Machine and Deep Learning for Resource Allocation in Multi-Access Edge Computing: A Survey

Hamza Djigal, Jia Xu, *Senior Member, IEEE,* Linfeng Liu, *Member, IEEE,* and Yan Zhang, *Fellow, IEEE*

***Abstract*—With the rapid development of Internet-of-Things (IoT) devices and mobile communication technologies, Multi-access Edge Computing (MEC) has emerged as a promising paradigm to extend cloud computing and storage capabilities to the edge of cellular networks, near to IoT devices. MEC enables IoT devices with limited battery capacity and computation/storage capabilities to execute their computation-intensive and latency-sensitive applications at the edge of the networks. However, to efficiently execute these applications in MEC systems, each task must be properly offloaded and scheduled onto the MEC servers. Additionally, the MEC servers may intelligently balance and share their computing resources to satisfy the application QoS and QoE. Therefore, effective resource allocation (RA) mechanisms in MEC are vital for ensuring its foreseen advantages. Recently, Machine Learning (ML) and Deep Learning (DL) have emerged as key methods for many challenging aspects of MEC. Particularly, ML and DL play a crucial role in addressing the challenges of RA in MEC. This paper presents a comprehensive survey of ML/DL-based RA mechanisms in MEC. We first present tutorials that demonstrate the advantages of applying ML and DL in MEC. Then, we present enabling technologies for quickly running ML/DL training and inference in MEC. Afterward, we provide an in-depth survey of recent works that used ML/DL methods for RA in MEC from three aspects: (1) ML/DL-based methods for task offloading; (2) ML/DL-based methods for task scheduling; and (3) ML/DL-based methods for joint resource allocation. Finally, we discuss key challenges and future research directions of applying ML/DL for resource allocation in MEC networks.**

***Index Terms*—Multi-access edge computing, resource allocation, task offloading, task scheduling, machine learning, deep learning, IoT applications.**

I. Introduction

With the recent progress in information and mobile communication technologies, such as fifth-generation mobile networks (5G), the quality of service (QoS), and the conjectures towards fantastic Quality of Experience (QoE) are widely increasing. Tens of billions of resource-limited wireless smart devices, such as mobile user equipment (UEs), sensors, and wearable devices can connect to the Internet through 5G networks [1]. The number of IoT devices is expected to increase to 25.2 billion by 2025, and it is estimated that 3.1 billion of the connected IoT devices will use cellular technology, establishing business opportunities for enterprises and mobile operators [2]. Also, the 5G networks bring a range of benefits to the end-users and IoT devices, including 5G's ultra-reliability (99.999%), very low latency (below 5ms), high bandwidth (10 Gbps), and the ability to support 1000 times higher data volumes [3]. The 5G networks will highly improve user's QoS and QoE, and facilitate the demand of smart applications (e.g., smart cars, smart energy grids, smart houses, etc.) [2].

Due to their limited resources (computation, storage, etc.) capacities and finite battery capacity, the IoT devices are not suitable for supporting latency or delay-intensive applications or IoT applications providing 5G services such as online gaming and video services [4]. To overcome this challenge, the concept of mobile cloud computing (MCC) is introduced to enable the IoT devices to offload their computation-sensitive applications to powerful centralized remote clouds which are accessible via the Internet or a core network (CN) of a mobile operator [5]. However, MCC also incurs high latency because data is offloaded to remote cloud servers that are located far away from the IoT devices. To address this issue, the emerged Mobile Edge Computing paradigm has been introduced by ETSI ISG [6] to move the cloud computation and storage capabilities closer to the end-users. In September 2017, ETSI ISG officially renamed it Multi-access Edge Computing (MEC) to better reflect that the edge is not only based on mobile networks but can also refer to various networks such as WiFi and fixed access technologies [7], [8], [3]. MEC comprises edge servers located at the edge of the network and implemented either at the following access points: base stations (BSs), radio access networks (RANs) for LTE/5G, hot spots, data centers (DCs), routers, switches, and wiFi access points (WAP).

The basic principle of MEC is to bring the cloud computation and storage capabilities closer to the edge of the networks. Hence, MEC can provide ultra-low latency and reliability compared to the MCC. In addition, due to its ultra-low latency and high bandwidth characteristics, MEC brings new business opportunities to the major stakeholders including mobile operators, application developers, telecom equipment and software vendors, IT platform and technology vendors [9]. For instance, a mobile operator can maximize its revenue by offering open access of the MEC platforms through Application Programming Interfaces (APIs) to service providers and suggest usage-

This work was supported in part by the National Natural Science Foundation of China under grants 61872193, 61872191 and 62072254, and in part by the National Foreign Expert Program of China under grant QN2022014001. (Corresponding author: Jia Xu)

Hamza Djigal, Jia Xu and Linfeng liu are with the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210023, China. (e-mail: djigalshamza@gmail.com, {xujia, liulf}@njupt.edu.cn).

Yan Zhang is with Department of Informatics, University of Oslo, 0316 Oslo, Norway (e-mail:yanzhang@ieee.org).

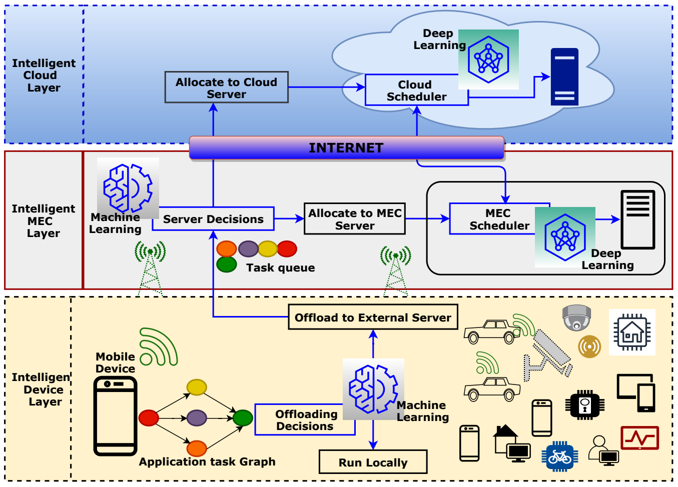


Fig. 1: The framework of ML/DL-enabled intelligent resource allocation in MEC

based charging for using resources (e.g., computation, storage, and bandwidth) [3]. Another advantage of MEC is that it enables IoT devices with limited battery capacity and limited computation/storage capabilities to execute their computation-intensive and latency-intensive applications at the edge of the networks. However, to efficiently execute these applications in MEC systems, each task must be properly offloaded and scheduled onto the edge/cloud servers. Moreover, multiple edge servers can collaboratively offload their computation-intensive tasks to each other through a backhaul network to offer better services for the end-users by balancing and sharing their computation resources [4]. For example, a nearby edge server can decide to offload the computation task of a connected mobile device to another edge server if it does not have the required computing resources to process the task within its deadline. Hence, efficient resource allocation mechanisms in MEC are vital for its foreseen advantages. The main objective of this paper is to provide an in-depth survey of recent works that used ML and DL methods to address the resource allocation problem in MEC, which we divide into three sub-problems: (1) task offloading problem, (2) task scheduling problem, and (3) joint resource allocation problem.

Traditionally, the resource allocation problem is solved using optimization methods such as heuristic or meta-heuristic, which are not globally optimal [10]. Also, these traditional solutions are not suitable for delay-sensitive and data-intensive applications since they are computationally expensive. Moreover, the time complexity of these traditional approaches proportionally increases with the increase in the number of tasks and network size. To overcome the drawbacks of traditional methods, some recent studies have begun to investigate ML and DL methods to solve the resource allocation problem in MEC. For instance, Huang et al. [11] proposed a DL-based task offloading and bandwidth allocation mechanism in MEC, which minimizes the overall offloading cost. Moreover, ML and DL techniques could be vital for optimizing the resource allocation process while meeting the QoS and QoE requirements of the application because they can intelligently predict unknown QoS or QoE requirements by learning from historical data.

Fig. 1 illustrates a ML/DL-enabled intelligent framework for resource allocation in MEC. In the intelligent device layer, a machine learning algorithm is embedded in each intelligent device (e.g., mobile device), which can make offloading decisions. In the intelligent MEC layer, an ML algorithm selects the suitable computation resources (server decisions) which can execute the offloaded tasks. If the offloaded tasks are allocated to an MEC server, a DL-based scheduler embedded in the MEC server makes the scheduling decision (i.e., task prioritization and assignment). Generally, the DL training process is done in the intelligent cloud layer because it requires powerful computing resources [12]. Additionally, the ML/DL-based offloading and scheduling decisions depend not only on the data size and computation resource capabilities but also on the ML/DL model to be executed.

A. *Existing Surveys on Resource Allocation*

In this section, we first present previous surveys on task offloading in MEC. Then, we discuss related surveys on task scheduling in various distributed systems, such as grid computing, cloud computing, fog computing, and MEC. Additionally, we classify the related surveys on traditional techniques for resource allocation (Table I). Finally, we compare the existing surveys with our survey in terms of different aspects as shown in Table II.

1) Existing Surveys on Task Offloading: In [13], the authors present a comprehensive survey on data offloading techniques in cellular networks. They classify the existing techniques into two main categories, namely delayed offloading and non-delayed offloading according to the delay that the data may tolerate. In delayed offloading, packet reception may be purposely delayed up to a certain time to achieve more beneficial delivery conditions. In non-delayed offloading, no additional delay is added to packet reception except the delay caused by the packet processing. Since there is no extra delay in no-delayed offloading, the QoS requirements are preserved. The authors of [5] address the computation offloading decision problem in MEC. They classify the offloading decision approaches into full offloading and partial offloading. The main idea of the full offloading approach is to offload the whole computation task to the MEC servers. In partial offloading, a part of the computation task is processed locally by the device while the rest is processed by the MEC servers. In [20], the authors present a survey on opportunistic offloading and classify them into two main categories: traffic offloading and computation offloading. In traffic offloading, the mobile devices download contents via the cellular network and send them to other nodes via opportunistic communication. In computation offloading, the computationally intensive tasks of the mobile device with limited computing resources are offloaded through opportunistic communication to other mobile devices nearby which have enough computing capacity to process the tasks. In [24], the authors present a survey and taxonomy on task offloading in edge-cloud environments. They analyze the task offloading solutions from five aspects: task types,

TABLE I: Summary of Existing Surveys on Conventional Techniques for Resource Allocation in Cloud/Edge/Fog

Ref.	**Traditional Techniques**			**Resource Allocation (RA) Aspects**			**Focus of Discussion**
	Heuristic	Meta-Heuristic	Game Theory	Offloading	Scheduling	Joint RA	
2015, [13]	✓			✓			Data offloading techniques in cellular networks.
2016, [14]					✓		Taxonomy for classifying 109 scheduling problems and solutions in DS.
2017, [15]					✓		Taxonomy of task allocation with temporal and ordering constraints
2017, [10]	✓	✓			✓		Taxonomy and survey on scheduling algorithms in IaaS cloud
2017, [5]	✓			✓			Survey on architecture and computation offloading in MEC
2017, [16]				✓	✓		Survey of economic and pricing models for resource management in cloud.
2018, [17]	✓				✓		Survey of cluster frameworks and scheduling strategies in data center networks
2018, [18]	✓	✓			✓		Survey of task scheduling methods in desktop grid computing systems
2018, [19]					✓		Taxonomy of the scheduling problem in cloud.
2018, [20]				✓			Survey of opportunistic offloading: traffic and computation offloading.
2019, [21]	✓	✓			✓		Systematic review and taxonomy of scheduling techniques in cloud.
2019, [22]	✓				✓		Survey of meta-heuristic scheduling techniques in cloud.
2019, [23]	✓	✓			✓		Taxonomy and survey on scheduling techniques in fog-cloud
2020, [24]	✓	✓		✓	✓		Survey and Taxonomy on Task Offloading for Edge-Cloud Computing
2020, [25]				✓			Survey of computation offloading modeling in edge computing
2020, [26]	✓				✓		Multiple workflows scheduling problems in multi-tenant distributed systems.
2020, [27]		✓	✓	✓			Survey of smartphone perspective on computation offloading
2021, [28]	✓			✓			Survey of task offloading in MEC
2021, [29]			✓		✓		Survey of resource allocation in NFV
2021, [30]	✓	✓			✓		Survey of collaborative task scheduling in edge computing
2021, [31]				✓	✓		Resource allocation in heterogeneous 5G networks

TABLE II: Comparison between Existing Surveys and our Survey: Emerging Techniques for Resource Allocation in MEC

Ref.	**ML/DL-Enabled MEC**	**Enabling Techniques for ML/DL Tasks in MEC**	**Taxonomy of ML/DL for RA**		**In-depth Review of Works focused on ML/DL for RA**			**Focus of Discussion**
			Focused on ML	Focused on DL	ML/DL-based Offloading	ML/DL-based Scheduling	ML/DL-based Joint RA	
2019, [32]		✓				✓		Discussed key factors that enables the implementation of DL in mobile networking applications
2019, [33]		✓		✓	✓			Discussed techniques for quickly running DL inference in MEC
2020, [34]		✓	✓	✓	✓			Discussed federated learning for MEC optimization
2020, [35]			✓	✓		✓		Mainly discussed ML-based resource allocation techniques for HetNets, MIMO, D2D, and NOMA networks.
2020, [25]		✓						Survey of computation offloading modeling in edge computing
2020, [36]			✓		✓			Survey of ML-based computation offloading modeling in edge computing
2020, [37]				✓				DL for 5G networks
2020, [38]					✓			Survey of stochastic-based offloading mechanisms in edge/cloud
2020, [39]	✓	✓						Investigated key techniques about the convergence of DL and MEC, MEC for DL, and DL for Edge
2020, [40]		✓			✓			Survey on edge intelligence, mainly discusses caching, training/inference, and offloading methods in MEC.
2020, [41]							✓	Survey on MEC for 5G and IoT. Mainly discusses enabling technologies for MEC in 5G
2021, [42]			✓					Survey on task offloading in edge and cloud computing
2021, [43]	✓		✓		✓	✓		Mainly focused on conventional techniques for resource scheduling in MEC
Our survey	✓	✓	✓	✓	✓	✓	✓	**In-dept survey of ML/DL-based resource allocation mechanisms in MEC (See Section I-B for key contributions)**

offloading schemes (i.e., full offloading and partial offloading), objectives, device mobility, and multi-hop cooperation. The authors of [25] provide a survey on computation offloading modeling in edge computing. Since the offloading problem is an optimization problem, they classify the offloading modeling approaches from different perspectives, including non and convex optimization, Markov Decision Process (MDP), game theory, Lyapunov optimization, and machine learning. In

[36], the authors present a survey on ML-based computation offloading in MEC systems and classify all the solutions into reinforcement learning, supervised learning, and unsupervised learning approaches. In contrast to Shakarami et al. [36], the authors of [38] provide a survey on stochastic-based offloading approaches in MCC, MEC and Fog computing systems. They propose a taxonomy to classify the approaches into three Markow models, namely, Markov chain, Markov process, and hidden Markov. In [39], the authors investigate the convergence of MEC and deep learning. They mainly discuss enabling techniques for the integration of MEC and DL, i.e., DL applications in MEC, DL training/inference in MEC, MEC for DL services, and DL for optimizing MEC.

2) Existing Surveys on Task Scheduling: In [44], the authors address the task allocation and load balancing problems in distributed systems. They focus on five main aspects: control, resource optimization, reliability, coordination strategy among heterogeneous nodes, and network structure. The authors of [23] provide a survey on scheduling techniques in different cloud models including traditional, serverless, and Fog-cloud environments. In [35], the authors investigate ML algorithms for resource management in wireless IoT networks. They mainly survey machine learning techniques for emerging cellular IoT networks such as MIMO, D2D communications, and NOMA networks. However, they did not address the task offloading problem which is vital for ensuring high performance in IoT networks, especially in the presence of a large number of computationally intensive tasks. The authors of [30] present a survey of collaborative task scheduling problems in edge computing. They analyze the problem from four main perspectives: computing architectures (e.g., device-edge, device-edge-cloud, etc.), computation task models (e.g, local execution, offloading types, etc.), optimization objectives, and scheduling methods. In [43], the authors present a comprehensive survey of resource scheduling in edge computing. They classify the existing works from three aspects including computation offloading, resource allocation, and resource provisioning. They also discuss different techniques of resource scheduling such as heuristic, approximation, game theory, and machine learning. Compared to [30], the authors of [43] provide more details about the scheduling methods and the optimization objectives. However, they also don't provide an in-depth review of ML/DL-based methods for resource scheduling.

In summary, most of the existing surveys on resource allocation either focused on the task offloading problem or task scheduling problem, ignoring the joint task offloading and scheduling problem. Also, as shown in Table I, the majority of the existing surveys focused on the traditional resource allocation methods such as heuristics, meta-heuristics, and game theory ignoring the emerging ML and DL techniques. Moreover, the emerging ML and DL techniques that have been used to solve the resource allocation problem in MEC have not been comprehensively discussed in the existing surveys.

To the best of our knowledge, there is no survey that thoroughly discussed the application of ML and DL for resource allocation in MEC while considering the other aspects shown in Table II, i.e., the ML/DL-enabled MEC; enabling techniques for ML/DL tasks in MEC; taxonomy of ML/DL for resource allocation, and in-depth review of works focused on ML/DL for resource allocation.

Motivated by this, we propose an in-depth survey of ML/DL-based resource allocation methods in MEC. Particularly, we survey recent works that used ML and DL techniques to address the task offloading, task scheduling, and joint resource allocation problems in MEC while considering the other aspects shown in Table II.

B. Contributions

In contrast to the existing surveys depicted in Table I and Table II, this survey focuses on ML and DL techniques for resource allocation in MEC. The key contributions of this article are as follows:

- We discuss the advantage of applying ML and DL for MEC (ML/DL-enabled MEC) by presenting three use cases from three perspectives: end-users, service providers, and networking services.
- We discuss potential technologies for quickly running ML and DL tasks (i.e., training and inference) in MEC.
- We discuss potential ML and DL algorithms for resource allocation in MEC, and their advantages and disadvantages are summarized.
- We conduct a comprehensive and in-depth survey of recent works that used ML and DL methods to address the resource allocation problem in MEC. Particularly, we discuss and classify current ML and DL-based methods for resource allocation from three aspects: task offloading, task scheduling, and joint resource allocation.
- We discuss lessons learned from the state-of-the-art ML and DL-based methods for resource allocation in MEC, which will help researchers to well-understand how and when ML and DL-based methods outperform the traditional techniques for resource allocation in MEC.
- We discuss key challenges and present future research directions of applying ML and DL for resource allocation in MEC.

The rest of the paper is organized as follows. Fig. 2 shows the structure of the paper. Table III shows all the acronyms used in the paper. Section II presents an overview of MEC, ML/DL, and resource allocation. Section III discusses the advantages bring by ML and DL for MEC. Section IV presents enabling technologies for ML and DL tasks in MEC. Section V presents potential ML and DL techniques for resource allocation. Section VI presents an in-depth survey of ML/DL-based methods for task offloading in MEC. In Section VII, we thoroughly survey state-of-the-art ML and DL methods for task scheduling in MEC. Section VIII thoroughly reviews recent works addressing the joint resource allocation problem using ML/DL methods. In section IX, we discuss challenges and future research directions. Finally, Section X concludes this survey.

II. Overview

In this section, we summarize the basics of MEC, ML/DL, and resource allocation.

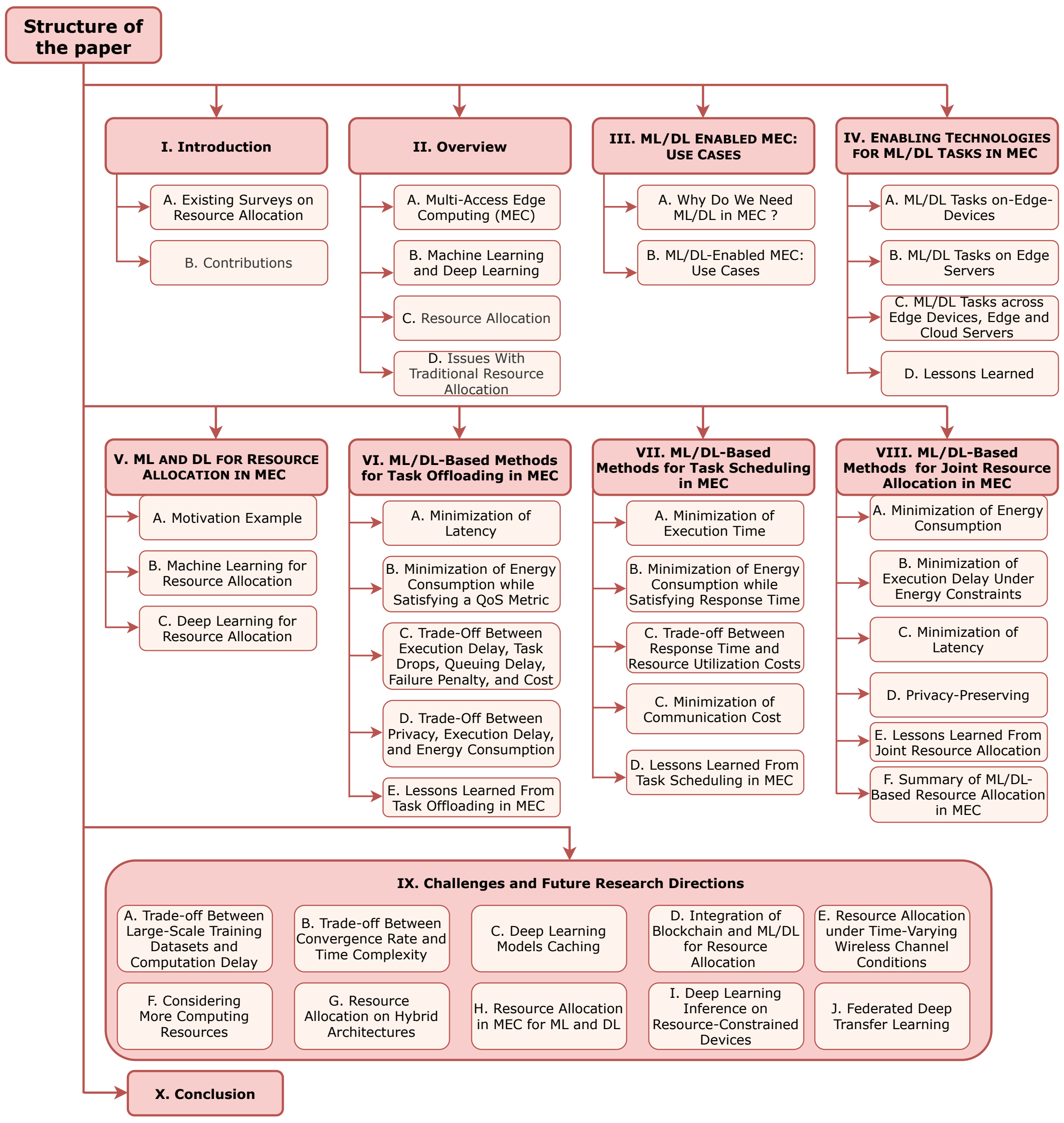

Structure of the paper
I. Introduction
A. Existing Surveys on Resource Allocation
B. Contributions
II. Overview
A. Multi-Access Edge Computing (MEC)
B. Machine Learning and Deep Learning
C. Resource Allocation
D. Issues With Traditional Resource Allocation
III. ML/DL ENABLED MEC: USE CASES
A. Why Do We Need ML/DL in MEC ?
B. ML/DL-Enabled MEC: Use Cases
IV. ENABLING TECHNOLOGIES FOR ML/DL TASKS IN MEC
A. ML/DL Tasks on-Edge-Devices
B. ML/DL Tasks on Edge Servers
C. ML/DL Tasks across Edge Devices, Edge and Cloud Servers
D. Lessons Learned
V. ML AND DL FOR RESOURCE ALLOCATION IN MEC
A. Motivation Example
B. Machine Learning for Resource Allocation
C. Deep Learning for Resource Allocation
VI. ML/DL-Based Methods for Task Offloading in MEC
A. Minimization of Latency
B. Minimization of Energy Consumption while Satisfying a QoS Metric
C. Trade-Off Between Execution Delay, Task Drops, Queuing Delay, Failure Penalty, and Cost
D. Trade-Off Between Privacy, Execution Delay, and Energy Consumption
E. Lessons Learned From Task Offloading in MEC
VII. ML/DL-Based Methods for Task Scheduling in MEC
A. Minimization of Execution Time
B. Minimization of Energy Consumption while Satisfying Response Time
C. Trade-off Between Response Time and Resource Utilization Costs
C. Minimization of Communication Cost
D. Lessons Learned From Task Scheduling in MEC
VIII. ML/DL-Based Methods for Joint Resource Allocation in MEC
A. Minimization of Energy Consumption
B. Minimization of Execution Delay Under Energy Constraints
C. Minimization of Latency
D. Privacy-Preserving
E. Lessons Learned From Joint Resource Allocation
F. Summary of ML/DL-Based Resource Allocation in MEC
IX. Challenges and Future Research Directions
A. Trade-off Between Large-Scale Training Datasets and Computation Delay
B. Trade-off Between Convergence Rate and Time Complexity
C. Deep Learning Models Caching
D. Integration of Blockchain and ML/DL for Resource Allocation
E. Resource Allocation under Time-Varying Wireless Channel Conditions
F. Considering More Computing Resources
G. Resource Allocation on Hybrid Architectures
H. Resource Allocation in MEC for ML and DL
I. Deep Learning Inference on Resource-Constrained Devices
J. Federated Deep Transfer Learning
X. Conclusion

Fig. 2: Structure of the survey.

A. Multi-Access Edge Computing (MEC)

Since the fundamental of MEC have been widely studied in the literature [3], [5], [8], [45], [46], in this sub-section, we discuss key networking technologies for MEC realization, including Network Function Virtualization (NFV) and Software Defined Networking (SDN). Before we introduce these networking technologies, it is natural to answer to the following question: Why Do We Need Multi-Access-Edge Computing?

1) Why Do We Need Multi-Access-Edge Computing?: Traditionally, the huge volume of data (big data) are mainly stored and analyzed on powerful remote cloud data centers. However, today, with the increasing number of smart devices (with limited resources and battery capacity) connecting to the internet through 4G/5G networks, processing the data generated by the devices on the remote cloud will incur high latency. The reason is that cloud servers are located far away from the smart devices (i.e., data sources). Hence, MEC paradigm is introduced by ETSI ISG [6] to bring the cloud services closer to the data sources and the end-users. MEC is characterized by ultra-low latency and high bandwidth.

2) Network Function Virtualization (NFV): NFV is a new network concept that aims to manage networking functions by evolving virtualization technology [47]. It has been proved that NFV has many benefits such as reducing the monetary cost of hardware infrastructure, optimizing the quality of service deployment, orchestrating many virtual networks, and scaling of network services [8]. NFV allows network service providers and vendors to implement network functions in software by evolving virtualization technologies rather than run on purpose-built hardware. NFV has also several use cases and applications including, traffic analysis [48] and security threats analysis [49] [50]. In NFV, services are implemented by a sequence of Virtual Network Functions (VNFs) that can run on servers [29]. These VNFs are also called Service Function Chaining (SFC). Furthermore, NFV offers a more efficient

TABLE III: Acronyms Used in the Paper

Acronym	Definition	Acronym	Definition	Acronym	Definition
A3C	Asynchronous Advantage Actor–Critic	FCM	Fuzzy C-Means	PPO	Proximal Policy optimization
ABC	Artificial Bee Colony	FIFO	First In, First Out	PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization	FCFS	First Come First Serve	QL	Q-Learning
AI	Artificial Intelligence	FF	First Fit	QoE	Quality of Experience
ANN	Neural Network	GA	Genetic Algorithm	QoS	Quality of Service
AE	AutoEncoder	Gbps	Gigabit billions of bits per second	RA	Resource Allocation
API	Application Programming Interface	GPU	Graphics Processing Unit	RAE	Relative Absolute Error
BS	Base Station	HCA	Hierarchical Clustering Algorithm	RAN	Radio Access Network
CART	Classification and Regression Tree	ILP	Integer Linear Programming	RF	Random Forest
CN	Corps Network	IoT	Internet-of-Things	RL	Reinforcement Learning
CMDP	Constrained Markov Decision Process	JTOS	Joint Task Offloading and Scheduling	RNN	Recurent Neural Network
CNN	Convolutional Neural Networks	KNN	K-Nearest Neighbors	RR	Round Robin
CPU	Central Processing Unit	LP	Linear	RSU	Road Side units
CRL	Clustered Reinforcement Learning	LPL	Local Processing Policy	SA	Simulated Annealing
CRN	Cognitive Radio Network	LTE	Long Term Evolution	Seq2Seq	Sequence-to-Sequence
CSI	Channel State Information	LSTM	Long Short-Term Memory	SL	Supervised Learning
D3QN	Dueling Double Deep Q-Network	MAQL	Multi-Agent Q-Learning	SLA	Service Level Agreement
D2D	Device-to-Device	MCC	Mobile Cloud Computing	SDN	Software Defined Networking
DAG	Direct Acyclic Graph	MCTS	Monte Carlo Tree Search	SVM	Support Vector Machine
DC	Data Center	MDP	Markov Decision Process	SOM	Self organizing Map
DCS	Distributed Computing System	MEC	Multi-access Edge Computing	TD	Temporal Difference
DE	Differential Evolution	MGM	Markov Game Model	TL	Transfer Learning
DL	Deep Learning	MIMO	Multiple Input Multiple Output	UE	User Equipment
DNN	Deep Neural Network	MINLP	Mixed Integer Nonlinear Programming	USL	Unsupervised Learning
DRL	Deep Reinforcement Learning	ML	Machine Learning	V2I	Vehicle-to-Infrastructure
DT	Decision Tree	MLP	Multilayer Perceptron	V2V	Vehicle-to-Vehicle
DTL	Deep Transfer Learning	MRL	Meta-Reinforcement Learning	VM	Virtual Machine
EFT	Earliest Finished Time	ms	millisecond	WAP	Wifi Access Point
EL	Ensemble Learning	MTL	Multi-task Transfer Learning		
EPG	Exact Potential Game	NFV	Network Functions Virtualization		
EST	Earliest Start Time	NOMA	Non-Orthogonal Multiple Access		
ET	Execution Time	PDS	Post Decision State		

and scalable network resources allocation for network functions, and therefore it can significantly reduce both operating expenses (OPEX) and capital expenses (CAPEX) of network service providers [51].

Although NFV brings a range of benefits for both academia and industry, the resource allocation in NFV, in particular, in MEC brings new challenges. For instance, since in NFV, the network functions must be executed in a specific order, it is crucial to investigate an efficient offloading and placement mechanism of VNF. Also, the scheduling of SFC is another challenge that needs to be considered in MEC. Towards this, the authors of [52] propose a deep learning based approach to address the SFC scheduling problem in MEC. In [53], the authors present a survey on hardware acceleration techniques for NFV. Moreover, the work in [29] provides a compressible survey of resource allocation problems in NFV.

3) Software Defined Networking (SDN): SDN is another networking technology for designing cost-effective and adaptable networks [54]. Many studies focus on the convergence of SDN and NFV at the MEC network. For instance, Light-MANO [55] is a multi–access networking framework, which converges SDN and NFV into a single lightweight platform for the management and orchestration of network services over distributed NFV systems. Other studies investigate the integration of networking technologies with MEC, such as [56] which investigates the convergence of O-RAN with MEC, SON, and network slicing in 5G networks; and [57] which studies the pairing of cloud RAN and MEC. The authors of [51] present a comprehensible survey of NFV and its relationship with SDN. The works in [58], [59], [60] investigate the advantages of SDN and NFV in the ecosystem of MEC. LayBack [61] is an architecture that facilitates the communication and computation of resource sharing among different networking technologies. Moreover, the authors of [62] provide a comprehensible survey of networking technologies for ultra-low latency (ULL) applications.

B. Machine Learning (ML) and Deep Learning (DL)

There are many surveys that have already discussed the basics of ML and DL techniques, such as [73], [63], [74], [75], [69], [70]. For this reason, in this section, we discuss popular artificial neural networks, which are commonly used for training and inference in MEC. Table IV summarizes existing surveys on ML/DL in MEC networks.

1) Popular Neural Networks: Artificial neural networks (ANNs) commonly called Neural Networks (NNs) are series of mathematical functions that attempt to retrieve the desired output from an input dataset by mimicking the way biological neurons operate. A neural network comprises layers of interconnected nodes which contains an input layer, one or many

TABLE IV: Summary of Existing Surveys on ML/DL

Ref.	Focus of Discussion
2017, [63]	Survey of ML techniques in cellular networks
2018, [64]	DL for wireless networks. Mainly focused on the applications of DL for different network layers, and DL to enhance network security
2018, [65]	DL for IoT big data and streaming data analytics
2019, [66]	Discussed DRL approaches in communications and networking
2020, [67]	Integration of Blockain and ML in communications and networking
2020, [68]	Relationship between ML and privacy protection in 6G networks
2020, [69]	ML and DL for IoT security
2021, [70]	Survey of DL and it applications
2021, [71]	ML techniques for network optimization to meet the end-to-end QoS and QoE
2021, [72]	ML techniques in the edge network. Mainly discusses models compression techniques, hardware, and software stacks

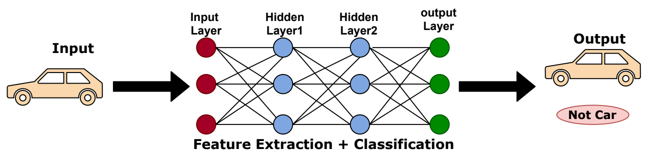


Fig. 3: Deep Learning.

hidden layers, and an output layer. Each node (or perception or artificial neuron) is associated with a weight and threshold. Neural networks are at the core of DL algorithms. A neural network has mainly two phases: training and inference, which will be discussed in the next sub-section.

Deep Learning (DL) is a particular type of ML model based on ANNs with multiple non-linear processing layers to automatically extract complex representation from data and to design its high-level abstractions [66]. As shown in Fig. 3, the input data passes through multiple hidden layers which perform some operations (e.g., matrix multiplications). The output of a layer is generally the input to the next layer. The output of the final layer is either a feature or a classification output. In contrast to traditional ML algorithms which first partition the feature extraction and classification, then solve them separately; deep learning models integrate the feature extraction and classification, and the end-to-end approach is adopted to solve the problem [76]. A DL model with many hidden layers in sequences is called a deep neural network (DNN). There are three main models of DNNs including Multilayer Perceptron (MLP), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs).

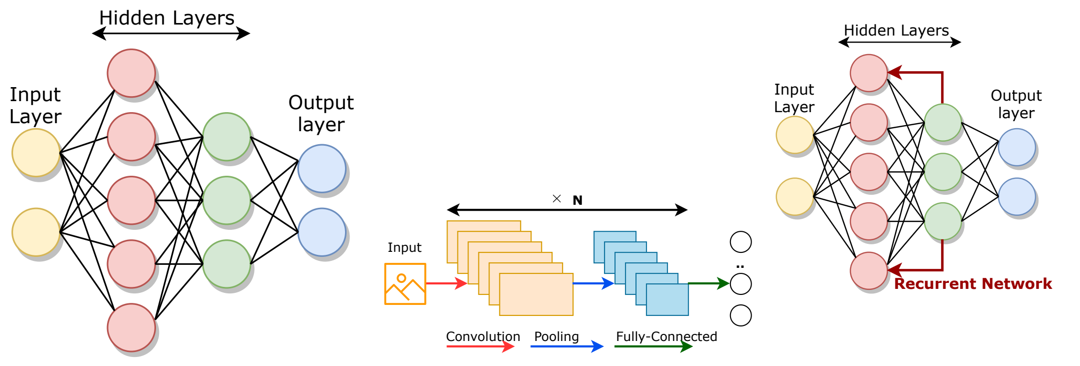
a) Multilayer Perceptron (MLP): MLPs are type of feedforward neural networks, which comprise a series of feedforward fully connected layers (Fig. 4a). MLPs use a backpropagation method (supervised learning) for training. Backpropagation helps to adjust the weights of the perceptrons to get the expected output or an output closer to the expected one. MLPs are generally applied for classification and regression problems.

b) Convolutional Neural Networks (CNNs): CNNs (or ConvNets) contain multiple feature extraction layers, including a series of convolutional layers, pooling layers, and fully connected layers [77] (Fig. 4b). CNNs are commonly used in computer vision. A CNN learns to extract the features of the inputs (e.g., images or videos) to achieve a specific task, such as image classification and pattern recognition. Compared to MLPS, CNN models extract the simple features from inputs by executing mathematical operations (convolution operations), in particular, matrix multiplication. There are many CNN models such as AlexNet [78], VGG-16 [79], GoogleNet [80], ResNet [81], SqueezeNet [82], and MobileNets [83]. More details about these CNN models can be found in [77], [84].

c) Recurrent Neural Networks (RNNs): RNNs contain intralayer recurrent connections, which make them different from MLP and CNN models [85] (Fig. 4c). RNNs are mainly used for time-series of sequential input data to make predictions about future outputs (e.g., sales forecasting).

2) ML/DL Training and Inference: In the context of machine learning, training is the process of learning an ANN or a DNN using dataset to achieve a specific AI task (e.g., image or voice recognition). The training is performed by feeding the ANN/DNN data, so that it can make a prediction about the type of data [86]. For example, suppose that we are training a DNN to differentiate three different objects, such as a cup, car, and bicycle as shown in Fig. 5a. The first step is to gather a dataset that consists of thousands of images that contain cups, cars, and bicycles. The second step is to feed the images to the DNN so that it can make a prediction about what is the image. If the prediction is inaccurate, the DNN is updated by correcting errors until obtaining more accurate predictions. The training process continues until the DNN makes prediction that satisfies the desired accuracy. Once the required accuracy is obtained, the DNN training is finished, and the trained model is ready to be used to make inference (prediction).

Inference consists of using a trained ANN/DNN model to make prediction on novel data. ML/DL inference is accomplished by feeding new input data to the neural network, allowing ANN/DNN to classify the input data. Considering the previous example (Fig. 5b), the DNN can be fed new input images of cups, cars, bicycles, and other images. After fully trained a DNN, it is simplified (compressed) before it can be deployed to the resource-constrained device. This is due to the fact that fully trained DNN models require more computational resources in terms of storage, CPU/GPU, energy, and latency. On the other hand, compressing a DNN model will impact the model accuracy. In [87], the authors present a method to minimize the inference time in embedded devices while meeting the user requirement. They propose an adaptive method to determine at the runtime, the best DNN model to use for a given dataset. To achieve this goal, the authors use an ML technique to automatically create a predictor that can quickly selects the optimum model to use. Firstly, the predictor trains the model off-line, then determines the optimum DNN by using the learned model of the input data. The proposed method is applied to the image classification problem and evaluated on a Jetson TX2 embedded DL system. The 50K images from the ImageNet ILSVRC 2012 validation dataset is used to evaluate the proposed method. The experimental results show that the proposed approach obtains



(a) Multilayer Perceptron (b) Convolutional Neural Network (c) Recurrent Neural Network

Fig. 4: Popular Neural Networks

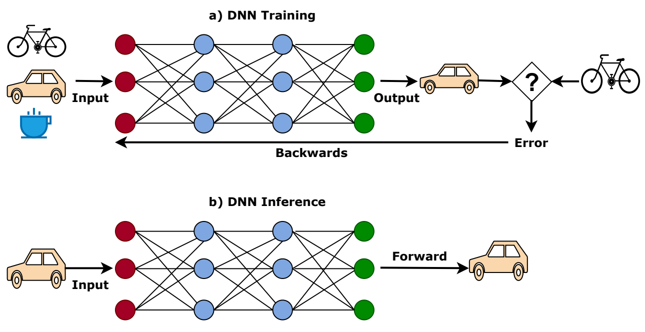


Fig. 5: DNN Traning versus DNN Inference

7.52% improvement in inference accuracy and 1.8x reduction in inference time over the individual DNN models.

C. Resource Allocation

In the literature, the resource allocation problem in Cloud/MEC is often referred to: (i) task offloading problem for deciding whether, where, how much, and what should be offloaded to the cloud/edge servers [5]; or (ii) resource provisioning problem for determining the adequate computation resources (e.g., servers) that will be used to execute each task; or (iii) task allocation problem [88] for ordering and mapping each task onto the best-suited computation resource. Also, the term "task scheduling problem" is often used to refer to the combination of the sub-problems (ii) and (iii) [10]. The term "joint resource allocation" is often used to refer to the combination of task offloading and task scheduling problems for jointly offloading and scheduling the application's tasks to the best-suited edge/cloud servers [4]. Hence, we follow the same motif throughout the rest of this paper and discuss the resource allocation problem from three main aspects: (1) task offloading problem, (2) task scheduling problem, and (3) joint resource allocation problem.

1) Task Offloading: Task offloading is the process of transferring computation-intensive tasks to a set of remote computing machines (e.g., cloud or edge servers) that can process the tasks. An efficient task offloading strategy can significantly reduce the latency and the total energy consumption of the IoT devices [1]. For instance, the authors of [89] propose a secure task offloading mechanism that can minimize the virtual reality (VR) devices' computation load while satisfying the VR's QoE and resisting malicious attacks. To this end, the authors introduce a blockchain to detect malicious attacks during tasks offloading and data processing, and use a reinforcement learning algorithm to properly allocate resources based on the defined-QoE requirements. In the proposed mechanism, the main information of each viewport providing offloading is stored by a transaction $< TXN_v, m >_{t-1}$ of a blockchain controller (BC) implemented at the edge access point (EAP), where TXN_v denotes the transaction ID, m denotes the BC ID, and t represents the time slot. As shown in Fig. 6, each EAP m needs to carry out the task offloading and blockchain consensus in parallel at each time slot. There are mainly two phases for each EAP m to perform over time slot t, namely the transaction generation and blockchain consensus. During the transaction generation at time slot t, the BC m generates transactions $< TXN_v, m >_{t-1}$ according to the offloaded records at time slot $t-1$. Then, the number of transactions gathered by BC m at time slot t denoted by $T_m(t)$ is calculated by (1)

$$T_m(t) = \sum_{v=1}^{V_m} x_{m,v}(t-1), \tag{1}$$

where $x_{m,v}(t-1)$ is the offloading decisions, and V_m the number of VR devices. Concerning the blockchain consensus process, it comprises five phases including request, pre-prepare, prepare, commit, and reply as shown Fig. 6. This consensus is based on the Byzantine fault tolerance (PBFT) protocol which is widely used in the literature.

The challenges of task offloading in MEC are as follows [88], [27]:

- *Decision on task offloading*: a key step regarding task offloading is to decide whether to offload or not the computational tasks of IoT devices. This decision may result in: *i) local execution*, that is, the task is processed locally by the IoT device due to the cost constraints or MEC resources constraints; *ii) full offloading* which means that the whole task is offloaded and processed by the MEC server, and *iii) partial offloading*, i.e., the task

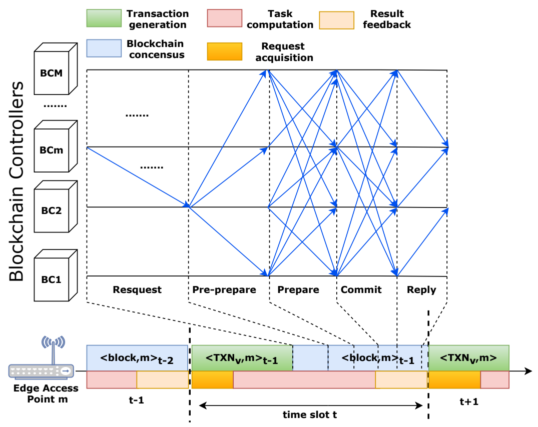


Fig. 6: Blockchain-enabled secure task offloading [89]

is divided into two parts, one part is executed by the IoT device and the rest is offloaded to the MEC server [5].

- *What to offload:* It consists of determining the parts of the IoT application that should be offloaded (i.e, *offloadable parts*) and the parts that should be executed locally (i.e., *non-offloadable parts*, e.g., user input or camera).
- *Where to offload:* It consists of selecting the target computing infrastructure (e.g., to MEC server, cloud server, or cloudlets) that will execute the offloaded tasks.
- *How to offload:* the answer to this question solves technical issues related to the task offloading mechanism. For example, a good task offloading scheme should satisfy the QoS requirements of the IoT application.
- *When to offload:* It determines the appropriate time for transferring the offloadable tasks from IoT devices to the selected MEC servers.

2) Task Scheduling: Task scheduling is the process of assigning an application's tasks to the computation resources and ordering their execution so that the dependencies between them are maintained while meeting the required QoS [10]. Efficient task scheduling mechanisms are vital for maximizing the QoS performance [90], maximizing the revenue earned by MEC service providers [91], and minimizing the energy consumption and delay of the application [92]. Also, processing the high volume of data generated by billions of IoT devices onto MEC servers requires appropriate task scheduling mechanisms. Besides, these data may require different QoS that a single MEC server cannot provide. Therefore, a proper task scheduling strategy is vital to meet the required QoS.

3) Use case of task offloading and scheduling in MEC: To help readers understand the task offloading and scheduling problem, we illustrate a use case from the perspective of the end-users, namely, virtual reality (VR) application which is considered to be the most key application market in future [93]. We first suppose a full task offloading and scheduling scenario, where the offloading algorithm implemented in the edge device decides to offload the whole VR application represented by a direct cyclic graph (or tasks graph) to the edge server (see Fig. 7a). Then, the scheduling algorithm gives priority to each task and allocates the tasks to the available edge servers based on their priority. After the execution of the tasks, the result from the edge servers is sent back to the edge device. In this scenario, the scheduling length (makespan) is equal to 65. The second scenario is the partial task offloading and scheduling, where the offloading algorithm decides to execute a part of the VR tasks locally and the rest (e.g., computation-intensive tasks) is offloaded to the MEC servers (see Fig. 7b). The makespan in the second scenario is 45 because only three tasks are executed on the edge servers.

4) Traditional Techniques for Resource Allocation in MEC: The traditional resource allocation techniques can be categorized into approximation-based, heuristic-based, meta-heuristic-based, and game-theoretic-based. Approximation methods find a quasi-optimal solution (within polynomial time [94]) to NP-hard problems that are guaranteed to be close to the optimal solution [95]. In the context of resource allocation, the methods for designing approximation algorithms include greedy-based [96], [97], [98], local search-based [99], [100], primal-dual-based [101], [102], and LP-rounding-based [103], [104], [105].

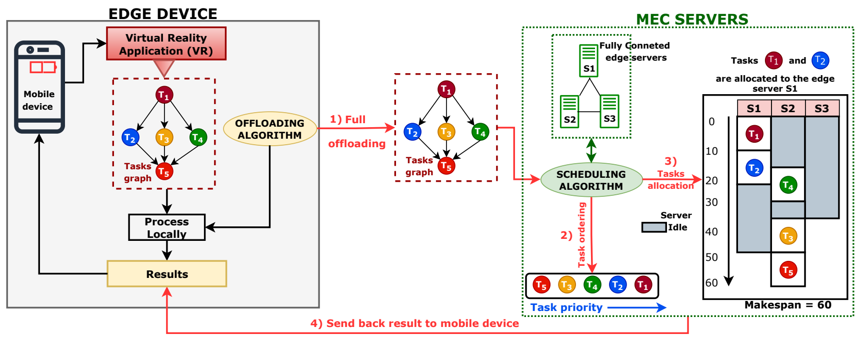
A heuristic technique is defined as "any approach to problem-solving that employs a practical method that is not guaranteed to be optimal, perfect or rational, but it is nevertheless sufficient for reaching an immediate short-term goal [106]". Heuristic methods are designed for specific problems, and they can find reasonably good solutions in an acceptable time interface [10], [107]. In the context of resource allocation, heuristic techniques can be classified into three groups: list-based [108], [109], [110] [111], [112], [113]; clustering-based [114], [115], [116], [117], [118], and duplication-based [119], [120], [121]. Among the three heuristic methods, the list-based approach is the simplest one with quadratic time complexity, i.e., $O(t^2 \times p)$ for t tasks and p processors.

While the heuristic techniques are designed for specific problems, the meta-heuristic approaches are designed for general-purpose optimization problems [10]. In terms of total execution times of an application (or makespan), the meta-heuristic techniques are better than heuristic-based due to their ability to search in a larger space of solutions. However, compared to heuristic methods, the running time of meta-heuristic algorithms increases rapidly when the number of tasks in the application increases [122]. Thus, meta-heuristic methods are not suitable for large-scale IoT applications. The meta-heuristic techniques can be categorized into genetic algorithm (GA) [123], [124], [125], particle swarm optimization (PSO) [126], [127], [128], ant colony optimization (ACO) [129], [130], [131], and simulated annealing (SA) [132], [133].

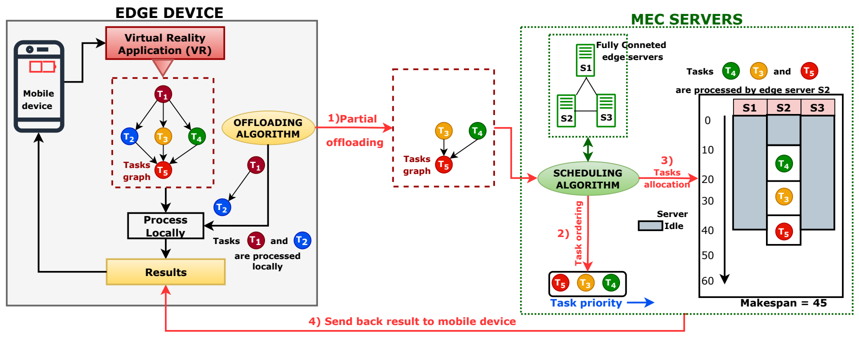
Game theory is a branch of applied mathematics that studies interactive decision-making, where the outcome for each player depends on the actions of all [134], [135], [136]. Game theoretical approaches can be classified into two groups: the classical game, which assumes that all players are rational, and the evolutionary game, which considers that a player may play for his interest and has limited information about available choices of strategies [137].

D. Issues With Traditional Resource Allocation Techniques

Due to the NP-hardness of the resource allocation problem, the solutions obtained by the conventional methods are not



(a) Full task offloading and scheduling



(b) Partial task offloading and scheduling

Fig. 7: Use case of task offloading and scheduling in MEC

globally optimal. They have difficulties in adapting to various QoS requirements and dynamic environments. In general, the traditional resource allocation techniques have the following main limitations:

- *Computationally expensive*: The execution time of traditional methods proportionally increases with the increase in the application size (i.e., number of tasks), which leads to extra overheads in terms of computational time. Therefore, they are unappropriated for delay-sensitive and data-intensive applications. Also, meta-heuristic methods such as GA are time-consuming since they maintain large solutions in memory.
- *Slow convergence*: The traditional resource allocation methods have a slow convergence rate since they cannot learn from previous sub-optimal solutions.
- *Lack of adaptability*: The solutions obtained through traditional resource allocation methods are sensitive to the environment changing. They supposed that the computing environment is static and known by mobile users. If a parameter about the computing environment (e.g., the wireless channel information) changes, the optimization problem would be reformulated to take into account the new changed parameter to achieve the desired objective. Therefore, conventional resource allocation schemes are not adaptable in time-variant dynamic environments.

III. ML/DL Enabled MEC: Use Cases

A. Why Do We Need ML/DL in MEC ?

The growing numbers of IoT devices connected to the Internet have generated a massive amount of data (pictures, audios, videos). Since the data is generated at the network edge, it is more beneficial to analyze them at the network edge. In this context, ML/DL techniques are necessary due to their ability to efficiently analyze and quickly extract features from a huge volume of data. Additionally, to efficiently execute and analyze the generated data, it is crucial to properly allocate (offload and schedule) them to the edge computational resources, which can satisfy the data requirements (e.g., latency, privacy, QoE). Since ML/DL are key techniques for data prediction, they can accurately predict both data requirements and the MEC computing nodes which will process the data.

B. ML/DL-Enabled MEC: Use Cases from three Perspectives

In this section, we present ML/DL use cases from three perspectives (see Fig. 8): 1) end-users, 2) service providers, and 3) networking.

1) End-User Perspectives: The execution of ML/DL tasks in MEC, in particular, on edge devices is beneficial to the end-users. In general, by running ML/DL at the edge of the network, the user QoE is maximized since they can predict user's requirements. For instance, a DL approach called LiveDeep is proposed to predict the user's viewport for live virtual reality (VR) streaming [138]. Facebook presents a data-driven approach to enable ML inference on smartphones with the objective to increase the QoE and reduce the latency [139]. Running ML/DL tasks on edge devices also enables the end-user to rapidly analyze and obtain its health report. For example, HealthFog [140] is a framework that implements deep learning in edge devices for automatic heart disease analysis. HealthFog delivers healthcare as a service using IoT

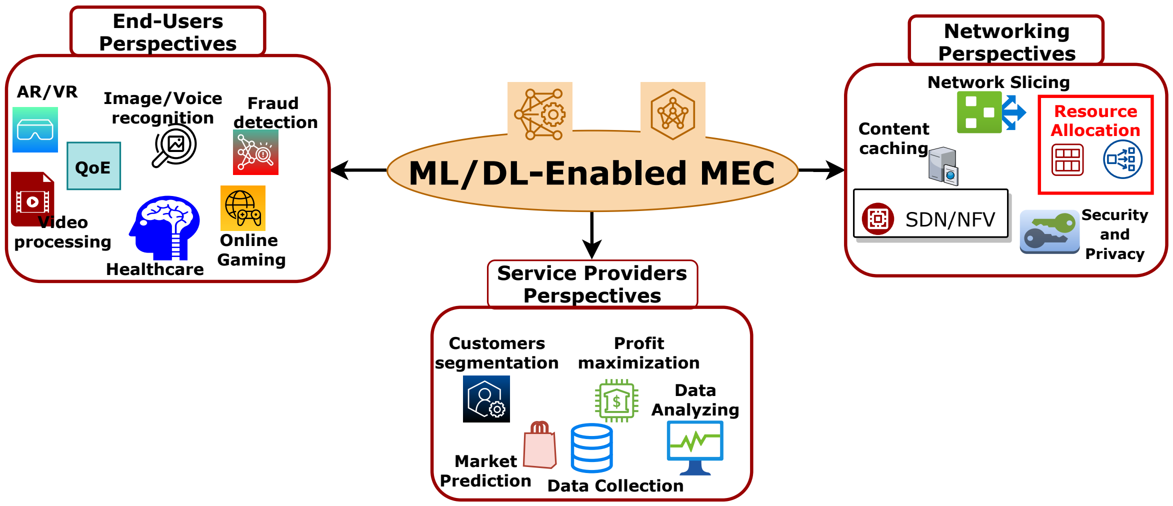


Fig. 8: ML/DL-Enabled MEC: Use Cases from three Perspectives

devices and efficiently executes user requests (i.e., the patient heart information). Additionally, bringing ML/DL models to the edge network enhances user privacy because the raw data required for DL tasks is stored on the edge devices instead of the cloud. Moreover, bringing ML/DL techniques at the edge of the network enables rapid access to the huge volume of real-time data generated by the IoT devices for rapid AI tasks, which in turn gives the devices the ability to respond to real-time events in an intelligent manner.

On top of this, ML/DL techniques enable end-users to intelligently offload their latency-sensitive applications (e.g., online gaming and mobile augmented reality (MAR)) to other devices or to the edge servers. Chakrabarti [141] proposes a DRL-based mechanism to offload MAR application' tasks to the nearby devices. The authors of [142] propose a DRL-based joint task offloading and migration approach, where DRL and LSTM are combined to solve the task offloading problem in MEC networks. The proposed algorithm called "Online Predictive Offloading (OPO)" uses LSTM to predict the load of the edge server with the objective to improve the convergence speed and accuracy of the DRL model during the offloading process. The experimental results shows that the proposed approach reduces the latency by 6.25% on average.

2) Service Providers Perspectives: One of the benefits of bringing ML/DL models to the edge of the network for the service providers or any stakeholder is the processing and analyzing of data generated by users or IoT devices at the edge network. This significantly reduces the cost and latency of sending data to the remote cloud for processing and analyzing. Also, the analyzed data can be exploited for security and safety (e.g., park monitoring, fire prevention), or for marketing purposes (e.g., users segmentation to determine users wishes).

Besides, the service providers can also maximize the long-term profit by intelligently selling their computation resources to the end-users. For instance, the authors of [143] present a public blockchain application in MEC with the objective to maximize the long-term profit of the service providers. In the public blockchain-enabled MEC network, each blockchain user (i.e., miner) offloads its proof-of-work puzzle tasks to the associated base stations operated by the MEC service providers. The goal of the miners is to maximize their mining rewards. To achieve these goals, a framework-based reinforcement learning is proposed. The proposed framework enables the service providers to maximize their long-term profit by dynamically adjusting the price per unit hash rate to a miner while taking into account the highest price that the miner can pay during the whole mining process. The framework also allows the miners to select the best response strategies, and therefore satisfy miners' objectives.

3) Networking Perspectives: The third ML/DL use case is the one optimizing networking technologies for MEC such as network slicing [144], [145], [146]; NFV [147], [148]; SDN [149] [150]; mobility management [151], [152]; content caching [153], [154]; resource allocation [155], [156], [157]; and security/privacy [158], [159].

Network slicing is a networking technology that enables network providers to split the physical network infrastructure into multiple logical networks [160]. Abidi et al. [145] propose a 5G network slicing approach based on deep belief network (DBN) and neural network (NN) (Fig. 9). Firstly, the number of IoT devices (e.g., mobile phones, cars, and cameras) in the 5G network is observed. Then, the attributes of the devices (e.g., device type, packet information, bandwidth) are collected. After that, the collected data are normalized into the interval [0,1] to reduce the redundant data. Then, the feature extraction is performed by multiplying a weight function with the attributes values of the network to obtain high-scale variation. The weight function is optimized by using a hybrid metaheuristic method, namely the glowworm swarm optimization method and the deer hunting optimization method. Next, the network slicing prediction is performed using DBN and NN to maximize the accuracy. The types of the predicted network slices are enhanced mobile broadband (eMBB), massive machine type communication (mMTC), and ultrareliable low-latency communication (URLLC). The experimental results prove that the accuracy of the proposed hybrid learning approach is better than the benchmark methods. For example, the simulation results proved that the proposed

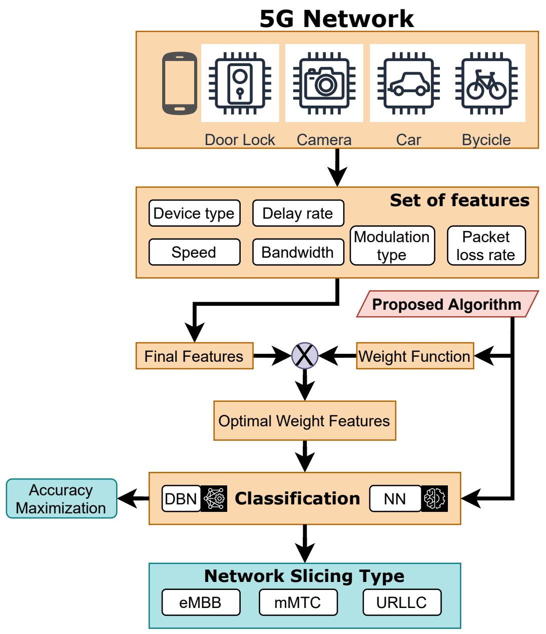


Fig. 9: ML and DL based 5G Network Slicing [145]

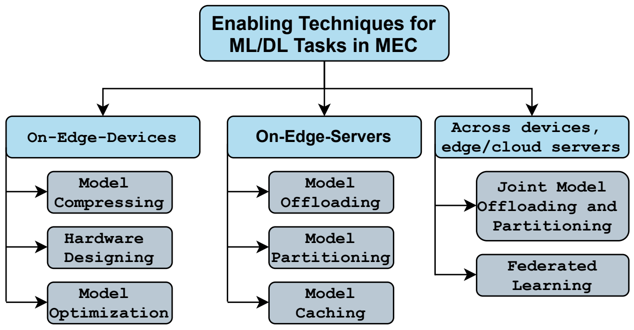


Fig. 10: Major Enabling Techniques for ML/DL Tasks in MEC

approach is 0.61% better than the PSO+NN+DBN based approach in terms of accuracy.

IV. Enabling Technologies for ML/DL Tasks in MEC

As the number of IoT devices increases, the need for researchers to understand how to design architectures that integrate ML/DL training and inference with MEC grows rapidly. Additionally, given the fact that MEC systems are distributed, a key question that arises is "where should we perform the training and inference and where should we deploy the fully trained model in MEC ?". In the literature, they are different approaches for performing ML/DL training and inference in MEC. Here, we discuss three major approaches (see Fig. 10): 1) On-Edge-Devices, where the ML/DL models training and inference are executed on the IoT device; 2) On-Edge-Servers, where data generated from the IoT devices are offloaded to one or more edge servers for training/inference; and 3) models training and inference across the edge devices, edge servers, and cloud servers.

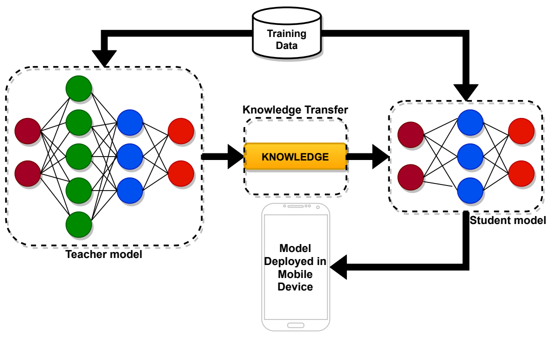
A. *ML/DL Tasks on-Edge-Devices*

At the intelligence devices layer, ML/DL techniques are applied for a range of applications, including virtual reality video streaming [161], image recognition [162], and pandemic tracking [163]. The execution of ML/DL models on the edge devices can reduce the latency of the training and inference time [4]. However, it is computationally expensive for ML/DL models, in particular, DNN, to make training and inference on resource-constrained devices due to millions of parameters that need to be refined over several time periods. For this reason, many studies have proposed strategies to reduce the training and inference times of DNN models running on resource-constrained devices while optimizing the QoS and QoE requirements. Such studies can have benefits throughout the MEC ecosystem by minimizing the latency of the DNN models while running on the IoT device or edge server. In this section, we discuss key approaches for enabling ML/DL tasks on smart devices such as model compression, hardware designing, and neural networks optimization.

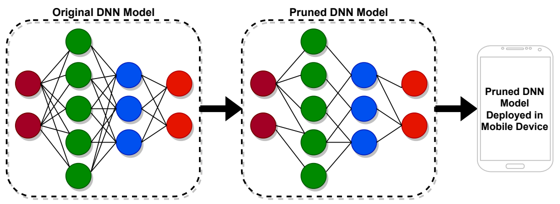
1) Model Compression: The common strategy for enabling ML/DL models on intelligent devices is to compress the model. Model compression reduces the resource and computational requirements, but it leads to an accuracy reduction compared with the original model. The most popular models compression techniques include knowledge distillation [164], pruning [165], and quantization [166].

The Knowledge Distillation (KD) method (also called as "student-teacher networks") transfers the knowledge learned from a larger DNN model (teacher model) to a model that has fewer parameters and layers (student model) (see Fig. 11a). The first step consists of training a larger DNN to generate labeled data. After that, the generated data is used to train a smaller and shallower mimic model, then the student model is deployed. Several studies have used this model compression technique. For instance, the authors of [167] combine knowledge distillation and auto-encoder methods to visually interpret and diagnose image classifiers. Furthermore, a small locally accurate model is trained to mimic the behavior of an original cumbersome DNN (big model) around one image of interest. In this approach, knowledge distillation is used to transfer the knowledge from the big model to the small model, while the auto-encoder is used to generate neighbors around the image of interest. Tanghatari et al. [168] propose a knowledge distillation approach to distribute the DNN training over IoT edge devices with the objective to protect data privacy on the edge devices and decrease the load on cloud servers. Furthermore, the knowledge of the main network is transferred to the generated small network. The experiments results show that the proposed approach preserves the IoT device data privacy and obtains on average 2.3% accuracy loss compared to the conventional centralized training on the cloud.

Pruning is a powerful compression method that removes redundant parameters of a neural network that are not necessary for training or inference (Fig. 11b). In [169], the authors propose a pruning technique based on activation maximization for CNN model acceleration and compression. Activation maximization is a simple method to visualize the features



(a) Knowledge Distillation



(b) Pruning

Fig. 11: Model compression techniques enabled ML/DL tasks on edge device

of a trained neural network with the objective to maximize the activation of certain neurons [170]. The experimental results based on RadioML2016.10a dataset [171] show that the proposed model obtains a higher accuracy compared to the weight sum (WS) [172] and average percentage of zeros [173] approaches. PruneFL [174], a federated learning approach with adaptive and distributed parameters pruning is proposed to minimize the training time on edge devices while ensuring a similar accuracy as the original model. PruneFL has two main phases: initial pruning at a selected edge device and further pruning which involves both the edge device and edge server during the federated learning process.

Concerning DNN quantization, it aims to reduce the number of bits required to store the weights of the neural networks [175] [176]. Coelho et al., [177] propose a novel heterogeneous quantization approach to minimize the energy consumption of the DNN model on Field Programmable Gate Arrays (FPGAs) while achieving high accuracy. The evaluation of parameters quantization during DNN training on edge device is the objective in [178]. Furthermore, it aims to understand how to select the quantization parameters during training to optimize neural networks for inference. The authors of [179] propose a quantization method in federated learning to enhance the efficiency of data exchange between edge servers and cloud servers. In this approach, the model training is performed on the edge servers, and the model aggregation is done on the cloud servers. The main idea is to quantize the neural network weights when the models are transmitted from the edge servers to the clouds and vice versa. Experiment results based on WikiText-2 [180] show that the proposed method reduces up to 19× the volume of data exchanged between the edge servers and cloud servers. Simulations results also prove that the impact on the validation loss of the final model is around 5%.

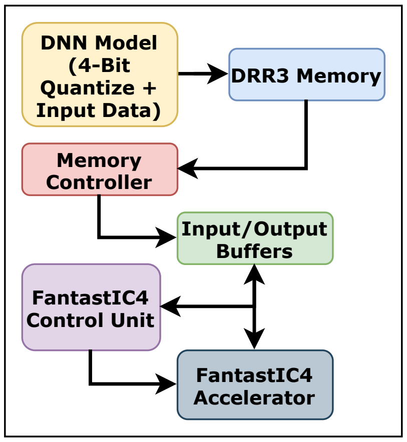


Fig. 12: FantastIC4 hardware accelerator architecture for DNN [181]

2) Hardware Designing: Although model compression is a powerful technique to enable model training and inference on edge devices, it can impact the final accuracy of the model. For instance, the pruning technique can reduce the model size but does not improve the training or inference time [182]. Also, most of the existing model acceleration approaches used compression techniques to reduce the complexity of the model. However, few compression techniques are implemented on popular DL frameworks such as Tensorflow Lite and Core ML [176]. For these reasons, recent studies have been focused on hardware architecture to facilitate the execution of ML/DL tasks on resource-constrained devices. For instance, the authors of [181] present FantastIC4, a new hardware architecture, which efficiently executes highly compact representations of DNNs based on fully-connected layers. As shown in Fig. 12, the FantastIC4 hardware-accelerator system is a combination of a CPU and an FPGA. FantastIC4 has three main parts: the software program, the DDR3 memory, and the hardware architecture on the FPGA chip. The software program comprises the CPU that transfers the input data and DNN model to the FPGA chip. Since the input data is usually very large, and cannot fully be stored on an on-chip BRAM (Block RAM), some of the data is stored in an off-chip DRAM Dynamic RAM). The input data is accessed through a memory controller built across a MIG (memory interface generator) IP. Concerning the FPGA chip part, it comprises the FantastIC4 control unit, memory controller, I/O Buffers, and the FantastIC4 accelerator. The memory controller aims to facilitate the transfer of the input data from the off-chip DRAM to the accelerator, then stores the execution results into the DRAM. The control unit manages the behaviour of other modules on the FPGA, the data movement, and the process inside the accelerator. Concerning the I/O buffers, they store the data for execution and cache the PSum data from the accelerator for inference of the subsequent layer. Finally, the heart of the system is the FantastIC4 accelerator, which reads the input data from the DRAM, performs the execution, and caches the results into the DRAM memory.

CMSIS-NN [183] is an efficient kernel designed to maximize the performance and minimize the memory footprint of neural networks on Arm Cortex-M CPUs targeted for IoT devices. MCUNet [184] is another framework designed for efficient neural network architecture (TinyNAS) and lightweight inference engine (TinyEngine) on microcontrollers. TinyNAS first optimizes the search space to fit the resource-constrained devices and then performs a neural network architecture search within the optimized space. TinyNAS is co-designed with TinyEngine to enhance the search space and fit large models. TinyEngine reduces the memory usage by 3.4× and accelerates the inference by 1.7-3.3× compared to CMSIS-NN [183]. Simulation results based on ImageNet proved that the MCUNet framework achieves 70.7% accuracy and accelerated the inference of wake word applications by 2.4-3.4×.

The authors of [185] present a hardware accelerator based on quantum annealer. Quantum annealer is a hardware architecture for discrete optimization problems. The proposed architecture outperformed GPUs and quantum annealers in terms of energy consumption. Hardware-Aware Automated Quantization (HAQ) [186] is a hardware architecture that uses reinforcement learning to automatically determine the quantization policy. HAQ also includes hardware architecture into a loop to minimize the latency, energy, and storage on the target hardware. Compared with conventional architectures (e.g., fixed bit width quantization), HAQ reduces the latency by 1.4-1.95× and the energy consumption by 1.9× with good accuracy.

There are other hardware accelerators for DNN tasks that have been proposed in the literature. For instance, DNNBuilder [187] can automatically build high-performance DNN hardware accelerators on FPGAs with the objective to satisfy the throughput and latency requirements of both cloud and edge devices. Kernel decomposition is another approach of hardware accelerators for DNN models. For example, ESCALATE [188] is an algorithm-hardware co-design for CNN accelerator based on kernel decomposition technique.

3) Neural Networks Models Optimization: Artificial network networks (ANNs) model optimization is another key approach to achieve high accuracy for ML/DL tasks. Recent studies have been focused on the architecture design of existing ANN models to create optimized ANN models instead of using compression methods to reduce their complexity. For instance, the authors of [189] propose an optimized ANN model to predict the thermal efficiency and water yield of solar still. The optimized ANN model used PSO and HWO (Humpback whale optimizer) [190] to optimize the traditional ANN model. Simulation results prove that the proposed optimized ANN model achieves the highest prediction accuracy compared to ANN, ANN-HWO, and ANN-PSO. RouteNet [191] is another NN model optimization, which is based on graph neural networks. RouteNet can optimize network representation in SDN. Compared to the conventional well-known NN (e.g, CNN, RNN) which are not designed to learn graph-based data, RouteNet is able to learn the complex relationship between topology, routing, and input traffic of a graph-structured network. The experimental results show that RouteNet obtains accurate delay prediction (Mean Relative Error) of 15.4%.

B. ML/DL Tasks on Edge Servers

While model compression and architecture optimization enable to run ML/DL tasks on edge devices, it is still challenging to deploy large DNNs models on resource-constrained devices with limited power, computation, and storage in real-time. Therefore, resource management approaches including task offloading, task partitioning, and content caching are good strategies to address this challenge.

1) ML/DL Tasks Offloading: Offloading ML/DL tasks from edge devices to more powerful servers such as edge servers or cloud servers is a good choice. Since the edge server is close to edge devices, it is natural to offload ML/DL computational tasks to the edge servers rather than the cloud. In [192], the authors propose an algorithm called "Multiple Algorithm Service Model (MASM)" to offload AI tasks to the cloudlet servers with the objective to minimize the energy consumption of the servers and the offloading delay cost while meeting the quality of the results (QoR). DNNOff [193] aims to automatically determine the DNN tasks that should be offloaded to the edge servers. To achieve this, the DNNOff algorithm first extracts the structure and parameters of the deep neural network model, then a random forest regression model is used to predict the execution cost of each layer. Finally, the DNNOff algorithm uses the prediction model to determine the parts that should be offloaded to the edge servers. The experiments based on real-world DNN applications with AlexNet, VGG, and ResNet models show that the DNNOff algorithm reduces the response time by 12.4–66.6%.

2) ML/DL Tasks Partitioning: He et al. [194] propose DNN tasks offloading approach to minimize the end-to-end inference delay. To achieve this goal, a tandem queueing model is used to analyze queueing and processing delays of DL tasks. Tandem queueing models are queueing theory models that consider the possibility that an end-user may request services from many sequentially arranged servers [195]. The authors of [194] first formulate the problem as a joint optimization problem, that is, DNN partitions deployment and resource allocation problems. Then, an algorithm based on Markov approximation is used to solve the problem. Simulation results prove that the proposed algorithm reduces the average end-to-end inference delay by 25.7%. In [196], the authors also present DNN tasks partitioning and offloading algorithm with the objective to minimize the processing delay and the computing burden of edge devices. Compared to [194] which used the Markov approximation method, in [196], a Mixed Integer Linear Programming (MILP) is used to solve the partitioning and offloading problems. The experiments results show that the proposed algorithm obtains up to 90.5% and 69.5% processing delay reduction compared with the MEC-server-only scheme (i.e., all DNN tasks are offloaded to the edge server) and mobile-device-only scheme (i.e., all DNN tasks are processed locally on the mobile devices), respectively. The work proposed in [197] extends [196] by considering not only the processing delay but also the energy consumption and price paid for DNN tasks execution on the edge server.

3) Content Caching: Edge servers can cache locally the user's related data near the location where the data have been generated to reduce latency. Therefore, content caching can

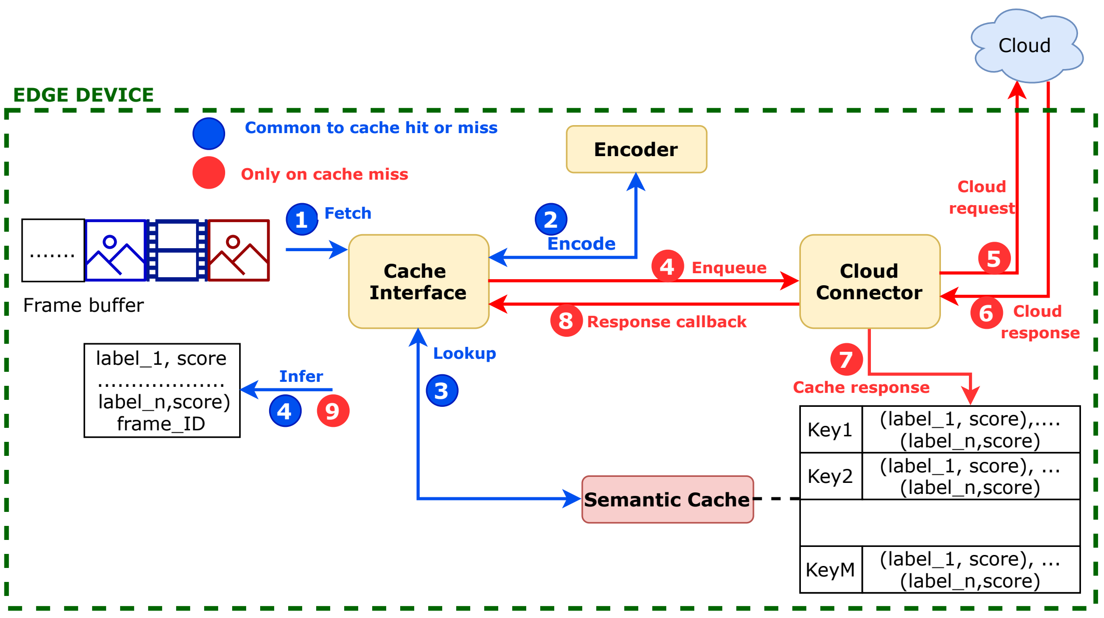


Fig. 13: Semantic cache to perform AI inference in edge [198]

enable model inference in the edge network. In [198], the authors propose a semantic cache approach to perform AI inference on unstructured data in edge nodes, which reduces the volume of data that needs to be sent to the remote cloud server. As shown in Fig. 13, the user first submits the input data (e.g., image or video) for AI inference to the cache service via the cache interface. Then, the interface forwards the input data to the encoder for features extraction. The encoder first searches in the cache. If there is a cache miss, then the image is sent to the cloud server which will perform the inference. The inference result from the cloud is stored in the cache indexed by a key, and finally, it is sent back to the user.

CacheNet [199] is a novel DNN model caching framework, which caches the low-complexity DNN models on the end devices and the high-complexity DNN models on edge/cloud servers. The basic idea of CacheNet is inspired by the caching approach in computer architecture, where the computer elements (e.g., register, cache, RAM) are separated by memory hierarchy based on response time. Compared to the memory hierarchy in a computer that only stores data, CacheNet stores DNN models. Especially, CacheNet generates multiple small sub-models. Then, each sub-model captures a partition of the knowledge represented by the large DNN model instead of training a single large-scale DNN model. Experiments results based on CIFAR-10 [200] and FVG dataset prove that CacheNet is 58 - 217% faster than the benchmark approaches.

C. ML/DL Tasks across Edge Devices, Edge Servers, and Cloud Servers

1) ML/DL Tasks Offloading and Partitioning: The most widely approach used to efficiently enable ML/DL inferences on MEC is tasks offloading and partitioning among the participating nodes in MEC (i.e., edge devices, edge servers, and cloud servers). As shown in Fig. 14, the joint DNN tasks partitioning and offloading approach first determines

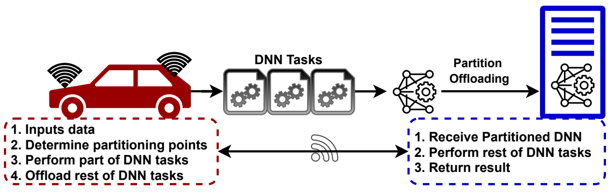

DNN Tasks
Partition Offloading
1. Inputs data
2. Determine partitioning points
3. Perform part of DNN tasks
4. Offload rest of DNN tasks
1. Receive Partitioned DNN
2. Perform rest of DNN tasks
3. Return result

Fig. 14: Joint DNN tasks partitioning and offloading

the partitioning points and perform some part of the DNN tasks on the edge device (vehicle). The rest of the DNN task is offloaded to the edge server, which will identify the partitioning point, processes it, and returns the result to the vehicle.

The authors of [201] propose a joint multi-device DNN partitioning, offloading, and allocation mechanism. The main objective is to minimize the maximum DNN execution latency among all the edge devices, and therefore reduce the global latency. In the proposed approach, multiple devices cooperate with an edge server, and each device can make a DNN partitioning decision on its own DNN model. The offloaded DNN layers of edge device are executed on the edge server to accelerate the learning process. To partition the DNN tasks, the logical layers are divided into two types, one type that will be executed locally on the device, and another type that will be executed on the edge server. Only intermediate outputs are offloaded from an edge device to an edge server. The offloading decision at the device side is modeled as an integer variable $S_i \in \{0, 1, 2..., k\}$, denoting that the layer 0 to S_i are executed locally on the edge device while the rest of the layers are offloaded to the edge server. The simulation results prove that the proposed approach outperforms the local-execution method by 67.6% and the edge-only-execution scheme by 41% when there are enough resources and bandwidth.

In [202], the authors present an efficient energy-aware DNN offloading algorithm for intelligent IoT systems in cloud-edge environments. The main objective is to minimize the overall energy consumption of all participating nodes (i.e., devices, edge servers, and cloud servers). The offloading algorithm called SPSO is based on two meta-heuristic methods, namely the particle swarm optimization (PSO) and the Genetic Algorithm (GA). Layers partitioning is also introduced to reduce the encoding dimension and the execution time. The DNN partitioning is performed as follows: firstly, the branches in a DNN are divided into isolated modules. Then, for each module, the actual layer is initialized as the start layer. After that, every two adjacent layers are checked based on a defined fitness function. Once a partition point is found, the actual layer is updated to the next layer. This process is repeated until the last two adjacent layers are checked. Finally, the layers between each two adjacent partition points are merged to form a deployment unit.

Qadeer and Lee [203] investigate the computation and wireless resource allocation problem in edge-based cloud (with limited computational resources) and traditional cloud environments. Specially, they propose an algorithm based on a deep deterministic policy gradient with a pruning approach. The learning process is performed on the edge-based cloud to achieve dynamic resource allocation for the edge devices. Experimental results show that the proposed algorithm achieves up to 55% reduction in terms of operational cost, and up to 86.5% reduction in rejection rate on average. Furthermore, the proposed algorithm obtains up to 115% gain in terms of QoE. Another offloading approach in edge-cloud networks is proposed in [204] with the objective to minimize the task processing delay. To achieve this, a DRL method, in particular, a DQN is used to learn the optimal offloading schemes through exploration and exploitation processes. Shi et al., [205] investigate the trade-off between the inference latency and data privacy during the DNN tasks partitioning in a MEC network.

2) Federated Learning: Federated Learning (FL) is another powerful enabling approach for ML/DL models training and inference in MEC while guaranteeing data privacy. FL is a decentralized ML approach that allows smart devices to cooperatively train a shared learning model while keeping the raw data on their devices, thus protecting their privacy. In FL, the edge devices (e.g., mobile phones) use their local dataset to collaboratively train and learn the model required by an FL server without sending their data. Then, they send the model updated to the server for aggregation. These steps are repeated several times until an expected accuracy is obtained.

Although, the learning techniques such as RL and DRL can effectively address the resource allocation problem in wireless networks, their learning speed may be slower in complex networks. Indeed, in a complex wireless network, a new learning policy should be updated for a newly-arrived system because of the lack of network adaptability [206]. To solve this issue, recent efforts focus on the application of FL for DRL. For instance, the authors of [206] present a federated learning framework for resource allocation in wireless networks with the objective to accelerate the learning speed. In the proposed FL framework, the edge server and the distributed systems share a DRL model that represents the learning policy as shown in Fig. 15. Also, each system k learns its local policy model W_k by a DRL method. The edge server updates a central policy model W_{cs} by aggregating the learned policy models from the systems W_k's. Then, the updated central policy model of the server is redistributed to each distributed system which replaces its local policy model with the central one. By repeating this process, the FL mechanism can accelerate the learning speed of the resource allocation policy and ensure adaptability to newly-arrived systems because of the central policy model at the edge server.

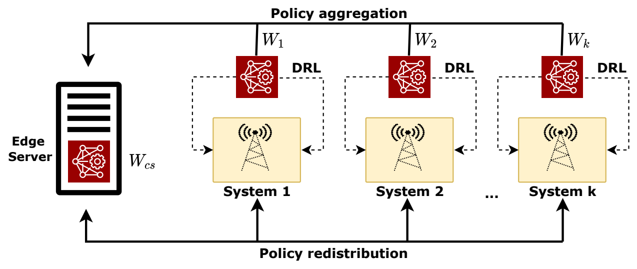


Fig. 15: Federated learning for DRL [208]

The authors of [207] also apply FL to accelerate the training of the DRL agents for task offloading in MEC. The FL recursively selects a random set of IoT devices to first download the parameters of the DRL agents from the edge server, then uses their own data to perform the training process on the downloaded model. Finally, it uploads only the updated model parameters of the DRL agent to the edge server for model aggregation. This FL-based approach enables resource-limited devices to learn a shared DRL agent without centralizing the training data.

In [209], the authors propose a task offloading and resource allocation algorithm based on federated learning and DRL. The algorithm distributed the DRL tasks from the edge servers to the edge devices for training, which improve the accuracy. The proposed algorithm called FDOR has four components: offloading action generation, offloading policies updating, DNN model aggregation, and adaptive learning rate approach. The authors of [210] present a gradient-descent-based federated learning approach, which comprises two main phases: local update and global aggregation. In the local update phase, each edge node performs gradient descent to locally adjust the model parameters with the objective to minimize the loss function defined on its own raw data. In the aggregation step, the model parameters obtained at each edge node are sent to an aggregator, which regroups the parameters and sends back an updated parameter to each edge node for the next round of iteration.

The authors of [208] investigate resource optimization techniques in FL. Especially, an NN-aware resource management mechanism based on FL is proposed, where the sub-networks of the global model are assigned to the mobile clients based on their local resources. They also present a use case of FL, namely virtual keyboard application (VKA) used by Google AI group [211]. The VKA on mobile devices uses a natural language processing (NLP) DL model to predict a word. The

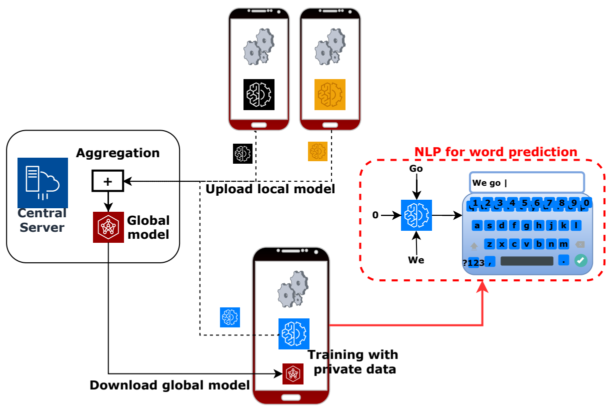


Fig. 16: Federated learning: virtual keyboard use case [208]

process of VKA with FL is described as follows (Fig. 16): firstly, the mobile device uploads the global model from the central server, then enhances it by training on local data. After that, the updated local model is sent to the central server through a secure encrypted communication link. Next, the global model is improved by aggregating the updated local models to the central server. This iteration of model training and aggregation is repeated until the global model converges.

In [34], the authors present a comprehensible survey of federated learning in MEC networks. They discuss several aspects of FL including fundamentals of FL, applications of FL in MEC, and challenges.

D. Lessons Learned

In this section, we discuss lessons learned from enabling techniques for ML/DL tasks (i.e., training and inference). Table V summarizes the potential techniques that enable to quickly perform ML/DL tasks in MEC. As shown in Table V, the majority of the enabling approaches for ML/DL tasks in MEC are evaluated using software stacks such Raspeberry Pi, PyTorch, TensorFlow, and QKeras. ImageNet [219] dataset is widely used to evaluate the performance of the proposed ML/DL approaches because it contains enough labeled high-resolution images belonging to different categories, which facilitate the training of large CNNs [78].

Although there are several hardware and software stacks (e.g., edge tensor processing unit, FPGAs, Tensorflow Lite, Core ML, and EdgeML [72]) for ML and DL techniques, there is a need to evaluate these tools and propose a standard testbed for ML/DL tasks in MEC. A standard testbed consisting of neural network models, datasets, networking models, IoT devices, edge/cloud servers, and resource allocation mechanisms is perceptible in enabling ML and DL tasks in MEC.

In MEC, DNN training and inference are mainly performed across edge devices, edge servers, and cloud servers. The main reason is that DNN training and inference require not only more computing power, but also less latency that a single computing node (e.g., edge only, edge server only, or cloud server only) cannot provide.

There are mainly three types of strategies to enable fully DNN models training and inference in MEC. The first one is model compression, which reduces the DNN model inference times. This approach has advantages in MEC by minimizing the latency of DNN models running on resource-constrained edge devices. However, the drawback of model compression techniques is that they do not guarantee to maintain the original model accuracy. In other words, the compressed DNN model can obtain an accuracy less than that of the original model. The second approach to facilitate the execution of ML/DL tasks in MEC is hardware architecture designing. The main challenge of these architectures is that they are designed for a specific DNN model as explained in the previous section. Therefore, it is challenging to design an architecture that can support different DNN models.

The third approach allowing DNN training and inference in MEC is ML/DL tasks offloading from the resource-constrained device to powerful servers (e.g, edge servers or cloud servers). Generally, for latency-aware applications, the tasks are offloaded to other edge devices or to the edge servers, which are close to the specific edge device. It is important to notice that the models compression method can be jointly used with the offloading technique to meet QoS requirements as in [228], [229]. The main challenge of offloading methods is to decide which part of the DNN tasks should be offloaded and where to offload them. Scheduling the offloaded DNN tasks is also challenging during DNN inference in MEC. Therefore, it is crucial to investigate the joint DNN tasks offloading and scheduling in MEC. Finally, a natural question that arises is how ML/DL methods can improve the tasks offloading and scheduling (i.e., resource allocation) in MEC. The answer to this question is the purpose of the next sections.

V. ML AND DL FOR RESOURCE ALLOCATION IN MEC

A. Motivation Example

To illustrate the importance of ML/DL methods for resource allocation in MEC, let's consider that we would want to write an algorithm **to offload "untrusted tasks" of an application to an edge server**. By using traditional programming approaches (Fig. 17), we would first look at what an "untrusted task" looks like. We might observe that some characters or words (such as "3pay", "money", "free", "bank") tend to appear several times. Perhaps we would also remark some patterns in the mobile user's id, name, location, and so on. Then, we would write a discovery algorithm for each of these patterns that would flag a task as "untrusted task" if a number of these patterns are found. Finally, we would test the algorithm and repeat the above steps until it is good enough. Since this problem is not trivial, the algorithm will likely become enough hard to maintain.

In contrast to the traditional approaches, an ML-based "untrusted tasks" offloading algorithm automatically learns which characters and words are good predictors of "untrusted tasks" by discovering unusually frequent patterns of characters or words (Fig. 18). Also, the ML-based mechanism is much easier to maintain and more accurate. Indeed, if malicious senders remark that all their tasks containing the word "3pay" are blocked, they might start writing "pay" instead. An "untrusted tasks" offloading algorithm using traditional approaches would

TABLE V: Summary of Works focused on Enabling ML/DL Tasks in MEC

Computation Layer	Ref.	NN Models	Techniques	Application	Simulation Tools	Dataset	Key Metric
On Edge Devices	2021, [179]	LSTM	Quantization	Image classification	PyTorch	WikiText-2 [180], MNIST, Bar Craw	Validation loss
	2019, [186]	MobileNet-V1	Quantization		Xilinx Zynq-7020 FPGA and Xilinx VU9P	ImageNet	Latency, energy, and model size
	2021, [181]	MLPs, ResNet-50 and ResNet-34, ResNet-20, and EfficientNet-B0	Hardware accelerator	Hand-gesture recognition, image classification	System Verilog + Mentor Graphics Simulator	Google speech sommands, MNIST, and CIFAR-10	Throughput and power consumption
	2019, [167]	LeNet [212] and VGG16 [213]	Knowledge distillation and Auto-encoder	Image classification	TensorFlow, Javascript, and Flask library	MNIST, QuickDraw, and CelebA [214]	Interpret and diagnostic image classifiers with high accuracy
	2022, [168]	LeNet5, AlexNet, MobileNet-V3, VGG-19, ResNet-18, ResNet-32	Knowledge distillation	-	TensorFlow	CIFAR-10	Privacy preserving
	2018, [87]	Inception, ResNet, and MobileNet	KNN	Image classification	Jetson TX2 embedded DL	ImageNet ILSVRC 2012 validation dataset	Minimize inference time while meeting user QoS
	2020, [169]	VT-CNN2	Pruning based on activation maximization	Automatic modulation classification	TensorFlow	RML2016.10a dataset [215]	Model acceleration
	2019, [174]	Conv-2 [216], VGG-11 [79], ResNet-18 [217]	Pruning	Image classification	PyTorch, Raspberry Pi	FEMNIST [216], CIFAR-10 [218], and ImageNet [219]	Accuracy and time trade-of
	2021, [177]	Fully-connected NN [220]	Quantization	Particle identification	QKeras [221], AutoQKeras, and hls4ml	Hls4ml lhc jet dataset [222]	Energy minimization and accuracy
	2019, [178]	Resnet-v1-50	Quantization		TensorFlow	Cifar10	Accuracy
On Edge Servers	2019, [192]	-	Offloading	-	MATLAB	Numerical	Energy consumption, delay cost, and QoR
	2022, [193]	AlexNet, VGG, and ResNet	Offloading	Image recognition	Python and Caffe2	unknown	Response time
	2020, [194]	VGGNET-16	Offloading + Partitioning		NVIDIA Tesla V100 GPU	Numerical	End-to-End Inference delay
	2020, [223]	MobileNet, Inception, and ResNet	Graph-based partition	Unknown	Scikit-learn, Caffe	Mobility datasets [224] and CRAWDAD [225]	Throughput
	2019, [196]	, VGG16, VGG13, ALEXNET, and LENET	a Offloading + partitioning (MILP)	Unknown	Orange Pi Win Plus, MATLAB	Numerical	Delay
	2018, [198]	RESNET-152	Caching	Object classification	NVIDIA Tegra TK1 + Raspberry Pi3	Youtube-Objects video	Latency
	[199], 2020	CacheNet, Shake-Shake and ResNet	Caching	-	TensorFlow, TensorFlow Lite and NCNN	CIFAR-10 and Frontal View Gait (FVG) dataset [226]	Latency
Across devices, edge and cloud servers	2021, [201]	MobilenetV2 and VGG19	Partitioning	Unknown	Raspberry Pis, NVIDIA Jetson Nanos	Unknown	Latency
	2022, [202]	AlexNet, VGG19, GoogleNet, and ResNet101	Offloading	Unknown	Python	Unknown	Energy Consumption
	2022, [203]	Conv1D and gated recurrent unit (GRU)	Resource allocation + Pruning		Python	Numerical	Operational cost, rejection rate, and QoE
	2022, [227]	MobileNetV1	Reinforcement learning	Image classification	AWS a1.medium, AWS a1.large, ARM-core, and ARM-NN SDK	Numerical	Response time and Accuracy
	2022, [209]	Fully connected DNN	Offloading, FL	Wireless communication	PyTorch	Numerical	convergence speed, execution delay
	2019, [210]	Squared-SVM, Linear regression, K-means, and CNN	Federated learning	Unknown	Raspberry Pi	MNIST, SGD, DGD, CIFAR-10	Minimize the loss function

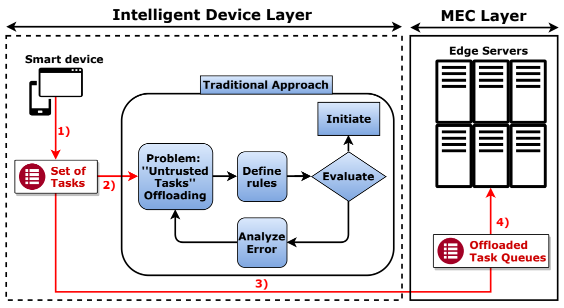


Fig. 17: Traditional approach for untrusted task offloading.

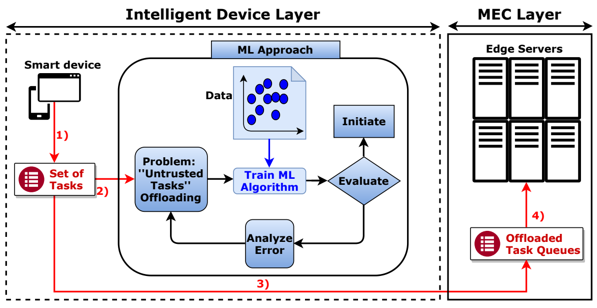


Fig. 18: ML approach for untrusted task offloading.

need to be updated to flag the word "pay". If malicious senders keep working around the traditional offloading algorithm, we will need to keep writing new rules forever. In contrast, an ML-based "untrusted tasks" offloading algorithm will automatically observe that "pay" has become unusually frequent in "untrusted tasks" flagged, and it starts flagging them without writing new rules (Fig. 19).

Now, suppose that we want to **schedule the offloaded tasks to the MEC servers** (Fig. 20). To provide a more realistic scenario, we consider the problem as a flow shop scheduling problem, which is widely used in manufacturing [230]. Also, recently, many manufacturing factories are integrated edge computing and AI to solve the daily job scheduling problem. Hence, this tutorial will help both generalists and specialists to understand how ML and DL can be used to solve the resource allocation problem in MEC. The flow shop scheduling problem can be formulated as follows: given a set of J jobs, the tasks of each job need to be processed by m machines (servers) in a specific order. In our case, we have one job with 3 tasks that need to be scheduled on 3 servers (Table VI). The main challenge in flow shop scheduling is to find the optimal sequence in which the tasks will be executed. For instance, if

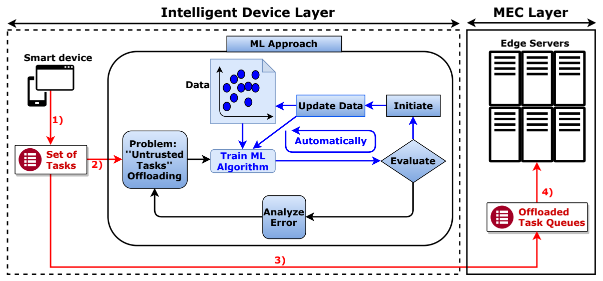


Fig. 19: ML approach for untrusted task offloading: automatically adapting to environment.

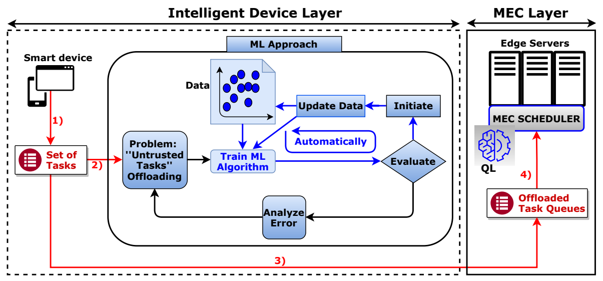


Fig. 20: QL-based offloaded task scheduling.

Algorithm 1 Q-Learning Algorithm

Input: Random state;
Output: Q-Table;

```
1: Initialize $Q^{\pi}(s,a)$ arbitrary for all (state, action) pairs;
2: repeat(for each episode $\tau$)
3:     Initialize state $s$;
4:     repeat(each step of episode)
5:         Choose action $a$ from $s$ using policy derived from $Q^{\pi}$ (e.g., $\epsilon$-greedy, $\epsilon \in [0,1]$);
6:         Take action $a$, observe $r$, $s'$;
7:         Update $Q^{\pi}(s,a)$ using (2);
8:         $s \leftarrow s'$;
9:     until $s$ is terminal
10: until convergence ($|Q^{\pi}_{\tau}(s,a) - Q^{\pi}_{\tau-1}(s,a)| < \omega$); where $\omega$ is the infinitesimal value.
```

we have n tasks, we will have $n!$ feasible sequences. Here, we aim to find the optimal sequence using a Q-learning method described in Algorithm 1, where $Q^{\pi}(s,a)$ is the value-function to maximize. At each step t, the $Q^{\pi}(s,a)$ value function is iteratively updated using Bellman equation (2).

$$Q^{\pi}_{t+1}(s,a) = Q^{\pi}_{t}(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q^{\pi}_{t}(s',a') - Q^{\pi}_{t}(s,a)], \tag{2}$$

where α is the learning rate, $R(s,a)$ is the reward for taking action a at state s. The $\max_{a'} Q^{\pi}_{t}(s',a')$ value is the maximum expected future reward for any action in state s'. These maximum values are stored in a table commonly called Q-table. The Q-table has one row and one column for each possible state and action, respectively.

The parameters used for this tutorial are given in Table VII. The values of the Q-table are initially set to zero. The learning steps to find the optimal sequence are described as follows:

TABLE VI: Processing time of 3 tasks on 3 machines

Task	M_1	M_1	M_3
T_1	4	3	1
T_2	3	5	5
T_3	3	2	4

First episode:

- Current state: $[T_1, T_2, T_3]$;
- Select a random action: T_2;
- Sequence : T_2;
- Reward: $R([T_1,T_2,T_3],T_2) = \frac{1}{makespan(T_2)} = \frac{1}{3+5+5} = 0.0769$;
- Calculate the new $Q^{\pi}(s,a)$: $Q^{\pi}([T_1,T_2,T_3],T_2)$

$$= Q^{\pi}([T_1,T_2,T_3],T_2) + \alpha\Big[R([T_1,T_2,T_3],T_2) + \gamma \max[Q^{\pi}([T_1,T_3],T_1), Q^{\pi}([T_1,T_3],T_3)] - Q^{\pi}([T_1,T_2,T_3],T_2)\Big]$$

$$= 0 + 0.4 \times \Big[0.0769 + 0.8 \times \max[0,0] - 0\Big] = 0.03076;$$

- The Q-table is updated (Table VIII index 1), and we process the current state $[T_1, T_3]$;
- Select a random action: T_3;
- Sequence: $[T_2, T_3]$;
- Reward: $R([T_1,T_3],T_3) = \frac{1}{makespan([T_2,T_3])} = \frac{1}{17} = 0.0588$;
- Calculate the new $Q^{\pi}(s,a)$: $Q^{\pi}([T_1,T_3],T_3)$

$$= Q^{\pi}([T_1,T_3],T_3) + \alpha\Big[R([T_1,T_3],T_3) + \gamma \max[Q^{\pi}([T_1],T_1)] - Q^{\pi}([T_1,T_3],T_3)\Big]$$

$$= 0 + 0.4 \times \Big[0.0588 + 0.8 \times 0 - 0\Big] = 0.0235;$$

- The Q-table is updated (Table VIII index 3), and we process the current state $[T_1]$;
- Select a random action: T_1;
- Sequence: $[T_2, T_3, T_1]$;
- Reward: $R([T_1],T_1) = \frac{1}{makespan([T_2,T_3,T_1])} = \frac{1}{18} = 0.0555$;
- Calculate the new $Q^{\pi}(s,a)$: $Q^{\pi}([T_1],T_1)$

$$= Q^{\pi}([[T_1],T_1) + \alpha\Big[R([T_1],T_1) + \gamma \max[Q^{\pi}([T_1],\emptyset)] - Q^{\pi}([T_1],T_1)\Big]$$

$$= 0 + 0.4 \times \Big[0.0555 + 0.8 \times 0 - 0\Big] = 0.0222;$$

- The Q-table is updated (Table VIII index 5), then the next episode starts. This process is repeated until the convergence of the algorithm. Fig. 21 shows the makespan obtained for each episode. After a certain number of episodes we obtain the best sequence that is $[T_3, T_2, T_1]$.

TABLE VII: Q-Learning parameters

Name	Notation	Value
Greedy policy	ϵ	ϵ= 0.2
Learning rate	α	α= 0.4
Discount factor	γ	γ= 0.8
Reward	R	$R = \frac{1}{makespan}$

TABLE VIII: Q-table of three tasks

		Actions		
Index	States	T_1	T_2	T_3
1	T_1, T_2, T_3	0	**0.03076**[a]	0
2	T_1, T_2	0	0	0
3	T_1, T_3	0	0	**0.0235**
4	T_2, T_3	0	0	0
5	T_1	**0.0222**	0	0
6	T_2	0	0	0
7	T_3	0	0	0
8	$\emptyset$	0	0	0

[a]Bold values indicate which action was selected in a specific state

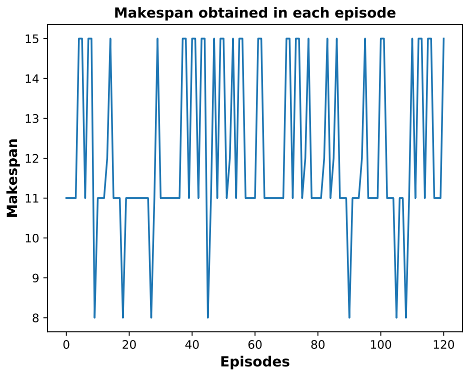


Fig. 21: Q-Learning for flow shop scheduling in MEC.

B. Machine Learning for Resource Allocation in MEC

ML has been widely used for solving resource allocation problems in MEC networks. For instance, the authors of [231] propose an ML-based algorithm for resource allocation in edge and IoT networks. Especially, they use a clustering approach to categorize the IoT users into clusters. The cluster with the highest priority offloads and executes its tasks at the edge server, while the cluster with the lowest priority executes its tasks locally. For the other clusters, the task offloading decision is modeled by a Markov Decision Process (MDP), and a DQN is used to train the optimal policy. In the same spirit, the authors of [232] use a Q-learning (QL) approach for cross-layer resource allocation in cognitive radio networks (CRN). Since the QL approach leads to a long convergence time for large state and action spaces, they also use a DQN to address this challenge. ML is also used for spectrum sharing cellular networks [233] and for resource trading in fog environment [234]. Furthermore ML is applied for resource allocation in various networking systems such as IoT [235], vehicular communication networks [236], and SDN [237].

C. Deep Learning for Resource Allocation in MEC

Deep learning has also been used for resource allocation in recent wireless communication technology such as 5G. For instance, the authors of [247] propose a deep learning method (LSTM) for resource allocation in 5G wireless networks with the objective to minimize the energy usage of the remote radio heads while considering the QoS constraints of the users. In [253], the authors present a DRL-based resource allocation and power management in the cloud. Using an autoencoder and a weight sharing structure, the convergence speed is accelerated, while the LSTM and RL are used to manage the server power usage. The simulation results based on Google cluster traces show that the proposed mechanism minimizes the power consumption and energy usage compared to the traditional schemes. Similarly, the authors of [254] propose a DRL-based resource allocation algorithm for V2V communications with the objective to minimize the interference of the V2V links to the V2I links while satisfying the latency constraints on V2V links.

In summary, several ML and DL methods have been used for resource allocation in MEC networks. Table IX summarizes potential ML and DL methods for resource allocation and their advantages and disadvantages in MEC.

In the next sections, we provide an in-depth survey of recent works in this area from three perspectives: ML/DL-based methods for task offloading (Section VI), ML/DL-based methods for task scheduling (Section VII), and ML/DL-based methods for joint resource allocation (Section VIII).

VI. ML/DL-Based Methods for Task Offloading in MEC

In this section, we survey state-of-the-art ML/DL-based methods for task offloading in MEC. We classify the research in this area into works focused on the minimization of latency (VI-A), minimization of the energy consumption while satisfying a QoS metric such as slowdown, response time, execution delay (VI-B), finding a proper trade-off between multiple QoS (VI-C), and finding a desirable trade-off between both the privacy protection, the execution delay, and the energy consumption (VI-D).

A. Minimization of Latency

The main objective of [12] is to find a proper offloading mechanism, which can minimize the latency of the application. To this end, the authors propose a DRL-based offloading algorithm, which can effectively learn the offloading policy represented by a Seq2Seq neural network. The algorithm has three main phases. In the first phase, the priority of each task is calculated based on the EFT, which is the summation of the running cost of the current task and the maximal EFT of the previous tasks. Then, the tasks are sorted in the ascending order of EFT. In the second phase, the offloading policy is designed. In this phase, the sorted task is converted into embedding vectors, that is 1, 0, where 1 indicates that

TABLE IX: Potential ML and DL Methods for Resource Allocation (RA) in MEC

Method	**Main Idea**	**Advantage**	**Disadvantage**	**Potential Applications in RA**
ML Methods				
DT	Use a tree-like prediction model to learn from training datasets represented as a set of rules and decision branches.	Easy to understand and has logarithmic time complexity.	Local optimal decision is obtained at each leaf. Hence, it cannot guarantee globally optimal decision tree.	Task orchestration [238], task offloading with the objective to minimize the energy consumption of mobile devices [239].
RF	Randomly use many decision trees to acquire robust prediction made by the individual trees to increase the overall accuracy	Low prediction errors even for noisy workloads	Time consuming	Modulation and coding prediction in 5G [240].
SVM	SVM learns the optimal separating hyperplane between the two classes of training dataset in the feature space [241]	Less computational resources	Monopolization issue: node with more assets have more power than others.	Cooperative offloading in balloon networks [242].
QL	QL learns an action-value function $Q(s;a)$ for each state-action pair. The $Q(s;a)$ value is the cumulative return value obtained after executing action a in state s.	Simple to implement since it can be formulated by a single equation	Slow convergence rate since all Q-values must converge before attaining the optimal policy.	Accurate task ordering and assignment [243], [244], Adaptive resource allocation [245].
Bagging	Bagging is an ensemble machine learning technique that aggregates multiple prediction models to reduce the variance.	Take the advantage of different learners to avoid the overfitting of models.	Computationally expensive and loss of interpretability	Prediction task's attributes [246]
DL Methods				
Seq2Seq	Train a model to generate a sequence of items from one domain to a sequence of items in another domain.	Easy to convert a resource allocation decision into a sequence prediction process.	Challenging with large input sequences since the output sequence is heavily related to the hidden state in the final output.	Binary offloading decision by converting the task into embedding vectors ”0” and ”1”, where ”1” indicates that the task is offloaded and ”0” indicates the task is executed locally [12].
LSTM	LSTM is a type of RNN that can learn long-term dependencies in prediction problems.	Easy to capture and store long-term dependencies.	Hard to remember a decision after a long period. It also requires high memory bandwidth due to the linear layers in each cell.	resource allocation for TV Service in 5G wireless network [247].
Autoencoder (AE)	AE is a type of NN used to train a compressed representation of raw data. It has two main layers, namely the encoder that converts the input into the code, and a decoder that uses the code to reconstruct the initial input.	Good for feature extraction	Computationally time consumption	Keep track of the long-term dependencies that exist between the tasks' requirements and the VMs' specifications [248].
CNN	CNN has two main steps: feature extraction, which performs a series of convolutions and pooling operations for detecting features, and the classification phase, which assigns a probability to the object for prediction. [66]	CNN can strongly reduce the complexity of the network model and the weigh sharing.	Require large training datasets. CNN cannot encode the position of IoT devices.	Feature extraction of the task queue [244], power allocation for secure industrial IoT [249].
RNN	RNN save the output of a particular layer and feed the result back to the input to predict the output of the layer.	RNN can remember information over time.	Very difficult to train an RNN to solve problems with long-term temporal dependencies because of the vanishing gradient problem [66].	Interference pattern prediction and power allocation in D2D network [250].
DRL	DRL allows an agent to approximate its policy and get the optimal solution without requiring any prior training data knowledge.	High convergence rate compared to RL. DRL enable IoT device to learn optimal policies(e.g., channel selection) without knowing the environment (e.g., wireless channel information or mobility pattern).	A DRL agent requires a million of training data samples to learn optimal policies.	Big data task scheduling [251], Optimal task ordering [251], decision-making process [252]

TABLE X: Average latency (ms) of the DRL-based offloading method proposed in [12]

Tasks (n)	**DRL-based**	**HEFT-based**	**Round-Robin-based**
n=10	**349.7**	365.05	436.53
n=25	**790.79**	833.90	948.71
n=40	**1185.59**	1262.06	1372.12

the task is offloaded on the MEC server, and 0 denotes that the task is executed locally on the device. The proximal policy optimization (PPO) [255] method is used to train the Seq2Seq neural network. Finally, the task is executed according to the results of the offloading decision of the second phase. The experimental results demonstrated that the proposed algorithm achieves lower latency than the HEFT-based and Round-Robin-based algorithms. For instance, when the number of tasks is equal to 10, the proposed DRL-based offloading algorithm obtains a latency of 349.7, while the latency of the HEFT-based and RR-based are 365.05 and 436.53, respectively (see Table X). The drawback of the proposed algorithm is that it has weak adaptability to a new environment. Therefore, it requires full retraining to update policy for the new environment, which is time-consuming.

To solve this issue, the authors of [256] introduce a task offloading scheme based on meta-reinforcement learning (MRL) called MRLCO that can learn a meta-offloading strategy for all IoT devices and rapidly obtain the proper policy for each IoT device. The main objective of [256] is to find an efficient offloading mechanism that minimizes the total latency. Compared to the previous work that has weak adaptability for a new environment, the method in [256] can quickly adapt to new environments. The proposed offloading strategy is also modeled as a seq2seq neural network. The novelty of the MRLCO algorithm is its ability to achieve fast adaptation to new MEC environments with a small number of gradient updates and samples. The experimental results show that the MRLCO algorithm obtains the lowest latency compared to the scheme proposed in [12].

Another idea aiming at minimization of the latency is introduced in [257]. When compared to the previous studies, the

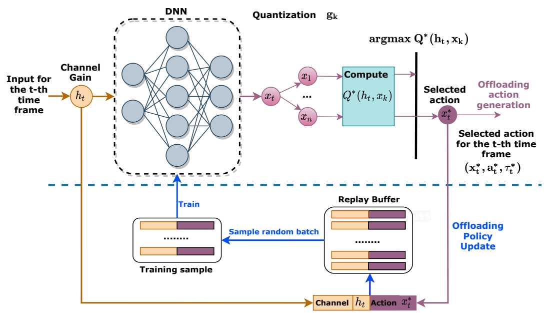


Fig. 22: Illustration of DRL-based offloading scheme proposed in [257]

authors of [257] also maximize the weighted sum computation rate of all offloaded devices. The main objective of the paper is to find a mechanism that can achieve optimal task offloading policies and wireless resource allocations while considering the time-varying wireless channel conditions. To this end, the authors propose a DRL-based online offloading (DROO) algorithm with DNN that learns the offloading decisions from the experience. The DROO algorithm has two main phases: offloading action generation and offloading policy updating (see Fig. 22). The generation of the offloading action uses a DNN, which is represented by its embedded parameters θ. In the tth time slot, the DNN takes the channel gain h_t as input and gives an offloading action x_t ($x_t \in [0, 1]$) as output based on its current offloading policy π_{θ_t}. The action is then quantified into K binary offloading actions, and the best action x_t^* is selected based on the feasible computation rate. The solution for h_t is the output (x_t^*, a_t^*, τ_t^*), which guarantees that all the constraints are satisfied. Finally, the network takes the action x_t^*, obtains a reward $Q^*(h_t, x_t^*)$, and adds the obtained state-action pair (h_t, x_t^*) to the replay memory. In the policy action updating phase, a group of training samples is extracted from the memory to train the DNN, which appropriately updates its offloading policy from θ_t to θ_{t+1}. The new policy θ_{t+1} is used in the next time slot to generate offloading decision x_{t+1}^* based on the new channel h_{t+1}. Simulation results show that the DROO algorithm minimizes the latency by more than an order of magnitude compared to the benchmark approaches.

In [258], the authors investigate the task offloaded scheduling problem in edge computing to minimize the long-term cost defined as a trade-off between task latency and energy consumption. To this end, a novel DRL-based algorithm called DRLOSM is proposed to solve the problem, which is modeled as MDP. To improve the training efficiency, a proximal policy optimization (PPO) algorithm is introduced, which is a policy gradient scheme with good stability and reliability [259]. Additionally, a convolutional neural network (CNN) is integrated with the DNN scheme to better extract the features of the task queue. The DNN scheme estimates the offloading scheduling policy. In the training phase of the proposed algorithm, two DNNs are initialized one with the parameter θ_{old} for sampling $\pi_{\theta_{old}}$, and another with θ for optimizing π_θ. In the sampling phase, N trajectories are sampled following the policy $\pi_{\theta_{old}}$. To obtain an efficient training model, the Generalized Advantage Estimations (GAEs) for each time step in each trajectory are calculated in advance. Then, the sampled data are cached for the optimization phase. In the optimization phase, the value θ of the policy π_θ is updated for a certain number of epochs. After that, the policy π_θ is enhanced in each epoch by applying stochastic gradient ascend on the cached data. Finally, the sampling policy $\pi_{\theta_{old}}$ is updated with π_θ, the cached data is dropped, and the next iteration continues.

B. Minimization of Energy Consumption while Satisfying a QoS Metric

The minimization of the energy consumption while satisfying the slowdown is the main objective of [260]. The authors propose a DRL-based offloading algorithm with DNN as an approximator function that achieves the desired trade-off between energy consumption and slowdown. The simulation results show that the algorithm can learn the approximate optimal task offloading policy after several training rounds and it is better than the baseline algorithms in terms of mean energy consumption and mean slowdown.

The goal to minimize the energy consumption and response time is pursued in [261]. This is accomplished by an algorithm based on classification and regression tree (CART), which finds the optimal offloading policy according to the device' QoS such as capacity, availability, authentication, speed, and cost. The proposed algorithm is compared with the First Fit (FF) and local processing policy (LPL) (i.e., execution is done always locally on the device). The simulation results show that the proposed CART-based offloading policy is better than FF and LPL methods in terms of energy consumption and response time. For instance, the proposed algorithm is better than FF method by 44 % in terms of energy consumption. In terms of response time, the proposed algorithm outperforms FF method by 50%.

Compared to the above studies, which assume that the wireless channel state information (CSI) is static and known by mobile users, the authors of [262] present an offloading mechanism in a static and time-varying MEC system. Under a static channel state information, the offloading problem is formulated as a noncooperative exact potential game (EPG), where each mobile user offloads its computation tasks to the edge server to selfishly maximize its processed CPU cycles in each time slot and reduce its energy consumption. Then, under time-variant and unknown channel state information, a payoff game theory is adopted, which is also proved to be an EPG. In this case, the offloading problem is solved using a Q-learning method and a best-response approach [263], which helps mobile users to adapt their offloading policies to dynamic wireless environments. The simulation results prove that the proposed mechanism outperforms the local processing and random approaches, and achieves at least 87.87% average payoff compared to the full CSI case.

In [264], the authors propose a novel architecture called Space-Air-GroundEdge (SAGE) to offload task-intensive services in maritime MEC networks to jointly minimize latency and energy consumption. The offloading problem is formulated as Multi-Armed Bandits (MABs) learning problem. The

Algorithm 2 D-DRL Algorithm Proposed in [266]

```
1: Input: computational tasks, states, and actions;
2: Output: Optimal offloading policy;
3: for user $n \in N$ do
4:     Initialize the learning parameters;
5: end for
6: for time slot $k \in 1, 2, ...$ do
7:     for user $n \in N$ do
8:         Observe $b_n^k$ and update its observation $o_n^{k-1}$ into $o_n^k$;    ▷ $b_n^k$ is the user $n$ bandwidth at time $k$
9:         n;
10:        Store $(o_n^{k-1}, x_n^{k-1}, o_n^k, r_n^{k-1})$ into the replay buffer;
11:        Input $o_n^k$ into actor network $\pi_{\theta_n}$ and determine the size of input data $x_n^k$ uploaded to the edge server.
12:        Compute its reward $r_n^k = u_n(x_n^k, x_{-n}^k)$
13:    end for
14:    if $k\%D == 0$ then
15:        for $m \in 1, 2, ...M$ do
16:            for user $n \in N$ do
17:                Compute the gradients;
18:                Update user $n$ actor and critic networks in every time slots $D$;
19:            end for
20:        end for
21:        Clear the replay buffer;
22:    end if
23: end for
```

MAB problem is a classic reinforcement learning problem of exploration and exploitation [265]. To solve the problem, an Upper Bound of Confidence interval (UCB) based algorithm is proposed. Firstly, the proposed algorithm analyzes the historical records and the associated reward and cost values of the maritime IoT devices to find the edge sever that will be selected. Then, the algorithm updates the confidence interval of each server forward degree and the number of times that the server is selected. Finally, the edge server with the lowest regret value that satisfies the offloading QoS is selected as the optimal server. The simulation results prove that the proposed algorithm outperforms the traditional UCB and ϵ-greedy algorithms under various conditions in terms of latency and energy consumption.

The authors of [266] propose an algorithm called D-DRL (Algorithm 2), which is based on DRL and differential neural computer (DNC) for decentralized computation offloading in edge computing to achieve the optimal offloading policy. DNC is a particular RNN with internal memory, which is capable of training and remembering the previous hidden states of the inputs data. Hence, with DNC, not only the learning process is accelerated but also the agent can continue learning policy when the network is uncertain and time-varying. Compared to previous approaches which assume that all users should share their information, in [266], the users have the possibility to share or not their QoS's information. The offloading problem is formulated as a "multi-agent partially observable Markov decision process (POMDP)" and an algorithm based on DRL is proposed to solve the problem. This approach enables each user to determine the approximately optimal computation offloading scheme directly from game history without any preliminary information about other users. The D-DRL has the following components: actor-critic network, replay buffer, policy optimizer, and actor-critic laws updating. Firstly, each user initializes the parameters of their actor and critic networks (line 2). At each time slot, each user notices its bandwidth and updates its parameters (lines 4-6). Then, each user inserts its previous observation, new observation, strategy, and reward into its replay buffer (line 8). After that, the current notice of each user is considered as the input of its actor-network and a part of its input data is uploaded to the edge sever based on the output of the actor-network. Next, each user calculates its reward and updates its actor and critic networks for each time slot. The data stored in their replay buffers are taken as one mini-batch, and they update the actor and critic networks by calculating the gradients (lines 13-16). Finally, each user optimizes its actor and critic networks by mini-batch stochastic gradient and clears its replay buffer. Simulation results show that the D-DRL algorithm outperforms the baselines algorithms. For instance, the D-DRL takes about 8000 time slots to converge to the stable state, while the MAPPO algorithm [267] and MAA2C algorithm [268] take about 14000 time slots and 18000 time slots, respectively.

C. Trade-Off Between Execution Delay, task drops, queueing delay, failure penalty, and cost

While previous surveyed works focused on a single objective or bi-objective offloading problems, the authors of [269] present a trade-off analysis between the execution delay, task drops (i.e. once the task queue is full), queueing delay, failure penalty, and execution cost for the offloading decision. Also, compared to [262] which considers only the CSI, in [269], the authors consider both the task queue state, the energy queue state, and the CSI between the mobile users and the base stations. The task offloading decision is formulated as MDP. Two DDQN learning algorithms are proposed to train the optimal task offloading policy without any prior knowledge of CSI. Simulation results show that the proposed algorithms called "DARLING" and "Deep-SARL" are better than the baselines in terms of long-term utility. The proposed algorithms obtain an optimal trade-off among the task execution delay, the task drops, the task queueing delay, the task failure penalty, and the MEC service cost compared to the baselines. Furthermore, the Deep-SARL algorithm outperforms the DARLING algorithm by considering the additive structure of the utility function.

The drawback of both previous surveyed works is that the offloading decision scheme does not take into account the application security and privacy requirements. Hence, the next section surveys studies that address the security-awareness and privacy-awareness task offloading problems.

D. Trade-Off Between Privacy, Execution Delay, and Energy Consumption

The protection of the mobile's user location and usage pattern privacy in MEC networks while minimizing the delay and energy consumption of the offloaded tasks is the

Algorithm 3 RL-Based Privacy-Aware Offloading Algorithm for healthcare IoT Devices Proposed in [270]

```
1:  **Input:** Healthcare data, states, and actions;
2:  **Output:** Optimal offloading policy;
3:  Initialize the learning parameters $\alpha$, $\gamma$, and $\delta$;
4:  HotBooting process;
5:  **for** time slot $k \in 1, 2, 3...$ **do**
6:      Observe $C_1^{(k)}, C_0^{(k)} b^{(k)}$;
7:      Compute $\chi^{(k)}$, $h^{(k)}$, and $\rho^{(k)}$;
8:      $s^{(k)} = \{C_1^k, \chi^k, h^k, \rho^k, C_0^k, b^k\}$;
9:      Divide the healthcare IoT data with size of $C_0^k + C_1^k$ into N equivalent tasks;
10:     Choose $x^{(k)} = [x_k^0, x_k^1]$ using $\epsilon$-greedy policy;
11:     Offload $x_1^{(k)}(C_0^k + C_1^k$ healthcare data to the edge device, process $x_0^{(k)}(C_0^k + C_1^k$ of the data locally, and store the rest in the replay buffer;
12:     Evaluate the achieved privacy, total energy consumption, and computation latency;
13: **end for**
```

main objective in [271]. To achieve this goal, the authors formulate the optimization problem as a constrained Markov decision process (CMDP) and solved it using Q-learning and Lagrangian approaches. The proposed scheme is described as follows: at each time slot n, the mobile user takes an action $a_n = arg \min_{a \in \mathcal{A}} Q_n(s_n, a)$ based on the observation of the buffer and channel state s_n. Then, it observes the next state s_{n+1}. Next, it updates the $Q_n(s, a)$ of the Q-function $Q(s, a)$ and the estimate value of the Lagrangian multiplier. The numerical experiments prove that when the privacy level of the mobile user is equal to 0.6, the extra delay and energy costs caused by the baseline algorithm are 100%, while the delay and energy costs incurred by the proposed algorithm are 45%. However, the proposed algorithm suffers from a slow learning convergence.

To overcome the drawback of the above study, the authors of [270] propose a novel offloading algorithm with the objective to protect both the user location privacy and the usage pattern privacy while minimizing the computation latency and the energy consumption cost of healthcare IoT devices. This is accomplished by an RL-based algorithm that achieves the optimal offloading decision while protecting the user's privacy and minimizing both the energy consumption and the computation latency of the IoT devices. To accelerate the learning process a transfer learning technique, i.e., PDS (Post Decision State) scheme [272] is used. The proposed algorithm is described as follows (see Algorithm 3): at each time slot k, the IoT device observes the current state $s^{(k)}$ that depends on the new generated healthcare data size and its priority, the state of the current radio channel, the current renewable energy generated, the previous computation records in the buffer, and battery level of the IoT device. After evaluating the healthcare data of size C_1^k, the IoT device calculates the priority of the healthcare data denoted by χ^k, and evaluates the channel power gain denoted by h^k. Based on historical data and the previous offloading experiences, the IoT device computes the amount of the harvested energy denoted by ρ^k and evaluates the current battery level b^k. Hence, the state is selected as $s^{(k)} = \{C_1^k, \chi^k, h^k, \rho^k, C_0^k, b^k\}$, where C_0^k indicates the healthcare data records in the buffer. Finally, the offloading policy $x^{(k)} = [x_0^k, x_1^k]$ is chosen with probability $(1 - \epsilon)$ and other feasible offloading decisions are randomly selected with a small probability. The simulation results show that the proposed RL-based mechanism converges faster than the CMDP-based method proposed in [271]. For instance, when the required privacy of the user is equal to 11, the proposed algorithm saves 40% of time slots to meet this requirement compared to the CMDP-based approach. Compared with the CMDP-based method, the proposed algorithm also saves 9.63% of the energy consumption cost and reduces 68.79% of the computation latency while improving 36.63% of the privacy level for a larger time slot.

Another idea aiming at maximizing the user privacy level while minimizing the offloading latency and energy consumption in a MEC-based blockchain network is introduced in [273]. When compared to the previous work, the authors of [273] address not only the computation tasks offloading problem but also the blockchain mining tasks offloading issue. In the MEC-based blockchain network, each miner (i.e, mobile user) can offload its IoT processing and mining tasks to the MEC server. To achieve this goal, the authors propose a DRL-based algorithm that can efficiently find optimal offloading actions without any prior knowledge of the environment dynamics. The authors first propose a QL-based offloading algorithm that enables miners to obtain optimal offloading decisions (see Algorithm 4). An ϵ-greedy policy is used to balance the exploration/exploitation and to update the QL. At each time step, the miner selects an offloading action based on the blockchain state and evaluates the privacy value and system costs (lines 6-12). After performing each action, the miner goes to the next step and updates the new state (lines 13-15). This process is repeated until the optimal offloading policy is obtained. To overcome the slow convergence of the QL-based algorithm for a larger state-action, the authors propose another algorithm called DRLO that uses a DNN to approximate the Q-values instead of using the traditional Q-table. The simulation results show that the DRLO algorithm obtains a better long-term reward compared to the conventional QL-based method. For instance, when the trade-off value is equal to 0.8, the experiment results proves that the convergence of the DRLO algorithm is 9% higher than that of the QL-based scheme. The experiment results also prove that the DRLO algorithm outperforms the benchmark algorithms in terms of offloading cost, energy consumption, and privacy-preserving. For example, for mining 100 kb blockchain transactions, the privacy level of the DRLO algorithm is 5.5% better than the conventional QL-based approach and 13.4% better than the CMDP-based method proposed in [271].

E. Lessons Learned from ML/DL-based Task Offloading

A summary of studies on ML and DL for task offloading in MEC is illustrated in Table XI. The majority of ML and DL-based offloading decision algorithms aim to minimize latency

TABLE XI: Summary of Studies on ML and DL for Task Offloading in MEC

Ref.	Learning Type	Algorithm	Mathematical Model	Simulation Tools	Optimization Criteria
2020, [266]	Deep RL	DRL	Game + POMDP	Python-Based	Minimize the energy consumption and delay cost.
2019 [274]	SL	SVM	-	-	Maximize the throughput.
2020, [273]	Deep RL	QL+DQN	MDP	-	Maximize the user privacy level while minimizing the offloading latency and energy consumption of the mobile user device.
2018, [262]	RL	QL	Game theory	MATLAB	Minimize the energy of the mobile device.
2019, [269]	Deep RL	DDQN	MDP	Tensorflow	Trade-off analysis between the execution delay, task drops, queueing delay, failure penalty, and execution cost.
2020, [261]	RL	CART	MDP	Cloudsim	Minimize the energy consumption and response time.
2017, [271]	RL	QL	CMDP	-	Protect the mobile' user location and usage pattern privacy while minimizing the delay and energy consumption of the offloaded tasks.
2019, [270]	RL	QL+TL+PDS	MDP	-	Protect both the user location privacy and the usage pattern privacy while minimizing the computation latency and the energy consumption cost of healthcare IoT devices.
2019, [260]	RL	DRL + DNN	-	-	Minimize the energy consumption while satisfying the slowdown.
2019, [12]	DRL	Seq2Seq	MDP	-	Minimize the latency of the application.
2020, [258]	Deep RL	PPO + CNN + DNN	MDP	Tensorflow	Minimize the long-term cost defined as a trade-off between task latency and energy consumption.
2021, [256]	MRL	Seq2Seq NN	MDP	Tensorflow	Minimizes the total latency.
2020, [257]	Deep RL	DNN	MIP non-convex	Tensorflow	Minimize the latency while maximizing the weighted sum computation rate of all offloaded devices.

Algorithm 4 QL-Based Task Offloading Algorithm for Mobile Blockchain Networks Proposed in [273]

```
1: Input: blockchain transaction data, channel states, and actions;
2: Output: Optimal offloading policy;
3: Initialize the learning parameters $\alpha$, $\gamma$, and $\epsilon$;
4: Initialize $Q(s,a)$ value function, and Q-table $Q(s^0,a^0)$
5: Set t=1;
6: while $t \leq T$ do
7:     Observe blockchain transaction $(D_1^t, D_0^t)$;
8:     Estimate the channel gain state $g^t$, and set $s^t = \{D_1^t, D_0^t, g^t\}$;
9:     Select a random action $a^t$ with probability $\epsilon$, otherwise $a^t = argmaxQ(s^t, a, \theta)$;
10:    Offload data processing task $x^t(D_1^t + D_0^t)$ to edge server or execute $x^t(D_1^t + D_0^t)$ locally on mobile device;
11:    Calculate the reward $r^t = P^t(s,a) + R_n^{mining} + C^t(s,a)$;    $\triangleright$ $R_n^{mining}$ mining reward of miner $n$
12:    Estimate the privacy level $P^t(s,a)$ and the system cost $C^t(s,a)$;
13:    set $s^{t+1} = \{D_1^{t+1}, D_0^{t+1}, g^{t+1}\}$;
14:    update $Q(s^t, a^t)$ using Bellman equation (2);
15:    $t \leftarrow t+1$;
16: end while
```

or to find a proper trade-off between the energy consumption at the IoT device and the latency. Besides, most of the studies use the DRL method or a combination of ML and DL methods to solve the offloading problem because these approaches converge faster than the standard reinforcement learning.

From the surveyed papers focused on task offloading, we learned the following key facts:

- **Designing an efficient task offloading scheme is a crucial challenge in MEC systems.** The main reason is that, although tasks offloading to MEC servers minimizes the energy consumption of the IoT device since the execution does not have to be done locally, it also incurs additional execution delays, including the time to send the offloaded tasks to the MEC server, the time to execute the offloaded tasks on the server, and the time to send the execution results back to the IoT device. Especially, there is a trade-off between the energy consumption of the IoT device and the execution delay. For this reason, the majority of the surveyed works aim to find a mechanism that minimizes the energy consumption of the IoT device while satisfying the execution delay.
- **The performance of an ML/DL-based offloading method is strongly related to the methods used to train the optimal offloading policy.** The reason is that the training method determines the long-term reward (i.e., the objective function). Also, the works that used two training methods (e.g., [76]) outperform those that use a single training method (e.g., [274]) because with two training models, each model can train its optimal offloading policy to cooperatively optimize the long-term reward.
- **For delay-sensitive and real-time applications, ML/DL-based methods outperform the traditional methods for task offloading in MEC.** Traditional offloading techniques (e.g., approximation, heuristic, and meta-heuristic) are computationally expensive. Therefore, they are not suitable for delay-sensitive and real-time IoT applications (e.g., online gaming, virtual reality, and augmented reality). Also, compared to traditional offloading approaches, the ML/DL-based offloading methods have the ability to predict both the delay sensitivities of each application and the MEC servers' computation capabilities by learning from historical data and previous offloading experiences. For instance, the DRL-based offloading algorithm outperforms the heuristic HEFT-based and Round-Robin-based algorithms in terms of latency [12].
- **The DRL-based offloading algorithm converges faster than the traditional RL-based algorithm when the number of states and actions spaces is large**. This is due to the fact that the traditional RL method uses Q-

table to approximate the Q-values, while the DRL method uses neural networks (e.g., DNN or DQN) to approximate the Q-values [273]. Therefore, for large state and action spaces, the RL method leads to a long convergence time compared to the DRL-based method.

- **Transfer learning (TL) can significantly accelerate the training process and improve the performance of a DL-based offloading algorithm**. The reason is that the TL method can learn from an existing offloading model to solve a similar offloading problem [275]. In the training phase of TL, the offloading model is divided into multiple sub-offloading models, where each sub-offloading model can learn from other similar sub-offloading models to accelerate its learning process. Transfer learning requires only small targeted training datasets to obtain high accuracy [276]. Therefore, TL reduces the re-training time and consumes less amount of bandwidth.

Algorithm 5 The QL-HEFT Algorithm [244]

```
Input: Tasks graphs;
Output: Makespan;
1: Initialize $Q$-table, learning rate, and discount factor;
2: Calculate the immediate reward ($rank_u$);
3: repeat(for each episode)
4:     Randomly select an entry task as current task $T_c$;
5:     repeat(for each episode)
6:         Randomly choose a legal task (expected $T_c$) as the next task $T_{next}$;
7:         Update $Q(T_c, T_{next})$ by Eq. (2)
8:         $T_c \leftarrow T_{next}$;
9:     until $T_{next}$ is terminal;
10:    Obtain a task order according to the updated $Q$-table;
11:    Allocate a processor to each task;
12:    Obtain the Makespan;
13: until convergence (makespan no longer changes);
```

VII. ML/DL-Based Methods for Task Scheduling in MEC

In this section, we survey state-of-the-art ML and DL methods for task scheduling in MEC. The main objective of the works focused on the ML/DL-based methods for task scheduling is to minimize the execution time (VII-A), to minimize energy consumption while satisfying response time (VII-B), to find a proper trade-off between response time and utilization costs of MEC resources (VII-C), and to minimize the communication cost (VII-D).

A. Minimization of Execution Time

One of the advantages of ML and DL techniques for task scheduling problems is their ability to minimize the application's tasks execution time by predicting the computation capabilities of the target environment. To minimize the total execution time (or makespan), the authors of [277] propose a two-phase learning-based algorithm, which schedules the data-intensive tasks into a cluster of resources. Firstly, the algorithm selects the cluster containing the nodes with the lowest data communication cost. Then, an adaptive assignment policy based on Q-learning is used to select a proper node in the selected cluster. The adaptive assignment policy comprises one global broker agent that selects the cluster with minimum data communication cost, and several local broker agents within the selected clusters which select the proper node that will execute the task. The experimental results show that the Q-learning-based scheduling algorithm outperforms the HCS [278] algorithm in terms of makespan for different workloads configuration.

The minimization of the execution time is also the main goal in [244]. This is achieved by an algorithm called QL-HEFT that combines the Q-learning algorithm and HEFT algorithm. The QL-HEFT algorithm has two main phases: a task prioritization phase for obtaining an optimal task order using the Q-learning method as described in Algorithm 5, and a processor selection phase for selecting the suitable server that will execute the ready task. Firstly, the QL-HEFT algorithm uses the Q-learning method to obtain an optimal task order by sorting the original order, i.e., the HEFT-based task order. A random selection approach is used to establish enough training in the state s and transfer to the state s' (line 4). After the agent selects an action a in the state s, the QL approach is used to update the corresponding Q-value $Q(s, a)$ in a Q-table (line 7). The iteration process continues until a final Q-table is obtained. After obtaining the optimal task order, the task with the highest priority is executed on the server that achieves its minimum earliest finished time (line 11). The QL-HEFT algorithm is compared with HEFT_D, HEFT_U, and CPOP algorithms using CloudSim. The experimental results show that the QL-HEFT algorithm outperforms these three algorithms in terms of makespan and speedup. The drawback of the QL-HEFT algorithm is that it requires a specific learning rate and discount factor to converge to the optimal solution, which is time-consuming.

While the authors of [244] use the QL method to prioritize the tasks, in [279], the authors propose a novel scheduling method called DeepSoCS that can learn the best task prioritization by using DRL. DeepSoCS comprises two main phases, namely the task ordering phase and the server selection phase. DeepSoCS employs the server selection strategy of the HEFT algorithm, i.e., the server that achieves the earliest finish time executes the ready task. In the task prioritization phase, two Message Passing Neural Networks (MPNNs) capture the task's features (i.e., task dependencies and communication costs). The first one denoted as g_1 takes a DAG as an input and calculates the task's features by considering information about its neighbor edges. The second one denoted as g_2 captures all task's features and takes jobs as inputs, then computes local and global features of the jobs. Next, tasks' features are constructed by the forward task information. Finally, a task is selected by using a conventional policy network which is defined as the probability of taking an action a in a state s. The experimental results show that the DeepSoCS outperforms the heuristic HEFT algorithm in terms of makespan and robustness under noise condition.

Another approach aiming at minimization of the task execution time is proposed in [280]. Compared to the previous work, the authors of [280] propose a scheduling algorithm based on Deep-Q-Network (DQN) called RLTS. The algorithm has three main phases including task ordering, state transition, and task scheduling training process. The application's tasks are ordered based on the upward rank value (as in HEFT [281]). In the state transition phase, after the action a is executed, i.e., after a task is allocated on the server a, the state space (i.e., the task's start time and finished time) change from the present state s to the next state s'. Then the reward at state s is calculated. In the task scheduling process, the DQN-based scheduling algorithm uses neural networks to calculate the action-value function rather than updating the Q-table by the Q-learning approach. Two neural networks namely target Q-network and evaluated Q-network are used. The output of each neural network is the probability to choose an action. A started episode with an empty state space represents that the tasks are assigned on the servers. For each task, an action (server) is selected by ϵ-greed policy, which selects the optimal server with probability $1 - \epsilon$ and selects a random server with probability ϵ. Then, the reward and the next state are obtained. The simulation results show that the proposed RLTS algorithm outperforms the HEFT and PEFT [109] algorithms in terms of makespan. For instance, when the number of tasks is equal to 100, the RLTS algorithm outperforms the HEFT algorithm and PEFT algorithm by 20% and 16%, respectively.

In [251], the authors propose a new cluster scheduler framework called "Spear" for dependent tasks scheduling to minimize the makespan. For this purpose, Spear uses "Monte Carlo Tree Search (MCTS)" and "Deep Reinforcement Learning (DRL)". The MCTS is a search approach for sequential decision-making problems where the result is a win or loss. MCTS keeps a state tree, where the nodes represent a path of actions and the edges indicate individual actions. In Spear, a neural network is first designed to represent a scheduling policy. Then, the network is trained to minimize the makespan. In the DRL model, a neural network takes as input a list of ready tasks and the state of the cluster, then provides a scheduling action. The DRL method comprises three phases: state, action, and reward. In the state phase, Spear first adopts the b-level approach as a task prioritization scheme. Since the b-level only obtains information about the execution time of the tasks, the b-load, which accumulates the load of the tasks is also considered. The b-load is defined as the product of the task execution time and the resource demands. In the action phase, once the DRL agent is called for an action, it draws one action from the actions space. Then, the tasks in the cluster are executed for one time slot. Finally, in the last phase, the agent received -1 reward each time the processing action is selected in order to obtain the scheduling length. The total accumulated reward is equal to the negative of the given graph makespan. The experimental results show that Spear outperforms the Graphene [282] algorithm in terms of makespan. For instance, on average, Spear surpasses Graphene by 90% in terms of makespan. In terms of algorithms running time Spear outperforms Graphene algorithm since the average running time of Spear is 500 seconds and the one of Graphene is 1000 seconds.

The drawback of the above-surveyed scheduling algorithms is that they consider only the makespan as a performance metric, but other metrics such as cost and robustness need to be considered to improve the convergence of the algorithm. For this reason, the authors of [283] present a multi-agent Deep-Q-network (DQN) algorithm with reinforcement learning for multi-objective application scheduling to minimize both the execution time and cost. To achieve this goal, a Markov Game Model (MGM) is used, which considers the scheduling goals as two agents. Each agent determines its actions based on a neural network by mixing the output of the neural network with random actions to sample its training data.

The authors of [245] address not only the makespan minimization but also the response time and robustness of the algorithm. To this end, they propose an online Q-learning scheduling algorithm that can adapt the task arrival and execution processes automatically. The scheduling problem is formulated as an MDP and solved by QL approach. The algorithm has three main phases: initialization, action selection for a task, and learning. The first stage aims to initialize the "discounted accumulative reward (DAR)" of each action. DAR is an objective function used in MDP problem which considers forthcoming allocations into account. The second phase aims to allocate a task to the processing units. If the task is explored with an ϵ-greedy exploration probability value p_e, the allocation is performed randomly. Otherwise, the allocation is performed using the learning scheme. Finally, the learning phase determines the estimated expected reward of an allocation and the estimated expected reward of a certain task type. The experimental results show that the online Q-learning scheduling algorithm outperforms the Min–Min, Min–Max, Suffrage, and ECT algorithms in terms of response time and robustness.

B. *Minimization of Energy Consumption While Satisfying Response Time*

The scheduling and resource allocation problem in IoT devices is addressed in [284] with the objective to minimize the energy consumption at the IoT device while satisfying the response time. The authors formulate the problem as an MDP problem and solved it using a reinforcement learning approach. Particularly, they present an algorithm called DA-DRLS that takes the advantage of DNN and RL. The algorithm can rapidly adapt to a sudden change in the IoT device requirements by continuously observing "demand drift" and dynamically updating the scheduling policy. Simulation results show that the proposed algorithm outperforms the benchmark algorithms in terms of energy consumption (reduces by 36.7%) and response time (reduces by 59.7%). It also increases the resource utilization by at least 10.4% compared to the benchmark methods.

The optimization of both the energy consumption and application performance is the main objective in [285]. This is pursued by an algorithm called ISVM-Q that combines the Q-learning and SVM methods for scheduling the application tasks in the wireless sensor network (WSN). The WSN comprises n sensor nodes which are always learning until the ending learning condition is reached. The SVM model is improved by using a linear basis function to get the current state of the system, and the Q-value (i.e., the output of the improved SVM model) is obtained by estimating the regression model. Then, the algorithm selects an action that corresponds to the current estimated Q-value. Finally, the selected action is executed to get the corresponding reward, and the Q-value is updated. Experiment results show that the ISVM-Q algorithm outperforms the baseline algorithms in terms of energy consumption and application performance. For instance, the performance of ISVM-Q is 0.55% higher than that of the IQ algorithm proposed in [286].

The minimization of the execution time of the task and energy consumption is the main goal in [287]. The authors investigate the task allocation problem for Multi-task Transfer Learning (MTL) in edge computing. The main idea consists of dividing a machine learning-based application into multiple machine learning tasks, where each task can learn from other tasks to improve its performance. To achieve this goal, they propose a "Data-driven Cooperative Task Allocation" approach based on clustered reinforcement learning (CRL) and SVM models. They formulated the problem as an MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \lambda >$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, r, and λ denote the states; actions; transition probability reward function, and discount factor for future rewards, respectively. The CRL model makes allocation decisions based on the relation between the observations of the current environment and those previously seen. Concerning the SVM model, it predicts the task importance and dynamically adjusts the CRL model allocation decisions based on real-time data. The experiment results show that the proposed mechanism reduces the processing time by 3.24 times and saves energy consumption by 48.4%.

C. Trade-off Between Response Time and Resource Utilization Costs

In [288], the authors address the task scheduling problem with the objective to minimize the task response time while maximizing the VM resources utilization. This is achieved by an algorithm based on queueing theory [289] and reinforcement learning schemes. Since the M/M/1 queueing system is hard to analyze in a cloud/edge environment (due to the complexity and dynamicity of the system), the authors propose a new queueing model divided into three submodels, namely, the task scheduling submodel (TSSM), task execution submodel (TESM), and task transmission submodel (TTSM). The TSSM receives user tasks (requests) and pushes them into a global queue $\mathbf{G}_q$ using FIFO approach. Then, the task dispatcher implemented in TSSM pushes user tasks to the corresponding buffer queue of the TESM module, which is then assigned to the VMs resided in TESM. Next, the VM submits the execution results to TTSM module. Finally, the TTSM module transmits the execution results to the requesting users. The main objective of the task dispatcher is to schedule the tasks in $\mathbf{G}_q$ to VM resources. This is accomplished by a Q-Learning approach which continuously interacts with the environment to obtain the optimal policy. The proposed method called $\mathcal{Q}$-sch is compared with the random scheduling scheme (i.e., users tasks are randomly scheduled to the VMs), equal scheduling scheme (i.e., users tasks are ordered then scheduled to the VMs), and mix-scheduling scheme (i.e., a user's task is first randomly scheduled to a VM. If the remaining buffer memory of the VM is equal to zero, then rescheduling the task to the VMs by maximizing the rest buffer memory). In terms of response time, the experiment results show that the $\mathcal{Q}$-sch algorithm outperforms the random scheduling, mix-scheduling, and equal scheduling schemes by 1.85%, 2.45%, and 4%, respectively, when the task arrival rate varies. The results also prove that the $\mathcal{Q}$-sch algorithm improves the resource utilization while reducing the average response time. The drawback of the proposed approach is that it used a random scheduler, i.e., the first Come First Serve (FCFS) approach to process user requests. However, the FCFS scheme is inappropriate in a cloud/edge environment because the performance of a cloud/edge platform depends on how well it can satisfy users' requirements specified in the SLA.

When compared to the previous work, the objective of [248] is to minimize not only the CPU utilization cost but also the Ram utilization cost. The authors use a DRL approach to model the task scheduling problem. In this approach, the *state* represents an "offloaded task", the *action* represents a VM, and the *reward* represents the task execution cost on the VM. A manager node observes the resource utilization of each VM to populate the reward values of each action. Then, the manager uses the DRL algorithm to assign the new arrival task to the optimal VM. The proposed DRL approach integrates a Long Short-Term Memory (LSTM) layer, which keeps track of the long-term dependencies that exist between the tasks' requirements and VMs' specifications. The DRL approach integrated with LSTM helps to improve the decision-making process and reduce the runtime of the DRL by storing the long-term dependencies into the LSTM's memory cell. The experiments based on real-world datasets show that the proposed method outperforms the Shortest Job First (SJF) algorithm by 28.8%, the RR algorithm by 14%, and the PSO algorithm by 14% in terms of CPU utilization. The results also prove that the proposed algorithm minimizes the RAM utilization cost by 31.25%, 25%, and 18.78% compared to the SJF, RR, and PSO, respectively.

While energy consumption is not considered in the two previous works, the main objective of [243] is to minimize the energy consumption while reducing task response time and maximizing CPU utilization. To this end, the authors propose a Q-learning-based task scheduling algorithm in data centers that has two main phases: task dispatcher for assigning user requests to servers in the data center and task scheduling phase for ordering the assigned tasks in each server. The dispatcher (i.e., global scheduler) is implemented at the data center level, where an M/M/S queueing system (i.e., a single queue with more than one parallel server) is used to reduce the energy consumption in the data center. In this M/M/S queueing model, the task arrivals are supposed to follow the Poisson distribution model, and the server's service time follows a negative exponential distribution. It is also supposed that the servers provide the same type of services. The task dispatcher distributes requests to the servers uniformly when all the servers in the data center are treated as a whole. In this way, the average task response time is shorter compared to M/M/1 queueing system [290], which considers the servers in the data center as independent. After a task is assigned to a server, it is first pushed onto a queue Q, waiting to be scheduled to a specific VM. A time window is used to determine when to process the tasks in Q. For each time window, the scheduler first removes all the tasks from the queue Q, then assigns each task to a specific VM by pushing it to the buffer queue implemented in that VM. A dynamic task prioritization approach based on task laxity and task lifetime is used to order all the tasks in Q, and a Q-learning-based approach is employed to reward task assignments to simultaneously optimize the task response time and CPU utilization. The Q-learning-based scheduler continuously learns from the environment to obtain the optimal policy that meets this objective. That is, if a is an action (i.e., assigning the task t to VM_i), the scheduler will receive a reward of 1 in case that (i) VM_i can satisfy the deadline of the task t; (ii) VM_i is the VM with the smallest waiting time; and (iii) VM_i gives the best CPU utilization for the task t.

Another idea aiming at minimization of the MEC resource utilization cost is introduced in [76]. Compared to the previous studies, the authors of [76] also minimize the rate of missed tasks. To this end, the authors propose a two-phase scheduling algorithm based on deep learning. The first phase of the algorithm determines the location to execute the task (i.e., on edge or cloud). The second phase of the algorithm assigns the task on the edge or cloud according to the execution location determined at the first phase. To determine the location of the task execution, i.e., whether the IoT task will be executed on the edge or cloud nodes, three different clustering methods based on self-organizing map method are used, namely "task clustering by self-organizing map (SOM)", "task clustering by hierarchical self-organizing map (H-SOM)", and "task clustering by autoencoder and self-organizing map (AE-SOM)". The SOM method directly sends the parameter of the tasks for clustering. Concerning the H-SOM, it is used in every layer. In AE-SOM method, the encoder extracts the task features (e.g., task type, task priority, task privacy, task execution time, etc.) before clustering. After the clustering step, the Earliest-Deadline-First (EDF) algorithm is used to schedule the tasks in each cluster to the edge or cloud nodes. The experimental results show that the AE-SOM method has a better result in terms of missed tasks rate. For instance, the AE-SOM method is better than SOM method by 3.19% and H-SOM method by 4.23% in terms of missed task rate. In terms of memory and bandwidth costs, the AE-SOM method has 305.75 (G$) less than the SOM and H-SOM average cost.

D. Minimization of Communication Cost

In [291], the authors investigate a machine learning approach to predict the tasks execution time and the scheduling failure probability. A Seq2Seq NN and RL approaches are used to improve the scheduling decisions. The proposed method can identify a near-optimal scheduling decision by presenting all possible scheduling choices to the system. During the task execution phase, a vector is generated for each possible scheduling decision based on the task models and data sources. The generated vector is then passed to the neural network for a scheduling decision. This process is repeated until all tasks are completed. The RL approach is compared with Round Robin (RR), First Come First Served (FCFS), and Random (RN). The authors use three bioinformatic applications to evaluate the performance of the algorithms, namely pangenome analysis, phylogenetic profiling, and metagenomics. The experimental results based on the pangenome analysis application prove that the RL approach is 20% faster than the FCFS algorithm. The RL approach also has 50% lower network transfers than the FCFS algorithm and achieves almost zero failed tasks. For the phylogenetic profiling application, the RL approach is 25% faster than the RR algorithm in terms of execution time. Finally, for the metagenomics application, the RL approach is 16% faster than the FCFS algorithm and reduces by 24% the network transfers cost compared to the FCFS algorithm. In sum, the experimental results show that the scheduling based on the RL approach outperforms the RR, FCFS, and RN algorithms in terms of execution time and network traffic cost.

In [292], the authors address the problem of N independent wireless links scheduling in a dense wireless network with the objective to maximize the sum-rate. The wireless link scheduling problem consists of selecting a subset of links in any given transmission time slot with the goal to maximize a certain network service function of the reached long-term average rates. To achieve this goal, the authors propose a deep learning-based algorithm, which trains the optimal scheduling policy based on the geographical locations of the neighboring transmitters and receivers. Particularly, they propose a DNN with three main phases: convolution phase, fully connected phase, and feedback connection phase. The convolution aims to capture the interference patterns of neighboring links using geographic location information. In this stage, a spatial convolution approach is used to estimate the total interference generated by the transmitter and the receiver. Concerning the fully connected phase, it is responsible for capturing the non-linear functional allocation of the optimized schedule. It takes a vector of links as input and gives an output $x_i \in [0, 1]$, where $x_i = 1$ if the link is scheduled, and $x_i = 0$ otherwise. Finally, the feedback connection phase is proposed between each iteration of the neural network to update the optimization's state. This phase is described as follows: after the execution of $(t-1)$th of the convolution phase and the fully connected phase, the vector $x_i \in [0, 1]$ which represents the activation status of the links is obtained. Then, a new convolution stage and fully connected stage begin with density grids so that the activation status for all N wireless links are updated for the next interference estimations. Finally, the scheduling decisions of the N links are determined after a fixed number of neural network iterations by quantifying the x vector from the last iteration into binary values.

E. Lessons Learned From ML/DL-based Task Scheduling in MEC

A summary of studies on ML and DL for task scheduling in MEC is illustrated in Table XII. The majority of ML/DL-based

TABLE XII: Summary of Studies on ML and DL for Task Scheduling in MEC

Ref.	Learning Type	Algorithm	Mathematical Model	Simulation Tools	Optimization Criteria
2020, [293]	SL	CNN	-	-	Security-awareness of deep network embedded devices
2019, [291]	SL+RL	Seq2Seq	-	Okeanos Cloud	Minimize the execution time, network traffic cost, and failure rate
2021, [76]	USL + SL	SOM+AE	-	MATLAB	Minimize the rate of missed tasks and cost
2020, [280]	RL	DQN	-	-	Minimize the execution time and running time
2019, [292]	USL	DNN	-	-	Maximize the sum-rate
2020, [287]	RL + SL	SVM + Clustered RL	MDP	AIOPS	Minimize the execution time and energy consumption
2019, [285]	RL + SL	QL + SVM	-	-	Minimize the energy consumption while maximizing the application performance
2014, [245]	RL	QL	MDP	MATLAB	Minimize the response time
2018, [277]	RL	QL	-	OptorSim	Minimize the execution time
2016, [294]	RL	QL	-	CloudSim	Minimize the execution time
2020, [243]	RL	QL	Queueing theory	CloudSim	Minimize the energy consumption while reducing task response time and maximizing CPU utilization
2019, [284]	RL	DRL	MDP	Python-based	Minimize the energy consumption at the IoT device while satisfying the response time
2019, [283]	RL	DQN	Markov Game	EC2 Cloud	Minimize both the execution time and execution cost.
2020, [279]	RL	DRL	POMDP	DS3	Minimize the execution time
2019, [251]	RL	DRL	MCTS	Python(Theano)	Minimize the execution time and running time
2019, [248]	RL+SL	DRL+LSTM	MDP	Python-Based	Minimize the CPU utilization cost and Ram utilization cost
2019, [244]	RL	Q-Learning	-	CloudSim	Minimize the execution time.

scheduling methods aim to minimize the application execution time or to find a trade-off between response time and resource utilization costs. Besides, most of the studies used QL and DRL methods to solve the task scheduling problem.

From the surveyed papers focused on ML/DL for task scheduling in MEC, we learned the following main lessons:

- **For a large dataset, ML/DL-based scheduling methods outperform the traditional heuristic scheduling methods**. The first reason is that the performance of a learning method strongly depends on the amount and quality of the training dataset. The second reason is that the ML/DL methods can predict and extract the task's features (e.g., task dependencies, communication costs, and QoS requirements) by learning from previous experiences [295]. Therefore, ML/DL methods can provide the optimal task priority order, which can significantly reduce the scheduling length (makespan). Also, DL methods, in particular, differential neural computer (DNC) is capable of training and remembering previous hidden states of inputs data. Hence, DNC can accelerate the learning process and enable the agent to continue learning policy when the network is uncertain and time-varying.
- **Dividing an ML-based application into multiple machine learning tasks can significantly improve the scheduling decision**. In this way, each ML-task can learn from each other to improve its reward. For instance, by dividing the scheduling problem of multi-task transfer learning in MEC into clustered reinforcement learning (CRL) and SVM models, the processing time and energy consumption is reduced [287]. Furthermore, the CRL model can make scheduling decisions based on the relation between each cluster, while the SVM model can predict the task's features and dynamically adjusts the CRL model scheduling decisions.
- **The efficiency of ML/DL-based scheduling schemes is strongly related to the type of algorithms used for both task prioritization and server selection**. For instance, when a Q-Learning method is used to calculate the task priority and the earliest finish time (EFT) is used to select a server for a task, the convergence rate of the algorithm is lower as proved in [244]. This is due to the fact that the Q-learning algorithm uses a Q-table to calculate the action-value function, in which each Q-value must converge before attaining the optimal policy. On the other hand, when the upward rank value [281], i.e., the critical path approach is used to calculate the task priority, and a DQN method is used to select a server for a task, the convergence rate is higher [280]. Therefore, it is crucial to choose the appropriate ML/DL method for both task prioritization and server selection.
- **The running time of DRL-based mechanisms for task scheduling can be significantly reduced by integrating the LSTM method in the learning process.** The reason is that LSTM can predict the long-term dependencies that exist between the task's QoS and the specifications of the MEC servers by exploring its memory cell, in which the previous long-term dependencies have been stored. This is also proved in [296], where an automated task scheduling method based on DRL and LSTM has been proposed to minimize both the CPU utilization cost and Ram utilization cost.

VIII. ML AND DL-BASED METHODS FOR JOINT RESOURCE ALLOCATION IN MEC

While the above surveyed works address the task offloading problem and the task scheduling problem separately, this section surveys current works addressing the joint resource allocation (i.e., joint task offloading and scheduling) problem in MEC using ML/DL techniques. The offloading decision directly impacts the scheduling strategy because the offloaded tasks have different QoS requirements (e.g, latency, security, execution time, etc.), and the MEC resources are limited [4]. Furthermore, the task offloading decision in MEC impacts the application transmission delay and power consumption, which leads to an extra scheduling length. Hence, it is vital to jointly address the task offloading problem and task scheduling problem. We classify the research in this area into studies focused on minimization of the energy consumption (VIII-A), minimization of the execution delay under energy constraints (VIII-B), minimization of latency (VIII-C), and privacy-preserving (VIII-D).

A. Minimization of Energy Consumption

In [11], the authors investigate the joint task resource allocation problem in MEC. They consider a MEC system with N mobile users with M independent tasks to be offloaded to the edge servers to minimize the overall offloading cost in terms of energy consumption, computation cost, and delay cost. In this regard, the authors propose a DQN-based joint task offloading and bandwidth allocation algorithm. The authors formulate the problem as a DQN problem with state space, action space, and reward function. The state space is considered as a $1 \times (NM + 2N)$ vector, which involves all users' offloading decisions x_{nm} and the bandwidth allocations. That is, the offloading decision space $x_{nm} \in \{0, 1\}$ for $n = 1, 2, ...N$ and $m = 1, 2, ...M$. Concerning the action space, it is defined as an index selection, which determines how the offloading decision is changed. This index also indicates if the uplink and downlink bandwidth is increased or decreased for mobile users. Finally, the reward of the state-action pair is $r_{s,a} \in \{1, -1, 0\}$. The experimental results show that the proposed DQN-based algorithm outperforms the MUMTO algorithm [297] in terms of overall offloading cost and convergence.

While in [11] the application's tasks are considered as independent, the dependent tasks offloading decision and the resource allocation in MEC is investigated in [1]. The main objective in [1] is to simultaneously minimize the task execution time and energy consumption of the mobile device. To achieve this goal, the authors first formulate the problem as a mixed-integer optimization problem, then propose a DRL-based algorithm with deep neural networks (DNNs) to learn the optimal allocation between the states and the actions. The wireless channels and edge CPU frequency represent the state, and the task offloading decisions represent the actions. The DRL uses the actor-critic learning scheme, which trains a DNN periodically in the actor-network from past experiences to learn the optimal allocation between the states and the actions. The proposed DRL-based algorithm has two main phases: the task offloading action generation based on DNN, and the offloading policy updating. To generate an offloading decision, the output of the DNN (i.e., offloading action) is first quantified into candidate offloading actions. Then, the critic network evaluates the performance of the candidate actions. Next, the candidate action with the lowest energy-time cost is selected as the solution. After generating the candidate offloading actions, the energy-time cost (ETC) of each action is evaluated, and the best offloading action is selected. Finally, the optimal actions learned in the offloading action generation phase are used to update the parameters of the DNN. Simulation results show that the proposed DRL-based algorithm attains up to 99.1% of the optimal ETC.

The authors of [298] investigate a joint resource allocation algorithm for hybrid mobile edge computing (H-MEC) systems. The H-MEC comprises ground stations (GSs), ground vehicles (GVs), and unmanned aerial vehicles (UAVs), all with MEC-enhanced to enable IoT devices to offload their computationally intensive tasks. The authors proposed a DL-based online offloading algorithm called "H2O" with the objective to minimize the energy consumption of all IoT devices. The H2O algorithm has two main phases: the offline training phase and the online optimization phase. The offline training step which requires high computation and storage capacities is performed in the remote cloud. To find a training sample, which is required to train the DNN, they first propose a clustering method called "large-scale path-loss fuzzy c-means (LS-FCM)" to find the positions of UAVs and GVs. The UAVs and GVs are then deployed based on their locations. Compared to the conventional FCM method [299], the LS-FCM approach does not allow some GS cluster centers to participate in the iteration process. Then, a PSO-based algorithm called "U-PSO" is applied to solve the offloading decision and resource allocation problems. After that, a supervised learning algorithm is used to train the DNN that can be used when the number of IoT devices varies. Next, the trained DNN is implemented for online decisions. That is, after an UE processes its membership values, the DNN outputs its offloading decision and resource allocation results. Compared to the previous work, the H2O algorithm uses both DL and meta-heuristic approaches. Moreover, it exploits the advantage of the PSO algorithm in giving global optimal solutions and uses the advantage of DNN in speeding up the real-time decision. Also, compared to traditional DL-based methods that need to input the information of all UEs, the H2O is efficient in hybrid MEC networks with a large number of IoT devices and UEs. The experiment results show that the H2O has better efficiency and accuracy compared to the random offloading, greedy offloading, and standard PSO offloading approaches.

The drawback of both above-mentioned works on joint resource allocation is that they do not consider the user's QoE. For this reason, the authors of [300] introduce a DRL-based method to address the joint offloaded task scheduling and resource allocation in vehicular networks. Compared to the previous work, the main goal of [300] is to minimize not only the execution delay but also the energy consumption while maximizing the user's QoE. Due to the complexity of the joint offloading and scheduling problem, the authors divide the problem into two sub-problems: vehicle offloading task scheduling and decision of resource allocation. The first sub-problem is solved by a two-sided matching method with the aim to maximize the total utilities reflecting the user's QoE level. An algorithm called "Dynamic V2I Matching (DVIM)" is introduced to find the optimal match. The DVIM algorithm first initializes the forbidden list (i.e., offloading tasks rejected by an RSU) and accepted list (i.e., offloading tasks accepted by an RSU). Then, each vehicle i calculates its utility $u_{i,k}$ if its task is offloaded to RSU k. Next, all vehicles make their preference RSU list $\mathcal{P}_i$ in descending order of $u_{i,k}$. After that, the vehicles that have been matched with less than q_v RSUs submit offloading requests to the preferred RSU in $\mathcal{P}_i$. The selected RSU is then removed from the preference list. After the vehicles submit their requests, the RSUs accept those requests that increase the overall utility values. The vehicles continue submitting requests until their preference lists are empty. Concerning the second sub-problem, it is solved by an improved DRL method called MADD. The MADD algorithm first initializes the experience replay buffer D with N transitions, the action-value function Q with random weight θ, and the target Q-network, which gives the temporal difference (TD) target. Then, it schedules the offloading requests. For each phase, a random RSU is selected from the available list with probability ϵ. Otherwise, the RSU with the largest Q-value is selected using a greedy approach. After that, the immediate reward r_t and the next state are observed. In the NN training phase, a DQN randomly samples a transition from the buffer D. For each sample, if the next state is the last state, then the TD target is r_j. Otherwise, the DQN is used to calculate the TD target. Then, the gradient descent method is used to update the Q-network parameters. Finally, the TD target network parameters and the random probability ϵ are updated every step to accelerate the convergence speed. Simulation results show that the DVIM and MADD algorithms outperform the baseline algorithms. For instance, in terms of utilities, the DVIM algorithm is 20% better than the greedy algorithm and 50% better than the random algorithm. In terms of average QoE, the MADD algorithm outperforms the DQN method by 15%, the Q-learning method by 25%, and the greedy method by 35%.

In [301], the authors use three learning techniques, namely LA, LSTM, and RL to address the joint computation offloading and resource provisioning problems in an edge-cloud environment. The main objective is to maximize the CPU utilization while minimizing the execution time and energy consumption. To this end, the authors first propose a learning automata (LA)-based algorithm to make a decision about offloading the incoming workload tasks to the edge servers or cloud servers. If the task is to be computed in the edge environment, the master edge server submits the job to the slave servers. For n numbers of requests as input, the LA-based algorithm first initializes the actions probabilities denoted by $P(1), P(2), \ldots, P(n)$, which are equal. Then, the first action "a" is selected randomly. In each period, "a" number of requests are executed in the edge server and cloud server. If the total execution time of the "a" number of requests in the edge server is less than the one in the cloud server, then the selected action is penalized, and the corresponding probability is reduced, while the remaining actions probabilities are increased. Otherwise, the selected action is rewarded. Finally, the algorithm selects the optimal action, which corresponds to the number of requests that can be executed in the cloud server. After the procedure of the LA-based algorithm for the offloading decision, LSTM, and QL techniques are used for resource provisioning. In particular, LSTM model is used to predict the future number of requests, while the QL is used to find a suitable number of edge servers required to process the dynamic workloads. The experimental results based on real workloads from [301] show that the proposed hybrid learning technique can reduce the average execution time by up to 8.3% compared with the fuzzy-based offloading (FO) algorithm proposed in [302], and by up to 11.3% compared to the post-decision state (PDS)-based online learning algorithm presented in [303].

B. Minimization of Execution Delay under Energy Constraints

The task offloading to the MEC servers reduces the energy consumption of the IoT device since the execution is done on the remote edge servers. On the other hand, the task offloading, especially in ultra-dense networks also incurs additional execution delays, which include the delay to send the application to MEC servers and the delay to receive results of the computation from the MEC servers [304]. Therefore, it is vital to investigate the trade-off between task execution delay and energy consumption of IoT devices.

The authors of [305] present a joint task offloading and scheduling decision mechanism to minimize the energy consumption of the application while satisfying the overall execution delay. The optimization problem is formulated as a Lyapunov optimization problem. To solve the problem, an online Q-learning algorithm is proposed. It first transforms the joint problem of task offloading and scheduling into Lyapunov drift-plus-penalty (LDPP) optimization, which is a popular method used for the optimization of queueing networks and stochastic systems [306]. Then, the algorithm obtains the optimal offloading mechanism using a QL-based offloading method. The QL-based offloading method is formulated as MDP. It first initializes the offloading vector of all tasks with random values. Then, the state transition is decided, either by choosing the task that has the highest reward $LDPP$ value with a probability $1-\delta$ or by choosing a task randomly with a probability δ. The next phase of the QL-based offloading method is the action selection policy. To select an action, the agent first identifies a set of actions with a positive Q-value. If there is no action with a positive Q-value, an action is chosen randomly. Otherwise, it uses ϵ-$greedy$ policy to choose an action. Furthermore, given a state, the agent with ϵ-$greedy$ policy chooses the action with the maximum Q-value with a probability $1-\epsilon$, and chooses a random action with a probability ϵ, where ϵ is very small. Next, the Q-matrix is updating by a positive or negative reward generated by the state transitions. Finally, the optimal offloading decision is obtained when the reward function cannot generate any positive reward. After obtaining the optimal offloading policy, the next stage of the algorithm is the scheduling and migration decision to improve resource utilization. To this end, a Lagrange migration (L-migration) method is introduced. The L-migration method first calculates the edge premigration cost also called "premigration marginal migration cost (MMC)". After that, the sum of the minimal

MMC and transmission cost is compared with the maximal one to decide whether to start the migration. If this sum is greater or equal to the maximal MMC value, then no migration is required. Otherwise, the algorithm starts the iterative process to determine the optimal migration policy. The task with the lowest delay requirements has higher priority to migrate in order to avoid migration congestion delay for delay-sensitive tasks. It is then assigned to the edge node with the highest load rate. The simulation results show that the proposed joint offloading and scheduling algorithm outperforms the delay-optimal algorithm [307] (which only considers the delay cost of task offloading ignoring the energy consumption), the energy-optimal algorithm [303] (that only considers the energy cost ignoring delay cost), and T2E algorithm [308] (that only focuses on delay optimization between the edge and terminal layer ignoring energy constraints) in terms of energy-saving while achieving low delay cost.

The minimization of the application execution delay while saving the battery power of the mobile user's equipment is the main goal in [309]. When compared to the previous study, the authors of [309] integrate SDN and MEC to propose a novel software-defined edge cloudlet (SDEC) framework for task offloading and scheduling. Particularly, QL and co-operative Q-learning (C-QL) schemes are proposed to solve the joint task offloading and scheduling problem in SDEC. The QL scheme aims to minimize the total execution delay (Sum_{delay}), while the C-QL aims to reduce the search time of the optimal resource scheduling. Like most of the RL learning approaches, the proposed QL scheme also has three main elements, including the set of states $\mathcal{S}$, the set of actions $\mathcal{A}$, and reward R. $\mathcal{S}$ comprises two components: the Sum_{delay} of the system and the available computation resource capability C_{avail} of the MEC server. The set of actions $\mathcal{A}$ performed by the agent is the resource allocation v and the computation ratio α. After executing an action a in each time step, the agent obtains a reward $R(s;a)$. Then, each state-action pair obtains a long-term reward $Q(s;a)$. After that, the agent computes and stores $Q(s;a)$ in a Q-table. This iteration is repeated until the Q-learning method converges to the optimal value of Q. Concerning the C-QL scheme, it allows agents to learn from each other in order to reduce the search time in the proposed QL scheme. Furthermore, the C-QL scheme reduces the communication time among the base stations by sharing useful information among them. Simulation results show that the proposed scheme reduces the execution delay by up to 62.10%. Also, the proposed C-Q-learning scheme achieves better performance than other benchmark approaches in terms of delay requirements.

Another idea aiming at minimization of the execution delay of the mobile application while minimizing the energy consumption is introduced in [310]. Compared to the previous study that used LTE as wireless technology, [310] uses Narrowband-IoT (NB-IoT) to transmit data to the base station (BS). The optimization problem is formulated as a continuous-time MDP (CTMDP) model, where the states of the system transit only when a packet (arrival/departure) event occurs, but not at every time slot. To solve the problem, the authors propose a combination of methods including approximate dynamic programming (ADP), temporal-difference learning (TDL), and semi-gradient descent (SGD). The main idea of the TDL method is to make the learner's current prediction for the current input pattern more closely match the next prediction at the next time step [311]. While most of the RL approaches are based on Q-value, the proposed algorithm uses TDL with a post-decision state (PDS). The main advantage of the PDS is that it does not require information about the transition probabilities to find the optimal action to take. It also has a much smaller state-space compared to Q-value. The proposed algorithm applies uniformization to the CTMDP model to avoid a heavy signaling overhead of the IoT devices. Particularly, the proposed algorithm has five main steps: initialization, local state updating, optimal control action, post-decision local state updating, and per-node value functions updating. The first phase initializes the per-node value functions $V_{n,k}$ of each IoT device n, where k denotes the index of the decision epoch. Then, when an event e_k (i.e., packet arrival or departure) occurs at the IoT device, the BS fixes $k = k + 1$, and the kth decision epoch begins. The BS notifies the second and third IoT devices to update their local states. Based on the event e_k an action is determined. After that, each IoT device updates its post-decision local state. Finally, each IoT device updates its per-node value function under the PDS.

C. Minimization of the Latency

The latency minimization of IoT users in large-scale MEC networks is the main goal in [312]. Toward this end, a DRL algorithm is proposed, which comprises three methods, namely 2r-SAE, ASA, and 2p-ER. The 2r-SAE is a stacked autoencoder approach, which provides quality data to the DRL model by compressing and representing the high dimensional data. Hence, the 2r-SAE can reduce the state space and improve the learning efficiency of the DRL. Concerning the ASA method, it tries to find the optimal action for the DRL model to produce an offloading decision with the observed state. The 2p-ER is introduced to enhance the learning process of DNN in the DRL model. In the proposed DRL algorithm, the agent interacts with the environment in discrete decision epochs. At each epoch t, the agent takes an action based on the state s_t, then the environment generates a reward r_t. After that, the ASA is introduced to find the optimal action a_t^* and the state–action pairs (h_t, a_t^*) are putted into the experience replay (ER) for DNN (agent) learning. Next, a batch of transitions is selected from the buffer by priority. Moreover, the transition which induces evident loss function decrease will have higher priority, while the transition which cannot enhance the performance of DNN will have the lower priority. Simulation results prove that the proposed algorithm outperforms existing benchmarks in terms of latency.

A joint spectrum allocation and scheduling in V2V broadcast communications is proposed in [313]. The main objective is to address the strict latency constraints on V2V links while minimizing the interference to V2I links. To this end, the authors propose a DRL mechanism, where a DQN is used to find the optimal policy for the problem. The simulation results show that each vehicle satisfies the latency constraints and minimizes the interference to V2I links.

The minimization of the task's offloading latency is also the main objective in [314]. Toward this end, the authors propose an online learning algorithm based on the Multi-Armed Bandits (MAB) framework by jointly optimize the task offloading decision and the spectrum scheduling decision. The simulation results show that the proposed algorithm is better than the UCB algorithm [315] in terms of performance delay. Another MAB-based learning algorithm for tasks-intensive offloading and balancing in MEC-enabled vehicular networks is proposed in [316]. The proposed algorithm enables individual MEC servers to learn global knowledge iteratively. Multiple players compete for multiple arm machines, where each player selects one of the arm machines and accordingly obtains a reward.

D. Privacy Preserving

In [252], the authors address three main issues faced by existing methods for computation offloading and resource allocation in MEC. These issues include security and privacy, cooperative computation offloading, and dynamic optimization. To address the first two issues, they employ blockchain technology, in particular, a consensus approach to ensure data security, and use cooperative communication to offload the computation tasks from mobile devices to the MEC system. Concerning the dynamic optimization problem, a Markov decision process (MDP) is formulated and a deep reinforcement learning algorithm is proposed to solve the MDP problem. Particularly, an algorithm based on "asynchronous advantage actor-critic (A3C)" reinforcement learning is proposed to solve the problem. A3C is a fast parallel reinforcement learning method that utilizes multiple CPU threads on a single machine to learn more intelligently and efficiently. The experimental results show that the proposed algorithm converges fast and performs better than the fixed block size (FBS) and fixed block interval (FBT) schemes.

E. Lessons Learned From ML/DL-based Joint Resource Allocation

A comprehensive summary of studies on ML/DL for joint resource allocation in MEC is given in Table XIII. The majority of ML/DL-based Joint Task Offloading and Scheduling (JTOS) methods in MEC aim to minimize the energy consumption of the IoT device or to minimize the execution delay while satisfying the energy constraints. Most of the works in this area formulated the problem as an MDP problem. Then, the MDP problem is solved using DRL techniques to find the optimal JTOS decisions. Besides, the majority of the proposed mechanisms are implemented using Tensorflow.

After deeply surveying the works addressing joint resource allocation issues in MEC, we list the following key lessons:

- **The performance of the ML/DL-based algorithm for joint resource allocation strongly depends on both the offloading policy and the scheduling policy used by the algorithm**. This is due to the fact that the offloading results depend on the target computing edge servers capabilities that are responsible to schedule and execute the offloaded tasks.
- **A combination of learning techniques can significantly improve the joint resource allocation algorithm** because each learning method can accomplish a specific task to maximize the long-term reward. For instance, it is proved in [298] that by combining the DNN, clustering, and PSO methods, the joint resource allocation algorithm obtains better efficiency and accuracy compared to the greedy, random, and PSO algorithms.
- **Stacked autoencoder (SAE) is an efficient method for latency minimization in large-scale MEC networks** because it can collect a good quality training data required by the DL-based method. Indeed, the SAE method can compress and represent the high-dimensional data generated by IoT devices. Hence, it can reduce the state space and provide quality data to the DL method used for the joint resource allocation [312]. Therefore, it can accelerate the learning process.
- **To provide better QoS/QoE for the end-users, it is crucial to execute the DL training process on the remote cloud** because the DL training procedure requires powerful computing resources. Furthermore, by executing the offloading and scheduling training procedures in the cloud servers, we are saving the limited resources of edge servers to meet the QoS (e.g., for delay-sensitive applications) and QoE (e.g., video quality received by end-users).
- **The efficiency of ML and DL methods for joint resource allocation is strongly related to the type of task being offloading** in the sense that the training of dependent tasks (i.e., tasks graph) is more complex than the training of independent tasks. Indeed, in the case of dependent tasks, the task dependencies must be preserved during both the offloading and scheduling process. Furthermore, if a task t_i preceding a task t_k is offloaded to the MEC server for training (due to IoT device resource constraints), the result must be sent back to the IoT device before considering the task t_k. One possible solution to this challenge is gathering all dependent tasks to the same MEC server as proposed in [12].

*F. **Summary of ML/DL-based Resource Allocation in MEC***

In summary, ML and DL methods enable efficient resource allocation in MEC by their ability to predict both the task's features (e.g., QoS, QoE, security, privacy requirements, etc.) and the target MEC resource capabilities. Particularly, DL and DRL can extract complex features from large amounts of high-dimensional data generated by IoT devices. Furthermore, with the recent progress in mobile communication such as 5G and beyond, it is crucial to embed artificial intelligence (AI) into MEC systems. Current ML and DL algorithms are good approaches to address the resource allocation problem in MEC networks. ML-based mechanisms for resource allocation require datasets to train from. Then, the trained model is applied to the real dataset to achieve the optimal resource allocation policy. However, the trained model may not be adapted well to the entire features and properties of the data.

TABLE XIII: Summary of Studies on ML and DL for Joint Resource Allocation in MEC

Ref.	Learning Type	Algorithm	Problem formulation	Simulation Tools	Optimization Criteria
2020, [1]	DRL	DNN	Mixed integer optimization	TensorFlow	Minimize the task execution time and energy consumption of the mobile device
2019, [11]	RL	DQN	-	TensorFlow	Minimize the execution time, energy consumption, and delay Cost
2020, [301]	RL + SL	LA + QL + LSTM	-	iFogSim	Maximize the CPU utilization while minimizing the execution time and energy consumption
2020, [305]	RL	QL	Lyapunov theory + MDP	MATLAB	Minimize the energy consumption of the application while satisfying the overall execution delay.
2020, [309]	RL	Q-Learning	-	OpenAI Gym	Minimize the execution delay of the mobile application while saving battery power of the mobile device.
2019, [310]	RL	TD-learning	MDP	-	Minimize the execution delay of the mobile application while minimizing the energy consumption.
2020, [298]	SL	DNN + PSO	MINLP	-	Minimize the energy consumption of all user devices.
2018, [313]	Deep RL	DQN+DNN	MDP	-	Minimize the latency on V2V links while minimizing the interference to V2I links.
2019, [317]	RL	MADL+D3QN	Game+MDP	-	Maximize the long-term downlink utility while satisfying the UE's QoS requirements.
2019, [318]	Deep RL	DQN	-	Tensorflow	Maximize both the short-term reward and long-term reward of the DQN model.
2020, [194]	SL	DNN	Queueing theory	-	Minimize the end-to-end inference delay of DL tasks.
2019, [319]	RL	QL	MDP	-	Minimize the power consumption while maximizing the system throughput.
2020, [312]	USL	AE +SA + DNN	-	-	Minimize the latency of IoT users.
2019, [300]	Deep RL	DDQN	MDP	Tensorflow	Minimize the execution delay and the energy consumption while maximizing the user's QoE.
2020, [252]	RL	DRL	MDP + A3C	Tensorflow	Protect data security while maximizing the computation rate and throughput of blockchain systems.

Hence, DL techniques have been used to address some of the limitations of ML mechanisms.

Nevertheless, the application of ML and DL techniques for resource allocation brings new challenges that need to be addressed. In particular, these techniques require a large amount and good quality of datasets to learn from, which are often scarce and also difficult to generate. For instance, the training model of the DRL-based task offloading method in large-scale MEC networks requires quality data from the IoT users to improve the learning efficiency [312]. Also, although some datasets can be available online, their privacy protection must be considered during the learning process. Therefore, data generation and privacy-preserving are other issues that need to be addressed to develop efficient ML/DL-based resource allocation algorithms.

Another challenge is that the collected dataset cannot be generalized for all types of MEC tasks because different tasks have different structures, attributes, and requirements. For instance, in most research papers, MEC tasks are considered as bag of tasks (BoT), where the tasks are independent, and each task has its input location id, input size, number of task execution instructions. Other studies considered the MEC tasks as dependent tasks represented by a Direct Acyclic Graph (DAG) $G = (T, E)$, where T is a set of t tasks, and E is a set of e edges. Each task t represents a set of instructions that must be executed on the same computing resource. Each edge corresponds to the precedence constraints among the tasks. Therefore, the collected dataset from different sources cannot be generalized to all MEC tasks due to the heterogeneity and distributed features of the datasets sources. Consequently, it is challenging to find a unified task model for resource allocation in MEC.

Additionally, the ML/DL training models are rarely static. Therefore, the models may need to be retrained on slightly changed datasets (e.g., when datapoints have been added or deleted) [320]. Generally, a ML/DL model need to be retrained from scratch when the data distributions have deviated significantly from those of the original training dataset. This is known as model drift. In terms of latency and promptness, model retraining at the edge of the networks has the advantage of reducing the communication bandwidth and latency between the IoT device and the remote server since it does not require submitting data from IoT networks to the cloud [321]. However, it is expensive to retrain models from scratch. The training model needs to be refined and adapted with the new sub-dataset when the new dataset is similar to the dataset observed in the past on which the model was trained.

The main reason a model needs to be retrained is that the environment in which the model is being predicted keeps changing, and consequently the dataset changes, causing model drift. Nevertheless, how and why the dataset changes depend on the use case, such as small dataset and adversarial environments. For instance, if initially the model wasn't trained with a dataset large enough the model accuracy may vary significantly between training and testing. Therefore, the model can be retrained with a training dataset that contains new observations and increases its size. In an adversarial environment (e.g., intrusion and fraud detection), where the environment behavior is actively trying to reduce the rewards given to the agent, model retraining becomes much more crucial. Hence, models retraining mechanisms capable of considering the adversarial environment should be further investigated.

The next section discusses challenges and future research directions in detail.

IX. Challenges and Future Research Directions

In this section, we discuss key challenges and potential future research directions of applying ML and DL techniques for resource allocation in MEC.

A. Trade-off Between Large-Scale Training Datasets and Computation Delay

The main challenge of DL algorithms is that they require enough high-quality training datasets, which are difficult to collect or generate, and also may not be available. Indeed, the training dataset needs to be collected from multiple IoT devices, which is challenging due to the heterogeneity and distributed features of these devices [322]. One possible solution is to move the trained model to mobile devices. However, most IoT devices have limited battery capacity and computation/storage capabilities to execute deep learning models which required high computation and storage capabilities. It is for this reason that many studies have proposed methods to offload the trained model to the edge servers rather than running it on mobile devices. For instance, DNNOff [193] aims to automatically determine the DNN models that should be offloaded to the edge servers.

Also, large training datasets are required to well analyze and compare the efficiency of different related DL-based RA methods. For example, it is proved in many studies (e.g., [244]) that different learning algorithms can provide similar performance for small datasets. On the other hand, high dimensional datasets for model training can lead to unacceptable delays because the training process of DL algorithms, in particular, DRL algorithms is computationally intensive [34]. Therefore, a vital research direction is the trade-off analysis between large-scale training datasets and computation delay. One potential solution is the compression of the training dataset as proposed in [312], where the authors use a stacked autoencoder to compress and represent the high-dimensional dataset.

B. Trade-off Between Convergence Rate and Time Complexity

The acceleration of the convergence rate of a training model is one of the most challenging issues of ML and DL techniques. Furthermore, RL-based methods require a large number of datasets, which increases the number of states and actions spaces, and therefore leads to a slow convergence rate. To solve this challenge, existing studies combine RL and DL methods. For instance, the work in [76] exploits the features of the autoencoder method in an RL model to accelerate the convergence speed of the traditional RL algorithm. By combining autoencoder and RL, the encoder can extract the task's features (e.g., task type, task priority, task execution time, etc.). Then, the feature obtained by the encoder can help the RL algorithm to find the best action in a given state, instead of using the traditional Q-table, which suffers from a slow convergence rate. On the other hand, the combination of different learning methods may increase the algorithm time complexity, which will lead to a high computation delay. However, a learning method that leads to high time complexity is not suitable for delay-sensitive and real-time applications (e.g., video streaming). Consequently, researchers need to investigate new mechanisms that accelerate the convergence rate of RL-based methods while minimizing the time complexity.

C. Deep Learning Models Caching

Most of the studies that focused on content caching approaches used deep learning to improve the traditional caching strategies. Furthermore, few works have investigated strategies to cache the DNN model at the edge of the network. However, DNN model caching at the edge network may enhance the learning process and reduce inference time. Indeed, caching the DNN model on the edge node reduces the volume of input data that needs to be sent to the remote cloud server and therefore reduces the inference latency [198]. Therefore, researchers need to investigate novel DNN model caching approaches instead of DL methods to improve traditional content caching approaches. A recent study on DNN model caching is CacheNet [199], which caches the low-complexity DNN models on IoT devices, and the high-complexity DNN models on edge/cloud servers.

D. Integration of Blockchain and ML/DL for Resource Allocation in MEC

Blockchain can be defined as a distributed data structure consisting of a chain of blocks that keeps records of all transactions in the blockchain network while ensuring security [323]. Each block is identified by a unique cryptography hash function. The blockchain network comprises nodes that record the same transactions. Due to its security and decentralization features, blockchain technology has been used in many areas, including banking systems [324], finance [325], cloud computing [326], healthcare applications [327], and MEC [328]. The most well-known application of blockchain technology is the Bitcoin created by Satoshi Nakamoto in 2008 [329]. Blockchain technology can also be used to solve the security and privacy challenges of resource allocation in MEC. A recent investigation towards this approach is introduced in [252], where a consensus approach is used to ensure data security. The authors of [330] depicts the limitations of edge intelligence (EI) and why blockchain technology could benefit from EI.

The integration of blockchain and ML/DL may improve the resource allocation method in terms of different metrics. On the one hand, the main challenge of ML/DL techniques is that they require enough high-quality training datasets which need to be collected from multiple distributed IoT devices. Also, the training datasets may contain security-sensitive information which make the training and inference more challenging since the data privacy must be considered during the learning and inference processes. Hence, Blockchain technology can be regarded as a complementary technology to solve the above challenge due to its decentralized, privacy persevering, and secure features. Particularly, distributed training and inference can be performed securely.

On the other hand, Blockchain also faces many challenges when it comes to performing edge intelligence, such as storage load, transaction capacity, and fault tolerance, which prevent many blockchain systems from being implemented [330]. Blockchain also faces a technical challenge when it comes to performing ML/DL tasks in the edge network, such as the training and inference. Hence, DL techniques can benefit blockchain for resource utilization. Specially, since a DL technique can predict data, it may also facilitate the prediction of computational tasks that a miner (or a consensus node) needs to offload to the edge/cloud server.

Therefore, the integration of blockchain and ML/DL for resource allocation in MEC is feasible due to the important current interest in blockchain and ML/DL techniques. Nevertheless, few studies have focused on the integration of lightweight blockchain (i.e., a blockchain that can be applied on resource-constrained devices without affecting the security features [331]) and DL techniques. Consequently, a vital future research direction is the development of mechanisms based on lightweight blockchain and DL for resource allocation in large-scale MEC networks.

E. Resource Allocation under Time-Varying Wireless Channel Conditions

Most of the surveyed works ignored the time-varying wireless channel state information (CSI) in joint resource allocation. However, the CSI significantly impacts the joint resource allocation decision of a wireless-powered MEC system because of the uncertain channel. The existing methods focused either on task offloading under CSI or task scheduling under CSI ignoring the joint resource allocation problem under CSI. Therefore, ML/DL algorithms for joint task offloading and scheduling in MEC networks with time-varying wireless channels should be further investigated. A recent study towards this approach is presented in [257].

F. Considering More Computing Resources

Most of the existing ML/DL-based mechanisms for resource allocation in MEC define the target computation resource only in terms of CPU processing capacity. However, ML techniques, in general, and DRL techniques in particular, require more computation resources such as memory and storage to obtain a high accuracy model. Particularly, the training model of a DRL algorithm might have specific memory requirements that the target MEC servers have to meet in order to execute the model. Hence, one topic of interest is the investigation of ML/DL-based algorithms that consider more computing resources constraints such as memory, storage, and bandwidth.

G. Resource Allocation on Hybrid Architectures

Many studies assumed that the target MEC system comprises a bounded number of similar processors (i.e., related CPU resources). However, nowadays more and more high-performance computing systems use hybrid architectures (i.e., unrelated resources), such as multicore and accelerators like GPU. These novel architectures have introduced new resource allocation issues. For instance, how to jointly select the required CPU and GPU resources to execute the offloaded task or the training model is a big challenge. In particular, DL algorithms require GPU-enabled servers to accelerate the training model [332]. Therefore, it is vital to investigate mechanisms that can jointly share the hybrid computation resources (i.e., CPU and GPU) to both the offloaded task and the training model.

H. Resource Allocation in MEC for ML/DL

In the literature, the majority of works applied ML/DL techniques to improve traditional resource allocation methods without investigating how resource allocation mechanisms in MEC can improve ML/DL techniques. Indeed, since ML/DL are increasingly integrated into MEC, it is vital to investigate resource allocation methods (i.e., offloading, caching, and scheduling), which can accelerate the learning process and the convergence of ML/DL methods. Also, when a DL algorithm is implemented on a resource-constraint IoT device, some DL tasks, in particular, delay-intensive tasks (e.g., visual target tracking, online video editing) cannot be processed on the IoT device because they require a lot of computational resources. Therefore, a potential future research direction is the application of offloading, caching, and scheduling techniques to improve ML/DL methods. A recent work towards this approach is presented in [333]. The authors proposed a novel DL task distributed framework, where the lower layers of the convolutional neural network (CNN) model are executed on the unmanned aerial vehicles, while the higher layers of the CNN model are offloaded to the MEC server.

I. Deep Learning Inference on Resource-Constrained Devices

Most of the surveyed papers focused on the deep learning training phase to obtain the optimal resource allocation policy, ignoring the inference (prediction) phase of DL. While DL training "teaches" a DNN using datasets to perform an AI task, DL inference, on the other hand, uses a trained DNN model to make predictions on new data that the model has never seen before [86]. DL inference is usually a production phase where a model is deployed to predict real-world datasets. The DL inference is computationally expensive because of the high dimensional data and millions of operations that need to be performed on the data [33]. In addition, most DL models are offloaded and executed in cloud data centers because they require more computation resources that resource-constrained devices cannot provide. However, offloading and executing DL models on the cloud servers or MEC servers [194] cannot satisfy the delay requirement of real-time services such as real-time video analytics and intelligent manufacturing. Therefore, a vital research direction is to propose novel DL models that can be embedded in resource-constrained IoT devices while considering the trade-off between the DL inference accuracy and latency.

J. Federated Deep Transfer Learning

Although transfer learning (TL) can significantly accelerate the training process by using knowledge of an existing trained model, it can negatively impact the learning performance of a target domain if there is a high dissimilarity between the source domain and the target domain. This issue called negative transfer, is one of the most challenging problems in TL [334]. Negative transfer reduces the accuracy of a machine learning model after retraining. It is mainly caused by a poor dependency between the source and the target domains. To overcome this issue, deep transfer learning (DTL) has been introduced as a new paradigm that uses DL methods to perform an efficient knowledge transfer (i.e., positive transfer) [335]. Indeed, DL has a strong dependence on large training datasets compared to traditional ML techniques because it requires a huge amount of data to understand and extract the hidden patterns. Hence, the utilization of DL to perform TL tasks (i.e., DTL) can significantly mitigate the negative transfer issue. Also, DTL has been proved to achieve high prediction accuracy in various research fields including fault diagnosis in manufacturing for cross-domain prediction [336], medical (e.g., magnetic resonance imaging (MRI), classification for covid-19 disease [337]), machine fault diagnosis [338], and networking [339]. However, its application for resource allocation in MEC is still limited. Further research on resource allocation in MEC needs to investigate new DTL mechanisms that consider not only the negative transfer issue but also the challenges of IoT data scarcity and privacy faced by deep learning techniques. A key research direction is the combination of federated learning (to address the privacy and security issues) and DTL (to solve the datasets scarcity and the negative transfer issues), which we called federated deep transfer learning (FDTL).

X. Conclusion

This paper provides a comprehensive survey and tutorial of ML and DL methods for resource allocation problems in MEC. We first present tutorials that demonstrate the advantages of applying ML and DL techniques in MEC. Then, we discuss potential technologies for quickly running ML and DL tasks (e.g., training and inference) in MEC. We also discuss and summarize key ML and DL methods and their importance for resource allocation in MEC. Afterward, we provide a comprehensive and in-depth survey of recent works that applied ML and DL techniques to address the resource allocation problem in MEC from three aspects including task offloading problem, task scheduling problem, and joint resource allocation problem. Furthermore, the state-of-the-art ML/DL-based resource allocation techniques are reviewed and classified within the scope of this study. Finally, we present an extensive list of challenges and future research directions related to the application of ML and DL for resource allocation in MEC. This survey provides an effective manual that can motivate readers to advance this research field, and help them to well understand how and when ML/DL-based resource allocation techniques perform better than the traditional methods.

References

[1] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans Wireless Commun*, vol. 19, no. 8, pp. 5404–5419, 2020.

[2] GSMA, "Iot in the 5g era opportunities and benefits for enterprises and consumers." [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2019/11/201911-GSMA-IoT-Report-IoT-in-the-5G-Era.pdf

[3] 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, 2017.

[4] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE J Sel Areas Commun*, vol. 37, no. 3, pp. 668–682, 2019.

[5] 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, 2017.

[6] ETSI, "Multi-access edge computing." [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[7] N. Boyd, "Mobile edge computing vs multi-access edge computing," Mar. 2018. [Online]. Available: https://www.sdxcentral.com/edge/definitions/mobile-edge-computing-vs-multi-access-edge-computing/

[8] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2961–2991, 2018.

[9] C. o. M. I. Alex Reznik, "Mec proof of concept."

[10] M. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," *Concurr. Comput. Pract. E.*, vol. 29, no. 8, 2017.

[11] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit Commun Netw*, vol. 5, no. 1, pp. 10 – 17, 2019.

[12] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun Mag*, vol. 57, no. 5, pp. 64–69, 2019.

[13] F. Rebecchi, M. Dias de Amorim, V. Conan, A. Passarella, R. Bruno, and M. Conti, "Data offloading techniques in cellular networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 580–603, 2015.

[14] R. V. Lopes and D. Menascé, "A taxonomy of job scheduling on distributed computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3412–3428, 2016.

[15] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Rob. Auton. Syst.*, vol. 90, pp. 55–70, 2017.

[16] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han, "Resource management in cloud networking using economic analysis and pricing models: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 954–1001, 2017.

[17] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frame-

works for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3560–3580, 2018.

[18] E. Ivashko, I. Chernov, and N. Nikitina, "A survey of desktop grid scheduling," *IEEE Trans Parallel Distrib Syst*, vol. 29, no. 12, pp. 2882–2895, 2018.

[19] L. F. Bittencourt, A. Goldman, E. R. Madeira, N. L. da Fonseca, and R. Sakellariou, "Scheduling in distributed systems: A cloud computing perspective," *Comput Sci Rev*, vol. 30, pp. 31 – 54, 2018.

[20] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, and J. Crowcroft, "A survey of opportunistic offloading," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2198–2236, 2018.

[21] M. Kumar, S. Sharma, A. Goel, and S. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J Netw Comput Appl*, vol. 143, pp. 1 – 33, 2019.

[22] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generat. Comput. Syst.*, vol. 91, pp. 407 – 415, 2019.

[23] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Comput Surv*, vol. 52, no. 4, Aug. 2019.

[24] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186 080–186 101, 2020.

[25] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *J Netw Comput Appl*, vol. 169, p. 102781, 2020.

[26] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions," *ACM Comput Surv*, vol. 53, no. 1, Feb. 2020.

[27] Q.-H. Nguyen and F. Dressler, "A smartphone perspective on computation offloading—a survey," *Comput Commun*, vol. 159, pp. 133 – 154, 2020.

[28] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty, "A survey on task offloading in multi-access edge computing," *J Syst Architect*, vol. 118, p. 102225, 2021.

[29] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Trans Parallel Distrib Syst*, vol. 32, no. 2, pp. 295–314, 2021.

[30] S. Chen, Q. Li, M. Zhou, and A. Abusorrah, "Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm," *Ah S Sens*, vol. 21, no. 3, p. 779, 2021.

[31] Y. Xu, G. Gui, H. Gacanin, and F. Adachi, "A survey on resource allocation for 5g heterogeneous networks: Current research, future trends, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 668–695, 2021.

[32] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 2019.

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

[34] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 2020.

[35] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain, "Machine learning for resource management in cellular and iot networks: Potentials, current solutions, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1251–1275, 2020.

[36] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Comm Com Inf Sc*, vol. 182, p. 107496, 2020.

[37] M. McClellan, C. Cervelló-Pastor, and S. Sallent, "Deep learning at the mobile edge: Opportunities for 5g networks," *Appl. Sci.*, vol. 10, no. 14, p. 4735, 2020.

[38] A. Shakarami, M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective," *J Grid Comput*, pp. 1–33, 2020.

[39] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2020.

[40] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," *arXiv preprint arXiv:2003.12172*, 2020.

[41] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multiaccess edge computing for 5g and internet of things," *IEEE Int. Things J.*, vol. 7, no. 8, pp. 6722–6747, 2020.

[42] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Comm Com Inf Sc*, vol. 195, p. 108177, 2021.

[43] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, pp. 1–1, 2021.

[44] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Trans Parallel Distrib Syst*, vol. 27, no. 2, pp. 585–599, 2016.

[45] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[46] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[47] N. F. V. ETSI, "Network functions virtualisation (nfv)," https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[48] R. Sairam, S. S. Bhunia, V. Thangavelu, and M. Gurusamy, "Netra: Enhancing iot security using nfv-based edge traffic analysis," *IEEE Sensors J*, vol. 19, no. 12, pp. 4660–4671, 2019.

[49] A. Pastor, A. Mozo, D. R. Lopez, J. Folgueira, and A. Kapodistria, "The mouseworld, a security traffic analysis lab based on nfv/sdn," in *Proc. 13th Int. Conf. Availability Rel. Secur.*, 2018, pp. 1–6.

[50] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi, "Nfv security survey: From use case driven threat analysis to state-of-the-art countermeasures," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3330–3368, 2018.

[51] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 2016.

[52] T. Wang, J. Zu, G. Hu, and D. Peng, "Adaptive service function chain scheduling in mobile edge computing via deep reinforcement learning," *IEEE Access*, vol. 8, pp. 164 922–164 935, 2020.

[53] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi, "Survey of performance acceleration techniques for network function virtualization," *Proc IEEE*, vol. 107, no. 4, pp. 746–764, 2019.

[54] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132 021–132 085, 2020.

[55] R. Riggio, S. N. Khan, T. Subramanya, I. G. B. Yahia, and D. Lopez, "Lightmano: Converging nfv and sdn at the edges of the network," in *NOMS 2018 - 2018 IEEE/IFIP Netw. Operations Manage. Symp.*, 2018, pp. 1–9.

[56] C.-L. I, S. Kuklinskí, and T. Chen, "A perspective of o-ran integration with mec, son, and network slicing in the 5g era," *IEEE Netw.*, vol. 34, no. 6, pp. 3–4, 2020.

[57] A. Reznik, L. M. C. Murillo, Y. Fang, W. Featherstone, M. Filippou, F. Fontes, F. Giust, Q. Huang, A. Li, C. Turyagyenda *et al.*, "Cloud ran and mec: A perfect pairing," *ETSI White paper*, no. 23, pp. 1–24, 2018.

[58] Z. Lv and W. Xiu, "Interaction of edge-cloud computing based on sdn and nfv for next generation iot," *IEEE Internet Things J*, vol. 7, no. 7, pp. 5706–5712, 2020.

[59] B. Blanco, J. O. Fajardo, I. Giannoulakis, E. Kafetzakis, S. Peng, J. Pérez-Romero, I. Trajkovska, P. S. Khodashenas, L. Goratti, M. Paolino *et al.*, "Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn," *Comput. Standards & Interfaces*, vol. 54, pp. 216–228, 2017.

[60] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun, "Cds-mec: Nfv/sdn-based application management for mec in 5g systems," *Comm Com Inf Sc*, vol. 135, pp. 96–107, 2018.

[61] P. Shantharama, A. S. Thyagaturu, N. Karakoc, L. Ferrari, M. Reisslein, and A. Scaglione, "Layback: Sdn management of multi-access edge computing (mec) for network access services and radio resource sharing," *IEEE Access*, vol. 6, pp. 57 545–57 561, 2018.

[62] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 2019.

[63] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2392–2431, 2017.

[64] 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, 2018.

[65] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 2018.

[66] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 2019.

[67] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung, "Blockchain and machine learning for communications and networking systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1392–1431, 2020.

[68] Y. Sun, J. Liu, J. Wang, Y. Cao, and N. Kato, "When machine learning meets privacy in 6g: A survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2694–2724, 2020.

[69] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for internet of things (iot) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, 2020.

[70] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Comput Sci Rev*, vol. 40, p. 100379, 2021.

[71] F. Tang, B. Mao, Y. Kawamoto, and N. Kato, "Survey on machine learning for intelligent end-to-end communication toward 6g: From network access, routing to traffic control and streaming adaption," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1578–1598, 2021.

[72] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[73] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg,

T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput Surv*, vol. 53, no. 2, Mar. 2020. [Online]. Available: https://doi.org/10.1145/3377454

[74] I. Portugal, P. Alencar, and D. Cowan, "The use of machine learning algorithms in recommender systems: A systematic review," *Expert Syst Appl*, vol. 97, pp. 205 – 227, 2018.

[75] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Comput. & Secur.*, vol. 81, pp. 123 – 147, 2019.

[76] S. Shadroo, A. M. Rahmani, and A. Rezaee, "The two-phase scheduling based on deep learning in the internet of things," *Comm Com Inf Sc*, vol. 185, p. 107684, 2021.

[77] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif Intell Rev*, vol. 53, no. 8, pp. 5455–5516, 2020.

[78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.

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

[80] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[81] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, June 2016.

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

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

[84] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, 2019.

[85] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[86] "Deep learning training vs deep learning inference," https://premioinc.com/blogs/blog/deep-learning-training-vs-deep-learning-inference.

[87] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, 2018.

[88] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An online algorithm for task offloading in heterogeneous mobile clouds," *ACM Trans Internet Technol*, vol. 18, no. 2, Jan. 2018.

[89] P. Lin, Q. Song, F. R. Yu, D. Wang, and L. Guo, "Task offloading for wireless vr-enabled medical treatment with blockchain security using collective reinforcement learning," *IEEE Internet Things J*, pp. 1–1, 2021.

[90] A. Samanta, Z. Chang, and Z. Han, "Latency-oblivious distributed task scheduling for mobile edge computing," in *2018 IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–7.

[91] J. Huang, S. Li, and Y. Chen, "Revenue-optimal task scheduling and resource management for iot batch jobs in mobile edge computing," *Peer-to-Peer Netw. Appl.*, no. 8, 2020.

[92] Y. Cui, D. Zhang, T. Zhang, P. Yang, and H. Zhu, "A new approach on task offloading scheduling for application of mobile edge computing," in *2021 IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–6.

[93] X. Jiang, F. R. Yu, T. Song, and V. C. M. Leung, "A survey on multi-access edge computing applied to video streaming: Some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 871–903, 2021.

[94] E. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, *Bin Packing Approximation Algorithms: Survey and Classification*, 01 2012, p. (to appear).

[95] P. Festa, "A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems," in *2014 16th Int. Conf. Transparent Opt. Netw. (ICTON)*. IEEE, 2014, pp. 1–20.

[96] J. Shen, N. Yi, B. Wu, W. Jiang, and H. Xiang, "A greedy-based resource allocation algorithm for multicast and unicast services in ofdm system," in *2009 Int. Conf. Wireless Commun. Signal Process.*, 2009, pp. 1–5.

[97] Y. Fan, L. Wang, W. Wu, and D. Du, "Cloud/edge computing resource allocation and pricing for mobile blockchain: An iterative greedy and search approach," *IEEE Trans Comput Social Syst*, vol. 8, no. 2, pp. 451–463, 2021.

[98] F. Wei, S. Chen, and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," *China Commun*, vol. 15, no. 11, pp. 149–157, 2018.

[99] M. T. Islam, A.-E. M. Taha, S. Akl, and S. Choudhury, "A local search algorithm for resource allocation for underlaying device-to-device communications," in *2015 IEEE Global Commun. Conf. (GLOBECOM)*, 2015, pp. 1–6.

[100] Q. Wei, W. Sun, B. Bai, L. Wang, E. G. Ström, and M. Song, "Resource allocation for v2x communications: A local search based 3d matching approach," in *2017 IEEE Int. Conf. Commun. (ICC)*, 2017, pp. 1–6.

[101] A. L. Stolyar, "Greedy primal-dual algorithm for dynamic resource allocation in complex networks," *Queueing Syst*, vol. 54, no. 3, pp. 203–220, 2006.

[102] M. Chen and J. Huang, "Optimal resource allocation for ofdm uplink communication: A primal-dual approach," in *2008 42nd Annu. Conf. Inf. Sci. Syst.*, 2008, pp. 926–

931.

[103] Y.-H. Chiang, T. Zhang, and Y. Ji, "Joint cotask-aware offloading and scheduling in mobile edge computing systems," *IEEE Access*, vol. 7, pp. 105 008–105 018, 2019.

[104] V. T. Chakaravarthy, A. R. Choudhury, S. Gupta, S. Roy, and Y. Sabharwal, "Improved algorithms for resource allocation under varying capacity," in *Eur. Symp. Algorithms*. Springer, 2014, pp. 222–234.

[105] K. Mukherjee, P. Dutta, G. Raravi, T. Rajasubramaniam, K. Dasgupta, and A. Singh, "Fair resource allocation for heterogeneous tasks," in *2015 IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 1087–1096.

[106] Wikipedia, "Heuristic." [Online]. Available: https://en.wikipedia.org/wiki/Heuristic

[107] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[108] H. Djigal, J. Feng, and J. Lu, "Task scheduling for heterogeneous computing using a predict cost matrix," in *Proc. 48th Int. Conf. Parallel Process.: Workshops*, ser. ICPP 2019. New York, NY, USA: ACM, 2019, pp. 25:1–25:10.

[109] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans Parallel Distrib Syst*, vol. 25, no. 3, pp. 682–694, 2014.

[110] H. Djigal, J. Feng, and J. Lu, "Performance evaluation of security-aware list scheduling algorithms in iaas cloud," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 330–339.

[111] P. Paymard and N. Mokari, "Resource allocation in pd-noma–based mobile edge computing system: Multiuser and multitask priority," *Trans Emerg Telecommun Technologies*, p. e3631, 2019.

[112] H. Djigal, F. Jun, J. Lu, and J. Ge, "Ippts: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Trans Parallel Distrib Syst*, pp. 1–1, 2020.

[113] H. Djigal, L. Liu, J. Luo, and J. Xu, "Buda: Budget and deadline aware scheduling algorithm for task graphs in heterogeneous systems," in *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, 2022, pp. 1–10.

[114] A. Yoosefi and H. R. Naji, "A clustering algorithm for communication-aware scheduling of task graphs on multi-core reconfigurable systems," *IEEE Trans Parallel Distrib Syst*, no. 10, pp. 2718–2732, 2017.

[115] H. Kanemitsu, M. Hanada, and H. Nakazato, "Clustering-based task scheduling in a large number of heterogeneous processors," *IEEE Trans Parallel Distrib Syst*, vol. 27, no. 11, pp. 3144–3157, 2016.

[116] L. Dong, M. N. Satpute, J. Shan, B. Liu, Y. Yu, and T. Yan, "Computation offloading for mobile-edge computing with multi-user," in *2019 IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 841–850.

[117] J. L. de Souza Toniolli and B. Jaumard, "Resource allocation for multiple workflows in cloud-fog computing systems," in *Proc. 12th IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, ser. UCC '19 Companion. New York, NY, USA: Association for Computing Machinery, 2019, p. 77–84. [Online]. Available: https://doi.org/10.1145/3368235.3368846

[118] M. Y. Özkaya, A. Benoit, B. Uçar, J. Herrmann, and Ü. V. Çatalyürek, "A scalable clustering-based task scheduler for homogeneous processors using dag partitioning," in *IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*. IEEE, 2019, pp. 155–165.

[119] A. Dogan and R. Ozguner, "Ldbs: A duplication based scheduling algorithm for heterogeneous computing systems," 2002, pp. 352 – 359.

[120] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Trans Parallel Distrib Syst*, vol. 28, no. 9, pp. 2674–2688, 2017.

[121] K. He, X. Meng, Z. Pan, L. Yuan, and P. Zhou, "A novel task-duplication based clustering algorithm for heterogeneous computing environments," *IEEE Trans Parallel Distrib Syst*, vol. 30, no. 1, pp. 2–14, Jan 2019.

[122] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *J. Supercomput.*, vol. 71, no. 9, pp. 3373–3418, 2015.

[123] Y. Liu, L. Meng, and H. Tomiyama, "A genetic algorithm for scheduling of data-parallel tasks on multicore architectures," *IPSJ Trans. Syst. LSI Des. Methodol.*, vol. 12, pp. 74–77, 2019.

[124] S. Basu, M. Karuppiah, K. Selvakumar, K.-C. Li, S. H. Islam, M. M. Hassan, and M. Z. A. Bhuiyan, "An intelligent/cognitive model of task scheduling for iot applications in cloud computing environment," *Future Gener. Comput. Syst*, vol. 88, pp. 254–261, 2018.

[125] R. L. Kadri and F. F. Boctor, "An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case," *Eur J Oper Res*, vol. 265, no. 2, pp. 454–462, 2018.

[126] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *J. Intell. Manuf.*, vol. 29, no. 3, pp. 603–615, 2018.

[127] Q. You and B. Tang, "Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things," *J Cloud Comput*, vol. 10, no. 1, pp. 1–11, 2021.

[128] R. S. Elhabyan and M. C. Yagoub, "Two-tier particle swarm optimization protocol for clustering and routing in wireless sensor network," *J. Netw. Comput. Appl.*, vol. 52, pp. 116 – 128, 2015.

[129] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem," *IEEE Access*, vol. 7, pp. 20 281–20 292, 2019.

[130] Y. Moon, H. Yu, J.-M. Gil, and J. Lim, "A slave

ants based ant colony optimization algorithm for task scheduling in cloud computing environments," *Human-centric Comput. Inf. Sciences*, vol. 7, no. 1, p. 28, 2017.

[131] J. Meshkati and F. Safi-Esfahani, "Energy-aware resource utilization based on particle swarm optimization and artificial bee colony algorithms in cloud computing," *J Supercomput*, vol. 75, no. 5, pp. 2455–2496, 2019.

[132] S. Chai, Y. Li, J. Wang, and C. Wu, "A list simulated annealing algorithm for task scheduling on network-on-chip." *JCP*, vol. 9, no. 1, pp. 176–182, 2014.

[133] C. Gallo and V. Capozzi, "A simulated annealing algorithm for scheduling problems," *J Appl Math Phys*, vol. 7, no. 11, pp. 2579–2594, 2019.

[134] J. J. F. S. Avinash Dixit, "Game theory explained," https://www.pbs.org/wgbh/americanexperience/features/nash-game/.

[135] W. Lu, W. Wu, J. Xu, P. Zhao, D. Yang, and L. Xu, "Auction design for cross-edge task offloading in heterogeneous mobile edge clouds," *Computer Communications*, vol. 181, pp. 90–101, 2022.

[136] W. Lu, S. Zhang, J. Xu, D. Yang, and L. Xu, "Truthful multi-resource transaction mechanism for p2p task offloading based on edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6122–6135, 2021.

[137] J. Moura and D. Hutchison, "Game theory for multi-access edge computing: Survey, use cases, and future trends," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 260–288, 2019.

[138] X. Feng, Y. Liu, and S. Wei, "Livedeep: Online viewport prediction for live virtual reality streaming using lifelong deep learning," in *2020 IEEE Conf. Virtual Real. 3D User Interfaces (VR)*, 2020, pp. 800–808.

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

[140] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, and R. Buyya, "Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated iot and fog computing environments," *Future Gener Comp Sy*, vol. 104, pp. 187–200, 2020.

[141] K. Chakrabarti, "Deep learning based offloading for mobile augmented reality application in 6g," *Comput. & Elect. Eng.*, vol. 95, p. 107381, 2021.

[142] Z. Wu and D. Yan, "Deep reinforcement learning-based computation offloading for 5g vehicle-aware multi-access edge computing network," *China Commun*, vol. 18, no. 11, pp. 26–41, 2021.

[143] A. Asheralieva and D. Niyato, "Learning-based mobile edge computing resource management to support public blockchain networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 1092–1109, 2021.

[144] X. Shen, J. Gao, W. Wu, K. Lyu, M. Li, W. Zhuang, X. Li, and J. Rao, "Ai-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 45–66, 2020.

[145] M. H. Abidi, H. Alkhalefah, K. Moiduddin, M. Alazab, M. K. Mohammed, W. Ameen, and T. R. Gadekallu, "Optimal 5g network slicing using machine learning and deep learning concepts," *Comput Standards Interfaces*, vol. 76, p. 103518, 2021.

[146] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, and T. Miyazawa, "Consideration on automation of 5g network slicing with machine learning," in *2018 ITU Kaleidoscope: Mach. Learn. a 5G Future (ITU K)*. IEEE, 2018, pp. 1–8.

[147] Y. Liu, H. Lu, X. Li, Y. Zhang, L. Xi, and D. Zhao, "Dynamic service function chain orchestration for nfv/mec-enabled iot networks: A deep reinforcement learning approach," *IEEE Int. Things J.*, vol. 8, no. 9, pp. 7450–7465, 2021.

[148] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Comput. Netw.*, vol. 166, p. 106980, 2020.

[149] B. Trinh and G.-M. Muntean, "A deep reinforcement learning-based resource management scheme for sdn-mec-supported xr applications," in *2022 IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2022, pp. 790–795.

[150] C. Li, C. Qianqian, and Y. Luo, "Low-latency edge cooperation caching based on base station cooperation in sdn based mec," *Expert Syst. Appl.*, vol. 191, p. 116252, 2022.

[151] H. Zhang, R. Wang, W. Sun, and H. Zhao, "Mobility management for blockchain-based ultra-dense edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7346–7359, 2021.

[152] D. Wang, X. Tian, H. Cui, and Z. Liu, "Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network," *China Commun.*, vol. 17, no. 8, pp. 31–44, 2020.

[153] A. Lekharu, M. Jain, A. Sur, and A. Sarkar, "Deep learning model for content aware caching at mec servers," *IEEE Trans Netw Service Manag*, 2021.

[154] W.-C. Chien, H.-Y. Weng, and C.-F. Lai, "Q-learning based collaborative cache allocation in mobile edge computing," *Future Gener. Comput. Syst.*, vol. 102, pp. 603–610, 2020.

[155] L. Sun, L. Wan, and X. Wang, "Learning-based resource allocation strategy for industrial iot in uav-enabled mec systems," *IEEE Trans. Ind. Inf.*, vol. 17, no. 7, pp. 5031–5040, 2021.

[156] H. Peng and X. Shen, "Multi-agent reinforcement learning based resource management in mec- and uav-assisted vehicular networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 131–141, 2021.

[157] S. Vimal, M. Khari, N. Dey, R. G. Crespo, and

Y. Harold Robinson, "Enhanced resource allocation in mobile edge computing using reinforcement learning based moaco algorithm for iiot," *Comput. Commun.*, vol. 151, pp. 355–364, 2020.

[158] L. Liu, J. Feng, Q. Pei, C. Chen, Y. Ming, B. Shang, and M. Dong, "Blockchain-enabled secure data sharing scheme in mobile-edge computing: An asynchronous advantage actor–critic learning approach," *IEEE Int. Things J.*, vol. 8, no. 4, pp. 2342–2353, 2021.

[159] U. Majeed and C. S. Hong, "Flchain: Federated learning via mec-enabled blockchain network," in *2019 20th Asia-Pacific Net. Operations Manage. Symp. (APNOMS)*, 2019, pp. 1–4.

[160] Z. Mlika and S. Cherkaoui, "Network slicing with mec and deep reinforcement learning for the internet of vehicles," *IEEE Netw*, vol. 35, no. 3, pp. 132–138, 2021.

[161] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Int. Things J.*, vol. 7, no. 10, pp. 9517–9529, 2020.

[162] S. Wan, L. Qi, X. Xu, C. Tong, and Z. Gu, "Deep learning models for real-time human activity recognition with smartphones," *Mobile Netw. Appl.*, vol. 25, no. 2, pp. 743–755, 2020.

[163] A. Feriani, A. Refaey, and E. Hossain, "Tracking pandemics: A mec-enabled iot ecosystem with learning capability," *IEEE Int. Things Mag.*, vol. 3, no. 3, pp. 40–45, 2020.

[164] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int J Comput Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[165] R. Reed, "Pruning algorithms-a survey," *IEEE Trans Neural Netw*, vol. 4, no. 5, pp. 740–747, 1993.

[166] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[167] J. Wang, L. Gou, W. Zhang, H. Yang, and H.-W. Shen, "Deepvid: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation," *IEEE Trans Vis Comput Graphics*, vol. 25, no. 6, 2019.

[168] E. Tanghatari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Distributing dnn training over iot edge devices based on transfer learning," *Neurocomputing*, vol. 467, pp. 56–65, 2022.

[169] Y. Lin, Y. Tu, and Z. Dou, "An improved neural network pruning technology for automatic modulation classification in edge devices," *IEEE Trans Veh Technol*, vol. 69, no. 5, pp. 5703–5706, 2020.

[170] Z. Zhou, H. Cai, S. Rong, Y. Song, K. Ren, W. Zhang, Y. Yu, and J. Wang, "Activation maximization generative adversarial nets," *arXiv preprint arXiv:1703.02000*, 2017.

[171] T. J. O'shea and N. West, "Radio machine learning dataset generation with gnu radio," in *Proc. GNU Radio Conf.*, vol. 1, no. 1, 2016.

[172] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[173] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[174] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *arXiv preprint arXiv:1909.12326*, 2019.

[175] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artif Intell Rev*, vol. 53, no. 7, pp. 5113–5155, 2020.

[176] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, "Deep model compression and architecture optimization for embedded systems: A survey," *J Signal Process. Syst.*, vol. 93, no. 8, pp. 863–878, 2021.

[177] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Mach. Intell.*, vol. 3, no. 8, pp. 675–686, 2021.

[178] A. Kwasniewska, M. Szankin, M. Ozga, J. Wolfe, A. Das, A. Zajac, J. Ruminski, and P. Rad, "Deep learning optimization for edge devices: Analysis of training quantization parameters," in *IECON 2019 - 45th Annu. Conf. IEEE Ind. Electronics Soc.*, vol. 1, 2019, pp. 96–101.

[179] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler, and F. Silvestri, "Neural network quantization in federated learning at the edge," *Inform Sciences*, vol. 575, pp. 417–436, 2021.

[180] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[181] S. Wiedemann, S. Shivapakash, D. Becking, P. Wiedemann, W. Samek, F. Gerfers, and T. Wiegand, "Fantastic4: A hardware-software co-design approach for efficiently running 4bit-compact multilayer perceptrons," *IEEE Open J Circuits Syst.*, vol. 2, pp. 407–419, 2021.

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

[183] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[184] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Adv Neur In*, vol. 33, pp. 11 711–11 722, 2020.

[185] S. Mücke, N. Piatkowski, and K. Morik, "Hardware acceleration of machine learning beyond linear algebra," in *Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Springer, 2019, pp. 342–347.

[186] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vision

Pattern Recognit., 2019, pp. 8612–8620.

[187] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM Int. Conf. Computer-Aided Des. (ICCAD)*. IEEE, 2018, pp. 1–8.

[188] S. Li, E. Hanson, X. Qian, H. H. Li, and Y. Chen, "Escalate: Boosting the efficiency of sparse cnn accelerator with kernel decomposition," in *MICRO-54: 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2021, pp. 992–1004.

[189] E. B. Moustafa, A. H. Hammad, and A. H. Elsheikh, "A new optimized artificial neural network model to predict thermal efficiency and water yield of tubular solar still," *Case Stud. Thermal Eng.*, vol. 30, p. 101750, 2022.

[190] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, 2016.

[191] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging graph neural networks for network modeling and optimization in sdn," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, 2020.

[192] 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, 2019.

[193] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "Dnnoff: Offloading dnn-based intelligent iot applications in mobile edge computing," *IEEE Trans Ind Informat*, vol. 18, no. 4, pp. 2820–2829, 2022.

[194] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet Things J*, vol. 7, no. 10, pp. 9241–9254, 2020.

[195] C. Kim, A. Dudin, O. Dudina, and S. Dudin, "Tandem queueing system with infinite and finite intermediate buffers and generalized phase-type service time distribution," *Eur J Oper Res*, vol. 235, no. 1, pp. 170–179, 2014.

[196] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *2019 IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[197] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li. (2021) Task partitioning and offloading in dnn-task enabled mobile edge computing networks.

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

[199] Y. Fang, S. M. Shalmani, and R. Zheng, "Cachenet: A model caching framework for deep learning inference on the edge," *arXiv preprint arXiv:2007.01793*, 2020.

[200] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do cifar-10 classifiers generalize to cifar-10?" *arXiv preprint arXiv:1806.00451*, 2018.

[201] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet Things J*, vol. 8, no. 12, pp. 9511–9522, 2021.

[202] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments," *IEEE Trans Parallel Distrib Syst*, vol. 33, no. 3, pp. 683–697, 2022.

[203] A. Qadeer and M. J. Lee, "Ddpg-edge-cloud: A deep-deterministic policy gradient based multi-resource allocation in edge-cloud system," in *2022 Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, 2022, pp. 339–344.

[204] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, pp. 1–19, 2022.

[205] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, "Privacy-aware edge computing based on adaptive dnn partitioning," in *2019 IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[206] H.-S. Lee and D.-E. Lee, "Resource allocation in wireless networks with federated learning: Network adaptability and learning acceleration," *ICT Express*, vol. 8, no. 1, pp. 31–36, 2022.

[207] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported internet of things," *IEEE Access*, vol. 7, pp. 69 194–69 201, 2019.

[208] R. Yu and P. Li, "Toward resource-efficient federated learning in mobile edge computing," *IEEE Netw.*, vol. 35, no. 1, pp. 148–155, 2021.

[209] L. Zang, X. Zhang, and B. Guo, "Federated deep reinforcement learning for online task offloading and resource allocation in wpc-mec networks," *IEEE Access*, vol. 10, pp. 9856–9867, 2022.

[210] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, 2019.

[211] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[212] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[213] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th Annu. Comput. Commun. Workshop Conf. (CCWC)*. IEEE, 2018.

[214] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proc. IEEE Int. Conf. Comput. vision*, 2015, pp. 3730–3738.

[215] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans Cogn Commun Netw*, vol. 3, no. 4, pp. 563–575, 2017.

[216] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf:

A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[217] S. Ayyachamy, V. Alex, M. Khened, and G. Krishnamurthi, "Medical image retrieval using resnet-18," in *Med. Imag. 2019: Imag. Inf. Healthcare Res. Appl.*, vol. 10954. International Society for Optics and Photonics, 2019, p. 1095410.

[218] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[219] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conf. comput. vision pattern recognition*. Ieee, 2009, pp. 248–255.

[220] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *J Instrum*, vol. 13, no. 07, p. P07027, 2018.

[221] [Online]. Available: https://github.com/google/qkeras

[222] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu *et al.*, "Compressing deep neural networks on fpgas to binary and ternary precision with hls4ml," *Mach Learn : Sci Technol*, vol. 2, no. 1, p. 015001, 2020.

[223] H.-J. Jeong, H.-J. Lee, K. Yong Shin, Y. Hwan Yoo, and S.-M. Moon, "Perdnn: Offloading deep neural network computations to pervasive edge servers," in *2020 IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2020, pp. 1055–1066.

[224] Y. Zheng, X. Xie, W.-Y. Ma *et al.*, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[225] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "Crawdad dataset cambridge/haggle (v. 2006-09-15)," *CRAWDAD wireless netw data arch*, 2006.

[226] Z. Zhang, L. Tran, X. Yin, Y. Atoum, X. Liu, J. Wan, and N. Wang, "Gait recognition via disentangled representation learning," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 4710–4719.

[227] S. Shahhosseini, D. Seo, A. Kanduri, T. Hu, S.-s. Lim, B. Donyanavard, A. M. Rahmani, and N. Dutt, "Online learning for orchestration of inference in multi-user end-edge-cloud networks," *ACM Trans Embedded Comput Syst (TECS)*, 2022.

[228] D. Xu, Q. Li, and H. Zhu, "Energy-saving computation offloading by joint data compression and resource allocation for mobile-edge computing," *IEEE Commun Lett*, vol. 23, no. 4, pp. 704–707, 2019.

[229] T. T. Nguyen, V. N. Ha, L. B. Le, and R. Schober, "Joint data compression and computation offloading in hierarchical fog-cloud systems," *IEEE Trans Wireless Commun*, vol. 19, no. 1, pp. 293–309, 2020.

[230] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep q network," *IEEE Trans Ind Informat*, vol. 15, no. 7, pp. 4276–4284, 2019.

[231] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in iot networks via machine learning," *IEEE Internet Things J*, vol. 7, no. 4, pp. 3415–3426, 2020.

[232] A. Kwasinski, W. Wang, and F. S. Mohammadi, "Reinforcement learning for resource allocation in cognitive radio networks," *Mach. Learn. Future Wireless Commun.*, pp. 27–44, 2020.

[233] H. Ghauch, H. Shokri-Ghadikolaei, G. Fodor, C. Fischione, and M. Skoglund, "Machine learning for spectrum sharing in millimeter-wave cellular networks," *Mach. Learn. Future Wireless Commun.*, pp. 45–62, 2020.

[234] N. C. Luong, Y. Jiao, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "A machine-learning-based auction for resource trading in fog computing," *IEEE Commun Mag*, vol. 58, no. 3, pp. 82–88, 2020.

[235] Z. Al-Makhadmeh and A. Tolba, "Sraf: Scalable resource allocation framework using machine learning in user-centric internet of things," *Peer-to-Peer Netw. Appl.*, pp. 1–11, 2020.

[236] S. C. Rajanarayanan, R. Misra, and R. J. Pandya, "Machine learning oriented resource allocation to achieve ultra low power, low latency and high reliability vehicular communication networks," in *2020 IEEE 17th India Council Int. Conf. (INDICON)*, 2020, pp. 1–5.

[237] J. Liu, T. Yang, J. Bai, and B. Sun, "Resource allocation and scheduling in the intelligent edge computing context," *Future Gener Comp Sy*, vol. 121, pp. 48–53, 2021.

[238] C. Mechalikh, H. Taktak, and F. Moussa, "A fuzzy decision tree based tasks orchestration algorithm for edge computing environments," in *Adv. Inf. Netw. Appl.*, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham: Springer International Publishing, 2020, pp. 193–203.

[239] H. Guo, J. Liu, and J. Lv, "Toward intelligent task offloading at the edge," *IEEE Netw*, vol. 34, no. 2, pp. 128–134, 2020.

[240] S. Imtiaz, H. Ghauch, G. P. Koudouridis, and J. Gross, "Random forests resource allocation for 5g systems: Performance and robustness study," in *2018 IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, 2018, pp. 326–331.

[241] M. E. Mavroforakis and S. Theodoridis, "A geometric approach to support vector machine (svm) classification," *IEEE Trans Neural Netw*, vol. 17, no. 3, pp. 671–682, 2006.

[242] S. Wang, M. Chen, C. Yin, W. Saad, C. S. Hong, S. Cui, and H. V. Poor, "Federated learning for task and resource allocation in wireless high altitude balloon networks," *IEEE Internet Things J*, pp. 1–1, 2021.

[243] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, "Q-learning based dynamic task scheduling for energy-efficient cloud computing," *Future Gener Comp Sy*, vol. 108, pp. 361 – 371, 2020.

[244] Z. Tong, X. Deng, H. Chen, J. Mei, and H. Liu,

"Ql-heft: a novel machine learning scheduling scheme base on cloud computing environment," *Neural Comput. Appl.*, pp. 1–18, 03 2019.

[245] Z. Tong, Z. Xiao, K. Li, and K. Li, "Proactive scheduling in distributed computing—a reinforcement learning approach," *J Parallel Distr Com*, vol. 74, no. 7, pp. 2662 – 2672, 2014.

[246] T. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans on Cloud Comput*, vol. 8, no. 1, pp. 256–268, 2020.

[247] P. Yu, F. Zhou, X. Zhang, X. Qiu, M. Kadoch, and M. Cheriet, "Deep learning-based resource allocation for 5g broadband tv service," *IEEE Trans Broadcast*, vol. 66, no. 4, pp. 800–813, 2020.

[248] G. Rjoub, J. Bentahar, O. Abdel Wahab, and A. Bataineh, "Deep smart scheduling: A deep learning approach for automated big data scheduling over the cloud," in *2019 7th Int. Conf. Future Internet Things Cloud (FiCloud)*, 2019, pp. 189–196.

[249] P. Goswami, A. Mukherjee, M. Maiti, S. K. S. Tyagi, and L. Yang, "A neural network based optimal resource allocation method for secure iiot network," *IEEE Internet Things J*, pp. 1–1, 2021.

[250] J. Shi, Q. Zhang, Y.-C. Liang, and X. Yuan, "Distributed deep learning power allocation for d2d network based on outdated information," in *2020 IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2020, pp. 1–6.

[251] Z. Hu, J. Tu, and B. Li, "Spear: Optimized dependency-aware task scheduling with deep reinforcement learning," in *2019 IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, July 2019, pp. 2037–2046.

[252] J. Feng, F. Richard Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet Things J*, vol. 7, no. 7, pp. 6214–6228, 2020.

[253] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *2017 IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 372–382.

[254] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for v2v communications," *IEEE Trans Veh Technol*, vol. 68, no. 4, pp. 3163–3173, 2019.

[255] W. Zhan, C. Luo, J. Wang, G. Min, and H. Duan, "Deep reinforcement learning-based computation offloading in vehicular edge computing," in *2019 IEEE Global Commun. Conf. (GLOBECOM)*. IEEE Press, 2019, p. 1–6.

[256] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans Parallel Distrib Syst*, vol. 32, no. 1, pp. 242–253, 2021.

[257] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans Mobile Comput*, vol. 19, no. 11, pp. 2581–2593, 2020.

[258] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J*, vol. 7, no. 6, pp. 5449–5465, 2020.

[259] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[260] H. Meng, D. Chao, and Q. Guo, "Deep reinforcement learning based task offloading algorithm for mobile-edge computing systems," in *Proc. 2019 4th Int. Conf. Math. Artif. Intell.*, ser. ICMAI 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 90–94.

[261] D. Rahbari and M. Nickray, "Task offloading in mobile fog computing by classification and regression tree," *Peer-to-Peer Netw. Appl.*, vol. 13, no. 1, 2020.

[262] T. Q. Dinh, Q. D. La, T. Q. S. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Trans Commun*, vol. 66, no. 12, pp. 6353–6367, 2018.

[263] 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, 2016.

[264] T. Yang, S. Gao, J. Li, M. Qin, X. Sun, R. Zhang, M. Wang, and X. Li, "Multi-armed bandits learning for task offloading in maritime edge intelligence networks," *IEEE Trans. Veh. Technol.*, pp. 1–1, 2022.

[265] A. Slivkins, "Introduction to multi-armed bandits," *arXiv preprint arXiv:1904.07272*, 2019.

[266] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans Comput*, vol. 69, no. 6, pp. 883–893, 2020.

[267] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *arXiv preprint arXiv:1710.03748*, 2017.

[268] S. Li, S. Bing, and S. Yang, "Distributional advantage actor-critic," *arXiv preprint arXiv:1806.06914*, 2018.

[269] 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, 2019.

[270] M. Min, X. Wan, L. Xiao, Y. Chen, M. Xia, D. Wu, and H. Dai, "Learning-based privacy-aware offloading for healthcare iot with energy harvesting," *IEEE Internet Things J*, vol. 6, no. 3, pp. 4307–4316, 2019.

[271] X. He, J. Liu, R. Jin, and H. Dai, "Privacy-aware offloading in mobile-edge computing," in *GLOBECOM 2017 - 2017 IEEE Global Commun. Conf.*, 2017, pp. 1–6.

[272] X. He, H. Dai, and P. Ning, "Improving learning and adaptation in security games by exploiting information asymmetry," in *2015 IEEE Conf. Comput. Commun.*

(INFOCOM), 2015, pp. 1787–1795.

[273] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Trans Netw Service Manag*, vol. 17, no. 4, pp. 2536–2549, 2020.

[274] K. Wang, X. Yu, W. Lin, Z. Deng, and X. Liu, "Computing aware scheduling in mobile edge computing system," *Wirel Netw*, pp. 1–17, 2019.

[275] N. Li, L. Hu, Z.-L. Deng, T. Su, and J.-W. Liu, "Research on gru neural network satellite traffic prediction based on transfer learning," *Kluw Commun*, vol. 118, no. 1, pp. 815–827, 2021.

[276] W. Sun, J. Liu, and Y. Yue, "Ai-enhanced offloading in edge computing: When machine learning meets industrial iot," *IEEE Netw*, vol. 33, no. 5, pp. 68–74, 2019.

[277] M. H. Moghadam and S. M. Babamir, "Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling," *J Comput Sci*, vol. 24, pp. 402 – 412, 2018.

[278] R.-S. Chang, J.-S. Chang, and S.-Y. Lin, "Job scheduling and data replication on data grids," *Future Gener Comp Sy*, vol. 23, no. 7, pp. 846–860, 2007.

[279] T. T. Sung, J. Ha, J. Kim, A. Yahja, C.-B. Sohn, and B. Ryu, "Deepsocs: A neural scheduler for heterogeneous system-on-chip (soc) resource scheduling," *Electronics*, vol. 9, no. 6, p. 936, 2020.

[280] T. Dong, F. Xue, C. Xiao, and J. Li, "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment," *Concurrency Computation: Pract. Experience*, vol. 32, no. 11, p. e5654, 2020.

[281] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans Parallel Distrib Syst*, vol. 13, no. 3, pp. 260–274, 2002.

[282] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 81–97.

[283] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39 974–39 982, 2019.

[284] A. Chowdhury, S. A. Raut, and H. S. Narman, "Da-drls: Drift adaptive deep reinforcement learning based scheduling for iot resource management," *J Netw Comput Appl*, vol. 138, pp. 51 – 65, 2019.

[285] Z. Wei, F. Liu, Y. Zhang, J. Xu, J. Ji, and Z. Lyu, "A q-learning algorithm for task scheduling based on improved svm in wireless sensor networks," *Comm Com Inf Sc*, vol. 161, pp. 138 – 149, 2019.

[286] K. Shah and M. Kumar, "Distributed independent reinforcement learning (dirl) approach to resource management in wireless sensor networks," in *2007 IEEE Int. Conf. Mobile Adhoc Sensor Syst.*, 2007, pp. 1–9.

[287] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "On-edge multi-task transfer learning: Model and practice with data-driven task allocation," *IEEE Trans Parallel Distrib Syst*, vol. 31, no. 6, pp. 1357–1371, 2020.

[288] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," *Cluster comput.*, vol. 18, no. 4, pp. 1595–1607, 2015.

[289] J. G. Shanthikumar, S. Ding, and M. T. Zhang, "Queueing theory for semiconductor manufacturing systems: A survey and open problems," *IEEE Trans Autom Sci Eng*, vol. 4, no. 4, pp. 513–522, 2007.

[290] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+r queuing systems," *IEEE Trans Parallel Distrib Syst*, vol. 23, no. 5, pp. 936–943, 2012.

[291] A. M. Kintsakis, F. E. Psomopoulos, and P. A. Mitkas, "Reinforcement learning based scheduling in a workflow management system," *Eng Appl Artif Intel*, vol. 81, pp. 94 – 106, 2019.

[292] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *IEEE J Sel Areas Commun*, vol. 37, no. 6, pp. 1248–1261, 2019.

[293] J. Zhou, "Real-time task scheduling and network device security for complex embedded systems based on deep learning networks," *Microprocess Microsy*, vol. 79, p. 103282, 2020.

[294] D. Cui, W. Ke, Z. Peng, and J. Zuo, "Multiple dags workflow scheduling algorithm based on reinforcement learning in cloud computing," in *Comput. Intell. Intell. Syst.*, 2016, pp. 305–311.

[295] C. Morariu, O. Morariu, S. Răileanu, and T. Borangiu, "Machine learning for predictive scheduling and resource allocation in large scale manufacturing systems," *Comput Ind*, vol. 120, p. 103244, 2020.

[296] G. Rjoub, J. Bentahar, O. Abdel Wahab, and A. Saleh Bataineh, "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems," *Concurrency Comput.: Pract. Experience*, vol. 33, no. 23, p. e5919, 2021.

[297] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *2016 IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–6.

[298] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid mec networks," *IEEE Internet Things J*, vol. 7, no. 7, pp. 6252–6265, 2020.

[299] M.-S. Yang, "A survey of fuzzy clustering," *Math Comput modelling*, vol. 18, no. 11, pp. 1–16, 1993.

[300] Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans Intell Syst Technol*, vol. 10, no. 6, Oct. 2019.

[301] A. Shahidinejad and M. Ghobaei-Arani, "Joint computation offloading and resource provisioning for e dge-cloud computing environment: A machine learning-based approach," *Softw. Pract. Experience*, vol. 50,

no. 12, pp. 2212–2230, 2020.

[302] H. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proc. 4th ACM workshop Mobile cloud comput. services*, 2013, pp. 9–16.

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

[304] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J*, vol. 5, no. 4, pp. 2633–2645, 2018.

[305] S. Xu, Q. Liu, B. Gong, F. Qi, S. Guo, X. Qiu, and C. Yang, "Rjcc: Reinforcement-learning-based joint communicational-and-computational resource allocation mechanism for smart city iot," *IEEE Internet Things J*, vol. 7, no. 9, pp. 8059–8076, 2020.

[306] L. Bracciale and P. Loreti, "Lyapunov drift-plus-penalty optimization for queues with finite capacity," *IEEE Commun Lett*, vol. 24, no. 11, pp. 2555–2558, 2020.

[307] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Trans Ind Informat*, vol. 14, no. 10, pp. 4642–4655, 2018.

[308] C. J. Watkins and P. Dayan, "Q-learning," *Mach learn*, vol. 8, no. 3-4, pp. 279–292, 1992.

[309] N. Kiran, C. Pan, S. Wang, and C. Yin, "Joint resource allocation and computation offloading in mobile edge computing for sdn based wireless networks," *J Commun Netw*, vol. 22, no. 1, pp. 1–11, 2020.

[310] L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, "Joint computation offloading and multiuser scheduling using approximate dynamic programming in nb-iot edge computing system," *IEEE Internet Things J*, vol. 6, no. 3, pp. 5345–5362, 2019.

[311] G. Tesauro *et al.*, "Temporal difference learning and td-gammon," *Commun ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[312] F. Jiang, K. Wang, L. Dong, C. Pan, and K. Yang, "Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale mec networks," *IEEE Internet Things J*, vol. 7, no. 10, pp. 9278–9290, 2020.

[313] H. Ye and G. Y. Li, "Deep reinforcement learning based distributed resource allocation for v2v broadcasting," in *2018 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2018, pp. 440–445.

[314] K. Wang, Y. Tan, Z. Shao, S. Ci, and Y. Yang, "Learning-based task offloading for delay-sensitive applications in dynamic fog networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 399–11 403, 2019.

[315] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. learn.*, vol. 47, no. 2, pp. 235–256, 2002.

[316] P. Dai, Z. Hang, K. Liu, X. Wu, H. Xing, Z. Yu, and V. C. S. Lee, "Multi-armed bandit learning for computation-intensive services in mec-empowered vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7821–7834, 2020.

[317] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans Wireless Commun*, vol. 18, no. 11, pp. 5141–5152, 2019.

[318] F. Xu, F. Yang, S. Bao, and C. Zhao, "Dqn inspired joint computing and caching resource allocation approach for software defined information-centric internet of things network," *IEEE Access*, vol. 7, pp. 61 987–61 996, 2019.

[319] Y. Fan, Z. Zhang, and H. Li, "Message passing based distributed learning for joint resource allocation in millimeter wave heterogeneous networks," *IEEE Trans Wireless Commun*, vol. 18, no. 5, pp. 2872–2885, 2019.

[320] Y. Wu, E. Dobriban, and S. Davidson, "Deltagrad: Rapid retraining of machine learning models," in *Int. Conf. Mach. Learn.* PMLR, 2020, pp. 10 355–10 366.

[321] R. Kolcun, D. A. Popescu, V. Safronov, P. Yadav, A. M. Mandalari, Y. Xie, R. Mortier, and H. Haddadi, "The case for retraining of ml models for iot device identification at the edge," *arXiv preprint arXiv:2011.08605*, 2020.

[322] H. Djigal, F. Jun, and J. Lu, "Secure framework for future smart city," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017, pp. 76–83.

[323] Z. Hong, Z. Wang, and W. Cai, "Blockchain-empowered fair computational resource sharing system in the d2d network," *Future Internet*, vol. 9, p. 85, 11 2017.

[324] Y. Guo and C. Liang, "Blockchain application and outlook in the banking industry," *Financial Innov.*, vol. 2, no. 1, p. 24, 2016.

[325] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harv Bus Rev*, vol. 1, no. 9, pp. 2–5, 2017.

[326] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proc. 17th IEEE/ACM int. symp. cluster cloud grid comput.* IEEE Press, 2017, pp. 468–477.

[327] E. J. De Aguiar, B. S. Faiçal, B. Krishnamachari, and J. Ueyama, "A survey of blockchain-based strategies for healthcare," *ACM Comput Surv*, vol. 53, no. 2, Mar. 2020.

[328] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song, "Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 695–708, 2019.

[329] J. Xie, H. Tang, T. Huang, F. R. Yu, R. Xie, J. Liu, and Y. Liu, "A survey of blockchain technology applied to smart cities: Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2794–2830, thirdquarter 2019.

[330] X. Wang, X. Ren, C. Qiu, Z. Xiong, H. Yao, and

V. Leung, "Integrating edge intelligence and blockchain: What, why, and how," 2022.

[331] Y. Liu, K. Wang, Y. Lin, and W. Xu, "LightChain: A lightweight blockchain system for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3571–3581, 2019.

[332] A. Sufian, A. Ghosh, A. S. Sadiq, and F. Smarandache, "A survey on deep transfer learning to edge computing for mitigating the covid-19 pandemic," *J Syst Architect*, vol. 108, p. 101830, 2020.

[333] B. Yang, X. Cao, C. Yuen, and L. Qian, "Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of uavs," *IEEE Internet Things J*, vol. 8, no. 12, pp. 9878–9893, 2021.

[334] T. V. Phan, S. Sultana, T. G. Nguyen, and T. Bauschert, "Qq - transfer: A novel framework for efficient deep transfer learning in networking," in *2020 Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, 2020, pp. 146–151.

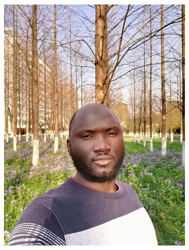
[335] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans Knowl Data Eng*, vol. 22, no. 10, pp. 1345–1359, 2010.

[336] K. I-Kai Wang, X. Zhou, W. Liang, Z. Yan, and J. She, "Federated transfer learning based cross-domain prediction for smart manufacturing," *IEEE Trans Ind Informat*, pp. 1–1, 2021.

[337] Y. Pathak, P. Shukla, A. Tiwari, S. Stalin, S. Singh, and P. Shukla, "Deep transfer learning based classification model for covid-19 disease," *IRBM*, 2020.

[338] S. Shao, S. McAleer, R. Yan, and P. Baldi, "Highly accurate machine fault diagnosis using deep transfer learning," *IEEE Trans Ind Informat*, vol. 15, no. 4, pp. 2446–2455, 2019.

[339] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for radio resource allocation with diverse quality-of-service requirements in 5g," *IEEE Trans Wireless Commun*, vol. 20, no. 4, pp. 2309–2324, 2021.



Hamza Djigal received the BSc degree in mathematics from Cheikh Anta Diop University, Dakar, Senegal, in 2009, the MSc degree in software engineering from Central China Normal University, Wuhan, China, in 2014, and the PhD degree in computer science and technology from Hohai University, Nanjing, China, in 2020. He is currently a postdoctoral researcher with the School of Computer Science, Nanjing University of Posts and Telecommunications, China. His main research interests include parallel and distributed computing, MEC, deep learning, and resource allocation in heterogeneous MEC networks.

Jia Xu (*SM*'21) received the M.S. degree in School of Information and Engineering from Yangzhou University, Jiangsu, China, in 2006 and the PhD. degree in School of Computer Science and Engineering from Nanjing University of Science and Technology, Jiangsu, China, in 2010. He is currently a professor in the School of Computer Science at Nanjing University of Posts and Telecommunications, China. He was a visiting Scholar in the Department of Electrical Engineering & Computer Science at Colorado School of Mines from Nov. 2014 to May. 2015. His main research interests include crowdsourcing, edge computing and wireless sensor networks.

Linfeng Liu (*M*'13) received the B. S. and Ph. D. degrees in computer science from the Southeast University, Nanjing, China, in 2003 and 2008, respectively. At present, he is a Professor in the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, China. His main research interests include the areas of vehicular ad hoc networks, wireless sensor networks and multi-hop mobile wireless networks. He has published more than 80 peer-reviewed papers in some technical journals or conference proceedings, such as IEEE TMC, IEEE TPDS, ACM TAAS, IEEE TSC, IEEE TVT, IEEE IoTJ, Computer Networks, Elsevier JPDC.

Yan Zhang (*F*'19) received the Ph.D. degree from the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore. He is currently a Full Professor with the Department of Informatics, University of Oslo, Oslo, Norway. His research interests include next-generation wireless networks leading to 5G beyond/6G, green and secure cyber-physical systems (e.g., smart grid and transport). Dr. Zhang is an Editor for several IEEE publications, including the IEEE COMMUNICATIONS MAGAZINE, IEEE NETWORK, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, and IEEE INTERNET OF THINGS JOURNAL. He is a Chair in a number of conferences, including the IEEE GLOBECOM and IEEE PIMRC. He is an IEEE Vehicular Technology Society Distinguished Lecturer. He is a Fellow of IET. He is the Chair of IEEE Communications Society Technical Committee on Green Communications and Computing. Prof. Zhang was a recipient of the Highly Cited Researcher Award (Web of Science top 1% most cited) by Clarivate Analytics.