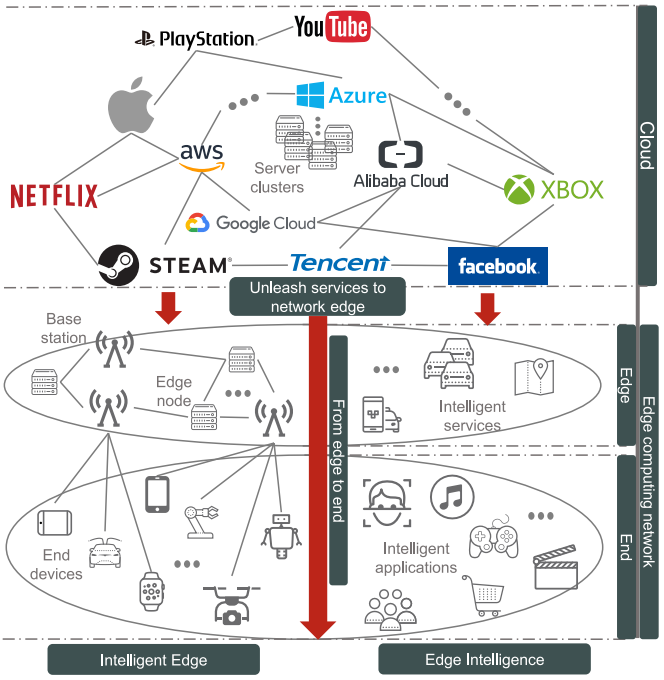


Fig. 1. Edge intelligence and intelligent edge.

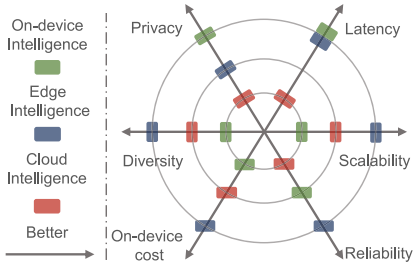


Fig. 2. Capabilities comparison of cloud, on-device and edge intelligence.

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.

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]. Thus the end devices, edge, and cloud are gradually merging into a community of shared resources. However, the maintenance and management of such a large and complex overall architecture (community) involving 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,

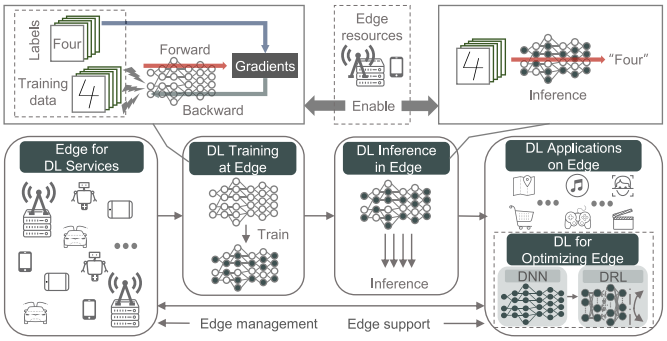


Fig. 3. Landscape of Edge DL according to the proposed taxonomy.

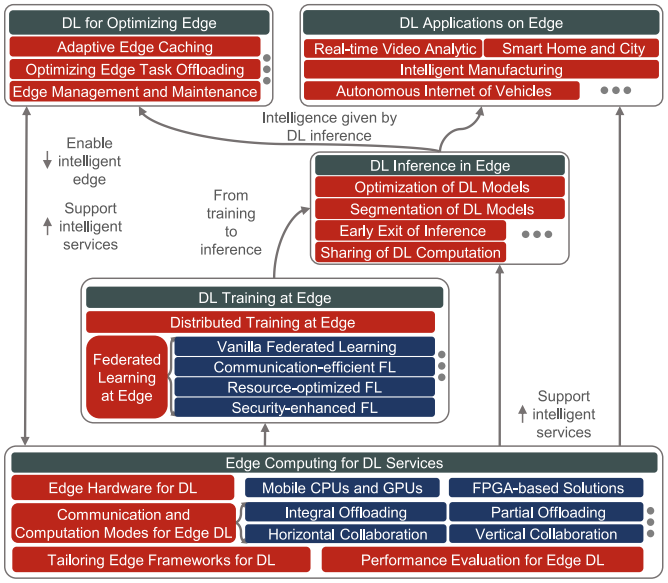


Fig. 4. Conceptual relationships of edge intelligence and intelligent edge.

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 that edge intelligence and intelligent edge, i.e., Edge DL, together face some of the same challenges and practical issues in multiple aspects, we identify the following five technologies that are essential for Edge DL:

1) *DL applications on Edge*, technical frameworks for systematically organizing edge computing and DL to provide intelligent services;
2) *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 computing for DL*, which adapts the edge computing platform in terms of network architecture, hardware and software to support DL computation;
4) *DL training at Edge*, training DL models for edge intelligence at distributed edge devices under resource and privacy constraints;
5) *DL for optimizing Edge*, application of DL for maintaining and managing different functions of edge computing networks (systems), e.g., edge caching [16], computation offloading [17].

As illustrated in Fig. 3, "DL applications on Edge" and "DL for optimizing edge" correspond to the theoretical goals of edge intelligence and intelligent edge, respectively. To support them, various DL models should be trained by intensive computation at first. In this case, for the related works leveraging edge computing resources to train various DL models, we classify them as "DL training at Edge". Second, to enable and speed up Edge DL services, we focus on a variety of techniques supporting the efficient inference of DL models in edge computing frameworks and networks, called "DL inference in Edge". At last, we classify all techniques, which adapts edge computing frameworks and networks to better serve Edge DL, as "Edge computing for DL".

To the best of our knowledge, existing articles that are most related to our work include [18]–[21]. Different from our more extensive coverage of Edge DL, [18] is focussed on the use of machine learning (rather than DL) in edge intelligence for wireless communication perspective, i.e., training machine learning at the network edge to improve wireless communication. Besides, discussions about DL inference and training are the main contribution of [19]–[21]. Different from these works, this survey focuses on these respects: 1) comprehensively consider deployment issues of DL by edge computing, spanning networking, communication, and computation; 2) investigate the holistic technical spectrum about the convergence of DL and edge computing in terms of the five enablers; 3) point out that DL and edge computing are beneficial to each other and considering only deploying DL on the edge is incomplete.

This paper is organized as follows (as abstracted in Fig. 4). We have given the background and motivations of this survey in the current section. Next, we provide some fundamentals related to edge computing and DL in Section II and Section III, respectively. The following sections introduce the 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 present lessons learned and discuss open challenges 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], [22], [23].

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,

TABLE I
LIST OF IMPORTANT ABBREVIATIONS IN ALPHABETICAL ORDER

Abbr.	Definition	Abbr.	Definition	Abbr.	Definition
A-LSH	Adaptive Locality Sensitive Hashing	DVFS	Dynamic Voltage and Frequency Scaling	NLP	Natural Language Processing
AC	Actor-Critic	ECSP	Edge Computing Service Provider	NN	Neural Network
A3C	Asynchronous Advantage Actor-Critic	EEoI	Early Exit of Inference	NPU	Neural Processing Unit
AE	Auto-Encoder	EH	Energy Harvesting	PPO	Proximate Policy Optimization
AI	Artificial Intelligence	FAP	Fog radio Access Point	QoE	Quality of Experience
APU	AI Processing Unit	FCNN	Fully Connected Neural Network	QoS	Quality of Service
AR	Augmented Reality	FL	Federated Learning	RAM	Random Access Memory
ASIC	Application-Specific Integrated Circuit	FPGA	Field Programmable Gate Array	RNN	Recurrent Neural Network
BS	Base Station	FTP	Fused Tile Partitioning	RoI	Region-of-Interest
C-RAN	Cloud-Radio Access Networks	GAN	Generative Adversarial Network	RRH	Remote Radio Head
CDN	Content Delivery Network	GNN	Graph Neural Network	RSU	Road-Side Unit
CNN	Convolutional Neural Network	IID	Independent and Identically Distributed	SDN	Software-Defined Network
CV	Computer Vision	IoT	Internet of Things	SGD	Stochastic Gradient Descent
DAG	Directed Acyclic Graph	IoV	Internet of Vehicles	SINR	Signal-to-Interference-plus-Noise Ratio
D2D	Device-to-Device	KD	Knowledge Distillation	SNPE	Snapdragon Neural Processing Engine
DDoS	Distributed Denial of Service	kNN	k-Nearest Neighbor	TL	Transfer Learning
DDPG	Deep Deterministic Policy Gradient	MAB	Multi-Armed Bandit	UE	User Equipment
DL	Deep Learning	MEC	Mobile (Multi-access) Edge Computing	VM	Virtual Machine
DNN	Deep Neural Networks	MDC	Micro Data Center	VNF	Virtual Network Function
DQL	Deep Q-Learning	MDP	Markov Decision Process	V2V	Vehicle-to-Vehicle
DRL	Deep Reinforcement Learning	MLP	Multi-Layer Perceptron	WLAN	Wireless Local Area Network
DSL	Domain-specific Language	NFV	Network Functions Virtualizatio	ZB	Zettabytes

such as Cloudlet [24], Micro Data Centers (MDCs) [25], Fog Computing [26], [27] and Mobile Edge Computing [5] (viz., Multi-access Edge Computing [28] now). However, the edge computing community has not yet reached a consensus on the standardized definitions, architectures and protocols of edge computing [23]. We use a common term "edge computing" for this set of emerging technologies. In this section, different edge computing concepts are introduced and differentiated.

1) Cloudlet and Micro Data Centers: Cloudlet 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 [24]. Similar to Cloudlets, MDCs [25] 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 [26], [27]. 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 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 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. The European Telecommunications Standards Institute (ETSI) further extends the terminology of MEC from Mobile Edge Computing to Multi-access Edge Computing by accommodating more wireless communication technologies, such as Wi-Fi [28].

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) include 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 created for processing computation-intensive tasks, such as DL, it cannot guarantee the delay requirements

TABLE II
SUMMARY OF EDGE COMPUTING AI HARDWARES AND SYSTEMS

	Owner	Production	Feature
Integrated Commodities	Microsoft	Data Box Edge [29]	Competitive in data preprocessing and data transmission
	Intel	Movidius Neural Compute Stick [30]	Prototype on any platform with plug-and-play simplicity
	NVIDIA	Jetson [31]	Easy-to-use platforms that runs in as little as 5 Watts
	Huawei	Atlas Series [32]	An all-scenario AI infrastructure solution that bridges "device, edge, and cloud"
AI Hardware for Edge Computing	Qualcomm	Snapdragon 8 Series [33]	Powerful adaptability to major DL frameworks
	HiSilicon	Kirin 600/900 Series [34]	Independent NPU for DL computation
	HiSilicon	Ascend Series [35]	Full coverage – from the ultimate low energy consumption scenario to high computing power scenario
	MediaTek	Helio P60 [36]	Simultaneous use of GPU and NPU to accelerate neural network computing
	NVIDIA	Turing GPUs [37]	Powerful capabilities and compatibility but with high energy consumption
	Google	TPU [38]	Stable in terms of performance and power consumption
	Intel	Xeon D-2100 [39]	Optimized for power- and space-constrained cloud-edge solutions
	Samsung	Exynos 9820 [40]	Mobile NPU for accelerating AI tasks
Edge Computing Frameworks	Huawei	KubeEdge [41]	Native support for edge-cloud collaboration
	Baidu	OpenEdge [42]	Computing framework shielding and application production simplification
	Microsoft	Azure IoT Edge [43]	Remotely edge management with zero-touch device provisioning
	Linux Foundation	EdgeX [44]	IoT edge across the industrial and enterprise use cases
	Linux Foundation	Akraino Edge Stack [45]	Integrated distributed cloud edge platform
	NVIDIA	NVIDIA EGX [46]	Real-time perception, understanding, and processing at the edge
	Amazon	AWS IoT Greengrass [47]	Tolerance to edge devices even with intermittent connectivity
	Google	Google Cloud IoT [48]	Compatible with Google AI products, such as TensorFlow Lite and Edge TPU

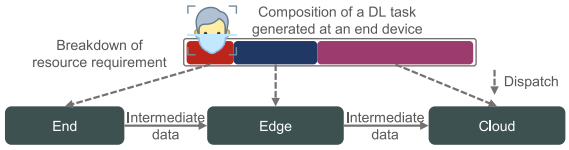


Fig. 5. A sketch of collaborative end-edge-cloud DL computing.

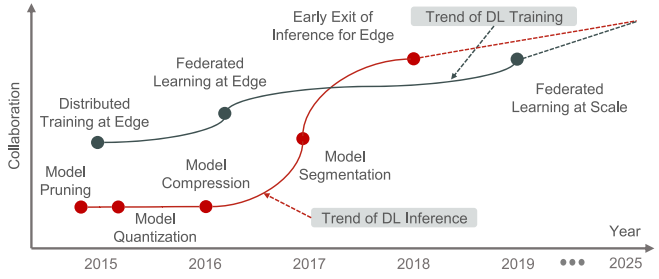


Fig. 6. Computation collaboration is becoming more important for DL with respect to both training and inference.

throughout the whole process from data generation to transmission to execution. Moreover, independent processing on the end or edge devices is limited by their computing capability, power consumption, and cost bottleneck. Therefore, collaborative end-edge-cloud computing for DL [12], abstracted in Fig. 5, is emerging as an important trend as depicted in Fig. 6. In this novel computing paradigm, computation tasks with lower computational intensities, generated by end devices, can be executed directly at the end devices or 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], [49], [50]. 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.

B. Hardware for Edge Computing

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 Hardware for Edge Computing: Emerged edge AI hardware can be classified into three categories according to their technical architecture: 1) Graphics Processing Unit (GPU)-based hardware, which tend to have good compatibility and performance, but generally consume more energy, e.g., NVIDIA' GPUs based on Turing architecture [37]; 2) Field Programmable Gate Array (FPGA)-based hardware [51], [52], which are energy-saving and require less computation resources, but with worse compatibility and limited programming capability compared to GPUs; 3) Application Specific Integrated Circuit (ASIC)-based hardware, such as Google's TPU [38] and HiSilicon's Ascend series [35], usually with a custom design that is more stable in terms of performance and power consumption.

As smartphones represent the most widely-deployed edge devices, chips for smartphones have undergone rapid developments, and their capabilities have been extended to the acceleration of AI computing. To name a few, Qualcomm first applies AI hardware acceleration [33] in Snapdragon and releases Snapdragon Neural Processing Engine (SNPE) SDK [53], which supports almost all major DL frameworks. Compared to Qualcomm, HiSilicon's 600 series and 900 series chips [34] do not depend on GPUs. Instead, they incorporate an additional 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 [36]. Performance comparison of most commodity chips with respect to DL can be found in [54], 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' Atlas modules [32] and Microsoft's Data Box Edge [29] can carry out preliminary DL inference and then transfer to the cloud for further improvement.

3) Edge Computing Frameworks: 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 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 [55]. Based on Kubernetes, Huawei develops its edge computing solution "KubeEdge" [41] for networking, application deployment and metadata synchronization between the cloud and the edge (also supported in Akraino Edge Stack [45]). "OpenEdge" [42] focus on shielding computing framework and simplifying application production. For IoT, Azure IoT Edge [43] and EdgeX [44] are devised for delivering cloud intelligence to the edge by deploying and running AI on cross-platform IoT devices.

C. Virtualizing the Edge

The requirements of virtualization technology for integrating edge computing and DL reflect in the following aspects: 1) The resource of edge computing is limited. Edge computing cannot provide that resources for DL services as the cloud does. Virtualization technologies should maximize resource utilization under the constraints of limited resources; 2) DL services rely heavily on complex software libraries. The versions and dependencies of these software libraries should be taken into account carefully. Therefore, virtualization catering to Edge DL services should be able to isolate different services. Specifically, the upgrade, shutdown, crash, and high resource consumption of a single service should not affect other services; 3) The service response speed is critical for Edge DL. Edge DL requires not only the computing power of edge devices but also the agile service response that the edge computing architecture can provide.

The combination of edge computing and DL to form high-performance Edge DL services requires the coordinated integration of computing, networking and communication resources, as depicted in Fig. 8. Specifically, both the computation virtualization and the integration of network virtualization, and management technologies are necessary. In this section, we discuss potential virtualization technologies for the edge.

1) Virtualization Techniques: Currently, there are two main virtualization strategies: Virtual Machine (VM) and container. In general, VM is better at isolating while container provides easier deployment of repetitive tasks [69]. With VM virtualization at operating system level, a VM hypervisor splits a physical server into one or multiple VMs, and can easily manage each VM to execute tasks in isolation. Besides, the VM hypervisor can allocate and use idle computing resources more efficiently by creating a scalable system that includes multiple independent virtual computing devices.

In contrast to VM, container virtualization is a more flexible tool for packaging, delivering, and orchestrating software infrastructure services and applications. Container virtualization for edge computing can effectively reduce the workload execution time with high performance and storage requirements, and can also deploy a large number of services in a scalable and straightforward fashion [70]. A container consists of a single file that includes an application and execution environment with all dependencies, which makes it enable efficient service handoff to cope with user mobility [71]. Owning to that the execution of applications in the container does not depend on additional virtualization layers as in VM virtualization, the processor consumption and the amount of memory required to execute the application are significantly reduced.

2) Network Virtualization: Traditional networking functions, combined with specific hardware, is not flexible enough to manage edge computing networks in an on-demand fashion. In order to consolidate network device functions into industry-standard servers, switches and storage, Network Functions Virtualization (NFV) enables Virtual Network Functions (VNFs) to run in software, by separating network functions and services from dedicated network hardware. Further, Edge DL services typically require high bandwidth, low latency, and dynamic network configuration, while Software-defined Networking (SDN) allows rapid deployment of services, network programmability and multi-tenancy support, through three key innovations [72]: 1) Decoupling of control planes and data planes; 2) Centralized and programmable control planes; 3) Standardized application programming interface. With these advantages, it supports a

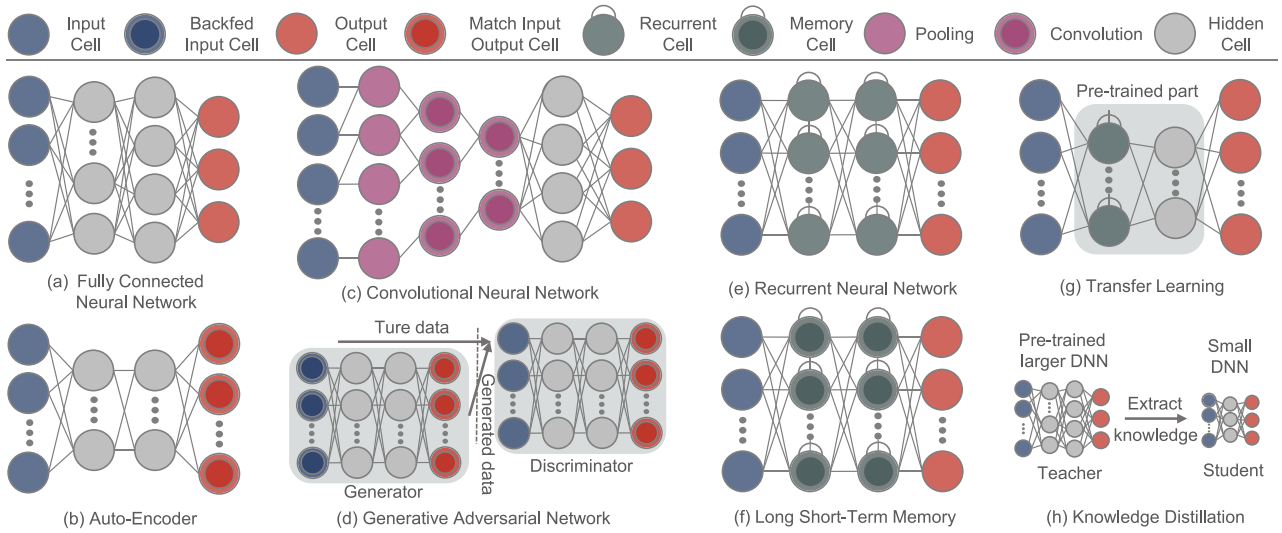


Fig. 7. Basic structures and functions of typical DNNs and DL.

highly customized network strategy that is well suited for the high bandwidth, dynamic nature of Edge DL services.

Network virtualization and edge computing benefit each other. On the one hand, NFV/SDN can enhance the interoperability of edge computing infrastructure. For example, with the support of NFV/SDN, edge nodes can be efficiently orchestrated and integrated with cloud data centers [73]. On the other hand, both VNFs and Edge DL services can be hosted on a lightweight NFV framework (deployed on the edge) [74], thus reusing the infrastructure and infrastructure management of NFV to the largest extent possible [75].

3) Network Slicing: Network slicing is a form of agile and virtual network architecture, a high-level abstraction of the network that allows multiple network instances to be created on top of a common shared physical infrastructure, each of which optimized for specific services. With increasingly diverse service and QoS requirements, network slicing, implemented by NFV/SDN, is naturally compatible with distributed paradigms of edge computing. To meet these, network slicing can be coordinated with joint optimization of computing and communication resources in edge computing networks [76]. Fig. 8 depicts an example of network slicing based on edge virtualization. In order to implement service customization in network slicing, virtualization technologies and SDN must be together to support tight coordination of resource allocation and service provision on edge nodes while allowing flexible service control. With network slicing, customized and optimized resources can be provided for Edge DL services, which can help reduce latency caused by access networks and support dense access to these services [77].

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 [78]. 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 [79]).

A. Neural Networks in Deep Learning

DL models consist of various types of Deep Neural Networks (DNNs) [79]. 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 [80]. 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 [81].

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

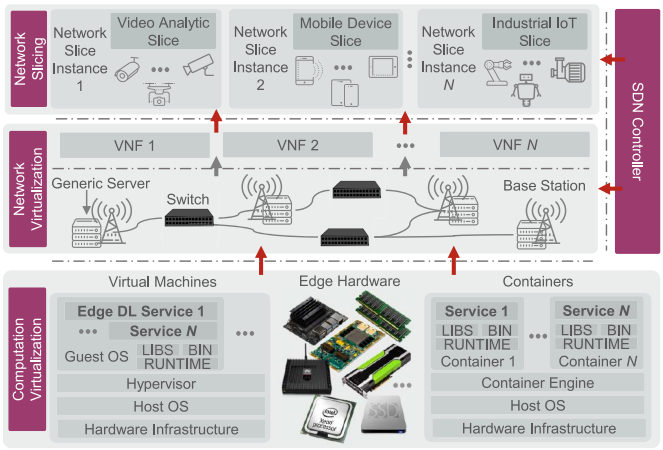


Fig. 8. Virtualizing edge computing infrastructure and networks.

the risk of overfitting [82]. 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 back-fed 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 [83]. 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 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 [84].

6) Transfer Learning (TL): TL can transfer knowledge, as shown in Fig. 7(g), from the source domain to the target domain so as to achieve better learning performance in the target domain [85]. 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) [86] 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.

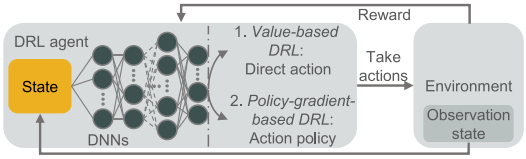


Fig. 9. Value-based and policy-gradient-based DRL approaches.

B. Deep Reinforcement Learning (DRL)

As depicted in Fig. 9, 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], [87].

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 [88]. 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 [89], and Dueling Deep Q-Learning (Dueling-DQL) [90] 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) [91], Asynchronous Advantage Actor-Critic (A3C) [92], Proximate Policy Optimization (PPO) [93], 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 [94]. 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

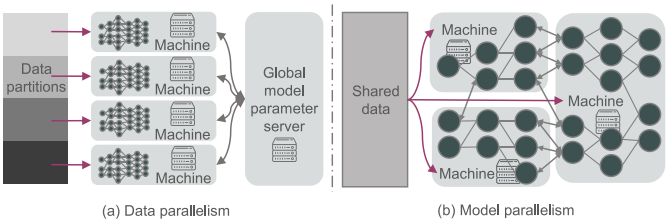


Fig. 10. Distributed training in terms of data and model parallelism.

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* [95]–[98] as illustrated in Fig. 10.

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 [99]. Data parallelism means dividing 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 [100].

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. Potential DL Libraries for Edge

Development and deployment of DL models rely on the support of various DL libraries. However, different DL libraries have their own application scenarios. For deploying DL on and for the edge, efficient lightweight DL libraries are required. Features of DL frameworks potentially supporting future edge intelligence are listed in Table III (excluding libraries unavailable for edge devices, such as Theano [101]).

TABLE III
POTENTIAL DL LIBRARIES FOR EDGE COMPUTING

Library	CNTK [56]	Chainer [57]	TensorFlow [58]	DL4J [59]	TensorFlow Lite [60]	MXNet [61]	(Py)Torch [62]	CoreML [63]	SNPE [53]	NCNN [64]	MNN [65]	Paddle-Mobile [66]	MACE [67]	FANN [68]
Owner	Microsoft	Preferred Networks	Google	Skymind	Google	Apache Incubator	Facebook	Apple	Qualcomm	Tencent	Alibaba	Baidu	XiaoMi	ETH Zürich
Edge Support	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Android	×	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×
iOS	×	×	×	×	×	✓	✓	✓	×	✓	✓	✓	✓	×
Arm	×	×	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
FPGA	×	×	×	×	×	×	✓	×	×	×	×	✓	×	×
DSP	×	×	×	×	×	×	×	×	✓	×	×	×	×	×
GPU	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
Mobile GPU	×	×	×	×	✓	×	×	✓	✓	✓	✓	✓	✓	×
Training Support	✓	✓	✓	✓	×	✓	✓	×	×	×	×	×	×	✓

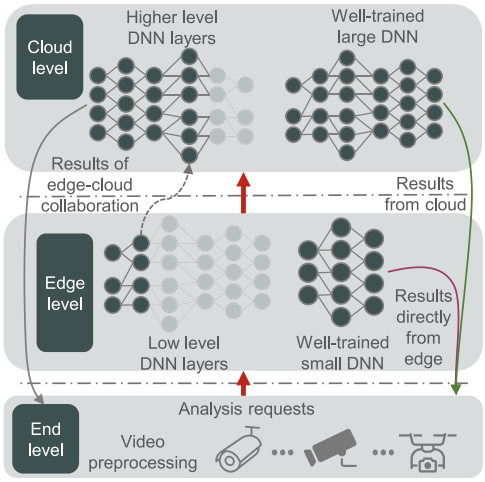


Fig. 11. The collaboration of the end, edge and cloud layer for performing real-time video analytic by deep learning.

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 edge DL applications and highlight their advantages over the comparing 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. 11, we summarize related works as a hybrid hierarchical architecture, which is divided into three levels: end, edge, and cloud.

1) End Level: At the end level, video capture devices, such as smartphones and surveillance cameras are responsible for video capture, media data compression [102], image pre-processing, and image segmentation [103]. 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 [104]. 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 [102].

If various DL tasks are executed at the end level independently, enabling parallel analytics requires a solution that supports efficient multi-tenant DL. With the model pruning and recovery scheme, *NestDNN* [105] transforms the DL model into a set of descendant models, in which the descendant model with fewer resource requirements shares its model parameters with the descendant model requiring more resources, making itself nested inside the descendent model requiring more resources without taking extra memory space. In this way, the multi-capacity model provides variable resource-accuracy trade-offs with a compact memory footprint, hence ensuring efficient multi-tenant DL at the end level.

2) Edge Level: Numerous distributed edge nodes at the edge level generally cooperate with each other to provide better services. For example, *LAVEA* [106] 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, compressing the DL model on the edge can improve holistic performance. The resource consumption of the edge layer can be greatly reduced while ensuring the analysis performance, by reducing the unnecessary filters in CNN layers [107]. Besides, in order to optimize performance and efficiency, [108] presents an edge service framework, i.e., *EdgeEye*, which realizes a high-level abstraction of real-time video analytic functions based on DL. To fully exploit the bond function of the edge, *VideoEdge* [109] implements an end-edge-cloud hierarchical architecture to help achieve load balancing concerning analytical tasks while maintaining high analysis accuracy.

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 [102]. 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 [110]. 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 [110], 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 [110]. 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 [111] to obtain the optimal task offloading policy in vehicular edge computing. Besides, Vehicle-to-Vehicle (V2V) 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 [112].

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 [113]. 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 [114]. For autonomous manufacturing inspection, *DeepIns* [113] 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 [115], 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 [116] should also be considered. Therefore, caching, communication with heterogeneous IoT devices, and computation offloading can be integrated [117] 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 [118], [119], 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 [120] and enables efficient music cognition system [121].

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 [122] 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 [123]) 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, dramatic 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 [124], [125] 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 [126] and deep residual networks [127] for CNNs. On the other hand, for more general neural network structures, existing optimization methods can be divided into four categories [128]: 1) parameter pruning and sharing [129], [130], including also weights quantization [131]–[133]; 2) low-rank factorization [124]; 3) transferred/compact convolution filters [107], [134], [135]; 4) knowledge distillation [136]. 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 Input:* Each application scenario has specific optimization spaces. Concerning object detection, *FFS-VA* uses two prepositive stream-specialized filters and a small full-function tiny-YOLO model to filter out vast but non-target-object frames [137]. In order to adjust the configuration of the input video stream (such as frame resolution and sampling rate) online with low cost, *Chameleon* [138] greatly saves the cost of searching the best model configuration by leveraging temporal and spatial correlations of the video inputs, and allows the cost to be amortized over time and across multiple video feeds. Besides, as depicted in Fig. 12, narrowing down the classifier's searching space [139] and dynamic Region-of-Interest (RoI) encoding [140] to focus on target objects in video frames can further reduce the bandwidth consumption and data transmission delay. Though this kind of methods can significantly compress the size of model inputs and hence reduce the computation overhead without altering the structure of DL models, it requires a deep understanding of the related application scenario to dig out the potential optimization space.
- *Model Structure:* Not paying attention to specific applications, but focusing on the widely used DNNs' structures is also feasible. For instance, point-wise group convolution and channel shuffle [142], paralleled convolution and pooling computation [143], depth-wise separable convolution [107] can greatly reduce computation cost while maintaining accuracy. *NoScope* [144] leverages two

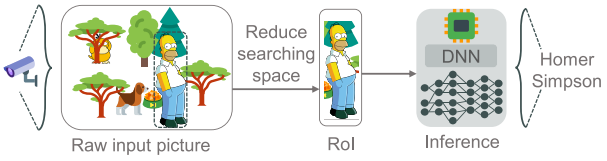


Fig. 12. Optimization for model inputs, e.g., narrowing down the searching space of DL models (pictures are with permission from [141]).

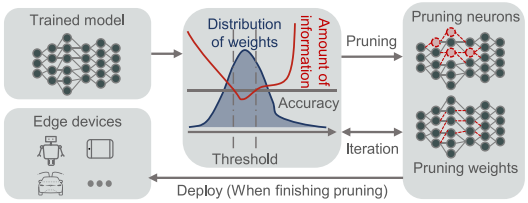


Fig. 13. Adaptive parameters pruning in model structure optimization.

types of models rather than the standard model (such as YOLO [9]): specialized models that waive the generality of standard models in exchange for faster inference, and difference detectors that identify temporal differences across input data. After performing efficient cost-based optimization of the model architecture and thresholds for each model, *NoScope* can maximize the throughput of DL services and by cascading these models. Besides, as depicted in Fig. 13, parameters pruning can be applied adaptively in model structure optimization as well [145]–[147]. 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 should be designed to work directly for optimized models [145].

- *Model Selection:* With various DL models, choosing the best one from available DL models in the edge requires weighing both precision and inference time. In [148], the authors use *k*NN to automatically construct a predictor, composed of DL models arranged in sequence. Then, the model selection can be determined by that predictor along with a set of automatically tuned features of the model input. Besides, combining different compression techniques (such as model pruning), multiple compressed DL models with different tradeoffs between the performance and the resource requirement can be derived. *AdaDeep* [149] explores the desirable balance between performance and resource constraints, and based on DRL, automatically selects various compression techniques (such as model pruning) to form a compressed model according to current available resources, thus fully utilizing the advantages of them.
- *Model Framework:* 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 [150]; 2) automatically compresses DL models into smaller dense matrices by finding the minimum number of non-redundant hidden elements [151]; 3) performs quantization and coding on all commonly used DL structures [146], [151], [152]; 4) specializes DL models to contexts and shares resources across multiple simultaneously executing DL models [152]. With respect to the hardware, running DL models on Static Random Access Memory (SRAM) achieves better energy savings compared to Dynamic RAM (DRAM) [146]. Hence, DL performance can be benefited if underlying hardware directly supports running optimized DL models [153] on the on-chip SRAM.

B. Segmentation of DL Models

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 [154], 2) distributed edge nodes [155], [156], or 3) collaborative "end-edge-cloud" architecture [12], [49], [157], [158].

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. 14, a general process for determining the partition point can be divided into three steps [12], [157]: 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 [156]. 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.

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 [159]. Nonetheless, this idea can be further improved by the emerged EEoI [160].

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

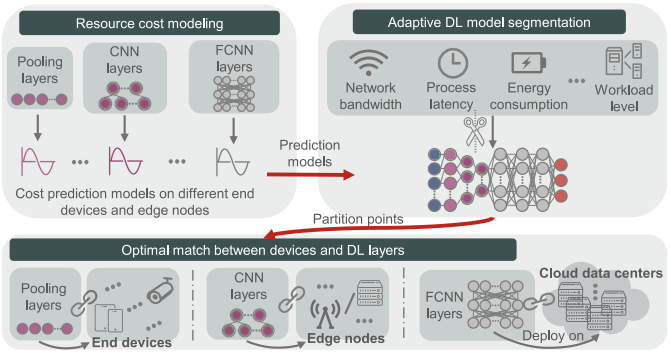


Fig. 14. Segmentation of DL models in the edge.

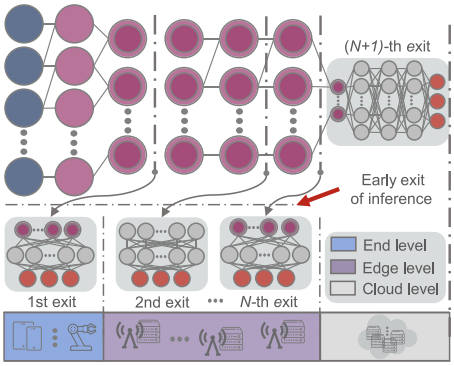


Fig. 15. Early exit of inference for DL inference in the edge.

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. 15, 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 [113], [161]. 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 [162] for adaptive DNN partitioning and EEoI.

D. Sharing of DL Computation

The requests from nearby users within the coverage of an edge node may exhibit spatiotemporal locality [163]. For instance, users within the same area might request recognition tasks for the same object of interest, and it may introduce redundant computation of DL inference. In this case, based on offline analysis of applications and online estimates of network conditions, *Cachier* [163] proposes to cache related DL models for recognition applications in the edge node and to minimize expected end-to-end latency by dynamically adjusting its cache size. Based on the similarity between consecutive frames in first-person-view videos, *DeepMon* [164] and *DeepCache* [165] utilize the internal processing structure of CNN layers to reuse the intermediate results of the previous frame to calculate the current frame, i.e., caching internally processed data within CNN layers, to reduce the processing latency of continuous vision applications.

Nevertheless, to proceed with effective caching and results reusing, accurate lookup for reusable results shall be addressed, i.e., the cache framework must systematically tolerate the variations and evaluate key similarities. *DeepCache* [165] performs cache key lookup to solve this. Specifically, it divides each video frame into fine-grained regions and searches for similar regions from cached frames in a specific pattern of video motion heuristics. For the same challenge, *FoggyCache* [166] first embeds heterogeneous raw input data into feature vectors with generic representation. Then, Adaptive Locality Sensitive Hashing (A-LSH), a variant of LSH commonly used for indexing high-dimensional data, is proposed to index these vectors for fast and accurate lookup. At last, Homogenized *k*NN, which utilizes the cached values to remove outliers and ensure a dominant cluster among the k records initially chosen, is implemented based on *k*NN to determine the reuse output from records looked up by A-LSH.

Differ from sharing inference results, *Mainstream* [167] proposes to adaptively orchestrate DNN stem-sharing (the common part of several specialized DL models) among concurrent video processing applications. By exploiting computation sharing of specialized models among applications trained through TL from a common DNN stem, aggregate per-frame compute time can be significantly decreased. Though more specialized DL models mean both higher model accuracy and less shared DNN stems, the model accuracy decreases slowly as less-specialized DL models are employed (unless the fraction of the model specialized is very small). This characteristic hence enables that large portions of the DL model can be shared with low accuracy loss in *Mainstream*.

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 frameworks can better maintain DL services running on the edge; 4) fair

platforms for evaluating Edge DL performance help further evolve the above implementations.

A. Edge Hardware 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 edge devices can be used to undertake lightweight DL computation, and hence avoiding communication with the cloud, but it still needs to face limited computation resources, memory footprint, and energy consumption. To break through these bottlenecks, in [143], the authors focus on ARM Cortex-M micro-controllers and develop *CMSIS-NN*, a collection of efficient NN kernels. By *CMSIS-NN*, 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.

With regard to the bottleneck when running CNN layers on mobile GPUs, *DeepMon* [164] decomposes the matrices used in the CNN layers to accelerate the multiplications between high-dimensional matrices. By this means, high-dimensional matrix operations (particularly multiplications) in CNN layers are available in mobile GPUs and can be accelerated. In view of this work, various mobile GPUs, already deployed in edge devices, can be potentially explored with specific DL models and play a more important role in enabling edge DL.

Other than DL inference [143], [164], important factors that affect the performance of DL training on mobile CPUs and GPUs are discussed in [168]. Since commonly used DL models, such as VGG [169], are too large for the memory size of mainstream edge devices, a relatively small Mentee network [170] 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 Field Programmable Gate Array (FPGA) is explored to study their feasibility for edge DL.

FPGA-based edge devices can achieve CNN acceleration with arbitrarily sized convolution and reconfigurable pooling [143], and they perform faster than the state-of-the-art CPU and GPU implementations [145] with respect to RNN-based speech recognition applications while achieving higher energy efficiency. In [52], the design and setup of an FPGA-based edge platform are developed to admit DL computation offloading from mobile devices. On implementing the FPGA-based edge platform, 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 one.

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 specific DL models.
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 process.
Timing latency	FPGA	Timing on FPGAs can be an order of magnitude faster than GPUs.

Nevertheless, it is still pended to determine whether FPGAs or GPUs/CPUs are more suitable for edge computing, as shown in Table IV. Elaborated experiments are performed in [171] to investigate the advantages of FPGAs 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 [52], 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.

B. Communication and Computation Modes for Edge DL

Though on-device DL computation, illustrated in Section V, can cater for lightweight DL services. Nevertheless, an independent end device still cannot afford intensive DL computation tasks. The concept of edge computing can potentially cope with this dilemma by offloading DL computation from end devices to edge or (and) the cloud. 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 modes for Edge DL computation, as exhibited in Fig. 16.

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. 16(a)). This kind of offloading is straightforward by extricating itself from DL task decomposition and combinatorial problems of resource optimization, which may bring about additional

TABLE V
DETAILS ABOUT EDGE COMMUNICATION AND COMPUTATION MODES FOR DL

	Ref.	DL Model	End/Edge/Cloud	Network	Dependency	Objective	Performance
Integral Offloading	*DeepDecision* [172]	YOLO	Samsung Galaxy S7 / Server with a quad-core CPU at 2.7GHz, GTX970 and 8GB RAM / N/A	Simulated WLAN & LAN	TensorFlow, Darknet	Consider the complex interaction between model accuracy, video quality, battery constraints, network data usage, and network conditions to determine an optimal offloading strategy	Achieve about 15 FPS video analytic while possessing higher accuracy than that of the baseline approaches
	MASM [173]	\	Simulated devices / Cloudlet / N/A	\	\	Optimize workload assignment weights and the computation capacities of the VMs hosted on the Cloudlet	\
	EdgeEye [108]	DetectNet, FaceNet	Cameras / Server with Intel i7-6700, GTX 1060 and 24GB RAM / N/A	Wi-Fi	TensorRT, ParaDrop, Kurento	Offload the live video analytics tasks to the edge using EdgeEye API, instead of using DL framework specific APIs, to provide higher inference performance	\
Partial Offloading	*DeepWear* [174]	MobileNet, GoogLeNet, DeepSense, etc.	Commodity smartwatches running Android Wear OS / Commodity smartphone running Android / N/A	Bluetooth	TensorFlow	Provide context-aware offloading, strategic model partition, and pipelining support to efficiently utilize the processing capacity of the edge	Bring up to 5.08× and 23.0× execution speedup, as well as 53.5% and 85.5% energy saving against wearable-only and handheld-only strategies, respectively
	IONN [175]	AlexNet	Embedded board Odroid XU4 / Server with an quad-core CPU at 3.6GHz, GTX 1080 Ti and 32GB RAM / Unspecified	WLAN	Caffe	Partitions the DNN layers and incrementally uploads the partitions to allow collaborative execution by the end and the edge (or cloud) to improves both the query performance and the energy consumption	Maintain almost the same uploading latency as integral uploading while largely improving query execution time
Vertical Collaboration	[176]	CNN, LSTM	Google Nexus 9 / Server with an quad-core CPU and 16GB RAM / 3 desktops, each with i7-6850K and 2×GTX 1080 Ti	WLAN & LAN	Apache Spark, TensorFlow	Perform data pre-processing and preliminary learning at the edge to reduce the network traffic, so as to speed up the computation in the cloud	Achieve 90% accuracy while reducing the execution time and the data transmission
	Neurosurgeon [12]	AlexNet, VGG, Deepface, MNIST, Kaldi, SENNA	Jetson TK1 mobile platform / Server with Intel Xeon E5×2, NVIDIA Tesla K40 GPU and 256GB RAM / Unspecified	Wi-Fi, LTE & 3G	Caffe	Adapt to various DNN architectures, hardware platforms, wireless connections, and server load levels, and choose the partition point for best latency and best mobile energy consumption	Improve end-to-end latency by 3.1× on average and up to 40.7×, reduce mobile energy consumption by 59.5% on average and up to 94.7%, and improve data-center throughput by 1.5× on average and up to 6.7×
	[161]	BranchyNet	\	\	\	Minimize communication and resource usage for devices while allowing low-latency classification via EEoI	Reduce the communication cost by a factor of over 20× while achieving 95% overall accuracy
	[102]	Faster R-CNN	Xiaomi 6 / Server with i7 6700, GTX 980Ti and 32GB RAM / Work station with E5-2683 V3, GTX TitanXp×4 and 128GB RAM	WLAN & LAN	\	Achieve efficient object detection via wireless communications by interactions between the end, the edge and the cloud	Lose only 2.5% detection accuracy under the image compression ratio of 60% while significantly improving image transmission efficiency
	VideoEdge [109]	AlexNet, DeepFace, VGG16	10 Azure nodes emulating Cameras / 2 Azure nodes / 12 Azure nodes	Emulated hierarchical networks	\	Introduce dominant demand to identify the best tradeoff between multiple resources and accuracy	Improve accuracy by 5.4× compared to VideoStorm and only lose 6% accuracy of the optimum
Horizontal Collaboration	*MoDNN* [177]	VGG-16	Multiple LG Nexus 5 / N/A / N/A	WLAN	MXNet	Partition already trained DNN models onto several mobile devices to accelerate DNN computations by alleviating device-level computing cost and memory usage	When the number of worker nodes increases from 2 to 4, MoDNN can speedup the DNN computation by 2.17-4.28×
	[130]	VGGNet-E, AlexNet	Xilinx Virtex-7 FPGA simulating multiple end devices / N/A / N/A	On-chip simulation	Torch, Vivado HLS	Fuse the processing of multiple CNN layers and enable caching of intermediate data to save data transfer (bandwidth)	Reduce the total data transfer by 95%, from 77MB down to 3.6MB per image
	DeepThings [156]	YOLOv2	Perfromance-limited Raspberry Pi 3 Model B / Raspberry Pi 3 Model B as gateway / N/A	WLAN	Darknet	Employ a scalable Fused Tile Partitioning of CNN layers to minimize memory footprint while exposing parallelism and a novel work scheduling process to reduce overall execution latency	Reduce memory footprint by more than 68% without sacrificing accuracy, improve throughput by 1.7×-2.2× and speedup CNN inference by 1.7×-3.5×
	DeepCham [104]	AlexNet	Multiple LG G2 / Wi-Fi router connected with a Linux server / N/A	WLAN & LAN	Android Caffe, OpenCV, EdgeBoxes	Coordinate participating mobile users for collaboratively training a domain-aware adaptation model to improve object recognition accuracy	Improve the object recognition accuracy by 150% when compared to that achieved merely using a generic DL model
	LAVEA [106]	OpenALPR	Raspberry PI 2 & Raspberry PI 3 / Servers with quad-core CPU and 4GB RAM / N/A	WLAN & LAN	Docker, Redis	Design various task placement schemes that are tailed for inter-edge collaboration to minimize the service response time	Have a speedup ranging from 1.3× to 4× (1.2× to 1.7×) against running in local (client-cloud confguration)

computation cost and scheduling delay, and thus simple to implement. In [172], the proposed distributed infrastructure *DeepDecision* ties together powerful edge nodes with less powerful end devices. In *DeepDecision*, DL inference can be performed on the end or the edge, depending on the trade-offs between the inference accuracy, the inference latency, the

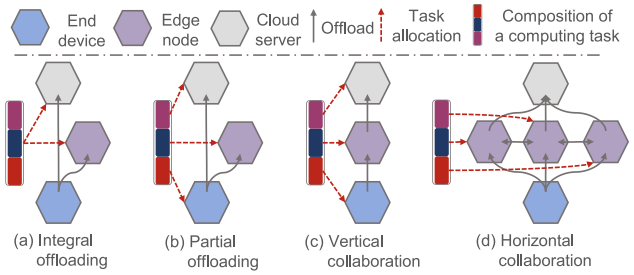


Fig. 16. Communication and computation modes for Edge DL.

DL model size, the battery level, and network conditions. With regard to each DL task, the end device decides whether locally processing or offloading it to an 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 sizes and performance in the edge can be adopted to fulfill one kind of task. Hence, multiple 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 is selected as the serving model. For instance, by optimizing the workload assignment weights and computing capacities of VMs, *MASM* [173] can reduce the energy cost and delay while guaranteeing the DL inference accuracy.

2) Partial Offloading: Partially offloading the DL task to the edge is also feasible (as depicted in Fig. 16(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 [178], *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.

Further, particularly for DL computation, *DeepWear* [174] abstracts a DL model as a Directed Acyclic Graph (DAG), where each node represents a layer and each edge represents the data flow among those layers. To efficiently determine partial offloading decisions, *DeepWear* first prunes the DAG by keeping only the computation-intensive nodes, and then grouping the repeated sub-DAGs. In this manner, the complex DAG can be transformed into a linear and much simpler one, thus enabling a linear complexity solution for selecting the optimal partition to offload.

Nevertheless, uploading a part of the DL model to the edge nodes may still seriously delay the whole process of offloading DL computation. To deal with this challenge, an incremental offloading system *IONN* is proposed in [175]. 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. 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 concerning the best partitions of the DL model and the uploading order. Specifically, on the one hand, DNN layers, performance benefit and uploading overhead of which are high and low, respectively, are preferred to be uploaded first, and thus making the edge node quickly build a partial DNN to achieve the best-expected query performance. On the other hand, unnecessary DNN layers, which cannot bring in any performance increase, are not 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 [176]. 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 [161]. 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 [177], executes DL in a 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 [130]. In [156], 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 [156], can adaptively distribute FTP partitions for balancing the workload of collaborated edge devices.

C. Tailoring Edge Frameworks for DL

Though there are gaps between the computational complexity and energy efficiency required by DL and the capacity of edge hardware [179], customized edge DL frameworks can help efficiently 1) match edge platform and DL models; 2) exploit underlying hardware in terms of performance and power; 3) orchestrate and maintain DL services automatically.

First, where to deploy DL services in edge computing (cellular) networks should be determined. The RAN controllers deployed at edge nodes are introduced in [180] to collect the data and run DL services, while the network controller, placed in the cloud, orchestrates the operations of the RAN controllers. In this manner, after running and feeding analytics and extract relevant metrics to DL models, these controllers can provide DL services to the users at the network edge.

Second, as the deployment environment and requirements of DL models can be substantially different from those during model development, customized operators, adopted in developing DL models with (Py)Torch, TensorFlow, etc., may not be directly executed with the DL framework at the edge. To bridge the gap between deployment and development, the authors of [181] propose to specify DL models in development using the deployment tool with an operator library from the DL framework deployed at the edge. Furthermore, to automate the selection and optimization of DL models, *ALOHA* [182] formulates a toolflow: 1) Automate the model design. It generates the optimal model configuration by taking into account the target task, the set of constraints and the target architecture; 2) Optimize the model configuration. It partitions the DL model and accordingly generates architecture-aware mapping information between different inference tasks and the available resources. 3) Automate the model porting. It translates the mapping information into adequate calls to computing and communication primitives exposed by the target architecture.

Third, the orchestration of DL models deployed at the edge should be addressed. *OpenEI* [183] defines each DL algorithm as a four-element tuple <Accuracy, Latency, Energy, Memory Footprint> to evaluate the Edge DL capability of the target hardware platform. Based on such tuple, *OpenEI* can select a matched model for a specific edge platform based on different Edge DL capabilities in an online manner. *Zoo* [184] provides a concise Domain-specific Language (DSL) to enable easy and type-safe composition of DL services. Besides, to enable a wide range of geographically distributed topologies, analytic engines, and DL services, *ECO* [185] uses a graph-based overlay network approach to 1) model and track pipelines and dependencies and then 2) map them to geographically distributed analytic engines ranging from small edge-based engines to powerful multi-node cloud-based engines. By this means, DL computation can be distributed as needed to manage cost and performance, while also supporting other practical situations, such as engine heterogeneity and discontinuous operations.

Nevertheless, these pioneer works are not ready to natively support valuable and also challenging features discussed in Section VI-B, such as computation offloading and collaboration, which still calls for further development.

D. Performance Evaluation for Edge DL

Throughout the process of selecting appropriate edge hardware and associated software stacks for deploying different kinds of Edge DL services, it is necessary to evaluate their performance. Impartial evaluation methodologies can point out possible directions to optimize software stacks for specific edge hardware. In [186], for the first time, the performance of DL libraries is evaluated by executing DL inference on resource-constrained edge devices, 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* [54] extensively evaluates 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 that of executing it. These discoveries imply that there are still

opportunities to further optimize the fusion of edge hardware, edge software stacks, and DL libraries.

Nonetheless, a standard testbed for Edge DL is missing, which hinders the study of edge architectures for DL. To evaluate the end-to-end performance of Edge DL services, not only the edge computing architecture but also its combination with end devices and the cloud shall be established, such as *openLEON* [187] and *CAVBench* [188] particularly for vehicular scenarios. Furthermore, 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., is ponderable in facilitating the evolution of "Edge Computing for DL".

VII. Deep Learning Training at Edge

Present DL training (distributed or not) in the cloud data center, namely cloud training, or cloud-edge training [50], 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 costs. 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 "DL Training at Edge". Such kind of DL training may require significant resources to digest distributed data and exchange updates. Nonetheless, FL is emerging and is promised to address these issues. We summarize select works on FL in Table VI.

A. Distributed Training at Edge

Distributed training at the edge can be traced back to the work of [189], where a decentralized Stochastic Gradient Descent (SGD) 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 [190], two different distributed learning solutions for edge computing environments are proposed. As depicted in Fig. 17, 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

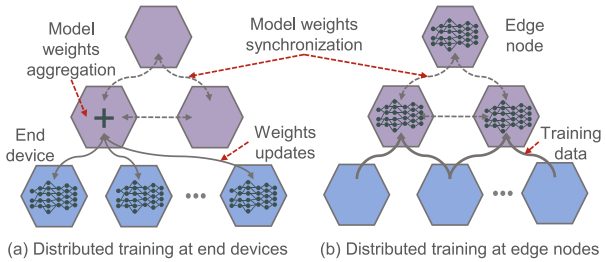


Fig. 17. Distributed DL training at edge environments.

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.

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 [191]). *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 [192]. The first is transmitting only important gradients by taking advantage of sparse training gradients [193]. 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

TABLE VI
SUMMARY OF THE SELECTED WORKS ON FL

	Ref.	DL Model	Scale	Dependency	Main Idea	Key Metrics or Performance
Vanilla FL	[198]	FCNN, CNN, LSTM	Up to 5e5 clients	TensorFlow	Leave the training data distributed on the mobile devices, and learns a shared model by aggregating locally-training updates	Communication rounds reduction: 10-100×
	[199]	RNN	Up to 1.5e6 clients	TensorFlow	Pace steering for scalable FL	Scalability improvement: up to 1.5e6 clients
Communication-efficient FL	[202]	ResNet18	4 clients per cluster / 7 clusters	\	Gradient sparsification; Periodic averaging	Top 1 accuracy; Communication latency reduction
	[203]	CNN, LSTM	Up to 1e3 clients	\	Sketched updates	Communication cost reduction: by two orders of magnitude
	[205]	CNN	Up to 500 clients	TensorFlow	Lossy compression on the global model; Federated Dropout	Downlink reduction: 14×; Uplink reduction: 28×; Local computation reduction: 1.7×
	[211]	CNN, RNN	Up to 37 clients	TensorFlow	Let faster clients continue with their mini-batch training to keep overall synchronization	Convergence acceleration: 62.4%
	[208]	CNN	5-500 clients (simulation); 3 Raspberry Pi and 2 laptops (testbed)	\	Design a control algorithm that determines the best trade-off between local update and global aggregation	Training accuracy under resource budget
	[204]	FCNN	50 clients	\	Periodic averaging; Partial device participation; Quantized message-passing	Total training loss and time
	[212]	CNN	500 clients	\	Global data distribution based data augmentation; Mediator based multi-client rescheduling	Top 1 accuracy imrpovement: 5.59%-5.89%; Communication traffic reduction: 92%
	[207]	LeNet, CNN, VGG11	10 Raspberry Pi	Py(Torch)	Jointly trains and prunes the model in a federated manner	Communication and computation load reduction
Resource -optimized FL	[218]	AlexNet, LeNet	Multiple Nvidia Jetson Nano	\	Partially train the model by masking a particular number of resource-intensive neurons	Training acceleration: 2×; Model accuracy improvement: 4%
	[219]	\	Up to 50 clients	TensorFlow	Jointly optimize FL parameters and resources of user equipments	Convergence rate; Test accuracy
	[220]	\	20 clients / 1 BS	\	Jointly optimize wireless resource allocation and client selection	Reduction of the FL loss function value: up to 16%
	[221]	LSTM	23-1,101 clients	TensorFlow	Modify FL training objectives with α-fairness	Fairness; Training accuracy
Security -enhanced FL	[201]	CNN	100 clients	MXNET	Use the trimmed mean as a robust aggregation	Top 1 accuracy against data poisoning
	[222]	\	2e10-2e14 clients	\	Use Secure Aggregation to protect the privacy of each client's model gradient	Communication expansion: 1.73×-1.98×
	[223]	\	10 clients	\	Leverage blockchain to exchange and verify model updates of local training	Learning completion latency

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 [194]. 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 [195], 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 [196]. Different from [190]–[192], in the scenario of [196], training data are trained at edge nodes as well as be uploaded to the cloud for further data analysis. Hence, Laplace noises [197] are added to these possibly exposed training data for enhancing the training data privacy assurance.

B. Vanilla 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) [198], [199] 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 [200], [201]. End devices, edge nodes and servers in the cloud can be equivalently deemed as clients in FL. These clients are assumed

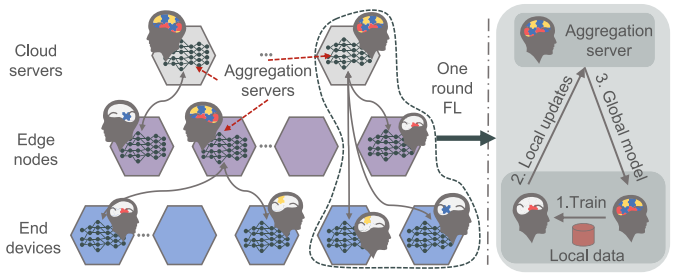


Fig. 18. Federated learning among hierarchical network architectures.

capable of handling different levels of DL training tasks, and hence contribute their updates to the global DL model. In this section, fundamentals of FL are discussed.

Without requiring uploading data for central cloud training, FL [198], [199] 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. 18, 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 [195], incurred by uploading training data to the cloud. Besides, FL introduces *FederatedAveraging* to combine local SGD on each device with a server performing model averaging. Experimental results corroborate *FederatedAveraging* 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.

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 *FederatedAveraging*; 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 *FederatedAveraging*, 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 [197], which are useful in avoiding the disclosure of privacy-sensitive data contained in local updates, can be applied naturally.

C. Communication-Efficient FL

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 DL model size is large enough, uploading updates, such as model weights, from edge devices to the central server may also consume nonnegligible communication resources. To meet this, we can let FL clients communicate with the central server periodically (rather continually) to seek consensus on the shared DL model [202]. In addition, *structured update*, *sketched update* can help enhance the communication efficiency when clients uploading updates to the server as well. *Structured update* means restricting the model updates to have a pre-specified structure, specifically, 1) low-rank matrix; or 2) sparse matrix [202], [203]. On the other hand, for *sketched update*, full model updates are maintained. But before uploading them for model aggregation, combined operations of subsampling, probabilistic quantization, and structured random rotations are performed to compress the full updates [203]. *FedPAQ* [204] simultaneously incorporates these features and provides near-optimal theoretical guarantees for both strongly convex and non-convex loss functions, while empirically demonstrating the communication-computation tradeoff.

Different from only investigating on reducing communication cost on the uplink, [205] 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 [203]. Naturally, such kind of model compression is lossy, and unlike on the uplink (multiple edge devices are uploading their models for averaging), the loss cannot be mitigated by averaging on the downlink. *Kashin's representation* [206] 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 train a model based on the whole global model locally, but only to train a smaller sub-model or pruned model [207] instead. Since sub-models and pruned models are more lightweight than the global model, the amount of data in updates uploading is reduced.

Computation resources of edge devices are scarce compared to the cloud. Additional challenges should be considered to improve communication efficiencies: 1) Computation resources are heterogeneous and limited at edge devices; 2) Training data at edge devices may be distributed non-uniformly [208]–[210]. For more powerful edge devices, *ADSP* [211] lets them continue training while committing model aggregation at strategically decided intervals. For general cases, based on the deduced convergence bound for distributed learning with non-IID data distributions, the aggregation frequency under given resource budgets among all participating devices can be optimized with theoretical guarantees [208]. *Astraea* [212] reduces 92% communication traffic by designing a mediator-based multi-client rescheduling strategy. On the one hand, *Astraea* leverages data augmentation [5] to alleviate the defect of non-uniformly distributed training data. On the other hand, *Astraea* designs a greedy strategy for mediator-based rescheduling, in order to assign clients to the mediators. Each mediator traverses the data distribution of all unassigned clients to select the appropriate

participating clients, aiming to make the mediator's data distribution closest to the uniform distribution, i.e., minimizing the KullbackLeibler divergence [213] between mediator's data distribution and uniform distribution. When a mediator reaches the max assigned clients limitation, the central server will create a new mediator and repeat the process until all clients have been assigned with training tasks.

Aiming to accelerate the global aggregation in FL, [214] takes advantage of over-the-air computation [215]–[217], 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. Then the server only needs to superpose these reshaped weights as the aggregation results, nonetheless, without other aggregation operations.

D. Resource-Optimized FL

When FL deploys the same neural network model to heterogeneous edge devices, devices with weak computing power (stragglers) may greatly delay the global model aggregation. Although the training model can be optimized to accelerate the stragglers, due to the limited resources of heterogeneous equipment, the optimized model usually leads to diverged structures and severely defect the collaborative convergence. *ELFISH* [218] first analyzes the computation consumption of the model training in terms of the time cost, memory usage, and computation workload. Under the guidance of the model analysis, which neurons need to be masked in each layer to ensure that the computation consumption of model training meets specific resource constraints can be determined. Second, unlike generating a deterministically optimized model with diverged structures, different sets of neurons will be dynamically masked in each training period and recovered and updated during the subsequent aggregation period, thereby ensuring comprehensive model updates overtime. It is worth noting that although *ELFISH* improves the training speed by $2\times$ through resource optimization, the idea of *ELFISH* is to make all stragglers work synchronously, the synchronous aggregation of which may not able to handle extreme situations.

When FL is deployed in a mobile edge computing scenario, the wall-clock time of FL will mainly depend on the number of clients and their computing capabilities. Specifically, the total wall-clock time of FL includes not only the computation time but also the communication time of all clients. On the one hand, the computation time of a client depends on the computing capability of the clients and local data sizes. On the other hand, the communication time correlates to clients' channel gains, transmission power, and local data sizes. Therefore, to minimize the wall-clock training time of the FL, appropriate resource allocation for the FL needs to consider not only FL parameters, such as accuracy level for computation-communication trade-off, but also the resources allocation on the client side, such as power and CPU cycles.

However, minimizing the energy consumption of the client and the FL wall-clock time are conflicting. For example, the client can save energy by always maintain its CPU at low frequency, but this will definitely increase training time. Therefore, in order to strike a balance between energy cost and training time, the authors of [219] first design a new FL algorithm *FEDL* for each client to solve its local problem approximately till a local accuracy level achieved. Then, by using Pareto efficiency model [224], they formulate a non-convex resource allocation problem for *FEDL* over wireless networks to capture the trade-off between the clients' energy cost and the FL wall-clock time). Finally, by exploiting the special structure of that problem, they decompose it into three sub-problems, and accordingly derive closed-form solutions and characterize the impact of the Pareto-efficient controlling knob to the optimal.

Since the uplink bandwidth for transmitting model updates is limited, the BS must optimize its resource allocation while the user must optimize its transmit power allocation to reduce the packet error rates of each user, thereby improving FL performance. To this end, the authors of [220] formulate resource allocation and user selection of FL into a joint optimization problem, the goal of which is to minimize the value of the FL loss function while meeting the delay and energy consumption requirements. To solve this problem, they first derive a closed-form expression for the expected convergence rate of the FL in order to establish an explicit relationship between the packet error rates and the FL performance. Based on this relationship, the optimization problem can be reduced to a mixed-integer nonlinear programming problem, and then solved as follows: First, find the optimal transmit power under a given user selection and resource block allocation; Then, transform the original optimization problem into a binary matching problem; Finally, using Hungarian algorithm [225] to find the best user selection and resource block allocation strategy.

The number of devices involved in FL is usually large, ranging from hundreds to millions. Simply minimizing the average loss in such a large network may be not suited for the required model performance on some devices. In fact, although the average accuracy under vanilla FL is high, the model accuracy required for individual devices may not be guaranteed. To this end, based on the utility function α-fairness [226] used in fair resource allocation in wireless networks, the authors of [221] define a fair-oriented goal q-FFL for joint resource optimization. q-FFL minimizes an aggregate re-weighted loss parameterized by q, so that devices with higher loss are given higher relative weight, thus encouraging less variance (i.e., more fairness) in the accuracy distribution. Adaptively minimizing q-FFL avoids the burden of hand-crafting fairness constraints, and can adjust the goal according to the required fairness dynamically, achieving the effect of reducing the variance of accuracy distribution among participated devices.

E. Security-Enhanced FL

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* [201] defines a trimmed mean operation. By filtering out not only the 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 [222] 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 [223], 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 VII.

A. DL for Adaptive Edge Caching

From Content Delivery Network (CDN) [227] to caching contents in cellular networks, caching in the network have been investigated over the years to deal with soaring demand for multimedia services [228]. Aligned with the concept of pushing contents near to users, edge caching [229], is deemed as a promising solution for further reducing the redundant data transmission, easing the pressure of cloud data centers and improving the QoE.

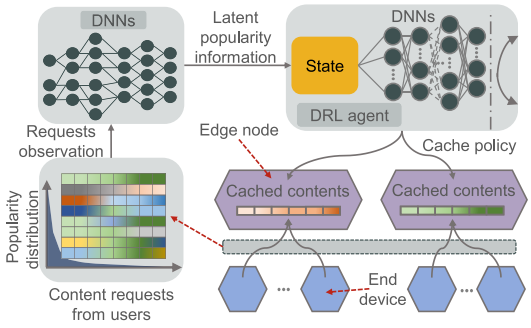


Fig. 19. DL and DRL for optimizing the edge caching policy.

Edge caching meets two challenges: 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 [230]; 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 [231]. 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. 19, 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 1) the features of users and contents and 2) the strategy of content placement and delivery.

For the first purpose, DL can be used to process raw data collected from the mobile devices of users and hence extract the features of the users and content as a feature-based content popularity matrix. By this means, the popular content at the core network is estimated by applying feature-based collaborative filtering to the popularity matrix [232].

For the second purpose, when using DNNs to optimize the strategy of edge caching, online heavy computation iterations can be avoided by offline training. A DNN, which consists of an encoder for data regularization and a followed hidden layer, can be trained with solutions generated by optimal or heuristic algorithms and be deployed to determine the cache policy [233], hence avoiding online optimization iterations. Similarly, in [234], 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 [233], [234], the complexity of optimization algorithms can be transferred to the training of DNNs, and thus breaking the practical limitation of employing them. In this case, DL is used to learn input-solution relations, and DNN-based methods are only available when optimization

TABLE VII
DL FOR OPTIMIZING EDGE APPLICATION SCENARIOS

		Ref.	DL	Comm. Scale	Inputs - DNN (States - DRL)	Outputs - DNN (Action - DRL)	Loss func. - DL (Reward - DRL)	Performance
DL for Adaptive Edge Caching		[232]	SDAE	60 users / 6 SBSs	User features, content features	Feature-based content popularity matrix	Normalized differences between input features and the consequent reconstruction	QoE improvement: up to 30%; Backhaul offloading: 6.2%
		[233]	FCNN	100-200 UEs per cell / 7 BSs	Channel conditions, file requests	Caching decisions	Normalized differences between prediction decisions and the optimum	Prediction accuracy: up to 92%; Energy saving: 8% gaps to the optimum
		[234]	FCNN	UEs with density 25-30 / Multi-tier BSs	Current content popularity, last content placement probability	Content placement probability	Statistical average of the error between the model outputs and the optimal CVX solution	Prediction accuracy: slight degeneration to the optimum
		[235]	FCNN CNN	Cars / 6 RSUs with MEC servers	Facial images - CNN; Content features - FCNN	Gender and age prediction - CNN; Content request probability - FCNN	N/A - CNN; Cross entropy error - FCNN	Caching accuracy: up to 98.04%
		[237]	RNN	20 UEs / 10 servers	User historical traces	User location prediction	Cross entropy error	Caching accuracy: up to 75%
		[241]	DDPG	Multiple UEs / Single BS	Features of cached contents, current requests	Content replacement	Cache hit rate	Cache hit rate: about 50%
DL for Optimizing Edge Task Offloading		[252]	FCNN	20 miners / Single edge node	Bidder valuation profiles of miners	Assignment probabilities, conditional payments	Expected, negated revenue of the service provider	Revenue increment
		[257]	Double-DQL	Single UE	System utilization states, dynamic slack states	DVFS algorithm selection	Average energy consumption	Energy saving: 2%-4%
		[253]	DQL	Multiple UEs / Single eNodeB	Sum cost of the entire system, available capacity of the MEC server	Offloading decision, resource allocation	Negatively correlated to the sum cost	System cost reduction
		[255]	DDPG	Multiple UEs / Single BS with an MEC server	Channel vectors, task queue length	Offloading decision, power allocation	Negative wighted sum of the power consumption and task queue length	Computation cost reduction
		[254]	DQL	Single UE / Multiple MEC servers	Previous radio bandwidth, predicted harvested energy, current battery level	MEC server selection, offloading rate	Composition of overall data sharing gains, task drop loss, energy consumption and delay	Energy saving; Delay improvement
		[251]	Double-DQL	Single UE / 6 BSs with MEC servers	Channel gain states, UE-BS association state, energy queue length, task queue length	Offloading decision, energy units allocation	Composition of task execution delay, task drop times, task queuing delay, task failing penalty and service payment	Offloading performance improvement
		[258]	DROO	Multiple UEs / Single MEC server	Channel gain states	Offloading action	Computation rate	Algorithn execution time: less than 0.1s in 30-UE network
DL for Edge Management and Maintenance	Communication	[259]	RNN & LSTM	53 vehicles / 20 fog servers	Coordinates of vehicles and interacting fog nodes, time, service cost	Cost prediction	Mean absolute error	Prediction accuracy: 99.2%
		[260]	DQL	4 UEs / Multiple RRHs	Current on-off states of processors, current communication modes of UEs, cache states	Processor state control, communication mode selection	Negative of system energy consumption	System power consumption
	Security	[261]	DQL	Multiple UEs / Multiple edge nodes	Jamming power, channel bandwidth, battery levels, user density	Edge node and channel selection, offloading rate, transmit power	Composition of defense costs and secrecy capacity	Signal SINR increasement
	Joint Optimization	[110]	Double-Dueling DQL	Multiple UEs / 5 BSs and 5 MEC servers	Status from each BS, MEC server and content cache	BS allocation, caching decision, offloading decision	Composition of received SNRs, computation capabilities and cache states	System utility increasement
		[262]	AC DRL	20 UEs per router / 3 fog nodes	States of requests, fog nodes, tasks, contents and SINR	Decisions about fog node, channel, resource allocation, offloading and caching	Composition of computation offloading delay and content delivery delay	Average service latency: 1.5-4.0s
		[112]	DQL	50 vehicles / 10 RSUs	States of RSUs, vehicles and caches, contact rate, contact times	RSU assignment, caching control and control	Composition of communication, storage and computation cost	Backhaul capacity mitigation; Resource saving

algorithms for the original caching problem exist. Therefore, the performance of DNN-based methods bounds by fixed optimization algorithms and is not self-adapted.

In addition, DL can be utilized for customized edge caching. For example, to minimize 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 [235]). 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 [236] 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, concerning taking full advantage of users' features, [237] points out that the user's willing to access the content in different environments is varying. Inspired by this, RNN is used to predict the trajectories of users. And based on these predictions, all contents of users' interests are then prefetched and cached in advance at 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 [238] 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 [239]. Compared to traditional RL irrelevant to DL, such as *Q*-learning [240] and Multi-Armed Bandit (MAB) learning [230], 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 [241], 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 [242] 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*NN 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. DL for Optimizing Edge Task Offloading

Edge computing allows edge devices offload part of their computing tasks to the edge node [243], under constraints of energy, delay, computing capability, etc. As shown in Fig. 20, 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 [244], 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 [245]–[255]. 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 [245], [256], [247] infeasible actually.

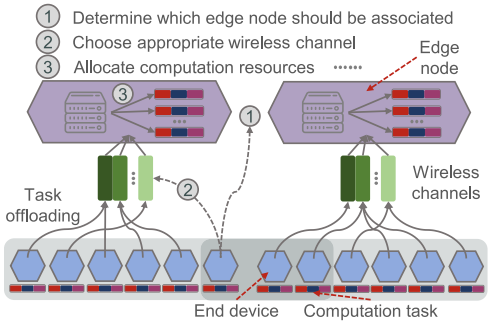


Fig. 20. Computation offloading problem in edge computing.

1) Use Cases of DNNs: In [249], 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 [252]. The computing and energy resources consumption of mining tasks on edge devices may 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 [252] 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 enhance the processing efficiency of the computation tasks, the reliability of offloading suffers from the potentially low quality of wireless environments. In [248], to maximize offloading utilities, the authors first quantify the influence of various communication modes on the task offloading performance and accordingly propose applying DQL to online select the optimal target edge node and transmission mode. For optimizing the total offloading cost, a DRL agent

that modifies Dueling- and Double-DQL [263] can allocate edge computation and bandwidth resources for end devices.

Besides, offloading reliability should also be concerned. The coding rate, by which transmitting the data, is crucial to make the offloading meet the required reliability level. Hence, in [250], effects of the coding block-length are investigated and an MDP concerning resource allocation is formulated and then solved by DQL, in order to improve the average offloading reliability. Exploring further on scheduling fine-grained computing resources of the edge device, in [257], Double-DQL [89] is to determine the best Dynamic Voltage and Frequency Scaling (DVFS) algorithm. Compared to DQL, the experiment results indicate that Double-DQL can save more energy and achieve higher training efficiency. Nonetheless, the action space of DQL-based approaches may increase rapidly with increasing edge devices. Under the circumstances, a pre-classification step can be performed before learning [253] to narrow the action space.

IoT edge environments powered by Energy Harvesting (EH) is investigated in [251], [254]. In EH environments, 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 [254]. Further, in [251], 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. To perform more fine-grained power control for local execution and task offloading, policy-gradient-based DRL should be considered. For example, compared tot he discrete power control strategy based on DQL, DDPG can adaptively allocate the power of edge devices with finer granularity [255].

Freely letting DRL agents take over the whole process of computation offloading may lead to huge computational complexity. Therefore, only employing DNN to make partial decisions can largely reduce the complexity. For instance, in [258], the problem of maximizing the weighted sum computation rate 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 than both, the action space of the DRL agent is narrowed, and the offloading performance is not impaired as well since the resource allocation problem is solved optimally.

C. DL for Edge Management and Maintenance

Edge DL services are envisioned to be deployed on BSs in cellular networks, as implemented in [264]. Therefore, edge management and maintenance require optimizations from multiple perspectives (including communication perspective). Many works focus on applying DL in wireless communication [265]–[267]. Nevertheless, management and maintenance at the edge should consider more aspects.

1) Edge Communication: When edge nodes are serving mobile devices (users), mobility issues in edge computing networks should be addressed. DL-based methods can be used to assist the smooth transition of connections between devices and edge nodes. To minimize energy consumption per bit, in [268], the optimal device association strategy is approximated by a DNN. Meanwhile, a digital twin of network environments is established at the central server for training this DNN off-line. To minimize the interruptions of a mobile device moving from an edge node to the next one throughout its moving trajectory, the MLP can be used to predict available edge nodes at a given location and time [259]. Moreover, determining the best edge node, with which the mobile device should associate, still needs to evaluate the cost (the latency of servicing a request) for the interaction between the mobile device and each edge node. 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.

Aiming at minimizing long-term system power consumption in the communication scenario with multiple modes (to serve various IoT services), i.e., Cloud-Radio Access Networks (C-RAN) mode, Device-to-Device (D2D) mode, and Fog radio Access Point (FAP) mode, DQL can be used to control communication modes of edge devices and on-off states of processors throughout the communicating process [260]. After determining the communication mode and the processors' on-off states of a given edge device, the whole problem can be degraded into an Remote Radio Head (RRH) transmission power minimization problem and solved. Further, TL is integrated with DQL to reduce the required interactions with the environment in the DQL training process while maintaining a similar performance without TL.

2) Edge Security: Since edge devices 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 the edge computing system should be enhanced. First, the system should be able to actively detect unknown attacks, for instance, using DL techniques to extract features of eavesdropping and jamming attacks [269]. According to the attack mode detected, the system determines the strategy of security protection. 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 [261] or protect user location privacy and the usage pattern privacy [270]. 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 of the signals, the bit error rate 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 [271]. 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) [272], 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 apply SDN to edge computing networks directly. In [273], 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 systems 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 [262].

IoVs is a special case of IoTs and focuses on connected vehicles. Similar to the consideration of integrating networking, caching and computing as in [262], 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 IoVs [110]. In addition, 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 [112], 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. Lessons Learned and Open Challenges

To identify existing challenges and circumvent potential misleading directions, we briefly introduce the potential scenario of "*DL application on Edge*", and separately discuss open issues related to four enabling technologies that we focus on, i.e., "*DL inference in Edge*", "*Edge Computing for DL*", "*DL training at Edge*" and "*DL for optimizing Edge*".

A. More Promising Applications

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.

For example, 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. Besides, intelligent driving involves speech recognition, image recognition, intelligent decision making, etc. Various DL applications in intelligent driving, such as collision warning, require edge computing platforms to ensure millisecond-level interaction delay. In addition, edge perception is more conducive to analyze the traffic environment around the vehicle, thus enhancing driving safety.

B. General DL Model for Inference

When deploying DL in edge devices, it is necessary to accelerate DL inference by model optimization. In this section, lessons learned and future directions for "*DL inference in Edge*", with respect to model compression, model segmentation, and EEoI, used to optimize DL models, is discussed.

1) Ambiguous Performance Metrics: For an Edge DL service for a specific task, there are usually a series of DL model candidates that can accomplish the task. However, it is difficult for service providers to choose the right DL model for each service. Due to the uncertain characteristics of edge computing networks (varying wireless channel qualities, unpredictable concurrent service requests, etc.), commonly used standard performance indicators (such as top-k accuracy [138] or mean average accuracy [164]) cannot reflect the runtime performance of DL model inference in the edge. For Edge DL services, besides model accuracy, inference delay, resource consumption, and service revenue are also key indicators. Therefore, we need to identify the key performance indicators of Edge DL, quantitatively analyze the factors affecting them, and explore the trade-offs between these indicators to help improve the efficiency of Edge DL deployment.

2) Generalization of EEoI: Currently, EEoI can be applied to classification problems in DL [160], 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.

3) 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.

4) 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. 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 challenges for "*Edge Computing for DL*" to build a complete edge computing framework for DL.

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 [102]. 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) Microservice for Edge DL Services: Edge and cloud services have recently started undergoing a major shift from monolithic entities to graphs of hundreds of loosely-coupled microservices [274]. 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. At present, the microservice framework, deployed on the edge for hosting DL services, is in its infant [275], due to several critical challenges: 1) Handling DL deployment and management flexibly; 2) Achieving live migration of microservices to reduce migration times and unavailability of DL services due to user mobilities; 3) Orchestrating resources among the cloud and distributed edge infrastructures to achieve better performance, as illustrated in Section VI-B3.

3) 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 [276].

Blockchain, as a decentralized public database storing transaction records across participated devices, can avoid the risk of tampering the records [277]. 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 [278]. However, this idea about blockchained edge is still in its infancy. Existing blockchains such as Ethereum [279] 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.

4) Integration With "DL for Optimizing Edge": End devices, edge nodes, and base stations in edge computing networks are expected to run various DL models and deploy corresponding services in the future. In order to make full use of decentralized resources of edge computing, and to establish connections with existing cloud computing infrastructure, dividing the computation-intensive DL model into sub-tasks and effectively offloading these tasks between edge devices for collaboration are essential. Owing to deployment environments of Edge DL are usually highly dynamic, edge computing frameworks need excellent online resource orchestration and parameter configuration to support a large number of DL services. Heterogeneous computation resources, real-time joint optimization of communication and cache resources, and high-dimensional system parameter configuration are critical. We have introduced various theoretical methods to optimize edge computing frameworks (networks) with DL technologies in Section VIII. Nonetheless, there is currently no relevant work to deeply study the performance analysis of deploying and using these DL technologies for long-term online resource orchestration in practical edge computing networks or testbeds. We believe that "*Edge Computing for DL*" should continue to focus on how to integrate "*DL for optimizing Edge*" into the edge computing framework to realize the above vision.

D. Practical Training Principles at Edge

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 most Edge DL frameworks or

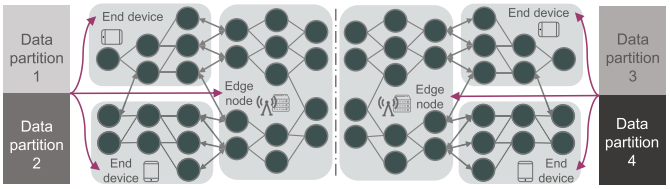


Fig. 21. DL training at the edge by both data and model parallelism.

libraries 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 lessons learned and challenges in "*DL Training 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 train them well by a single device. Therefore, large DL models are trained in distributed manners over thousands of CPU or GPU cores, in terms of data parallelism, model parallelism or their combination (Section III-C). 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, 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. 21.

2) Where Is Training Data From?: Currently, most of the DL training frameworks at the edge are aimed at supervised learning tasks, and test their performance with complete data sets. However, in practical scenarios, we cannot assume that all data in the edge computing network are labeled and with a correctness guarantee. For unsupervised learning tasks such as DRL, we certainly do not need to pay too much attention to the production of training data. For example, the training data required for DRL compose of the observed state vectors and rewards obtained by interacting with the environment. These training data can generate automatically when the system is running. But for a wider range of supervised learning tasks, how edge nodes and devices find the exact training data for model training? The application of vanilla FL is using RNN for next-word-prediction [199], in which the training data can be obtained along with users' daily inputs. Nonetheless, for extensive Edge DL services concerning video analysis, where are their training data from. If all training data is manually labeled and uploaded to the cloud data center, and then distributed to edge devices by the cloud, the original intention of FL is obviously violated. One possible solution is to enable edge devices to construct their labeled data by learning "labeled data" from each other. We believe that the production of training data and the application scenarios of DL models training at the edge should first be clarified in the future, and the necessity and feasibility of DL model training at the edge should be discussed as well.

3) Asynchronous FL at Edge: Existing FL methods [198], [199] 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 battery 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 [280]–[282]. Adequately selecting clients in each training period with resource constraints may also help. By setting a certain deadline for clients to download, update, and upload DL models, the central server can determine which clients to perform local training such that it can aggregate as many client updates as possible in each period, thus allowing the server to accelerate performance improvement in DL models [283].

4) 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 [284]. 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 [136]: 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 [136], [284] show some prospects, further research is needed to extend the TL-based training method to DL applications with different types of perceptual data.

E. Deployment and Improvement of Intelligent Edge

There have been many attempts to use DL to optimize and schedule resources in edge computing networks. In this

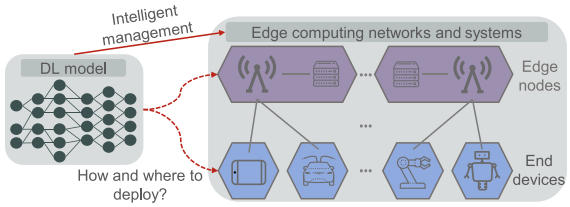


Fig. 22. Deployment issues of intelligent edge, i.e., how and where to deploy DL models for optimizing edge computing networks (systems).

regard, there are many potential areas where DL can be applied, including online content streaming [285], routing and traffic control [286], [287], etc. However, since 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 applying DL to optimize 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 optimizing Edge*" at the high level, but overlook the practical feasibility at the low level. Though DL exhibits its theoretical performance, the deployment issues of DNNs/DRL should be carefully considered (as illustrated in Fig. 22):

- 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 optimizing 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?

Besides, the abilities of the state-of-the-art DL or DRL, such as Multi-Agent Deep Reinforcement Learning [288]–[290], Graph Neural Networks (GNNs) [291], [292], 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. Conclusion

DL, as a key technique of artificial intelligence, and edge computing are expected to benefit each other. This survey has comprehensively introduced and discussed various applicable scenarios and fundamental enabling techniques for *edge intelligence* and *intelligent edge*. In summary, the key issue of extending DL from the cloud to the edge of the network is: under the multiple constraints of networking, communication, computing power, and energy consumption, how to devise and develop 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 intelligent edge will play an important supporting role to improve the performance of edge intelligence. We hope that this survey will increase discussions and research efforts on DL/Edge integration that will advance future communication applications and services.

Acknowledgment

Especially, the authors would like to thank the editors of IEEE COMST and the reviewers for their help and support in making this work possible.

References

[1] *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. Accessed: May 30, 2019. [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*. Accessed: May 30, 2019. [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, and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2013, pp. 1285–1293.

[4] W. Hu *et al.*, "Quantifying the impact of edge computing on mobile applications," in *Proc. 7th ACM SIGOPS Asia–Pac. Workshop Syst. (APSys)*, 2016, pp. 1–8.

[5] *Mobile-Edge Computing—Introductory Technical White Paper*, ETSI, Sophia Antipolis, France, 2018. [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, Q. Zhang, Y. Li, and L. Xu, "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. 55th Annu. Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[8] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019.

[9] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[11] H. Khelifi *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 *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2017, pp. 615–629.

[13] *Democratizing AI*. Accessed: Jun. 30, 2019. [Online]. Available: https://news.microsoft.com/features/democratizing-ai/

[14] Y. Yang, "Multi-tier computing networks for intelligent IoT," *Nat. Electron.*, vol. 2, no. 1, pp. 4–5, Jan. 2019.

[15] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Comput. Surveys*, vol. 51, no. 2, pp. 1–34, Apr. 2018.

[16] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "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, P. Pandey, and D. Pompili, "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," *Proc. IEEE*, vol. 107, no. 11, pp. 2204–2239, Nov. 2019.

[19] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[20] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[21] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," 2019. [Online]. Available: arXiv:1909.11875.

[22] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 3rd Quart., 2018.

[23] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Comput. Netw.*, vol. 130, no. 2018, pp. 94–120, 2018.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[25] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT," in *Proc. IEEE 29th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2015, pp. 687–694.

[26] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[27] F. Bonomi, R. A. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[28] *Multi-Access Edge Computing*. Accessed: Jun. 30, 2019. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[29] *What Is Azure Data Box Edge?*" Accessed: Jun. 30, 2019. [Online]. Available: https://docs.microsoft.com/zh-cn/azure/databox-online/data-box-edge-overview

[30] *Intel Movidius Neural Compute Stick*. Accessed: Jun. 30, 2019. [Online]. Available: https://software.intel.com/en-us/movidius-ncs

[31] *Latest Jetson Products*. Accessed: Jun. 30, 2019. [Online]. Available: https://developer.nvidia.com/buy-jetson

[32] *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*. Accessed: Jun. 30, 2019. [Online]. Available: https://e.huawei.com/en/solutions/business-needs/data-center/atlas

[33] *Snapdragon 8 Series Mobile Platforms*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms

[34] *Kirin*. Accessed: Jun. 30, 2019. [Online]. Available: http://www.hisilicon.com/en/Products/ProductList/Kirin

[35] *The World's First Full-Stack All-Scenario AI Chip*. Accessed: Jun. 30, 2019. [Online]. Available: http://www.hisilicon.com/en/Products/ProductList/Ascend

[36] *MediaTek Helio P60*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.mediatek.com/products/smartphones/mediatek-helio-p60

[37] *NVIDIA Turing GPU Architecture*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.nvidia.com/en-us/geforce/turing/

[38] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.

[39] *Intel Xeon Processor D-2100 Product Brief: Advanced Intelligence for High-Density Edge Solutions*. Accessed: May 30, 2019. [Online]. Available: https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/d-2100-brief.html

[40] *Mobile Processor: Exynos 9820*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/

[41] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeEdge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, 2018, pp. 373–377.

[42] *OpenEdge, Extend Cloud Computing, Data and Service Seamlessly to Edge Devices*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/baidu/openedge

[43] *Azure IoT Edge, Extend Cloud Intelligence and Analytics to Edge Devices*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/Azure/iotedge

[44] *EdgeX, the Open Platform for the IoT Edge*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.edgexfoundry.org/

[45] *Akraino Edge Stack*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.lfedge.org/projects/akraino/

[46] *NVIDIA EGX Edge Computing Platform: Real-Time AI at the Edge*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.nvidia.com/en-us/data-center/products/egx-edge-computing/

[47] *AWS IoT Greengrass: Bring Local Compute, Messaging, Data Caching, Sync, and ML Inference Capabilities to Edge Devices*. Accessed: Jun. 30, 2019. [Online]. Available: https://aws.amazon.com/greengrass/

[48] *Google Cloud IoT: Unlock Business Insights From Your Global Device Network With an Intelligent IoT Platform.* Accessed: Jun. 30, 2019. [Online]. Available: https://cloud.google.com/solutions/iot/

[49] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, and X. Feng, "Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2018, pp. 402–411.

[50] Y. Huang *et al.*, "Task scheduling with optimized transmission time in collaborative cloud-edge learning," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2018, pp. 1–9.

[51] E. Nurvitadhi *et al.*, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2017, pp. 5–14.

[52] S. Jiang *et al.*, "Accelerating mobile applications at the network edge with software-programmable FPGAs," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 55–62.

[53] *Qualcomm Neural Processing SDK for AI*. Accessed: Jun. 30, 2019. [Online]. Available: https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk

[54] A. Ignatov *et al.*, "AI benchmark: Running deep neural networks on Android smartphones," 2018. [Online]. Available: arXiv:1810.01109.

[55] D. Bernstein, "Containers and cloud: From LXC to docker to Kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[56] *Microsoft Cognitive Toolkit (CNTK), an Open Source Deep-Learning Toolkit*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/microsoft/CNTK

[57] S. Tokui *et al.*, "Chainer: A next-generation open source framework for deep learning," in *Proc. Workshop Mach. Learn. Syst. (LearningSys) 29th Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2015, pp. 1–6.

[58] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[59] *DeepLearning4j: Open-Source Distributed Deep Learning for the JVM, Apache Software Foundation License 2.0.* Accessed: Jun. 30, 2019. [Online]. Available: https://deeplearning4j.org

[60] *Deploy Machine Learning Models on Mobile and IoT Devices*. Accessed: Jun. 30, 2019. [Online]. Available: https://www.tensorflow.org/lite

[61] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015. [Online]. Available: arXiv:1512.01274.

[62] *PyTorch: Tensors and Dynamic Neural Networks in Python With Strong GPU Acceleration*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/pytorch/

[63] *Core ML: Integrate Machine Learning Models Into Your App*. Accessed: Jun. 30, 2019. [Online]. Available: https://developer.apple.com/documentation/coreml?language=objc

[64] *NCNN Is a High-Performance Neural Network Inference Framework Optimized for the Mobile Platform*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/Tencent/ncnn

[65] *MNN Is a Lightweight Deep Neural Network Inference Engine*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/alibaba/MNN

[66] *Multi-Platform Embedded Deep Learning Framework.* Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/PaddlePaddle/paddle-mobile

[67] *MACE Is a Deep Learning Inference Framework Optimized for Mobile Heterogeneous Computing Platforms*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/XiaoMi/mace

[68] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things," 2019. [Online]. Available: arXiv:1911.03314.

[69] Z. Tao *et al.*, "A survey of virtual machine management in edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.

[70] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[71] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.

[72] A. Wang, Z. Zha, Y. Guo, and S. Chen, "Software-defined networking enhanced edge computing: A network-centric survey," *Proc. IEEE*, vol. 107, no. 8, pp. 1500–1519, Aug. 2019.

[73] Y.-D. Lin, C.-C. Wang, C.-Y. Huang, and Y.-C. Lai, "Hierarchical CORD for NFV datacenters: Resource allocation with cost-latency tradeoff," *IEEE Netw.*, vol. 32, no. 5, pp. 124–130, Sep./Oct. 2018.

[74] L. Li, K. Ota, and M. Dong, "DeepNFV: A lightweight framework for intelligent edge network functions virtualization," *IEEE Netw.*, vol. 33, no. 1, pp. 136–141, Jan. 2019.

[75] *Mobile Edge Computing A Key Technology Towards 5G*. ETSI, Sophia Antipolis, France, 2015. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[76] H.-T. Chien, Y.-D. Lin, C.-L. Lai, and C.-T. Wang, "End-to-end slicing as a service with computing and communication resource allocation for multi-tenant 5G systems," *IEEE Wireless Commun.*, vol. 26, no. 5, pp. 104–112, Oct. 2019.

[77] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[78] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[79] S. S. Haykin and K. Elektroingenieur, *Neural Networks and Learning Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, 2009.

[80] R. Collobert and S. Bengio, "Links between perceptrons, MLPs and SVMs," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, p. 23.

[81] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.

[82] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 818–833.

[83] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. 27th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2014, pp. 2672–2680.

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

[85] 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.

[86] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: arXiv:1503.02531.

[87] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf. (IntelliSys)*, 2016, pp. 426–440.

[88] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[89] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double *Q*-learning," in *Proc. 13th AAAI Conf. Artif. Intell. (AAAI)*, 2016, pp. 2094–2100.

[90] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1995–2003.

[91] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.

[92] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1928–1937.

[93] J. Schulman *et al.*, "Proximal policy optimization algorithms," 2017. [Online]. Available: arXiv:1707.06347.

[94] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 1999, pp. 1057–1063.

[95] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 25th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2012, pp. 1223–1231.

[96] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1772–1777, 2014.

[97] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide, "Pipelined back-propagation for context-dependent deep neural networks," in *Proc. 13th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2012, pp. 26–29.

[98] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2014, pp. 1058–1062.

[99] A. Coates, B. Huval, T. Wang, D. J. Wu, B. Catanzaro, and A. Y. Ng, "Deep learning with cots HPC systems," in *Proc. 30th Int. Conf. Mach. Learn. (PMLR)*, 2013, pp. 1337–1345.

[100] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "SparkNet: Training deep networks in spark," 2015. [Online]. Available: arXiv:1511.06051.

[101] *Theano Is a Python Library That Allows You to Define, Optimize, and Evaluate Mathematical Expressions Involving Multi-Dimensional Arrays Efficiently*. Accessed: Jun. 30, 2019. [Online]. Available: https://github.com/Theano/Theano

[102] Y. Guo, B. Zou, J. Ren, Q. Liu, D. Zhang, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Netw.*, vol. 32, no. 6, pp. 137–143, Nov. 2018.

[103] C. Liu *et al.*, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 249–261, Mar./Apr. 2018.

[104] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "DeepCham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *Proc. 1st ACM/IEEE Symp. Edge Comput. (SEC)*, 2016, pp. 64–76.

[105] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 115–127.

[106] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. 2nd ACM/IEEE Symp. Edge Comput. (SEC)*, 2017, pp. 1–13.

[107] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. R. Faughnan, "Smart surveillance as an edge network service: From HARR-cascade, SVM to a lightweight CNN," in *Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, 2018, pp. 256–265.

[108] P. Liu, B. Qi, and S. Banerjee, "EdgeEye—An edge service framework for real-time intelligent video analytics," in *Proc. 1st Int. Workshop Edge Syst. Anal. Netw. (EdgeSys)*, 2018, pp. 1–6.

[109] C.-C. Hung *et al.*, "VideoEdge: Processing camera streams using hierarchical clusters," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, 2018, pp. 115–131.

[110] Y. He, N. Zhao, and H. Yin, "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.

[111] Q. Qi and Z. Ma, *Vehicular Edge Computing via Deep Reinforcement Learning*. 2018. [Online]. Available: arXiv:1901.04290.

[112] 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. 10190–10203, Nov. 2018.

[113] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4665–4673, Oct. 2018.

[114] L. Hu, Y. Miao, G. Wu, M. M. Hassan, and I. Humar, "iRobot-factory: An intelligent robot factory based on cognitive manufacturing and edge computing," *Future Gener. Comput. Syst.*, vol. 90, pp. 569–577, Jan. 2019.

[115] J. A. C. Soto, M. Jentsch, D. Preuveneers, and E. I. Zudor, "CEML: Mixing and moving complex event processing and machine learning to the edge of the network for IoT applications," in *Proc. 6th Int. Conf. Internet Things (IoT)*, 2016, pp. 103–110.

[116] G. Plastiras, M. Terzi, C. Kyrkou, and T. Theocharidcs, "Edge intelligence: Challenges and opportunities of near-sensor machine learning applications," in *Proc. IEEE 29th Int. Conf. Appl. Specific Syst. Archit. Processors (ASAP)*, 2018, pp. 1–7.

[117] Y. Hao *et al.*, "Smart-edge-CoCaCo: AI-enabled smart edge with joint computation, caching, and communication in heterogeneous IoT," 2019. [Online]. Available: arXiv:1901.02126.

[118] S. Liu, P. Si, M. Xu, Y. He, and Y. Zhang, "Edge big data-enabled low-cost indoor localization based on Bayesian analysis of RSS," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2017, pp. 1–6.

[119] A. Dhakal and K. K. Ramakrishnan, "Machine learning at the network edge for automated home intrusion monitoring," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, 2017, pp. 1–6.

[120] N. Tian *et al.*, "A fog robotic system for dynamic visual servoing," 2018. [Online]. Available: arXiv:1809.06716.

[121] L. Lu, L. Xu, B. Xu, G. Li, and H. Cai, "Fog computing approach for music cognition system based on machine learning algorithm," *IEEE Trans. Comput. Soc. Syst.*, vol. 5, no. 4, pp. 1142–1151, Dec. 2018.

[122] B. Tang *et al.*, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Trans. Ind. Informat.*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017.

[123] Y.-C. Chang and Y.-H. Lai, "Campus edge computing network based on IoT street lighting nodes," *IEEE Syst. J.*, early access, doi: 10.1109/JSYST.2018.2873430.

[124] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. 27th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2014, pp. 1269–1277.

[125] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2285–2294.

[126] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[127] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[128] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017. [Online]. Available: arXiv:1710.09282.

[129] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2015, pp. 1135–1143.

[130] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–12.

[131] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2015, pp. 3123–3131.

[132] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.

[133] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks," in *Proc. Int. Conf. Embedded Wireless Syst. Netw. (EWSN)*, 2017, pp. 168–173.

[134] F. N. Iandola *et al.*, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," 2016. [Online]. Available: arXiv:1602.07360.

[135] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: arXiv:1704.04861.

[136] R. Sharma, S. Biookaghazadeh, and M. Zhao, "Are existing knowledge transfer techniques effective for deep learning on edge devices?" in *Proc. 27th Int. Symp. High Perform. Parallel Distrib. Comput. (HPDC)*, 2018, pp. 15–16.

[137] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao, "FFS-VA: A fast filtering system for large-scale video analytics," in *Proc. 47th Int. Conf. Parallel Process. (ICPP)*, 2018, pp. 1–10.

[138] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2018, pp. 253–266.

[139] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. R. Faughnan, "Real-time human detection as an edge service enabled by a lightweight CNN," in *Proc. IEEE Int. Conf. Edge Comput. (IEEE EDGE)*, 2018, pp. 125–129.

[140] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2019, pp. 1–16.

[141] Fox. *Homer Simpson*. Accessed: Jun. 30, 2019. [Online]. Available: https://simpsons.fandom.com/wiki/File:Homer_Simpson.svg

[142] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 6848–6856.

[143] L. Du *et al.*, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.

[144] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1586–1597, Aug. 2017.

[145] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2017, pp. 75–84.

[146] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.

[147] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proc. 14th ACM Conf. Embedded Netw. Sensor Syst. CD-ROM (SenSys)*, 2016, pp. 176–189.

[148] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," in *Proc. 19th ACM SIGPLAN/SIGBED Int. Conf. Lang. Compilers Tools Embedded Syst. (LCTES)*, 2018, pp. 31–43.

[149] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices," in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2018, pp. 389–400.

[150] L. Lai and N. Suda, "Enabling deep learning at the IoT edge," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–6.

[151] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. F. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. 15th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2017, pp. 1–14.

[152] H. Shen, M. Philipose, S. Agarwal, and A. Wolman, "MCDNN: An execution framework for deep neural networks on resource-constrained devices," in *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2016, pp. 123–136.

[153] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.

[154] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2016, pp. 1–12.

[155] J. Zhang, S. Chen, B. Liu, Y. Ma, and X. Chen, "A locally distributed mobile computing framework for DNN based Android applications," in *Proc. 10th Asia–Pac. Symp. Internetware (Internetware)*, 2018, pp. 1–6.

[156] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[157] Z. Zhao, Z. Jiang, N. Ling, X. Shuai, and G. Xing, "ECRT: An edge computing system for real-time image-based object tracking," in *Proc. 16th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2018, pp. 394–395.

[158] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[159] S. S. Ogden and T. Guo, "MODI: Mobile deep inference made efficient by edge computing," in *Proc. {USENIX} Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–7.

[160] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464–2469.

[161] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 328–339.

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

[163] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 276–286.

[164] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based deep learning framework for continuous vision applications," in *Proc. 15th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2017, pp. 82–95.

[165] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "DeepCache: Principled cache for mobile deep vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 129–144.

[166] P. Guo, B. Hu, R. Li, and W. Hu, "FoggyCache: Cross-device approximate computation reuse," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 19–34.

[167] A. H. Jiang *et al.*, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *Proc. USENIX Conf. Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 29–41.

[168] Y. Chen, S. Biookaghazadeh, and M. Zhao, "Exploring the capabilities of mobile devices supporting deep learning," in *Proc. 27th Int. Symp. High Perform. Parallel Distrib. Comput. (HPDC)*, 2018, pp. 17–18.

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

[170] R. Venkatesan and B. Li, "Diving deeper into MENTEE networks," 2016. [Online]. Available: arXiv:1604.08220.

[171] S. Biookaghazadeh, F. Ren, and M. Zhao, "Are FPGAs suitable for edge computing?" 2018. [Online]. Available: arXiv:1804.06404.

[172] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 1421–1429.

[173] W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, and V. C. M. Leung, "MASM: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4216–4224, Jul. 2019.

[174] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "DeepWear: Adaptive local offloading for on-wearable deep learning," *IEEE Trans. Mobile Comput.*, vol. 19, no. 2, pp. 314–330, Feb. 2020.

[175] 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. ACM Symp. Cloud Comput. (SoCC)*, 2018, pp. 401–411.

[176] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, 2017, pp. 1–2.

[177] J. Mao, X. Chen, K. W. Nixon, C. D. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 1396–1401.

[178] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2010, pp. 49–62.

[179] X. Xu *et al.*, "Scaling for edge inference of deep neural networks," *Nat. Electron.*, vol. 1, no. 4, pp. 216–222, Apr. 2018.

[180] M. Polese *et al.*, "Machine learning at the edge: A data-driven architecture with applications to 5G cellular networks," 2018. [Online]. Available: arXiv:1808.07647.

[181] L. Lai *et al.*, "Rethinking machine learning development and deployment for edge devices," 2018. [Online]. Available: arXiv:1806.07846.

[182] P. Meloni *et al.*, "ALOHA: An architectural-aware framework for deep learning at the edge," in *Proc. Workshop Intell. Embedded Syst. Archit. Appl. (INTESA)*, 2018, pp. 19–26.

[183] X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, and W. Shi, "OpenEI: An open framework for edge intelligence," 2019. [Online]. Available: arXiv:1906.01864.

[184] J. R. Zhao, T. Tiplea, R. Mortier, J. Crowcroft, and L. Wang, "Data analytics service composition and deployment on IoT devices," in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2018, pp. 502–504.

[185] N. Talagala *et al.*, "ECO: Harmonizing edge and cloud with ML/DL orchestration," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–7.

[186] X. Zhang, Y. Wang, and W. Shi, "pCAMP: Performance comparison of machine learning packages on the edges," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–6.

[187] C. A. Ramiro, C. Fiandrino, A. B. Pizarro, P. J. Mateo, N. Ludant, and J. Widmer, "openLEON: An end-to-end emulator from the edge data center to the mobile users carlos," in *Proc. 12th Int. Workshop Wireless Netw. Testbeds Exp. Eval. Characterization (WiNTECH)*, 2018, pp. 19–27.

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

[189] G. Kamath, P. Agnihotri, M. Valero, K. Sarker, and W.-Z. Song, "Pushing analytics to the edge," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–6.

[190] L. Valerio, A. Passarella, and M. Conti, "A communication efficient distributed learning framework for smart environments," *Pervasive Mobile Comput.*, vol. 41, pp. 46–68, Oct. 2017.

[191] Y. Lin *et al.*, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017. [Online]. Available: arXiv:1712.01887.

[192] Z. Tao and C. William, "eSGD: Communication efficient distributed deep learning on the edge," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–6.

[193] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2015, pp. 1488–1492.

[194] E. Jeong *et al.*, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-IID private data," 2018. [Online]. Available: arXiv:1811.11479.

[195] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2015, pp. 1322–1333.

[196] 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, doi: 10.1109/TBDATA.2018.2829886.

[197] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. 3rd Theory Cryptography Conf. (TCC)*, 2006, pp. 265–284.

[198] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2017, pp. 1273–1282.

[199] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019. [Online]. Available: arXiv:1902.01046.

[200] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Trans. Commun.*, early access., doi: 10.1109/TCOMM.2019.2956472.

[201] C. Xie, S. Koyejo, and I. Gupta, "Practical distributed learning: Secure machine learning with communication-efficient local updates," 2019. [Online]. Available: arXiv:1903.06996.

[202] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," 2019. [Online]. Available: arXiv: 1909.02362.

[203] J. Konečný *et al.*, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.

[204] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," 2019. [Online]. Available: arXiv:1909.13014.

[205] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," 2018. [Online]. Available: arXiv:1812.07210.

[206] B. S. Kashin, "Diameters of some finite-dimensional sets and classes of smooth functions," *Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya*, vol. 41, no. 2, pp. 334–351, 1977.

[207] Y. Jiang, S. Wang, B. J. Ko, W.-H. Lee, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," 2019. [Online]. Available: arXiv:1909.12326.

[208] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 63–71.

[209] S. Wang *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.

[210] T. Tuor, S. Wang, T. Salonidis, B. Ko, and K. K. Leung, "Demo abstract: Distributed machine learning at resource-limited edge nodes," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2018, pp. 1–2.

[211] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," 2019. [Online]. Available: arXiv:1911.06949.

[212] M. Duan, "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," 2019. [Online]. Available: arXiv:1907.01132.

[213] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.

[214] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," 2018. [Online]. Available: arXiv:1812.11750.

[215] B. Nazer and M. Gastpar, "Computation over multiple-access channels," *IEEE Trans. Inf. Theory*, vol. 53, no. 10, pp. 3498–3516, Oct. 2007.

[216] L. Chen, N. Zhao, Y. Chen, F. R. Yu, and G. Wei, "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.

[217] G. Zhu, Y. Wang, and K. Huang, "Broadband analog aggregation for low-latency federated edge learning (extended version)," 2018. [Online]. Available: arXiv:1812.11494.

[218] Z. Xu, Z. Yang, J. Xiong, J. Yang, and X. Chen, "ELFISH: Resource-aware federated learning on heterogeneous edge devices," 2019. [Online]. Available: arXiv:1912.01684.

[219] C. Dinh *et al.*, "Federated learning over wireless networks: Convergence analysis and resource allocation," 2019. [Online]. Available: arXiv:1910.13067.

[220] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," 2019. [Online]. Available: arXiv:1909.07972.

[221] T. Li, M. Sanjabi, and V. Smith, "Fair resource allocation in federated learning," 2019. [Online]. Available: arXiv:1905.10497.

[222] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2017, pp. 1175–1191.

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

[224] J. E. Stiglitz, "Self-selection and Pareto efficient taxation," *J. Public Econ.*, vol. 17, no. 2, pp. 213–240, 1982.

[225] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logist. Quart.*, vol. 2, nos. 1–2, pp. 83–97, 1955.

[226] H. SHI, R. V. Prasad, E. Onur, and I. G. M. M. Niemegeers, "Fairness in wireless networks: Issues, measures and challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 5–24, 1st Quart., 2014.

[227] M. Hofmann and L. Beaumont, "Chapter 3—Caching techniques for Web content," in *Content Networking*. San Francisco, CA, USA: Morgan Kaufmann, 2005, pp. 53–79.

[228] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "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.

[229] E. Zeydan *et al.*, "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sep. 2016.

[230] J. Song, M. Sheng, T. Q. S. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Trans. Commun.*, vol. 65, no. 10, pp. 4309–4324, Oct. 2017.

[231] X. Li, X. Wang, P.-J. Wan, Z. Han, and V. C. M. Leung, "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.

[232] S. Rathore, J. H. Ryu, P. K. Sharma, and J. H. Park, "DeepCachNet: A proactive caching framework based on deep learning in cellular networks," *IEEE Netw.*, vol. 33, no. 3, pp. 130–138, May 2019.

[233] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "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.

[234] J. Yang, J. Zhang, C. Ma, H. Wang, J. Zhang, and G. Zheng, "Deep learning-based edge caching for multi-cluster heterogeneous networks," *Neural Comput. Appl.*, pp. 1–12, Feb. 2019. [Online]. Available: https://link.springer.com/article/10.1007/s00521-019-04040-z

[235] A. Ndikumana, N. H. Tran, and C. S. Hong, "Deep learning based caching for self-driving car in multi-access edge computing," 2018. [Online]. Available: arXiv:1810.01548.

[236] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[237] Y. Tang, K. Guo, J. Ma, Y. Shen, and T. Chi, "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.

[238] D. Adelman and A. J. Mersereau, "Relaxations of weakly coupled stochastic dynamic programs," *Oper. Res.*, vol. 56, no. 3, pp. 712–727, 2008.

[239] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov./Dec. 2018.

[240] K. Guo, C. Yang, and T. Liu, "Caching in base station with recommendation via Q-learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2017, pp. 1–6.

[241] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst. (CISS)*, 2018, pp. 1–6.

[242] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," 2015. [Online]. Available: arXiv:1512.07679.

[243] 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, 3rd Quart., 2017.

[244] 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.

[245] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.

[246] 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.

[247] 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.

[248] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[249] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *Proc. IEEE 28th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2017, pp. 1–6.

[250] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar, "Deep reinforcement learning based resource allocation in low latency edge computing networks," in *Proc. 15th Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2018, pp. 1–5.

[251] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[252] 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 *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.

[253] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2018, pp. 1–6.

[254] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[255] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," 2018. [Online]. Available: arXiv:1812.07394.

[256] T. Chen and G. B. Giannakis, "Harnessing bandit online learning to low-latency fog computing," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2018, pp. 6418–6422.

[257] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep Q-learning model for energy-efficient edge scheduling," *IEEE Trans. Services Comput.*, vol. 12, no. 05, pp. 739–749, Jan. 2019.

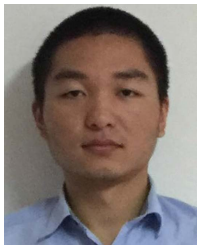
[258] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," 2018. [Online]. Available: arXiv:1808.01977.

[259] S. Memon and M. Maheswaran, "Using machine learning for handover optimization in vehicular fog computing," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput. (SAC)*, 2019, pp. 182–190.

[260] Y. Sun, M. Peng, and S. Mao, "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.

[261] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, and M. Guizani, "Security in mobile edge caching with reinforcement learning," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 116–122, Jun. 2018.
[262] 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.
[263] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Secure computation offloading in blockchain based IoT networks with deep reinforcement learning," 2018. [Online]. Available: arXiv:1908.07466.
[264] C.-Y. Li *et al.*, "Mobile edge computing platform deployment in 4G LTE networks: A middlebox approach," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–6.
[265] 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, 4th Quart., 2018.
[266] R. Li *et al.*, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 175–183, Oct. 2017.
[267] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "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.
[268] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for hybrid 5G services in mobile edge computing systems: Learn from a digital twin," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4692–4707, Oct. 2019.
[269] Y. Chen, Y. Zhang, S. Maharjan, M. Alam, and T. Wu, "Deep learning for secure mobile edge computing in cyber-physical transportation systems," *IEEE Netw.*, vol. 33, no. 4, pp. 36–41, Jul./Aug. 2019.
[270] M. Min *et al.*, "Learning-based privacy-aware offloading for healthcare IoT with energy harvesting," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4307–4316, Jun. 2019.
[271] T. E. Bogale, X. Wang, and L. B. Le, "Machine intelligence techniques for next-generation context-aware wireless networks," 2018. [Online]. Available: arXiv:1801.04223.
[272] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[273] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "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.
[274] Y. Gan *et al.*, "An open-source benchmark suite for microservices and their hardware–software implications for cloud and edge systems," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2019, pp. 3–18.
[275] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using Docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.
[276] J. Xu, S. Wang, B. Bhargava, and F. Yang, "A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3538–3547, Jun. 2019.
[277] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, 2017, pp. 557–564.
[278] J.-Y. Kim and S.-M. Moon, "Blockchain-based edge computing for deep neural network applications," in *Proc. Workshop Intell. Embedded Syst. Archit. Appl. (INTESA)*, 2018, pp. 53–55.
[279] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: http://gavwood.com/Paper.pdf
[280] S. Zheng *et al.*, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 4120–4129.
[281] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019. [Online]. Available: arXiv:1903.03934.
[282] W. Wu, L. He, W. Lin, RuiMao, and S. Jarvis, "SAFA: A semi-asynchronous protocol for fast federated learning with low overhead," 2019. [Online]. Available: arXiv:1910.01355.
[283] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," 2018. [Online]. Available: arXiv:1804.08333.
[284] T. Xing, S. S. Sandha, B. Balaji, S. Chakraborty, and M. B. Srivastava, "Enabling edge devices that learn from each other: Cross modal training for activity recognition," in *Proc. 1st Int. Workshop Edge Syst. Anal. Netw. (EdgeSys)*, 2018, pp. 37–42.
[285] J. Yoon, P. Liu, and S. Banerjee, "Low-cost video transcoding at the wireless edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, 2016, pp. 129–141.
[286] 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.
[287] Z. M. Fadlullah *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, 4th Quart., 2017.
[288] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. 29th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2016, pp. 2137–2145.
[289] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 2681–2690.
[290] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor–critic for mixed cooperative-competitive environments," in *Proc. 30th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 6379–6390.
[291] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," 2018. [Online]. Available: arXiv:1812.08434.
[292] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," 2018. [Online]. Available: arXiv:1812.04202.

Xiaofei Wang (Senior Member, IEEE) received the master's and doctoral degrees from Seoul National University in 2006 and 2013, respectively. He is currently a Professor with the Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, China. He was a Postdoctoral 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 100 technical papers in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE WIRELESS COMMUNICATIONS, the IEEE Communications, the IEEE TRANSACTIONS ON MULTIMEDIA, IEEE INFOCOM, and IEEE SECON. He was a recipient of the National Thousand Talents Plan (Youth) of China, 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, the Peiyang Scholar from Tianjin University, and the Fred W. Ellersick Prize from the IEEE Communication Society in 2017.



Yiwen Han (Student Member, IEEE) received the B.S. degree in communication engineering from Nanchang University, China, in 2015, and the M.S. degree in communication engineering from Tianjin University, China, in 2018, where he is currently pursuing the Ph.D. degree in computer science. His current research interests include edge computing, reinforcement learning, and deep learning. He received the Outstanding B.S. Graduates in 2015 and M.S. National Scholarship of China in 2016.

Victor C. M. Leung (Fellow, IEEE) is a Distinguished Professor of computer science and software engineering with Shenzhen University. He is also an Emeritus Professor of electrical and computer engineering and the Director of the Laboratory for Wireless Networks and Mobile Systems with the University of British Columbia (UBC). His research is in the broad areas of wireless networks and mobile systems. He has coauthored more than 1300 journal/conference papers and book chapters. He is serving on the Editorial Boards for the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE ACCESS, IEEE NETWORK, and several other journals. He received the IEEE Vancouver Section Centennial Award, the 2011 UBC Killam Research Prize, the 2017 Canadian Award for Telecommunications Research, and the 2018 IEEE TCGCC Distinguished Technical Achievement Recognition Award. He coauthored papers that won the 2017 IEEE ComSoc Fred W. Ellersick Prize, the 2017 IEEE Systems Journal Best Paper Award, the 2018 IEEE CSIM Best Journal Paper Award, and the 2019 IEEE TCGCC Best Journal Paper Award. He is a Fellow of the Royal Society of Canada, Canadian Academy of Engineering, and Engineering Institute of Canada. He is named in the current Clarivate Analytics list of "Highly Cited Researchers."

Dusit Niyato (Fellow, IEEE) received the B.Eng. degree from the King Mongkuts Institute of Technology Ladkrabang, Thailand, in 1999, and the Ph.D. degree in electrical and computer engineering from the University of Manitoba, Canada, in 2008. He is currently a Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests are in the area of Internet of Things and network resource pricing.

Xueqiang Yan is currently a Technology Expert with Wireless Technology Lab, Huawei Technologies. He was a Member of Technical Staff with Bell Labs from 2000 to 2004. From 2004 to 2016, he was the Director of Strategy Department, Alcatel-Lucent Shanghai Bell. His current research interests include wireless networking, Internet of Things, edge AI, future mobile network architecture, network convergence, and evolution.

Xu Chen (Member, IEEE) received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012. He is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He worked as a Postdoctoral Research Associate with Arizona State University, Tempe, USA, from 2012 to 2014, and a Humboldt Scholar Fellow with the Institute of Computer Science, University of Göettingen, Germany, from 2014 to 2016. He received the Prestigious Humboldt Research Fellowship awarded by Alexander von Humboldt Foundation of Germany, the 2014 Hong Kong Young Scientist Runner-Up Award, the 2016 Thousand Talents Plan Award for Young Professionals of China, the 2017 IEEE Communication Society Asia–Pacific Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the Honorable Mention Award of 2010 IEEE International Conference on Intelligence and Security Informatics, the Best Paper Runner-Up Award of 2014 IEEE International Conference on Computer Communications, and the Best Paper Award of 2017 IEEE Intranational Conference on Communications. He is currently an Associate Editor of the IEEE INTERNET OF THINGS JOURNAL and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and an Area Editor of the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY.