Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2022.021394

REVIEW

A Review of the Current Task Offloading Algorithms, Strategies and Approach in Edge Computing Systems

Abednego Acheampong[1], **Yiwen Zhang**[1,*], **Xiaolong Xu**[2] **and Daniel Appiah Kumah**[2]

[1]School of Computer Science and Technology, Anhui University, Hefei, 230039, China

[2]School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China

*Corresponding Author: Yiwen Zhang. Email: zhangyiwen@ahu.edu.cn

Received: 12 January 2022 Accepted: 18 March 2022

ABSTRACT

Task offloading is an important concept for edge computing and the Internet of Things (IoT) because computation-intensive tasks must be offloaded to more resource-powerful remote devices. Task offloading has several advantages, including increased battery life, lower latency, and better application performance. A task offloading method determines whether sections of the full application should be run locally or offloaded for execution remotely. The offloading choice problem is influenced by several factors, including application properties, network conditions, hardware features, and mobility, influencing the offloading system's operational environment. This study provides a thorough examination of current task offloading and resource allocation in edge computing, covering offloading strategies, algorithms, and factors that influence offloading. Full offloading and partial offloading strategies are the two types of offloading strategies. The algorithms for task offloading and resource allocation are then categorized into two parts: machine learning algorithms and non-machine learning algorithms. We examine and elaborate on algorithms like Supervised Learning, Unsupervised Learning, and Reinforcement Learning (RL) under machine learning. Under the non-machine learning algorithm, we elaborate on algorithms like non(convex) optimization, Lyapunov optimization, Game theory, Heuristic Algorithm, Dynamic Voltage Scaling, Gibbs Sampling, and Generalized Benders Decomposition (GBD). Finally, we highlight and discuss some research challenges and issues in edge computing.

KEYWORDS

Task offloading; machine learning; algorithm; game theory; dynamic voltage scaling

1 Introduction

The pervasiveness of the smart Internet of Things (IoT) [1] enables many electric sensors and devices to be connected and generates a large amount of data flow [2]. IoT applications, such as smart homes, disease prevention and control, and telecommunication are affecting and transforming our lives with the introduction of the IoT. These applications are time-sensitive and take a lot of energy, memory, and computational resources. Although IoT devices are increasingly becoming more powerful, the battery, CPU, and memory are still insufficient when running large apps on a single device. Computational offloading is one of the remedies to the difficulties mentioned above, in which the computation workloads are transferred to another system for execution [3].

Meanwhile, with robust data center architectures, the cloud is a well-tested and used option that may enhance the resource capabilities of end devices. Furthermore, the cloud is well-equipped with the essential automation tools and features to provide the required transparency to end devices while concealing this resource extensions' difficulty and logistical intricacies [4]. Consequently, the practice of offloading computation-intensive tasks of resource-intensive applications from the end devices to centralized Cloud infrastructure is a well-explored solution [4–6]. Nevertheless, applications such as media processing, online gaming, Augmented Reality (AR), Virtual Reality (VR), self-driving automotive applications, and recommendation systems [7–9] run on a wide range of mobile devices. When the focus of new applications shifted to high throughput and low latency communications, the cloud began to expose its significant drawbacks resulting from the resource constraints of these ubiquitous mobile devices. Running such resource-hungry applications requires fast response time data rates [10].

The need for alternative solutions arose due to the long distance between end devices and Cloud infrastructure, an unreliable, intermittent transport network, the cost of traversing the backhaul network, and the increased security surface throughout this long communication path [4]. *Edge computing* is a new computing paradigm that employs edge servers close to users. As alternative paradigms of edge computing, three related notions (technologies) are proposed: Open Edge Computing cloudlets [3], Multi-access Edge Computing (MEC) of European Telecommunications Standards Institute (ETSI) (2016), and Fog computing of OpenFog Consortium (2016) [11]. This novel infrastructure component, which establishes an extra resource layer between consumer devices and the cloud, can reduce rising bandwidth usage in backhaul, transport, and Cloud networks and communication latency and support real-time applications. Moreover, there is some research on AI's optimization of edge performance [12]. End devices, for example, can now offload resource-intensive operations to a nearby Edge device, reducing total execution time while avoiding the need for additional communication routes to a distant Cloud infrastructure. This method, also known as task offloading or computation offloading, improves the user's experience by reducing latency and improving energy efficiency for battery-powered devices [4]. Computation offloading entails dividing power-hungry mobile apps to use cloud resources from afar. Parts of code profiled as computation and energy heavy are identified for execution on cloud servers (partial offloading). Alternatively, the offloading method is used to decide whether to offload all tasks to a cloudlet or execute all tasks locally (Full offloading). Several research publications have highlighted the benefits of offloading to save energy and minimize system latency on mobile devices.

A distant or local execution technique, a transmission approach, and a result send-back procedure are all part of task offloading and resource allocation. The major components of the offloading scheme are as follows. (A) Job partitioning: If a task can be partitioned, we must divide it optimally before offloading it. If not, the entire task should be offloaded to an edge server or run locally. (B) Offloading decision: whether to offload to edge server(s) for execution or perform task locally. (C) Allocation of resources: determining the number of resources required. These resources include computing, communication, and energy, allocated for the tasks or components [11]. Fig. 1 shows a typical offloading process.

This study provides a thorough examination of task offloading in edge computing, including offloading strategies (full or partial), algorithms, and factors that influence offloading. The following are the contributions of this article:

1. A comprehensive survey of the current task offloading and resource allocation in edge computing, including offloading strategies, offloading algorithms, and factors affecting offloading, is presented.
2. We present the partitioning of offloading strategies into either full or partial offloading and present some comprehensive surveys.
3. We group several recent tasks offloading and resource allocation algorithms into two main categories. (i) machine learning algorithms and (ii) non-machine learning algorithms.
4. We elaborate on the factors affecting offloading and discuss some research challenges and issues.

The following is how the rest of the paper is structured: We examine related work to this article, an overview of offloading, and our paper selection criteria in Section 2. Offloading performance metrics and factors affecting offloading are presented in Section 3. Classification of offloading algorithms into two groups, machine learning, and non-machine learning algorithm and offloading strategies, are discussed in Section 4. Section 5 presents an open discussion and analysis of the various metrics, algorithms, systems, and utilized tools. A comparison of the major utilized offloading algorithms and some research challenges are presented in Section 6. Finally, a conclusion is presented in Section 7.

2 Related Work

The authors of [7] studied offloading modeling in edge computing. The article discussed some major edge computing architectures and classified past computation offloading research into different groups. Furthermore, the authors explored various basic models proposed in offloading modeling, such as the channel, computation and communication, and energy harvesting models. The authors also go over several offloading modeling techniques, including (non-) convex optimization, Markov decision processes, game theory, Lyapunov optimization, and machine learning. In Mobile Cloud Computing, Ahmed et al. [13] investigated mobile application frameworks and analyzed optimization methodologies affecting design, deployment, and offloading performance. The review paper featured an appropriate classification as a benefit.

Saeik et al. [4] explored how the edge and cloud can be coupled to help with task offloading. They focused on mathematical, artificial intelligence, and control theory optimization processes that can be used to satisfy dynamic requirements in an end-to-end application execution strategy.

Shakarami et al. [10] proposed a survey paper on stochastic-based offloading approaches in various computation environments, including Mobile Cloud Computing (MCC), Mobile Edge Computing (MEC), and Fog Computing (FC), in which a classical taxonomy is presented to identify new mechanisms. Markov chain, Markov process, and Hidden Markov Models are the three core fields of the proposed taxonomy.

Shakarami et al. [14] proposed ML-based computation offloading strategies in the MEC environment in the form of classical taxonomy. Reinforcement learning-based mechanisms, supervised learning-based mechanisms, and unsupervised learning-based mechanisms are the three primary categories in the proposed taxonomy. These classes are then compared based on essential characteristics such as performance measures, case studies, strategies used, evaluation tools, and their benefits and drawbacks.

Zheng et al. [3] provided a complete overview of computation offloading in edge computing, covering scenarios, influencing variables, and offloading methodologies. Authors specifically covered crucial problems during the offloading process, such as whether to offload, where to offload, and what to offload.

Bhattacharya et al. [15] looked into the adaption techniques of offloading systems. By defining the variable characteristics in the offloading ecosystem, presenting offloading solutions that adapt to these parameters, and highlighting the accompanying gains in user Quality of Experience, a monocular picture of the task offloading challenge was presented.

Carvalho et al. [16] surveyed computation offloading in edge computing using artificial intelligence and non-artificial intelligence techniques. Under AI techniques, they considered machine learning-based techniques that provide promising results in overcoming the shortcomings of current approaches for computing offloading and categorized them into classes for better analysis. They went on to discuss a vehicular edge computing environment offloading use case. Guevara et al. [17] looked at the primary issues with QoS constraints for applications running in traditional computing paradigms like fog and cloud. After that, they demonstrated a typical machine learning classification method. The paper's key benefit is that it professionally focuses on machine learning. Shan et al. [18] classified offloading methods into distinct purposes, such as minimizing execution delay, minimizing energy consumption, and minimizing the tradeoff between energy consumption and execution latency

However, the above surveys have several flaws, including a lack of paper selection criteria and consideration for other non-machine learning algorithms. The majority of the papers focused only on the machine learning aspect, ignoring the powerful algorithms that are not machine learning-based. Furthermore, several studies do not clearly distinguish between full and partial offloading. These are some of the areas we hope to address in our study. First, we want to look at both machine learning and non-machine learning algorithms, provide a clear definition of full and partial offloading and examine some of the recent research that has been done using both techniques.

2.1 Overview of Offloading

With the prevalence of smart devices and cloud computing, the amount of data generated by Internet of Things devices has exploded. Furthermore, emerging IoT applications such as Virtual Reality (VR), Augmented Reality (AR), intelligent transportation systems, smart homes, smart health, smart factories. And other IoT applications require ultra-low latency for data communication and processing, which the cloud computing paradigm does not provide [19]. Mobile Edge Computing (MEC) is an efficient technique to bring user and edge servers closer by offloading a computation-intensive task on deployed computation servers at the user side and minimizing backhaul traffic created by apps to a remote cloud data center. MEC decreases the time it takes to complete calculation operations for delay-sensitive cloud computing applications and saves energy. The offloading process usually entails two important decisions: what tasks should be offloaded and where they should be offloaded to, or where the offloading should be done. The latter is particularly important because it directly influences the system's quality of service and performance. Several researchers have proposed novel approaches to enhance offloading. Some approaches are MDP based, RL based and even link-prediction based. Zhang et al. [20] used the link-prediction approach in their offloading model; similarly, Liu et al. [21] adopted link-prediction in a recommendation system, making link-prediction a versatile approach. Unfortunately, there is not much research on link-prediction-based offloading currently. Offloading can take place between the device to device, a device to edge server, edge server to edge server, edge server to the cloud, cloud to the edge server, and device to cloud. Fig. 2 shows a typical MEC architecture and shows the various places offloading can occur. As illustrated, this structure has six divisions classified into three main layers.

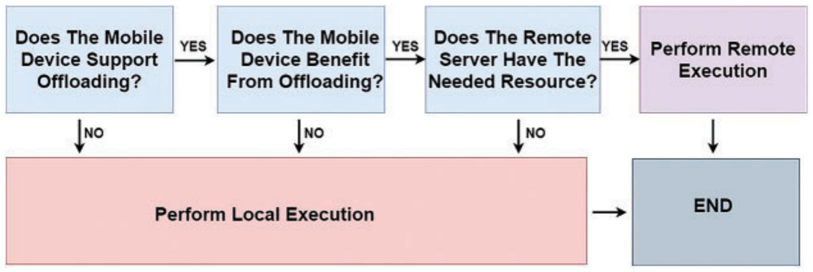
Does The Mobile Device Support Offloading?
YES
Does The Mobile Device Benefit From Offloading?
YES
Does The Remote Server Have The Needed Resource?
YES
Perform Remote Execution
NO
NO
NO
Perform Local Execution
END

Figure 1: A typical offloading process

- At the device layer (Device to device), IoT gadgets have become an integral part of our daily lives in recent years. They are both data producers and data consumers. Mobile devices can provide a limited number of computing resources, and most of these resources are idle most of the time. When a mobile device's compute capabilities are exceeded, it can split the application into smaller tasks and offload to adjacent mobile devices with idle processing resources. It can help reduce single device resource scarcity and increase overall resource consumption.
- At the edge layer (Device to edge server), most IoT devices cannot do sophisticated calculation tasks due to their low resources. A single device with limited resources (e.g., a smartphone) will not complete the tasks promptly. IoT devices can transfer compute activities to nearby edge servers, such as Cloudlets [22], or MEC servers [23], where they will be processed and evaluated, as shown in Fig. 1. It successfully lowers the cost and delay. It can help lessen the burden on embedded devices when running machine learning algorithms and enhance app performance.
- At the edge layer (edge server to edge server), edge computing can help IoT devices at the edge because of their computational capacity and quick response time. However, because a single edge server's processing capacity is limited, numerous edge nodes must be combined to maintain load balance and share data to provide cooperative services, such as collaborative edge.
- At the cloud layer (edge server to Cloud server), edge servers (e.g., MEC, Cloudlet, and Fog) have the processing and storage capability to complete most operations at the edge layer. However, they still require cloud services to store access data in many instances. Smart Healthcare uses fog to cloud offload to keep patient records for a long time. Due to the widespread distribution of resources, the edge server and cloud service must collaborate to complete a task. Fog computing requires the original spatial information provided by the cloud during rescue missions to direct rescuers to do on-site search and rescue activities. The fog and cloud give jobs to each other in this fashion and work together to provide service.

- At the cloud layer (Device to Cloud server), IoT devices may need to offload tasks to a remote cloud server for data storage or processing in some instances. Heavy tasks can be offloaded to powerful distant centralized clouds (e.g., Amazon EC2 [24], Microsoft Azure [25], and Google), and enormous volumes of data can be stored on Cloud storage. Mobile Cloud Computing (MCC) [26] benefits mobile consumers by prolonging battery life, enabling advanced applications, and increasing data storage capacities. Mobile games can offload computationally heavy operations, such as graphics rendering, to the cloud and display the results on the screen to interact with consumers. It can improve the gaming experience while conserving battery life on mobile devices.
- At the device layer (Edge server to device), IoT devices may need to receive external data for calculations in some cases, such as temperature, humidity, and other environmental parameters. Due to its restricted function and resources, it is difficult for an IoT device to measure and retain this information in significant amounts. As a result, IoT devices can connect to edge servers such as Cloudlet or MEC, located near IoTs. Edge servers have the computing power and may gather data from various sources. It can effectively lower the cost of connecting to the cloud and the load on equipment and transmission latency.
- Task offloading can also be divided into full offloading and partial offloading. Full offloading, also known as binary or coarse-grained or total offloading, occurs when the entire task is migrated to the edge cloud or edge for execution or when the entire task is executed locally. In contrast, partial offloading, also known as fine-grained offloading or dynamic offloading, occurs when the entire task is dynamically transmitted as little code as possible and offloads only the computation-intensive parts of the application (more details on offloading categories in Section 4). Fig. 3 shows the roadmap of this paper.

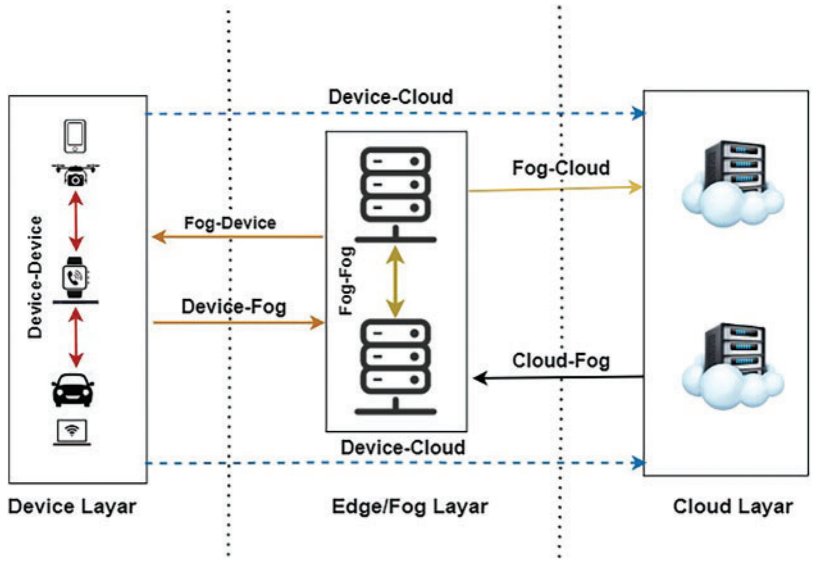

Figure 2: MEC architecture

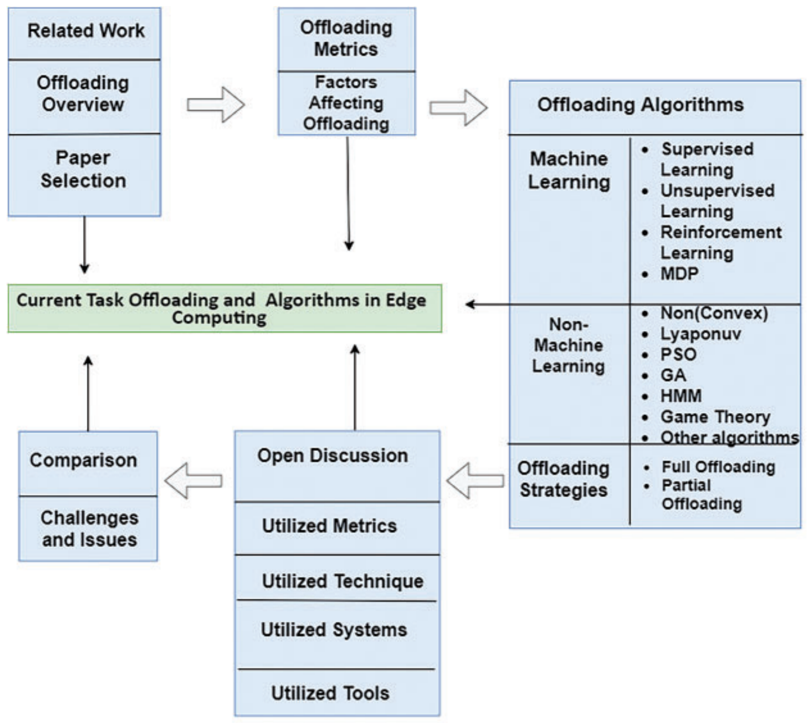

Figure 3: Roadmap of this paper

2.2 *Paper Selection Criteria*

A research approach for good papers in the MEC Offloading scenario is described in this part. Exploring and researching relevant papers is needed to build a more knowledge-rich survey.

2.2.1 Question Formulation

This survey intends to look at the essential features and techniques used in papers over time and the main concerns and obstacles in offloading, such as factors affecting offloading. Because an important goal of the current survey is to cover the entire study of MEC offloading and highlight relevant outstanding issues, specific key research questions must be answered to handle corresponding concerns.

Q1. What technique or algorithm is utilized in ML and Non-ML based offloading approaches? See Subsections 4.1 and 4.2.

Q2. What performance metrics are usually utilized in ML and Non-ML based offloading approaches? See Section 3.

Q3. What factors affect both ML and Non-ML based offloading approaches? See Subsection 3.1.

Q4. What evaluation tools are utilized for assessing the ML and Non-ML based approaches? See Subsection 5.4.

Q5. What utilized systems are considered in both ML and Non-ML based approaches? See Subsection 5.3.

Q6. What are the future research directions and open perspectives of ML and Non-ML based offloading approaches? See Subsection 6.1 .

Sections 3 to 6 try to elaborate more on the mentioned technical questions.

2.2.2 Article Selection

Suitable papers in the MEC have been explored in the academic databases. The principles of selecting articles in the process of exploring are summarized as follows:

1. Published papers between 2015 and 2021,
2. Published papers in the MEC,
3. Technical quality selection to choose appropriate papers in the MEC.

In the exploration process, appropriate keywords such as "mobile", "edge", "computing", "offloading", "MEC", "unsupervised learning", "deep learning", "reinforcement learning", "deep Q-networks", "game theory", "Lyapunov optimization", "supervised learning", "machine learning", "convex optimization", "heuristic", "Markov-decision process", and "resource allocation" were used. By narrowing the time limitations between 2015 and 2021, the exploration took place in April 2021. The outcome of the exploration was extraordinarily high in numbers since the issue of offloading spans many models in the literature, including stochastic and non-stochastic models with comprehensive techniques such as game theory, machine learning, and many more. As a result, 1147 articles were found due to the search. For the first step, 900 irrelevant papers were discarded by analyzing several essential components, such as the title, abstract, contributions, and conclusion. Following that, 152 papers were discarded as low quality after examining the organization of the remaining papers. As a result, the MEC approved 94 of the remaining articles. After removing 20 duplicate papers and 4 books, the remaining 71 papers related to ML and Non-ML are included in the current study, as shown in Fig. 4.

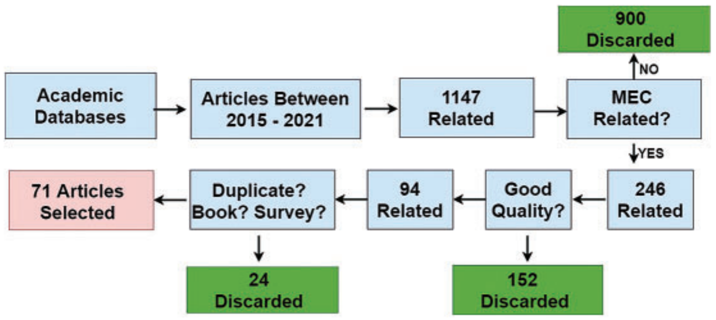

Figure 4: Paper selection

3 Offloading Metrics

The following technical question is addressed in this section:

Q2: What performance metrics are usually utilized in ML and Non-ML based offloading approaches

Researchers in the task offloading domain consider several offloading metrics; these metrics are either considered jointly as a multi-objective problem or individually. This paper will briefly explain these offloading metrics in the task offloading domain. When solving the task offloading problem, several different objectives may be applied. An objective function helps to formulate these goals and guide the offloading solution formally and mathematically. The objectives come in the form of offloading metrics. The offloading metrics covered include latency, energy, cost, bandwidth, and response time.

A. Energy

The overall energy required by offloading comprises the energy consumed to send the task from the device to the server, the energy consumed to execute the task in the edge server, and the energy consumed to return the results to the device [23,27]. Because mobile and IoT devices are typically battery-powered, maximizing the battery's lifetime through lowering the device's energy usage is a big concern. Inevitably, it is logical to expect that the most significant energy savings can be achieved by using a full offloading technique. Several other energy suppliers must be considered even when a total offloading strategy is used. From a complete, network-wide perspective, it is easy to see how the problem is moved to the Edge or Cloud infrastructures. As a result, energy consumption minimization must be followed at all tiers of an end-to-end communication paradigm. A variety of measures can be used to assess this goal; the most frequent is the average power consumption, which is calculated by summing the power consumption of the hardware equipment. Energy consumption, presented as power use over time, is another option. Typically, reducing power use leads to reducing energy consumption as well.

B. Latency

The total time it takes to transmit the work to the edge servers, the time it takes to execute the task on the servers, and the time it takes to return the results to the device are all factors in the offloading requirement [28]. There are a lot of various delay components that contribute to this. The first delay source is task processing, which can occur when a task is performed locally on the device, at the edge, or in the cloud [29,30]. When offloading a task to a remote edge or cloud location, the transmission and propagation delays at the various infrastructure tiers must be considered. Furthermore, processing and queuing delays at various processing and forwarding devices must be considered. Finally, during the task offloading decision, appropriately splitting the task can be an additional delay factor [31]. The delay goal can be described as either minimizing the average delay of each activity or minimizing the overall delay of all the mobile application's related tasks. This goal is equitable to the resources available and the network circumstances.

C. Cost

Different academics use different methods to calculate the cost of MEC offloading. The costs involved by delivering the tasks via the transmission media, executing them on the server, and receiving an acceptable answer from the request source are referred to as the cost. These costs are determined by the task's location, reaction time, task demand, and task energy consumption. Since consumed energy and delay are the essential components in computing total cost, making a trade-off among these two relevant metrics is critical. As a result, overall execution costs are regarded similarly to the two previously described measures, which comprise local and remote execution costs, as well as the processing and buffering delays.

D. Bandwidth

A crucial constraint is the available bandwidth at the access network and how different users might share it to offload workloads. Nevertheless, its significant impact on task offloading performance can also be regarded as a goal. Proper spectrum allocation becomes critical because available bandwidth is limited, particularly in IoT and congested cellular networks. Spectrum allocation is frequently linked to each end user's transmission rate and power level and the duration of each device's transmission to distribute bandwidth utilization better. When attempting to deploy spectrum utilization effectively, a useful statistic is to analyze spectrum usage regarding the number of offloaded jobs, power transfer, and bandwidth consumption [32]. In light of the dynamic wireless settings, the best bandwidth scheduling must consider the time-varying channel condition and the task average response time.

E. Response Time

The overall time it takes for a user to receive a response is called response time. Changes in a system's processing time and latency, which occur due to changes in hardware resources or utilization, can impact it. In the MEC scenario, *response time* is defined as the time between offloading tasks from local devices to remote servers and obtaining the appropriate response in the specified devices as a measure of performance. There are discrepancies between the system's response time and latency. The overall time between initiating a request and receiving an appropriate response is called response time. Latency, on the other hand, is defined as the time it takes for a transmitted request to arrive at its destination and be processed.

3.1 Factors Affecting Offloading

The following technical question is addressed in this section:

Q3: What factors affect both ML and Non-ML based offloading approaches

1) Wireless Channel

When it comes to computing offloading, the state of the wireless network is a critical aspect that has a considerable impact on offloading decisions. Wireless is the most common method of communication between edge devices and servers. Wireless channels have reflection, refraction, and multipath fading, which are not present in wired channels [11]. The fluctuations in channel strength across time and frequency are a defining feature of the wireless channel. The variants can be divided into two groups: (I) Fading on a large scale occurs when the mobile moves a distance on the magnitude of the cell size. It is typically frequency-independent due to shadowing by massive objects like buildings and hills and distance due to signal path loss. (II) Small-scale fading is caused by constructive and destructive interference between the transmitter and receiver's many signal channels; it occurs at the spatial scale of the carrier wavelength and is frequency dependent [33]. A typical channel's accessibility pattern can be classified as stochastic or deterministic. The former is only available sometimes, but the latter is expected to be available at all times [10].

2) Bandwidth and Network Interference

The data send rate is determined by the bandwidth between the IoT device and the server, and the data transmit time is affected. In some circumstances, the data transmission time between the device and the server may outweigh the offloading delay, resulting in too long latency to meet the time constraint. Furthermore, network interference is difficult to anticipate, influenced by device mobility, bandwidth variance, network congestion, and the distance between devices and servers. Network interference significantly impacts the offloading system's ability to fulfill latency-constrained applications [34].

3) End Device

Mobile devices, sensors, and IoT devices are just a few examples of edge devices. Their hardware architectures, processing capabilities, storage capacity, and operating systems are distinct. As a result, device heterogeneity will impact offloading effects such as execution time and energy usage. Moreover, device mobility significantly impacts offloading; if mobile users are continually moving around, this may generate interference amongst users. Furthermore, in the case of an IoV or a UAV, if the device, whether it is an automobile or a UAV, departs from the offloading range before the results obtained are provided, the device will most likely need to find other offloading sources, which will increase delay and waste device energy [35].

4) Near-End Device

The effect of computation offloading is mostly determined by the servers that provide processing task functionality. Before making an offloading decision, end devices should examine the computation capabilities of servers, available resources, distance, and access technologies. Computing capability refers to the rate with which data is processed at a server; it is one of the most critical, but not the only, considerations for deciding whether or not to offload. The response time is also affected by a lack of resources. When a task is offloaded to a server, but the CPU is overcrowded or occupied by other tasks, the task is paused and waits for the CPU, increasing response time and affecting offloading [36].

4 Offloading Algorithms

The following technical question is addressed in this section:

Q1. What technique or algorithm is utilized in ML and Non-ML based offloading approaches?

In this paper, we will group the offloading techniques adopted in mobile edge computing into two categories, (1) machine learning offloading algorithms, illustrated in Table 1, (2) non-machine learning offloading algorithms, illustrated in Table 2. First of all, most of the existing offloading reviews focus on machine learning methods, and provide very detailed classification methods [14]. Second, few reviews focus on non-machine learning methods. Therefore, this paper considers offloading methods into machine learning and non-machine learning, which can include all offloading comprehensively. Finally, dividing offloading into machine learning and non-machine learning can give us a better understanding of the development of offloading.

4.1 Machine Learning Offloading Algorithms

Here we will introduce several machine learning algorithms that have been adopted and invented by researchers to tackle problems with edge computing offloading. See Table 1 for more details.

4.1.1 What is Machine Learning?

According to IBM, machine learning is a branch of artificial intelligence (AI) and computer science that focuses on using data and algorithms to imitate the way that humans learn, gradually improving its accuracy [37]. Generally, we will define machine learning as an application of artificial intelligence (AI) that allows systems to learn and improve from experience without being explicitly programmed automatically. Machine learning focuses on developing computer programs that can access data and use it to learn for themselves [38]. Machine learning can be grouped into three categories.

Machine learning algorithms under **supervised algorithms** can use labeled data to apply what they have learned to new data and anticipate future events. The supervised learning algorithm creates an inferred function to generate predictions about the output values based on examining a known training dataset. After enough training, the algorithm can provide targets for any new instance. The supervised learning algorithm can also evaluate its output to the correct, expected result and detect failures, allowing the model to be modified as needed. Supervised learning is commonly utilized in bioinformatics, object identification, speech recognition, pattern recognition, handwriting recognition, and spam detection.

On the other hand**, unsupervised machine learning algorithms** are utilized when the trained data is not classed or annotated. Unsupervised learning investigates how machines might deduce a function from unlabeled data to determine underlying patterns. The algorithm does not sort out the proper output, but it examines the data and can deduce hidden patterns from unlabeled data using datasets. Liao et al. [39] utilized an unsupervised machine learning algorithm called clustering in their paper, Coronavirus Pandemic Analysis Through Tripartite Graph Clustering in Online Social Networks.

Reinforcement learning algorithms are machine learning algorithms that interact with their environment by creating actions and recognizing failures or rewards. Trial and error search and rewards are the most crucial aspects of reinforcement learning. This technology allows machines and software agents to dynamically select the appropriate behavior in a given environment to improve their efficiency. A primary reward feedback signal, called the reinforcement signal, is required for the agent to learn which behavior is superior. Model-based and model-free approaches are the two types of reinforcement learning methods in general. Model-based learning, which includes online and deep learning, is commonly used as a transition function, trial-and-error, and planning algorithms. Q-Learning (RL), Deep Q-Learning (DRL), are examples of model-free, which usually act as an erroneous model.

In this paper, the machine learning offloading algorithms that will be considered include; Support Vector Machine (SVM), Deep Neural Networks, Decision Tree (DT), Instance-Based Learning (IBL), Analytic Hierarchy Process (AHP), Markov Model (MM), Q-Learning, Deep Q-Learning, (or Deep Reinforcement Learning) and Actor-Critic Learning, example Deep Deterministic Policy Gradient (DDPG).

A) Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a standard Supervised Learning technique that may solve classification and regression issues. However, it is primarily utilized in Machine Learning for Classification difficulties. Because the Support Vector Machine (SVM) can handle classification issues quickly [40,41], it may also be used to make offloading decisions by transforming decision problems into classification problems. It allows for more accurate and speedy offloading progress.

Wu et al. [42] used an SVM for a vehicular edge computing scenario. The authors proposed an efficient offloading algorithm based on SVM to satisfy the fast-offloading demand in vehicular networks. Majeed et al. [43] Investigated the accuracy of offloading decisions to a cloud server and its impacts on overall performance. They then proposed an accurate decision-making system for mobile systems' adaptive and dynamic nature by utilizing SVM learning techniques for making offloading decisions locally or remotely. According to the authors, the proposed scheme can achieve up to 92% accuracy in decision-making and reduce energy consumption. Wang et al. [44] investigated the problem of energy minimization consumption for task computation and transmission in cellular networks. They formulated an optimization problem to minimize the energy consumption for task computation and transmission. They then proposed a Support Vector Machine (SVM)-based federated learning algorithm to determine the user association proactively. With the user association given, edge servers or BS can collect information related to the computational task of its associated users, using which the transmit power and task allocation of each user will be optimized, and the energy consumption of each user is minimized.

B) Deep Learning

DNN is a multi-layered ANN (Artificial Neural Network) commonly used for prediction, anomaly detection [45], and optimization issues to identify the best solution from under-trained input by modifying mathematics suitably. It is worth noting DNN itself can be vulnerable to attacks like adversarial samples [46]. While using the DNN technique, various parameters like the number of layers, initial weight, and learning rate must be carefully evaluated to avoid overfitting and computation time. These problems necessitate much computational power, in contrast to the offloading ecosystem's time-and resource-intensive entities. There are several use cases of a neural network; Hou et al. [47] used an encoder-decoder module to solve a time series problem. Moreover, Zhang et al. [48] utilized a Multilayered Perceptron (MLP) and LSTM modules to build a recommendation system for predicting live streaming services. Moreover, Wang et al. [45] used a deep residual Convolutional Neural Network (CNN) for anomaly detection in Industrial Control Systems (ICS) based on transfer learning.

Ali et al. [49] proposed a novel energy-efficient offloading algorithm based on deep learning. To train an intelligent decision-making algorithm that selects an optimal set of application components based on users' remaining energy, energy consumption by application components, network conditions, computational load, amount of data transfer, and delays in communication. Yu et al. [50] considered a small cell-based mobile edge cloud network and proposed a partial computation offloading framework (Deep supervised learning). The algorithm considers the varying wireless network state and the availability of resources to minimize the offloading cost for the MEC network. Huang et al. [28] aimed to solve the energy consumption problem and improve the quality of service, so they proposed a distributed deep learning-based offloading algorithm (DDLO) for MEC networks, where multiple parallel DNNs are used to generate offloading decisions. Zhao et al. [51] used the ARIMA-BP model to estimate the edge cloud's computation capacity to optimize energy for delay restrictions in the MEC environment through selective offloading. They proposed ABSO (ARIMA-BP-based Selective Offloading). To acquire the offloading policy, they devised a selective offloading algorithm.

C) Decision Tree (DT)

The decision tree is a machine learning model that may achieve high accuracy in various tasks while also being very easy to understand. What distinguishes decision trees from other ML models is their clarity of information representation. After being trained, a decision tree's "knowledge" is immediately articulated into a hierarchical structure. This structure organizes and presents information so that even non-experts can understand it. Offloading decision trees is useful because they set out the problem so that all choices may be evaluated and allow researchers to analyze the possible consequences of a decision comprehensively.

Rego et al. [26] proposed a scheme that involves using decision trees and software-defined networking to tackle general offloading challenges, specifically, the challenge of when to offload, where to offload, and what metrics to monitor. The proposed scheme further handles the offloading decision-making process and supports user mobility. Likewise, in [52], they proposed a scheme that utilized a decision tree for the offloading decision-making process and proposed an adaptive monitoring scheme to keep track of the metrics relevant to the offloading decision.

D) Instance Base Learning (IBL) (K-Nearest Neighbor (KNN))

K-Nearest neighbor is a regression and a classification machine learning algorithm or technique. However, in industry, it is mainly used to tackle classification and prediction problems. K-Nearest Neighbor assesses the labels of a set of data points surrounding a target data point to provide a prediction about the data point's class. Because it delivers highly precise predictions, the KNN algorithm can compete with the most accurate model. As a result, the KNN algorithm can be used for applications that need high precision, such as computation and task offloading.

Crutcher et al. [53] took a new approach with their proposed model, integrating the k-Nearest Neighbor method (kNN). They used aspects from Knowledge-Defined Networking (KDN) to create realistic estimates for historical data offloading costs. A predicted metric exists for each dimension of the high-dimensional feature space. After assessing the costs of computation offloading, input features for a hyper-profile and position node are computed. An ML-based query, the kNN, is executed within this hyper-profile.

E) Analytic Hierarchy Process (AHP)

AHP is a simple and effective hierarchical strategy for analyzing and organizing multi-objective decisions based on mathematics and psychology. It consists of three parts: the ultimate aim or problem, all feasible solutions (referred to as alternatives), and the criteria to evaluate the alternatives. By quantifying its criteria and alternative possibilities and tying those parts to the broader purpose, AHP gives a coherent foundation for a needed conclusion. This method may suit decision issues in computation offloading contexts because it may prepare the decision-making process with the most relevant solutions and evaluate alternative options.

Sheng et al. [54] proposed a compute offloading approach that considers the performance of devices and the resources available on servers. Offloading decision-making, server selection, and task scheduling are the three primary stages of their system. The first stage used job sizes, computational requirements, the server's computing capability, and network bandwidth. In the second stage, appropriate servers are chosen by executing the AHP and thoroughly analyzing candidate servers using multi-objective decision-making. They proposed a task scheduling model for an enhanced auction process in the third stage, taking time constraints and compute performance into account.

F) Q-Learning

Q-Learning is a model-free, off-policy reinforcement learning that will determine the best course of action given the state. The goal of Q-Learning is to learn a policy that will tell an agent what action to take in which situation. Q-Learning can handle problems with stochastic transitions and rewards without the need for adaptation. For any finite MDP, Q-Learning is the policy of maximizing the anticipated value. It can also help solve optimization decision-making problems by determining the best action-state selection policy. Hence Q-Learning is suitable for use to tackle offloading problems.

To deal with the uncertainty of MEC environments, Kiran et al. [55] presented an online learning mobility management (Q-Learning) system. Their proposed solution learns the best mobility management scheme from the environment through trial and error to reduce service delay. UEs use the highest Q-values in the current state to decide when to hand over. These Q-values will be adjusted regularly to maintain the system's dynamic nature. To solve the fundamental objective, the decision-making constraint, Hossain et al. [56] suggested a standard RL method, especially Q-Learning. The decision-making process in an edge computing scenario is based on whether data will be offloaded to edge devices or handled locally. They used a cost function with latency and power consumption effect to achieve their goal.

G) Deep Q-Network (DQN)

Deep reinforcement learning blends artificial neural networks with reinforcement learning to allow agents to learn the optimum actions in a virtual environment to achieve their objectives. Deep Q-Learning is one type of deep reinforcement learning that combines function approximation and target optimization, matching state-action pairs to future rewards. Deep Q-Learning, also known as Deep Q Network, is a method for approximating the Q-value of a reinforcement learning architecture (Q-Learning), Q (s, a), using Deep Neural Networks (DQN). The state is given as an input, and the output is the Q-value of all potential actions.

In the MEC context, Huang et al. [32] presented a Deep Q-Network (DQN)-based multiple tasks offloading and resource allocation algorithm. They converted mixed-integer nonlinear programming into an RL problem, which discovers the best solution in this way. To reduce overall costs, they developed collaboratively offloading decision-making and allocating bandwidth. In their model, Chen et al. [57] used a finite-state discrete-time Markov chain to define the Markov Decision Process (MDP). They also employed a deep neural network-based solution (DQN) for the optimal task computation offloading in a dynamically Ultra-Dense Network (UDN) of MEC environment with enabled wireless charging mobile devices in the high dimensionality state space of MDP. Zhao et al. [58] proposed a deep reinforcement learning algorithm, DQN, to learn the offloading decision to optimize system performance. At the same time, the neural network is trained to predict offloading actions, and a multi-user MEC was considered where computational tasks are assisted by multiple Computational Access Points (CAP) by offloading the computationally intensive task to the CAP.

Furthermore, multiple bandwidth allocation algorithms to optimize the wireless spectrum for links between users and CAPs were developed. Tong et al. [59] generated tasks using a Poisson distribution and then suggested a novel DRL-based online algorithm (DTORA), which employs a deep reinforcement learning method called DQN to evaluate if the job should be offloaded and distributes computer resources or not. The mobility of users between base stations was studied to make the system more practical. Xu et al. [60] aimed to minimize power consumption by improving resource allocation in cloud radio access networks (CloudRANs). The authors expressed the resource allocation problem as a convex optimization problem and proposed a novel DRL-based framework based on DQN. In the proposed algorithm, deep Q-Learning is adopted for the online dynamic control based on the offline-built DNN. Shan et al. [61] introduced a hybrid of DRL and federated learning (FL) by proposing an intelligent resource allocation model, "DRL + FL." In this model, an intelligent resource allocation algorithm DDQN-RA based on DDQN is proposed to allocate network and computing resources adaptively.

In contrast, the model integrates the FL framework with the mobile edge system to train the DRL agent in a distributed way. Ho et al. [62] defined a binary offloading model where the computation task is either executed locally or offloaded for MEC server execution. Which should be adaptive to the time-varying network conditions. Hence, a deep reinforcement learning-based approach was proposed to tackle the formulated nonconvex problem of minimizing computation cost in terms of system latency. Since the conventional RL method is not suitable for an ample action space, a DQN was considered in the proposed work. Chen et al. [63] investigated the situation where MEC is unavailable or cannot meet demand. They considered the surrounding vehicles a resource pool (RP) and split the complex task into smaller sub-task. They formulated the execution time for a complex task as a min-max problem. They then proposed a distributed computation offloading strategy based on DQN, that is, to find the best offloading policy to minimize the execution time of a complex task. Lin et al. [64] aimed to jointly optimize the offloading failure rate and the energy consumption of the offloading process. They established a computation offloading model based on MDP and later proposed an algorithm based on DQN and Simulated Annealing (SA-DQN). Alam et al. [65] studied the near-end network solution of computation offloading for mobile edge due to the problem of higher latency and network delay in the far-end network solution. Mobile devices' mobility, heterogeneity, and geographical distribution introduce several challenges in computation offloading in the mobile edge. They modeled the problem as an MDP and proposed a code offloading algorithm based on Deep Q-Learning to minimize the execution time, latency, and energy. Chen et al. [66] investigated a stochastic computation offloading policy for mobile users in an ultra-dense sliced radio access network (UDS-RAN). They formulated the problem of stochastic computation offloading as an MDP, for which they proposed two DQN based online offloading algorithms, DARLING and Deep-SARL. They used an RL-based framework double DQN (DDQN) for computation offloading in a high dimensionality state space of MDP for a dynamically UDS-RAN of a MEC environment. They aimed to achieve a better long-term utility performance.

H) Actor-Critic Methods

Actor-Critics aims to combine the benefits of both value-based and policy-based approaches while removing all of their disadvantages. The actor receives the state as input and produces the best action. It effectively directs the agent's behavior by learning the best policy (policy-based). On the other hand, the critic calculates the value function (value-based) or tells the actor how excellent or awful their actions are. The weights are updated in the form of a TD error. This scalar signal is the critic's sole output and is responsible for all learning in both the actor and the critic. Actor-Critic methods can realize offloading computing without knowing the transition probabilities among different network states.

Ke et al. [67], in a heterogeneous vehicular network, proposed an offloading model known as ACORL. An adaptive computation offloading method based on DRL, specifically DDPG, can address continuous action space in vehicular networks. The proposed algorithm considered multiple stochastic tasks, wireless channels, and bandwidth. Moreover, the authors combined the Ornstein-Uhlenbeck (OU) noise vector to the action space. Huang et al. [68] considered a multi-user single server system, where multiple Wireless Devices (WDs) are connected to a single Access Point (AP) responsible for both transferring Radio Frequency (RF) energy and receiving computation offloading from WDs. They proposed a DRL Online Offloading (DROO) to implement a binary offloading decision-making strategy generated by a DNN, maximizing the weighted sum of computing rates of all wireless devices and decreasing computation time significantly. The proposed algorithm learns from previous experience to eliminate complex mix integer problems and improve the offloading decisions. To attain a minimum average task latency and energy consumption, Liu et al. [69] reduced the average latency and energy's weighted sum by optimizing the task offloading decision-making. They proposed an optimization framework based on actor-critic and RL learning approaches to tackle computation offloading. Unlike the above methods, Wang et al. [70] considered a UAV-assisted MEC system, where the UAV provides computing resources to nearby users. The authors aimed to minimize the maximum processing delays in the entire period by optimizing user scheduling, task offloading ratio, UAV flight angle, and speed. They then formulated a nonconvex optimization problem and proposed a computation offloading algorithm based on DDPG. Chen et al. [71] considered a MEC-enabled MIMO system with stochastic wireless channels and task arrivals and proposed a dynamic offloading algorithm based on DDPG. They aimed to reduce the long-term average computation cost in terms of power consumption and to buffer delay at each user.

I) Markov Decision Process (MDP)

The Markov Decision Process (MDP) is a discrete-time stochastic control mechanism that can handle various problems. The result is partly random and partly under the control of the decision-makers. MDPs can be used to look at optimization problems solved by dynamic programming. MDP is highly suited for offloading optimization in cases requiring dynamic decision-making due to varying and changing environmental elements such as wireless channel characteristics and computational load. An MDP system is often made up of five tuples (E, S, A, P, R), where E stands for decision epochs, S for states, A for actions, P for state transition probability, and R for rewards. The five-tuple must be defined according to the offloading technique in this model. After that, we can develop the following: Bellman's formula.

$$V^{\pi *}(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, V^{\pi *}(s') \right\}$$

where γ is a discount factor, and $V^{\pi *}$ is the value function for the optimal policy, $a \in A$ and R are the immediate reward, P is the state transition probability. Then using a classical solution such as value iteration algorithm or policy iteration algorithm, we can make an optimal offloading decision, such as whether to run tasks locally or offload them and which edge server should handle the offloaded tasks. The fixed decision epochs are defined in the majority of MDP models. As a result, it is expected that computing tasks are requested at the start of each period or that incoming requests are gathered in a queue throughout a period. A decision is taken at the start of the following period. MDP with a set period makes modeling easier. However, it often adds latency owing to accumulated waiting time. This modeling eliminates the waiting time if a choice is taken soon after receiving a task. However, the unpredictable time gap necessitates anticipating the length of the following epoch, making the MDP model less reliable or accurate. However, most MDP research assumes that the decision epoch is long enough to complete a task; modeling becomes extremely difficult.

Zhang et al. [22] investigated mobile offloading in a system of intermittently linked cloudlets, looking at user mobility patterns and cloudlet admission control. Then, to establish an appropriate offloading policy for a mobile user in an infrequently connected cloudlet system, a Markov decision process (MDP) based approach was presented. The policy decides offloading or local execution activities based on the condition of the mobile user to achieve a minimum cost. Ko et al. [72] considered an edge-cloud enabled heterogenous network with diverse radio access networks and proposed ST-CODA (a spatial-temporal computation offloading decision algorithm). In their model, a mobile device decides where and when to process MDP tasks. They took into account energy consumption, processing time, and transmission cost. Then, the classic MDP algorithm, namely, the value iteration algorithm, is used to solve the formulated model. In Wei et al. [73], they demonstrated a single-user, single-server system before emphasizing the importance of data prioritization. Following that, the authors established distinct data priority levels. The system handles the data based on its priority level, with higher priority data contributing more to rewards.

Meanwhile, the authors assume that all connected probability distributions are unknown, which means that traditional MDP methods like the value iteration or policy iteration cannot be utilized. They instead employ the Q-Learning method. Zhang et al. [74] considered the movement of vehicles in a vehicular edge computing (VEC) scenario and then proposed a time-aware MDP-based offloading algorithm (TMDP) and a robust time-aware MDP-based offloading algorithm under certain and uncertain transition probabilities, respectively.

Table 1: Machine learning algorithms

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[67]	ACORL based on DDPG	Energy, Latency, bandwidth	TensorFlow 1.10, Python 3.5	Multi-user Single-server	Partial offloading	To tackle the trade-off between energy consumption, bandwidth allocation, and delay	Hyper-parameter sensitivity
[58]	Offloading algorithm based on DQN	Latency, Energy	Null	Multi-user Multi-server	Partial offloading	To minimize energy and latency	Low convergence in high mobility
[28]	DDLO algorithm based on DNN	Energy, Latency	TensorFlow Python	Multi-user Single-server	Full offloading	Reduce the total cost of energy and latency	Not applicable in real-time offloading
[59]	DTORA based on DQN	Response time, Energy, QoE	TensorFlow Python	Multi-user Multi-server	Full offloading	To minimize response time and improve system utility	High complexity
[61]	Offloading algorithm based on DDQN-RA	Energy, Latency, QoS, Load balancing	Null	Multi-user Multi-server	Full offloading	To reduce the average delay and energy	Under performance
[49]	EEDOS based on DNN	Energy, Cost	MATLAB	Single-user Single-server	Partial offloading	To minimize system cost and energy	Single-user. Not applicable in real-time offloading
[60]	Offloading DRL framework based on DQN	Power	TensorFlow	Multi-user Multi-server	Partial offloading	To minimize power consumption	Unable to perform in a high action space
[56]	Offloading algorithm based on Q-Learning	Latency, Energy	MATLAB	Multi-user Single-server	Full offloading	To reduce energy consumption and overall system cost	It cannot be utilized in a complex network

(Continued)

Table 1 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[50]	Deep Supervised Learning (DSL)	Cost	Null	Single-user Single-server	Partial offloading	To minimize the overall execution cost	Single-user Single-server
[32]	Offloading algorithm based on DQN	Energy, Latency, Cost	TensorFlow Python	Multi-user Multi-server	Full offloading	To reduce the overall cost of energy computation and delay	Centralized algorithm. Not applicable to higher users
[62]	Offloading algorithm based on DQN	Latency	TensorFlow	Multi-user Multi-server	Full offloading	To reduce the computation cost of delay	Not applicable to higher users and task
[68]	DROO base on DRL and DNN	throughput	TensorFlow Python	Multi-user Single-server	Full offloading	To maximize offloading task	Low convergence in high mobility
[75]	Offloading algo based on instance-based learning	Computation cost	OpenCL framework	Single-user Single-server	Full offloading	To tackle the adaptive scheduling problem in offloading framework	Single-user single-server
[44]	Offloading algorithm based on SVM and federated learning (FL)	Energy	Null	Multi-user Multi-server	Partial offloading	To minimize total energy consumption	High complexity
[42]	Offloading algorithm based on SVM	Latency	Null	Multi-user Multi-server	Partial offloading	To satisfy the fast-offloading demand in vehicular networks	High management overhead
[76]	Offloading algo based on DQN	Latency, Energy, Load balancing	iFogSim, Google cluster tree	Multi-user Multi-server	Partial offloading	To improve system performance in terms of energy, load balancing, latency	High complexity

(Continued)

Table 1 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[52]	Offloading decision-making algo based on decision tree	Make-offloading decision	Java	Single-user Single-server	Partial offloading	To handle offloading decisions and monitor offloading metrics	Single-user single server. Not scalable
[77]	Low complexity algo base on LR and DDLO algo base on DNN	Latency, Energy, QoS	Python	Multi-user Multi-server	Full offloading	To guarantee QoS and to minimize energy consumption	Not applicable in real-time offloading
[26]	Offloading algo based on decision tree and SDN	Energy	Java	Single-user Single-server	Full offloading	To minimize energy consumption	Single-user single-server. High overhead
[78]	Machine-learning techniques	Latency	CloudSim	Multi-user Multi-server	Full offloading	To improve response time	Not suitable to implement
[43]	SVM	Energy, Latency	Null	Single-user Single-server	Full offloading	To reduce response time and energy	Long training time
[79]	Offloading algorithm based on game theory and RL	Latency, Energy	MATLAB	Multi-user Single-server	Full offloading	To tackle multiple IoT devices computation offloading problems in MEC	Stationary model
[69]	Optimization framework based on actor-critic	Latency, Energy	Null	Single-user Multi-server	Full offloading	To attain a lower average task latency and energy consumption	May fail in a real-time environ-ment
[63]	A distributed offloading algorithm base on DQN	Latency, QoE	Null	Multi-user Multi-server	Partial offloading	To reduce the task execution time	Lack of adaptation to a general scene

(Continued)

Table 1 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[64]	Offloading algorithm based on SA-DQN	Energy, offloading failure	TensorFlow Python	Single-user Multi-server	Partial offloading	To minimize the rate of offloading failure and energy consumption	Not applicable in a stochastic environment and a high action space
[70]	Offloading algorithm based on DDPG	Latency	Null	Multi-user Single-server	Partial offloading	To minimize the maximum processing delay	Low convergence in high mobility
[65]	Code offloading algorithm based on deep Q-Learning	Latency, Energy	MATLAB	Multi-user Multi-server	Partial offloading	To minimize the latency of service computing	High complexity in real-time implementation
[71]	Offloading algorithm based on DDPG	Computation cost, Power-delay tradeoff	Null	Multi-user Single-user	Full offloading	To minimize average computation cost in terms of power consumption and buffering delay.	High complexity
[57]	Computation offloading algo based on DDQN	Throughput	TensorFlow	Single-user Multi-server	Full offloading	To maximize long-term utility performance	The unspecified prioritized queue for real-time tasks
[51]	ABSO base on ARIMA-BP	Energy	Null	Multi-user Single-server	Full offloading	To minimize the energy consumption of mobile devices	Lack of handover coverage. High complexity
[66]	Offloading algorithm based on DQN	Delay, Energy	Null	Single-user Multi-server	Full offloading	To minimize the long-term cost	Extensive data requirement

(Continued)

Table 1 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[54]	Offloading algorithm based on AHP	Latency, QoE, Energy	MATLAB	Multi-user Multi-server	Full offloading	To minimize latency and improve quality of experience	High complexity
[73]	An RL offloading algo base on MDP	Latency	Null	Single-user Single-server	Full offloading	To design an effective offloading scheme for energy harvest MEC system	Single-user single-server. High Complexity. Not applicable in a high state-action space
[72]	ST-CODA based on MDP	Energy, Latency, Cost	Null	Single-user Multi-server	Full offloading	To minimize energy consumption, latency, and transmission cost	May experience curse of dimensionality problem
[74]	RTMDP based on MDP	Latency	Null	Single-user Multi-server	Partial offloading	To minimize systems, delay	High complexity
[22]	Offloading algorithm based on MDP	Computation cost	Null	Multi-user Single-server	Full offloading	To minimize computation cost	May experience curse of dimensionality problem

4.2 Non-Machine Learning Offloading Algorithms

Here we will introduce several non-machine learning algorithms that have been adopted and used in both academia and industry to tackle problems with edge computing offloading. As illustrated in Table 2.

A) Lyapunov Optimization

The use of the Lyapunov function to regulate a dynamical system optimally is referred to as Lyapunov optimization. In control theory, Lyapunov functions are frequently utilized to ensure various types of system stability. A multi-dimensional vector is frequently used to describe the status of a system at a given point in time. A nonnegative scalar measure of this multi-dimensional state is a Lyapunov function. When the system approaches undesired states, the function is typically defined to get huge. Control measures that cause the Lyapunov function to drift in the negative direction towards zero create system stability. The drift-plus-penalty approach for combined network stability and penalty minimization is obtained by adding a weighted penalty term to the Lyapunov drift and reducing the sum. The goal is to reduce both the drift and the penalty simultaneously. It is crucial to define the drift and penalty when utilizing Lyapunov optimization in offloading design. The drift in an offloading context could be the energy queue drift or the task queue drift. At the same time, the penalty is often the offloading goal, such as task dropping reduction or execution latency reduction. The ideal offloading decision and other parameters might then be found by reducing the drift-plus-penalty expression for each time slot. Because the drift-plus-penalty expression is usually a deterministic problem, the Lyapunov optimization has a substantially lower computational complexity than (non-) convex optimization. Furthermore, unlike MDP, it does not necessitate the probability distribution of the random event process.

In [80], MEC systems with multiple devices were considered. They formulated a power consumption minimization problem with task buffer stability constraint to investigate the tradeoff between computation task execution delay and mobile devices' power consumption. Then proposed an online algorithm that decides the local execution and computation offloading policy based on Lyapunov optimization. Huang et al. [30] aimed to provide a low-latency energy-saving user interaction algorithm under different wireless conditions. So they proposed an adaptive offloading algorithm that can offload part of an application's computation to a server dynamically according to the wireless environment changes based on Lyapunov optimization. The authors used the concept of call graphs to represent the relationship between individual components A compute, and transmission power minimization problem with delay and reliability constraints was formulated by Liu et al. [81]. The authors employ results from extreme value theory [82] to put a probabilistic constraint on user task queue lengths and characterize the occurrence of low probability events in terms of queue length or delay violation. Finally, the optimization problem was solved using Lyapunov optimization tools.

B) Game Theory

For constructing distributed methods, game theory is a valuable tool. It can be utilized in a multi-user offloading scenario where every other end device chooses a suitable technique locally to obtain a mutually acceptable offloading solution. Every end device makes an offloading decision, receives incentives (utility), and then updates the decision in this model. This approach is repeated until the rewards are no longer improved, i.e., until Nash equilibrium is reached. An iterative technique is commonly utilized to identify the Nash equilibrium. It is fundamentally an optimization problem for each end device, i.e., maximization of its utility. As a result, the Nash equilibrium can be obtained using a traditional optimization solution, such as the Karush-Kuhn-Tucker conditions.

Chen et al. [83] studied the multiuser computation offloading problem in multi-channel wireless interference and contention environments. Authors showed that it is NP-hard to compute a centralized optimal solution for mobile-edge cloud computing, so they adopted a game theory approach to achieve efficient computation offloading in a distributed way. The distributed computation offloading decision-making problem among mobile users was formulated as a computation offloading game and then proposed a distributed offloading algorithm to achieve a Nash equilibrium. Zhou et al. [84] investigated the partial computation offloading problem for multiuser in mobile in MEC environment with multiple wireless channels. The authors modeled a game theory approach for the computation overhead and then proved the existence of Nash equilibrium. The paper aims to minimize the computation overhead. Cui et al. [79] proposed an offloading algorithm based on evolutionary game theory and RL. Multiuser computation offloading under a dynamic environment was investigated;

the authors then formulated the problem as a game model and proved that multiuser computation offloading has a unique Evolutionary Stability Strategy (ESS).

Also, Alioua et al. [85] proposed using game theory for computation offloading but instead in the context of an Un-man Aerial Vehicle (UAV) to assist in road traffic monitoring. They modeled the offloading decision-making problem as a sequential game, proved a Nash equilibrium, and proposed an algorithm to attain the Nash equilibrium. They aimed to improve the overall system utility by minimizing computation delay while optimizing the energy overhead and the computation cost. Liu et al. [86] studied fog to cloud offloading. They formulated the interaction between the fog and the cloud as a Stackelberg game to maximize the utilities of cloud service operators and edge owners by obtaining the optimal payment and computation offloading strategies and proved that the game is guaranteed to reach a Nash equilibrium. They then proposed two offloading algorithms to achieve low delay and reduced complexity, respectively. Anbalangan et al. [87] studied accumulated data at Macro base Stations (MBS). The MBS cannot accommodate all user demands and attempts to offload some users to nearby small cells, e.g., an access point (AP), and then investigate the trade-off between the economic incentive and the admittance of load between the MBS the AP to achieve optimal offloading. They proposed a software-defined networking-assisted Stackelberg game (SSG) model. In the model, the MBS aggregates users to the AP and prioritizes users experiencing a minor service. Kim et al. [88] proposed an optimal pricing scheme considering mobile users' need for resources, formulated a single-leader-multi-user Stackelberg game model, and then utilized Stackelberg equilibrium to optimize the overall system utility, i.e., mobiles users and edge cloud utilities.

C) (Non) Convex Optimization

Because it is solvable, convex optimization is a robust tool for solving optimization issues. The offloading objective(s) is(are) defined as an objective function, and the offloading limitations are formulated as constraint functions in this paradigm. If the specified optimization model is convex, traditional approaches like the Lagrange duality method can solve it and accomplish the global optimization goal. If the offloading model is a non-convex optimization issue, converting it to a convex optimization is a standard solution.

1) Alternating Direction Method of Multipliers (ADMM)

The Alternate Direction Method of Multipliers (ADMM) is a strategy for resolving convex optimization problems by breaking them down into smaller, easier chunks. Small local subproblem solutions are coordinated using a decomposition-coordination technique to solve a global problem.

Wang et al. [89] formulated the computation offloading decision, resource allocation, and content caching strategy as a non-convex optimization problem. They then transformed the non-convex problem into a convex one. Furthermore, decomposed the convex problem to solve it in a distributed and efficient manner. They then developed and applied an Alternating Direction Method of Multipliers (ADMM) based on distributed convex optimization to solve the problem. Bi et al. [90] considered a Wireless Powered Transfer (WPT) mobile edge computing system and proposed an offloading algorithm based on ADMM and Coordinate Descent (CD). They aim to maximize the weighted sum computation rate of all Wireless Devices (WD) in the network. The authors combined the recent advancement in WPT and MEC. Each WD followed a full offloading approach in the proposed architecture, i.e., all computation is either performed locally or executed remotely.

2) Successive Convex Approximation

For decreasing a continuous function among several block variables, successive convex approximations such as the Block Coordinate Descent (BCD) method are commonly utilized. A single block of variables is optimized at each iteration of this approach, while the remaining variables are kept constant.

Tang et al. [91] jointly optimized a weighted sum of the time and energy consumption in user experience. They formulated a mixed overhead of time and energy minimization problem (Nonlinear problem). The proposed a mixed overhead full granularity partial offloading algorithm based on Block Coordinate Descent (BCD) approach to deal with every variable step by step to tackle the problem. Baidas [92] formulated a joint subcarrier assignment, power allocation, and computing resource allocation problem to maximize the network sum offloading efficiency. A low complexity algorithm is proposed to solve the problem. The algorithm relaxes the problem into two subproblems and then solves with a successive convex approximation algorithm and polynomial-time complex Kuhn-Munkres algorithm.

3) Interior Point Method (IPM)

Interior point algorithms are a type of algorithm used to solve inequalities as constraints in both linear and nonlinear convex optimization problems. The linear programming Interior-Point approach requires a linear programming model with a continuous objective function and twofold continuously differentiable constraints. Interior point techniques approach a solution inside or outside the feasible region but never from the edge.

Yang [93] proposed a joint optimization scheme for task offloading and resource allocation based on edge computing and 5G communication networks to improve task processing efficiency by reducing time delay and energy consumption of terminal tasks. The problem of computing task offloading is transformed into a joint optimization problem and then developed corresponding optimization algorithms based on the interior point method

4) Semidefinite Programming

Semidefinite Programming (SDP) is a convex optimization discipline concerned with the optimization of a linear objective function (a user-specified function that the user desires to decrease or maximize) over the crossing of the cone of positive semidefinite matrices with an affine space [94–96].

Chen et al. [97] set out to jointly optimize all users' offloading decisions and communication resource allocation to reduce energy, computation, and delay costs for all users. They proposed an efficient approximate algorithm (MUMTO) Multi-user Multi-task offloading solution. A heuristic algorithm based on Semidefinite Relaxation (SDR), accompanied by restoration of the binary offloading decision and optimal communication resource allocation. To solve an NP-hard non-convex quadratically constraint quadratic optimization problem. Due to the inter-task dependency in various end devices, Liu et al. [27] sought to develop energy-efficient computation offloading. They used a Mixed Integer Programming problem (MIP) to solve an energy consumption minimization problem with two constraints: task dependency and the IoT service completion time deadline. To address this challenge and provide task computation offloading solutions for IoT sensors, they suggested an Energy-efficient Collaborative Task Computation Offloading (ECTCO) algorithm based on a semidefinite relaxation (SDR) and stochastic mapping technique.

D) Particle Swarm Optimization (PSO)

PSO is a metaheuristic optimization method that can solve continuous problems with imperfect information and computational limitations. As a result, this method may help offload optimization problems involving the ecosystem's dynamic behavior without ensuring a globally optimal solution. The outcome is dependent on the arbitrary spawn variables because PSO is a stochastic optimization method.

Particle Swarm Optimization (PSO) is used by Liu et al. [98] to reduce latency and dependability. They used code partitioning to depict the reliability of task elements as a Directed Acyclic Graph (DAG). They used the chance of service failure of MEC-enabled Augmented Reality (AR) service to establish a trade-off between latency and dependability. They presented an Integer PSO (ISPO) algorithm due to the problem's non-convexity. They then created a low-complexity heuristic technique for optimizing offloading because ISPO was not feasible in AR [10]. In Li et al. [99], authors formed a delay minimization problem. They proposed a computation offloading strategy dubbed EIPSO based on an improved PSO algorithm to reduce delay and improve load balancing of the MEC server.

E) Genetic Algorithm (GA)

At first look, GA appears to be a good fit for searching an ample state space for optimal solutions as an evolutionary computation technique. However, if the state space is not constructed correctly, GA will not discover an optimal solution in an acceptable amount of time. After that, the GA techniques are the best options when the optimizations are not possible at the local optimum. On the other hand, GA can be a candidate to solve many optimization problems because it requires several generations to create adequate outputs. However, it is only suitable for a select few. However, GA fails to achieve optimal results when measuring fitness is not straightforward.

Zhang et al. [100] studied the collaborative task offloading and data caching models. They proposed a hybrid of an online Lyapunov and genetic algorithms to tackle task offloading and data caching problems to minimize edge computation latency and energy consumption. The authors considered a multi-user MEC system, where multiple base stations serve multiple devices within its radio range through wireless cellular links. Similarly, Kuang et al. [101] also considered a multi-user, multi-server MEC system but instead focused on minimizing system communication latency and energy. Initially, an offloading algorithm based on backtracking was proposed, whose time complexity happens to be exponential with the number of servers. The authors proposed a genetic algorithm and a method based on a greedy strategy to reduce this time complexity.

Consequently, the authors proposed three offloading algorithms: backtracking, genetic, and greedy. They then compared the three algorithms in terms of the total cost of users, resource utilization of edge servers, and execution time. Zhao et al. [102] applied GA, too, but differently to achieve system optimality. They worked on a dynamic optimization offloading algorithm based on the Lyapunov method, namely LODCO. The original problem is time-dependent, converted to a per-time slot deterministic problem by Lyapunov optimization and using a perturbation parameter for each mobile device. Based on LODCO, they proposed their Multi-user Multi-server method by GA and Greedy approaches to achieve optimal results for the system. Shahryari et al. [103] considered a task offloading scheme that optimizes task offloading decision, server selection, and resource allocation and investigated the trade-off between task completion time and energy consumption. They then formulated the task offloading problem as a Mixed-Integer Nonlinear Problem (MINLP). They then proposed a sub-optimal offloading algorithm based on a Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). Hussain et al. [104] studied the problem of multi-hop computation offloading in a Vehicular Fog Computing (VFC) network and later formulated a multi-objective, non-convex, and np-hard quadratic integer problem and then proposed CODE-V, a computation offloading with differential evolution in VFC. The proposed algorithm considered the constraint of service latency, hop-limit, and computing capacity and optimally solved the problem of offloading the decision-making process, MEC node assignment, and multi-hop path selection for computation offloading.

F) Hidden Markov Model (HMM)

HMM is a durable and statistical Markov-based model with effective learning parameters gained from raw data. It may be identified as a finite state machine because of its hidden and observable states. HMM uses the Viterbi algorithm to find the optimum path for the system's probable following states, and by doing so, can perform better than a simple Markov model. However, because this technique is iterative, it is pretty resource and time-consuming to find the optimum path. With all of this, if HMM is chosen for the right problem and built effectively for that problem, it may forecast future system states in the decision-making nature of the offloading environment. Wang et al. [105] presented a smart HMM scheduling model that allows for dynamically modifying the processing technique while observing latency, energy, and accuracy in a mobile/cloud-based context Jazayeri et al. [106] set out to discover the ideal location for running offloading modules (end device, edge, or cloud). As a result, the authors presented a Hidden Markov model Auto-scaling Offloading (HMAO) based on HMM to discover the ideal place to execute offloading modules to make a tradeoff between energy usage and module execution time.

G) Other Algorithms

1) Heuristic Algorithms

In the style of a broad heuristic, a heuristic algorithm can generate an acceptable solution to a problem in a variety of actual settings, but for which there is no formal demonstration of correctness. NP-complete problems, a decision problem, are frequently solved using heuristic algorithms. Although solutions can be validated when offered, there are no known effective means to locate a solution fast and accurately in these circumstances. Heuristics can be used to generate a solution independently or combined with optimization techniques to offer a reasonable baseline. When approximate solutions are satisfactory but exact solutions are computationally expensive, heuristic algorithms are frequently used. Heuristic techniques might introduce rapid but sub-optimal results. Heuristics have the advantage of being simple algorithms designed to solve a specific problem with a short execution time.

Liao et al. [107] aimed to maximize the number of offloaded tasks for all UEs in uplink communication while maintaining low system latency. They formulated the problem as an NP-hard mix integer nonlinear problem and then proposed an efficient low complexity heuristic algorithm that provides a near-optimal solution. Ren et al. [108] studied three different offloading models, local compression, edge compression, and partial compression offloading. Close-form expressions of optimal resource allocation and minimum system delay for local and edge cloud compression are derived. A piecewise optimization problem was formulated for the partial compression based on a data segmentation strategy and then proposed an optimal joint computation resource allocation and communication algorithm. Xing et al. [109] aimed at minimizing computation delay subjected to individual energy constraints at the local user and the servers. They formulated the latency minimization problem as a mixed-integer nonlinear problem (MINLP) and then proposed a low complexity sub-optimal algorithm. You et al. [110] studied resource allocation for multi-user MEC systems based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA). In the TDMA MEC system, the authors formulated the resource allocation problem as a convex optimization problem. The aim is to minimize mobile energy consumption under computation latency constraints. In the OFDMA MEC system, the resource allocation problem is formulated as a mix-integer problem (MIP); to solve this, a low complexity sub-optimal offloading algorithm is proposed by transforming the OFDMA problem into a TDMA problem. The corresponding resource allocation is derived by defining an offloading priority function and having close to optimal performance in simulations. Similar to [90], Xu et al. [111] considered a WPT combined with MEC and proposed a metaheuristic search approach to maximize the weighted sum computation rate of all WD in the network.

2) Dynamic Voltage Scaling (DVS)

Modifying power and speed settings on a computing device's different CPUs, controller chips, and peripheral devices to optimize resource allotment for activities and optimum power savings when those resources are not needed is identified as Dynamic Voltage and Frequency Scaling (DVFS). By dynamically altering the voltage and frequency of a CPU, Dynamic Voltage and Frequency Scaling (DVFS) tries to reduce dynamic power usage. This method takes advantage of CPUs' discrete frequency and voltage settings.

Wang et al. [112] investigated partial computation offloading using Dynamic Voltage Scaling (DVS) by jointly optimizing the computational speed, transmit power, and the offloading ratio of mobile devices. Their study aims to minimize energy consumption and latency. They formulated both the energy minimization and latency minimization problems as non-convex optimization problems. They then converted the energy minimization non-convex problem was converted into a convex problem with the variable substitution technique and obtained its optimal solution. For the latency minimization non-convex problem, a locally optimal algorithm with a univariate search technique was proposed for the optimal solution. Like Guo et al. [113] studied an energy-efficient dynamic computation offloading using Dynamic Voltage Frequency Scaling (DVFS) by jointly optimizing computation offloading selection, clock frequency control, and transmission power allocation. The authors presented an energy-efficient dynamic offloading and resource scheduling (eDors) policy to reduce energy consumption and application completion time by formulating an energy-efficient cost reduction problem.

3) Gibbs Sampling

Gibbs Sampling is a Monte Carlo Markov Chain method for approximating complex joint distributions by iteratively drawing one instance from each variable's distribution, conditional on the current values of the other variables.

Yan et al. [114] considered a task dependency model where the input of a task at one wireless device requires the final task output at the other wireless device. They then investigated the optimal task offloading policy and resource allocation that minimizes the weighted sum of the wireless device's energy consumption and task execution time. They then proposed efficient algorithms to optimize the resource allocation and task offloading decision-making to minimize the weighted sum of energy consumption and task execution delay, based on the bisection search method and Gibb's sampling method.

4) Generalized Benders Decomposition

An approach for solving certain sorts of NLP and MINLP issues is known as Generalized Benders Decomposition. This approach has recently been proposed as a tool for resolving process design issues. When examining the solution of nonconvex problems using various GBD implementations, it is shown that in some circumstances, only local minima may be identified. In contrast, in others, even convergence to local optima is impossible.

Zhong et al. [115] looked into the coupling relationship between service caching and offloading decisions. They formulated the problem of joint computation offloading, service caching, and resource allocation as a mixed-integer non-linear programming problem. They then proposed a cooperative service caching and computation offloading (GenCOSCO) algorithm based on generalized benders decomposition, aimed to minimize the average task execution time in the MEC system.

Table 2: Non-machine learning algorithms

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[107]	A heuristic algorithm based on hypergraph	Maximize offloading task (Throughput)	MATLAB	Multi-user Single-server	Partial offloading	To maximize the number of offloading tasks for all UEs in uplink communication	Centralized
[83]	Offloading algorithm based on game theory	Energy, Latency	Null	Multi-user Single-server	Full offloading	To achieve efficient offloading by preventing interference	No consideration for power control. No-fault tolerance control
[80]	An online offloading algorithm based on Lyapunov	Power-delay tradeoff	Null	Multi-user Single-server	Full offloading	To balance the power consumption to mobile devices	Not applicable in real-time offloading. Centralized
[89]	Offloading algorithm based on ADMM	Latency, throughput	MATLAB	Multi-user Single-server	Full offloading	To reduce system latency and increase throughput	Static network. May fail under high mobility. Not applicable in real-time offloading
[97]	MUMTO based on SDR	Energy cost, Computation cost, Latency cost	MATLAB	Multi-user Single-server	Full offloading	To reduce the overall cost of energy, computation, and delay	Centralized algorithm. High latency. Not dynamic
[108]	Heuristic algo base on segmentation strategy	Latency	Null	Multi-user Single-server	Partial offloading	To minimize the overall system delay of all device	High complexity
[110]	Offloading algo based on TDMA and OFDMA	Energy, QoE	MATLAB	Multi-user Single-server	Partial offloading	To minimize the weighted sum mobile energy consumption	High complexity

(Continued)

Table 2 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[112]	Offloading algo based on DVFS, UST, and VSM	Energy, Latency	Null	Single-user Single-server Multi-user Multi-server	Partial offloading	To minimize system energy consumption and latency	Hard to implement
[30]	Dynamic offloading algo base on Lyapunov	Energy	Null	Single-user Single-server	Partial offloading	To achieve energy saving	No consid-eration for latency. Can achieve only local optimal
[81]	Offloading algo based on Lyapunov	Latency, Reliability	Null	Multi-user Multi-server	Partial offloading	To investigate the power-delay tradeoff of task offloading	Hard to implement. It may fail when many users are around. Can only achieve local optimal
[113]	eDors algorithm based on DVFS	Latency, Energy	Real testbed	Multi-user Single-server	Partial offloading	To provide an energy-efficient computation offloading	Less effective on reduction of latency. Limited by the trade-off between optimality and compu-tational complexity
[92]	A low complexity algorithm base on convex optimization	Throughput	Null	Multi-user Single-server	Full/partial	To maximize network sum offloading efficiency	Central controller. Not applicable in real-time offloading
[109]	A low complexity sub-optimal algo base on convex optimization	Latency	Null	Single-user Multi-server	Full offloading	To minimize user computation latency	Single-user system. Multi-user is not considered. Not scalable

(Continued)

Table 2 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[27]	ECTCO based on SDR and stochastic mapping	Energy	MATLAB	Multi-user Single-server	Partial offloading	To minimize the cost of energy	Central controller. Low convergence in high mobility
[88]	Offloading algo based on Stackelberg model	Energy	Null	Multi-user Single-server	Full offloading	To optimize the utility of the mobile users and the edge cloud	Central controller
[87]	SaSG	Latency, Throughput	Java	Multi-user Multi-server	Partial offloading	To minimize delay and increase user throughput	Not scalable. It may be challenging to implement in rea-time offloading
[114]	Offloading algo based on bi-section and Gibb's sampling	Energy, Latency	Null	Multi-user Single-server	Partial offloading	To minimize energy consumption and system latency	Not applicable. It may instead increase latency—computational complexity
[91]	Offloading algo based on BCD	Energy, Latency	Null	Multi/Single-user Single-server	Partial offloading	To minimize energy consumption and system latency	There was no consider-ation for change in the channel state information
[93]	Offloading algo based on IPM	Energy, Latency	Null	Multi-user Single-server	Partial offloading	To achieve better latency performance	It does not consider the execution order of the computing task

(Continued)

Table 2 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[104]	CODE-V	Energy, Latency	Null	Multi-user Multi-server	Full offloading	To minimize the average service latency and energy consumption	Too many assumptions for the algorithm to function. May incur latency when the network is scaled up
[19]	Offloading algo based on SDN	Latency, Throughput	Python	Multi-user Multi-server	Full offloading	To select the optimal offloading node and provide end-to-end bandwidth based on SDN	Controller vulnerability. Security issues
[86]	Offloading algo based on SG	Latency	Null	Multi-user Multi-server	Partial offloading	To cause cloud service operators and edge server owners to participate in computation offloading	Not scalable. Lack of practicability
[85]	Offloading algo based on game theory	Latency, Energy, Cost	Null	Multi-user Single-server	Partial offloading	To reduce computation delay while optimizing the energy and computation cost	The UAV's inflight energy usage was neglected
[84]	Offloading algo based on game theory	Latency, Energy	Null	Multi-user Single-server	Partial offloading	To minimize system computation overhead	May incur high computation overhead. Not applicable

(Continued)

Table 2 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[115]	GenCOSCO based on generalized benders decomposition	QoS, Latency	Null	Single-user Multi-server	Full offloading	To minimize the average task execution	The cached service on the edge server cannot be utilized among users; else, it raises a security risk
[98]	Offloading heuristic algorithm based on PSO	Reliability, Latency	Null	Single-user Multi-server	Partial offloading	To minimize the probability of service failure	High complexity
[99]	EIPSO based on an improved PSO algo	Latency, Load balancing	Null	Multi-user Multi-server	Full offloading	To reduce system latency and improve load balancing in MEC	High complexity
[106]	HMAO based on HMM	Latency, Energy	iFogsim	Multi-user Multi-server	Full offloading	To find the best place to execute a task	Central controller. May incur higher latency
[105]	Dynamic scheduling based on HMM	Reliability	Null	Single-user Single-server	Partial offloading	To dynamically adjust the offloading strategies based on environmental changes	High state space
[102]	LODCO based on GA, DVFS, and Lyapunov	QoE, Latency, Throughput	MATLAB	Multi-user Multi-server	Full offloading	To improve quality of experience and reduce delay	High complexity
[103]	A sub-optimal algo base on GA and PSO	Latency, Energy	Null	Multi-user Multi-server	Full offloading	To minimize overall offloading overhead	No consideration for user mobility

(Continued)

Table 2 (continued)

Reference	Algorithm (algo)	Performance metric	Simulation tool	System	Offloading type	Aim of algorithm	Algorithm drawbacks
[101]	GA algorithm and Greedy strategy	Cost	Python	Multi-user Multi-server	Full offloading	To reduce the cost of communication energy and delay	Static model
[100]	Online algo base on Lyapunov opt. and GA algo	Latency	Null	Multi-user Multi-server	Full offloading	To minimize the overall system latency of all device	May rather incur high latency. May get stuck in local optimal
[111]	Offloading algo based on Meta-heuristic search	Throughput	Python	Multi-user Single-server	Full offloading	To maximize the computation rate of user	Low convergence under high mobility
[90]	Offloading algo based on CD and ADMM	Throughput	Python TensorFlow	Multi-user Single-server	Full offloading	To maximize the computation rate of users	Rather high latency. Low convergence under high mobility

4.3 Offloading Strategies

Computation offloading strategies have been studied extensively [116–119] and generally fall under two categories, full offloading (coarse-grained) [120] and partial offloading (fine-grained) [91], [121–124]. For coarse-grained mode, the data set of a task has to be executed as a whole either locally or remotely on an edge server. In the case of partial offloading, task partition is allowed. Hence, a task is first partitioned into several components. These components are then offloaded to edge servers, or some components are executed locally while others are offloaded. In practice, binary offloading is easier to implement, and it is suitable for simple tasks that cannot be partitioned, while partial offloading is favorable for some complex tasks composed of multiple parallel segments.

4.3.1 Full Offloading

Full offloading is also called binary, coarse-grained, or total offloading. The complete task is migrated to the edge cloud for execution, or the whole task is executed locally. From Fig. 5, it could be seen that the task will not be divided. Either all the task is executed locally or offloaded to the edge for execution. Choosing where to offload may be based on data size, network condition, available resources, bandwidth, etc. This approach does not require estimating the computation overhead prior to the execution. The binary offloading problem can be solved using an exhaustive search. Enumerating all possible decisions can achieve the optimal global objective in a scenario with N end devices and one edge server. The end devices must decide whether to offload a task to the edge server. The number of possible decisions, on the other hand, is 2N. When N is huge, it is challenging to solve.

Some optimization procedures or methods should be used to lower the computational complexity. Consider the decomposition approach known as the Alternating Direction Method of Multipliers (ADMM).

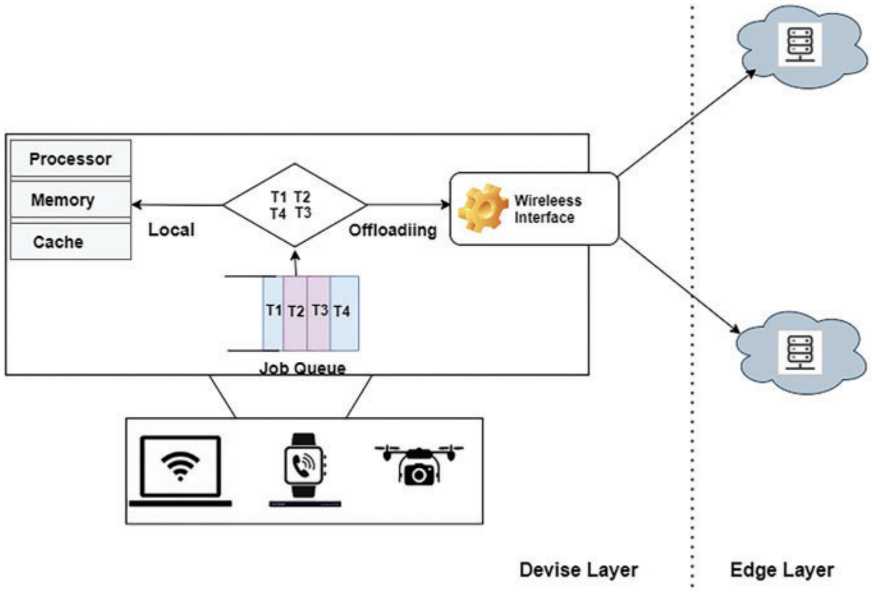

Figure 5: Full offloading

4.3.2 Partial Offloading

Partial offloading is also referred to as fine-grained offloading or dynamic offloading. This technique dynamically transmits as little code as possible and offloads only the computation-hungry parts of the application. From Fig. 6, it can be observed that the task is separated, one part is executed locally, and the other part is executed remotely. Despite the added processing expense and stress on application programmers, fine-grained offloading can eliminate needless transmission overhead, resulting in lower latency and energy usage. Data-partitioned-oriented tasks, code-partitioned-oriented tasks, and continuous-execution tasks are the three forms of partitionable jobs. We divide these partitionable tasks into two categories in the context of offloading modeling: parallel and sequential. The work is partitioned into components executed in parallel in the first scenario (parallel). As a result, the only decision the offloading strategy must make is where to place the components. There are dependencies among the components in the latter case (sequential), i.e., specific components cannot be executed until the completion of others, necessitating the definition of component placement and scheduling. The relationship between them is then described using Component Dependency Graphs (CDGs) or Correlation Graphs; a correlation graph is represented by G (V, E), where V and E denote its sets of nodes and edges respectively [125–127]. The CDG's vertices represent the application's various components or tasks. At the same time, the edges reflect the invocations between these components, or each vertex represents a component, and the direct edge represents the magnitude of data migration from one component to another. Fig. 7 shows an example of a CDG representation. The non-offloadable components must be executed locally, as indicated by the blue vertices. The offloadable components are represented by green vertices.

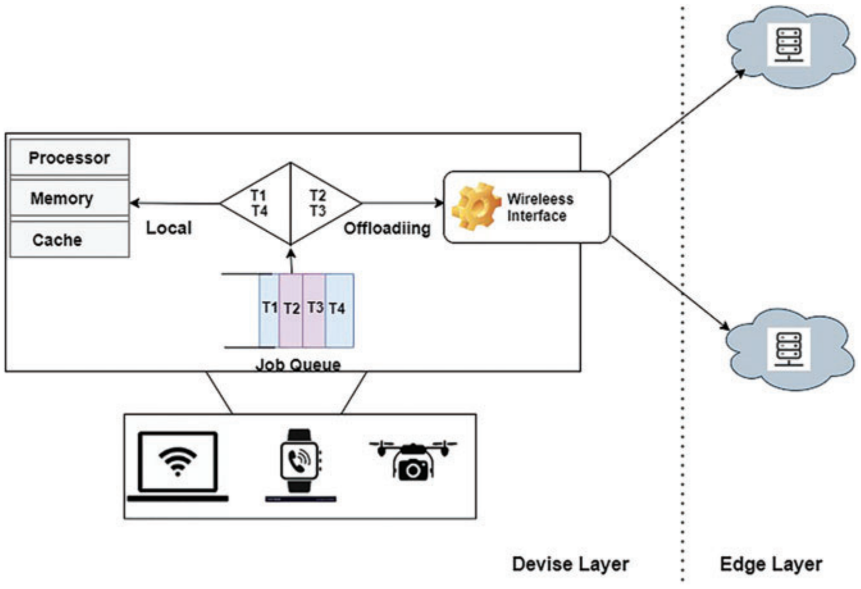

Figure 6: Partial offloading

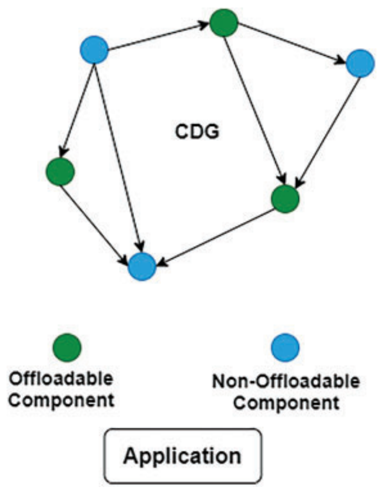

Figure 7: Component Dependency Graph (CDGs)

Table 3 groups all the proposed offloading algorithms, both the machine learning and the non-machine learning algorithms, into the two offloading strategies, full offloading or partial offloading.

Table 3: Full offloading and partial offloading

Full offloading (ML and Non-ML)	Partial offloading (ML and Non-ML)
[106], [102], [103], [101], [100], [128], [80], [89]	[107], [108], [110], [112], [30], [81], [113],
[97], [92], [109], [88], [104], [19], [115]	[92]
[99], [28], [59], [61], [56], [32]	[27], [87], [114], [91], [93], [86], [85]
[62], [68], [75], [77], [26]	[84], [98], [105], [67], [58], [49]
[78], [43], [79], [69]	[60], [50], [44], [42], [76]
[71], [57], [51]	[52], [63], [64]
[66], [54]	[70], [65]
[73], [72]	[74]
[22]	

5 Open Discussion

5.1 Utilized Metrics

In all, under the machine learning algorithms, 12 metrics were considered by researchers in the literature. From Fig. 8, it can be observed that the majority of the recently published works of researchers focus more on latency (Delay) and energy. From the chart, it can be observed that among the 36 chosen papers, 33.85% of work focused on latency and 32.31% on energy. We will like to state that different reviewed papers do not formulate a specific metric in the same way. Therefore, this study avoids introducing a general way to cover the formulation of reviewed metrics. Instead, we only review and present the metrics as they were put by authors. It is also worth mentioning that since some reviewed articles are multi-objective, some of the mentioned metrics might be considered in more than one article. Moreover, we will also like to mention that we considered all delays as latency and considered maximizing overall system utility and maximizing offloading task rate as throughput. See Fig. 8, for more details.

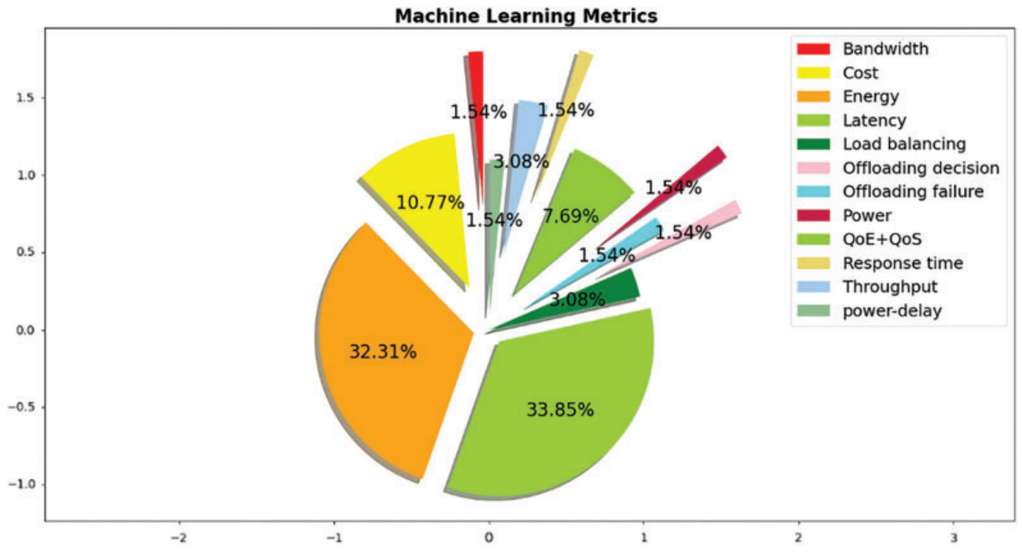

Figure 8: Machine learning offloading metrics

As for the non-machine learning algorithms, a total of 35 recently published papers were surveyed, and among all 8-performance metrics were considered by authors, from Fig. 9. It can be observed that energy and latency again are the dominant metrics considered by researchers. Among the 35 papers surveyed, 38.33% focused on latency, and 26.67% focused on energy. See Fig. 6 for details.

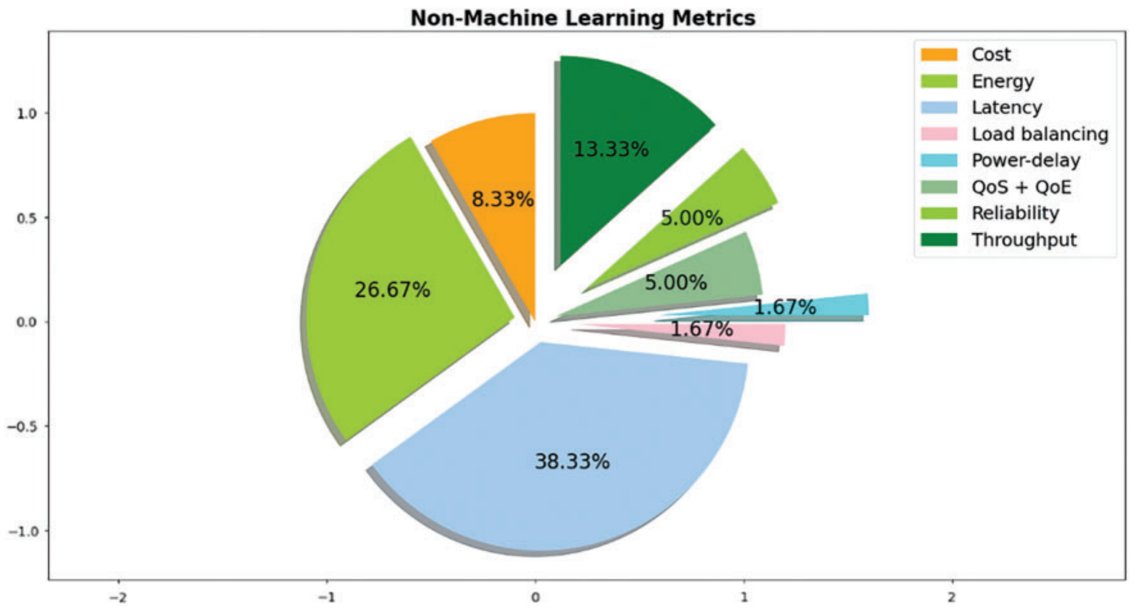

Figure 9: Non-machine learning metric

For machine learning and non-machine learning techniques, energy and latency are the main focus of research direction in computation offloading. Moreover, minimizing latency and energy consumption improves both service quality and experience, reduces both communication and computation costs, and improves overall system utility.

5.2 Utilized Techniques

Based on the above-mentioned reviewed papers, most of the offloading approaches in machine learning belong to reinforcement learning followed by supervised learning and then unsupervised learning, as shown in Fig. 10. RL is probably more suitable in a dynamic environment and covers ample state and action spaces. Moreover, RL removes the need for massive computational complexities with its DNN for approximation. RL methods can also be utilized for several scenarios. In the RL, most published papers considered in this article belong to DQN and Actor-Critic, with the least Q-Learning. Generally, the selection of the offloading approaches depends highly on the state-space of the problems. Fig. 11 shows the various distributions of the machine learning techniques utilized by various articles.

Since the offloading problems have commonly high complexities, pure Q-Learning is not applicable and suitable for solving such problems due to its tubular search nature. The time of the exploration to take the appropriate action by the agent to learn a policy will be close to impossible. This process will become considerably time-intensive and resource-intensive if the number of actions to select Q-values is significant. More specifically, in DQN or Actor-Critic, neural networks are generally approximating the Q-values. For the input (i.e., states), neural networks support the agent to learn the best possible actions that improve the decision-making performance of offloading. DQN and Actor-Critic perform well in discrete and continuous space, respectively. It makes it appropriate for implementation in various applications, such as AR, VR, online gaming, stock market, and Intelligent vehicle systems to enhance different metrics such as delay, energy, cost, bandwidth, and security. Since powerful open-source tools such as PyTorch and TensorFlow run appropriately with Python, such tools are in the attention of the researchers to implement their proposed models more precisely.

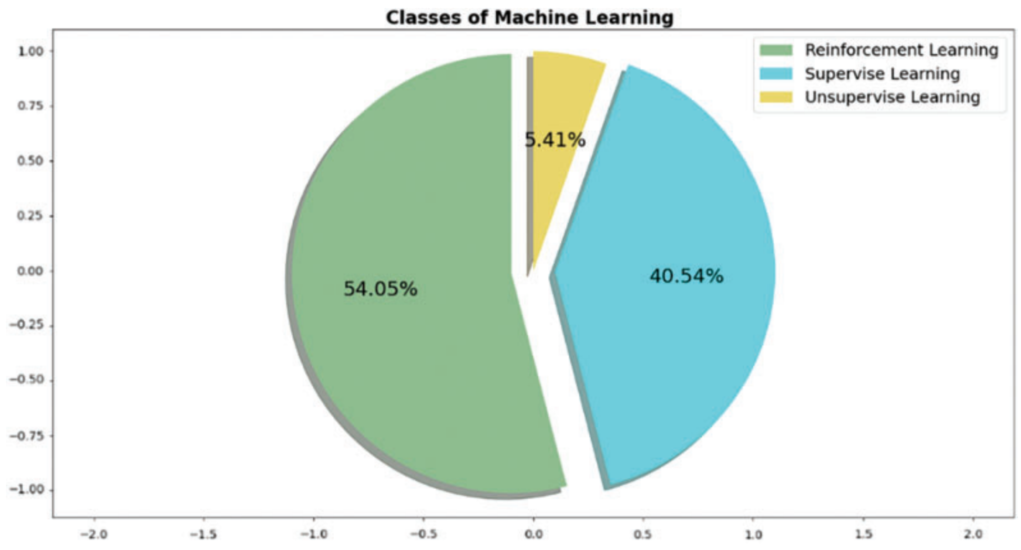

Figure 10: Classes of machine learning

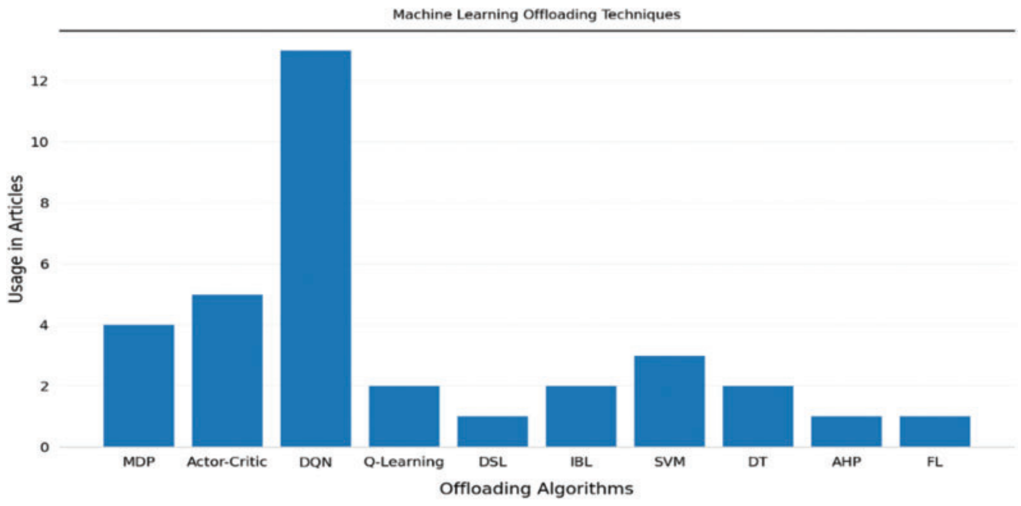

Figure 11: Machine learning offloading techniques

Meanwhile, for the non-machine learning techniques out of 35 papers considered in this article, convex optimization methods happen to have the maximum usage followed by Game theory and Lyapunov. Lately, there have been many game-theoretic approaches to deal with resource allocation problems and decision-making. For example, the problem of partial task offloading in a multi-user, Edge Computing Infrastructure, and a multi-channel wireless interference environment can be formulated as an offloading game [129,130]. This game maximizes the spectrum efficiency during offloading by allocating the proper channel to each user or player. On the other hand, Lyapunov optimization algorithms provide a unique property of finding sufficient conditions for stability in dynamical systems. Due to the stability theory of dynamical systems, Lyapunov-based optimization algorithms can study the task offloading problem. The key to using Lyapunov optimization in offloading modeling is to define the drift and the penalty. The drift could be the energy queue drift or the task queue drift in an offloading scenario. In contrast, the penalty is usually the offloading objective, e.g., the minimization of task dropping or execution latency. Then at each time slot, minimizing the drift-plus-penalty expression could determine the optimal offloading decision and other parameters.

Convex optimization is a powerful tool for optimization problems because it is solvable. In this model, the objective(s) of offloading is(are) formulated as an objective function, and the limitations of offloading are formulated as constraint functions. If the formulated optimization model is convex, classical methods such as the Lagrange duality method and Karush-Kuhn-Tucker (KKT) conditions can solve the model and achieve the global optimization objective. If the offloading model is a non-convex optimization problem, a usual way is to transform it into convex optimization. Some of the convex optimization methods considered here include ADMM, BCD, and IPM. Fig. 12. Shows the non-machine learning techniques considered here; more details can be found in Section 4.

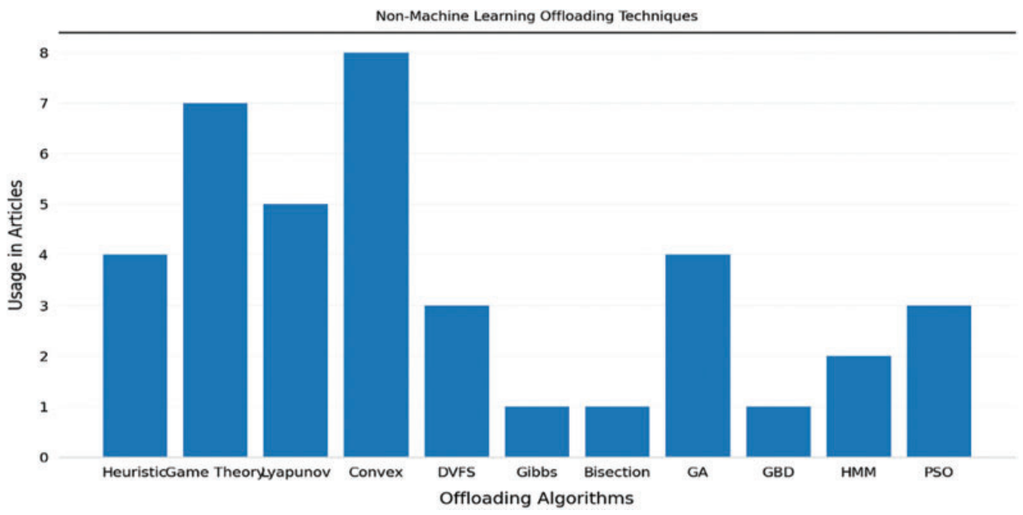

Figure 12: Non-machine learning offloading techniques

5.3 Utilized Systems

The following technical question is addressed in this section:

Q5: What utilized systems are considered in both ML and Non-ML based approaches?

The MEC systems considered in this article for both machine learning and non-machine learning techniques include Multi-user Multi-server, Multi-user Single-server, single-user multi-server, and Single-user Single-server.

There is greater complexity in the **multi-user multi-server offloading scenario**. It is a hybrid of single-user multi-server and Multi-user Single-server offloading scenarios. The information from both end devices and edge servers is necessary for decision making if a centralized offloading approach is employed for the Multi-user Multi-server offloading scenario. Although the centralized method can accomplish global optimization, solving the model is usually complex. As a result, a distributed technique is more suited to solving the problem of Multi-user Multi-server offloading [131].

In the Multi-user Single-server offloading scenario, multiple end devices offload their responsibilities to a single server. When modeling this type of scenario, all entities should be considered, and the decision should be made to maximize the entire system. There is no server selection in this scenario because there is only one server, and the server functions as the decision-maker for all end devices. A base station is a decision-maker in You et al. [132]. It gathers the essential data from the environment and all of its end devices before making decisions on their behalf.

Many edge servers are available in the **single-user multi-server offloading scenario**. Therefore, servers must be chosen based on several parameters (task size, availability of resources, mobility). The end device determines whether or not to offload the task and which server(s) it should be offloaded to. End devices may distribute components to numerous edge cloud servers if partitioning the work [133].

Only one end device selects whether to offload a computation work to an edge cloud server in a **single-user single-server offloading scenario**. The authors in [116] created a setup that included one edge device and one edge cloud server to show how to minimize the energy consumption at the mobile handset while guaranteeing a computational rate constraint imposed by the application. They later extended it to the multi-user scenario. Figs. 13 and 14 show the proportion of the various systems utilized under both machine learning and non-machine learning techniques.

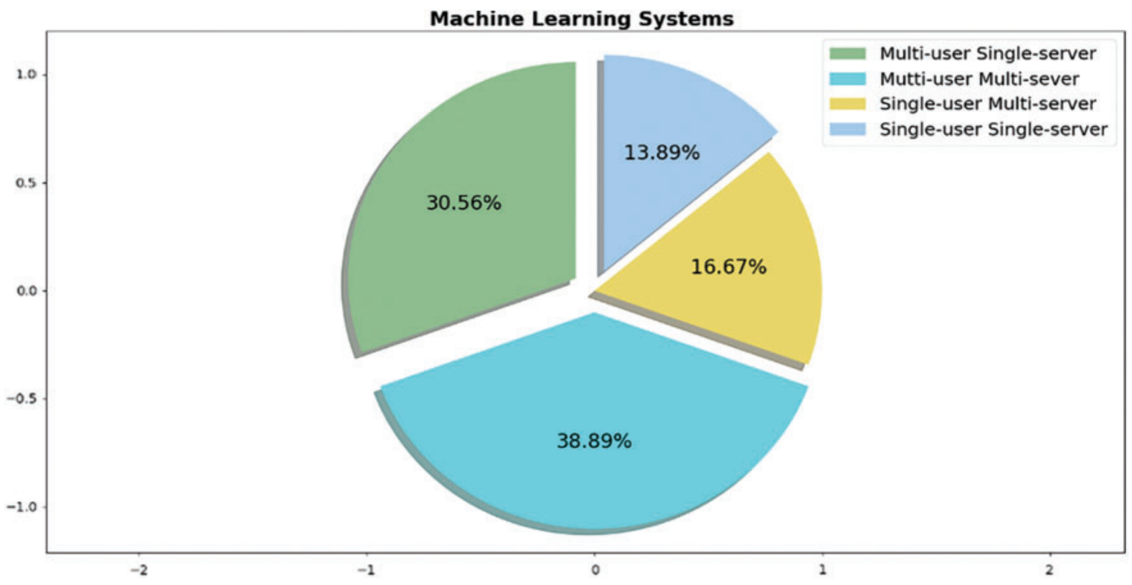

Figure 13: Machine learning systems

5.4 Utilized Tools

The following technical question is addressed in this section:

Q4: What evaluation tools are utilized for assessing the ML and Non-ML based approaches?

In Fig. 15, it can be observed that 52.78 percent of research publications do not specify evaluation methods for the proposed models. Furthermore, in the literature, 22.22 percent and 15.28 percent of articles employed Python and MATLAB, respectively, to develop their model. Furthermore, 4.17 percent and 2.78 percent of the study publications used the Java and iFogSim tools to assess and analyze the existing case studies. Finally, CloudSim and Google Cloud trace logs have the fewest participants of 1.39 percent each in the literature.

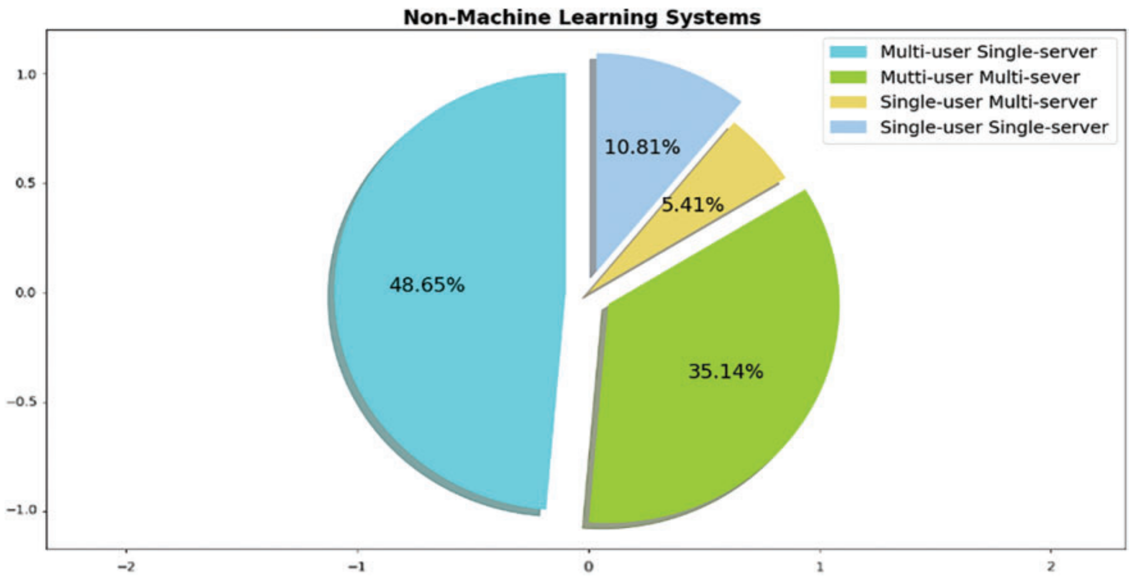

Figure 14: Non-machine learning systems

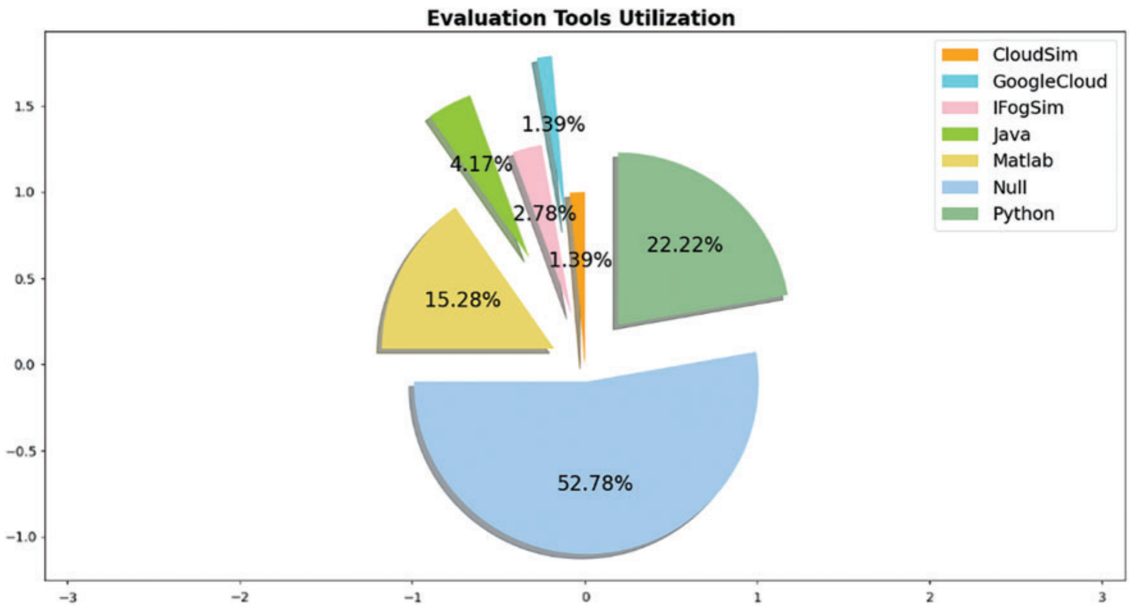

Figure 15: Evaluation tools utilization

6 Comparison of the Various Offloading Algorithms

In this subsection, we analyze the benefits and limitations of each algorithm, and we discuss the scenario for which these algorithms are well suited.

Because reinforcement learning can handle the curse of dimensionality problem mentioned in MDP, it is logical for a high-complexity offloading scenario like large-scale offloading. Furthermore, because reinforcement learning is an online method, it is appropriate for use in a dynamic environment. However, since end devices rarely support reinforcement learning, a powerful entity should be developed to implement reinforcement-learning for all end devices.

Offloading optimization and Lyapunov drift optimization can both be accomplished with Lyapunov optimization. As a result, it can be used in task offloading scenarios where load balancing and energy harvesting balance, for example, are essential. Although Lyapunov optimization is a centralized modeling method, it could be used in a large-scale offloading scenario because of its low computational complexity. On the other hand, Lyapunov optimization is a one-step optimization that can only achieve local optimization.

The inherently distributed nature of game theory models is an advantage. It is frequently utilized in multi-user offloading scenarios where users compete for computational or cellular resources. It, like MDP, can be used in a dynamic setting. Because a game theory requires each user to maximize their utility, it works well in large-scale offloading scenarios. On the other hand, Game theory models can only guarantee the Nash equilibrium, which may not be the optimal global solution.

MDP has the advantage of being used in both centralized and distributed environments. A centralized body decides for all users in the centralized situation, whereas each user makes their own offloading decision in the distributed scenario. The decision-maker must be aware of the system state at the decision to achieve optimal offloading. Therefore, data collecting remains a challenge. MDP is not scalable, which goes hand in hand with the curse of dimensionality. Nonetheless, MDP's dynamic decision-making makes it well suited for dynamic offloading scenarios in which a user can better their decision as the environment changes.

The advantage of (non-) convex optimization is that it can achieve global or near-global optimization. However, it necessitates the introduction of a centralized entity to collect all necessary data and make decisions for all users, making this approach non-scalable. As a result, this strategy is not appropriate for extensive offloading. However, it works effectively in a small-scale context, mainly when the offloading problem can be treated as a convex optimization problem, allowing for global optimization. Because (non-) convex optimization is a type of static modeling, it is ineffective for dynamic offloading scenarios in which the environment, such as channel state and network architecture, changes often.

6.1 Research Challenges and Open Issues

The following technical question is addressed in this section:

Q6: What are the future research directions and open perspectives of ML and Non-ML based offloading approaches?

1) Mobility

In many cases, the users' dynamic movement behavior becomes the decisive element in offloading the work. Even though few existing studies consider mobility, the case remains an open challenge. For new and emerging applications, building algorithms by learning the user behavior and network dynamics simultaneously, for example, is critical to cut communication and processing costs. Beyond

2021, high-user mobility applications such as UAVs, high-speed rails, moving hotspots, and 3D connectivity will be in great demand. Existing solutions would be impossible to apply in such intense conditions, not just in terms of accuracy but also minimal performance expectations. Since mobile devices with inherently high dynamic behavior can quickly move across different places, these devices might need to change their dedicated servers scattered in a geographically wide area. As the main challenge, it is required to have a proper mobility management technique to keep the connection with the edge server despite leaving the place of origin to receive the high dynamic content optimizations. On the other hand, to keep these connections, it is required for related organizations to offer standard protocols and unit platforms, as well. Notwithstanding their importance, mobility issues have been poorly or insufficiently addressed in the literature on MEC contexts.

2) Offloading Modeling in Edge Intelligence

With the proliferation of edge computing and IoT, a recent trend is to integrate artificial intelligence into edge computing, which gives rise to a new edge paradigm, namely, the emergence of edge Intelligence. In a three-tier edge intelligence architecture, a DNN with multiple layers is deployed at both the edge computing server and the cloud server, i.e., the front layers of the DNN are deployed at the edge server; the remaining layers are deployed at the cloud server. The output of the edge server is offloaded to the cloud, but the challenge here is deciding on which layer should be offloaded to the cloud, i.e., how to deploy the DNN. In a more typical scenario, multiple edge intelligence servers work together to provide Artificial Intelligence (AI) service for end devices. The offloading modeling for AI tasks should consider the AI performance, i.e., offloading to an edge intelligence server that has already been trained with similar tasks. Although some works [134–138] have contributed to this area, offloading modeling in edge intelligence is still an open issue.

3) Fault Tolerance

Aside from privacy and security, fault tolerance is crucial in establishing confidence in offloading at the edge. The tendency of a system to perform its intended purpose despite failures is known as fault tolerance [139,140]. Fault-tolerant computation refers to computing correctly even when a system contains mistakes. Mobility is one of the most vital requirements during offloading. The autonomy of communication and freedom of movement are critical factors for user experience. Even yet, achieving continuous connectivity and ongoing access to an edge server while moving presents particular challenges. Network bandwidth and data transfer speeds, for example, may fluctuate, or a connection may be lost. Following the application, some attributes are required to correctly analyze a system's behavior. Some of these characteristics, such as component reliability, availability, and failure probability, can be represented by various mathematical models, including the Binomial distribution, Markov model, and Poisson distribution, depending on the conditions and situations of the problems solved. Despite its importance, few studies on computation offloading consider fault-tolerance issues. Yang et al. [141] looked at service reliability and delays and errors. Through code-splitting, Liu et al. [98] represented the dependency of task components as a Directed Acyclic Graph. They used the service failure probability of MEC-enabled AR service to create a trade-off between latency and reliability. Qiao et al. [142] collaborated in a collaboratively self-organized D2D edge computing network to utilize mobile devices' processing capabilities while ensuring service availability utilizing energy harvesting technologies. Therefore, fault tolerance can be classified as a hot topic with a high significance level. To ensure the successful transmission and implementation of the task and decrease error, task offloading should be upgraded with fault tolerance mechanisms.

4) Interoperability

Interconnectivity, system model infrastructure, and interoperability interface are the three fundamental scope of interoperability. It is necessary to pay attention to the system's interoperability which is a complex challenge to incorporate security features as part of intercommunication. In addition, paying attention to interoperability factors while exchanging diverse data kinds across different levels and devices of a typical system or between different systems and paradigms as a form of offloading is a significant difficulty to be overcome. In general, interoperability, or the capacity to communicate between systems and concepts, is required to increase the system's vital metrics in mobile edge computing. Data could be transferred, transmitted, or handled together according to interoperability principles in interconnectivity and intercommunication. True interoperability requires an intermediate interface, known as a controller, to be in a structure that allows interaction between the system's pieces. The structure and system model in such a system should be highly adaptive to facilitate connectivity between mobile devices and servers because of the high dynamic behavior of the MEC environment, which involves high data rates on the one hand and heterogeneity on the other.

5) Privacy and Security

Because task offloading includes a large volume of data sent to third-party Edge infrastructures, security, and privacy [143] considerations are critical. These ideas can be approached from a variety of perspectives. One way to defend the computation offloading process is to provide confidentiality, integrity, availability, access control, and authentication between end devices and edge servers. Qi et al. [144] proposed a privacy-aware data fusion for smart cities to protect user-sensitive data during offloading. Many of the security concerns raised in this context are similar to those raised in the area of cloud computing. Edge computing's unique characteristics, such as restricted end-device resources and wireless access, make security procedures more difficult to implement. Another feature of the Edge server is that it serves as an authentication server or security source for end devices. End devices with limited resources are thought not to support complex security mechanisms [145]. To alleviate this issue, end-device security mechanisms should be offloaded to an edge server, which performs these mechanisms on behalf of the end devices due to the complex nature of end devices, including multiple types of communication protocols and dynamic security setups. Offloading compute or security functions to edge servers raise several privacy issues. Threats to Edge architectures are primarily focused on data transfer between the network's different nodes. Various encoding and critical cryptographic approaches and secure equipment execution are among the proposed solutions. Nonetheless, many of these technologies possess drawbacks in their implementations when used independently; for example, cryptographic keys may be too large, substantially increasing the volume of sent and stored data, and computations on encrypted data are still in the initial phases of the study. Edge-related security and privacy concerns are evolving rapidly, making it difficult to react to and defend against.

7 Conclusion

Task offloading is a critical technology for edge computing that can prolong battery life, reduce latency and improve application performance. In practice, many factors affect task offloading, which makes many offloading unable to achieve their expected objectives. This study presented a thorough examination of current task offloading and resource allocation in edge computing, covering offloading strategies, algorithms, and factors that influence offloading. We then grouped offloading strategies into two categories, Full and partial offloading. Additionally, we categorized offloading algorithms into machine learning and non-machine learning and discussed some general factors affecting offloading. Lastly, some research challenges and issues in edge computing were presented.

Funding Statement: This work is supported by the National Natural Science Foundation of China (Grant No. 61872002); Anhui Province Key Research and Development Program Project (Grant No. 201904a05020091).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Wang, F., Zhu, M., Wang, M., Khosravi, M. R., Ni, Q. et al. (2021). 6G-Enabled short-term forecasting for large-scale traffic flow in massive IoT based on time-aware locality-sensitive hashing. *IEEE Internet of Things Journal, 8(7),* 5321–5331. DOI 10.1109/JIOT.2020.3037669.
2. Wei, D., Ning, H., Shi, F., Wan, Y., Xu, J. et al. (2021). Dataflow management in the Internet of Things: Sensing, control, and security. *Tsinghua Science and Technology, 26(6),* 918–930. DOI 10.26599/TST.2021.9010029.
3. Zheng, T., Wan, J., Zhang, J., Jiang, C., Jia, G. (2020). A survey of computation offloading in edge computing. *Proceedings of the 2020 International Conference on Computer, Information and Telecommunication Systems*, Hangzhou, China.
4. Saeik, F., Avgeris, M., Spatharakis, D., Santi, N., Dechouniotis, D. et al. (2021). Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks, 195(3),* 108177. DOI 10.1016/j.comnet.2021.108177.
5. Zhao, T., Zhou, S., Guo, X., Zhao, Y., Niu, Z. (2015). A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. *2015 IEEE Globecom Work*, San Diego, CA, USA.
6. Xu, F., Yang, W., Li, H. (2020). Computation offloading algorithm for cloud robot based on improved game theory. *Computers and Electrical Engineering, 87(4),* 1–11. DOI 10.1016/j.compeleceng.2020.106764.
7. Wang, L., Zhang, X., Wang, T., Wan, S., Srivastava, G. et al. (2021). Diversified and scalable service recommendation with accuracy guarantee. *IEEE Transactions on Computational Social Systems, 8(5),* 1182–1193. DOI 10.1109/TCSS.2020.3007812.
8. Xu, Y., Qi, L., Dou, W., Yu, J. (2017). Privacy-preserving and scalable service recommendation based on simhash in a distributed cloud environment. *Complexity, 2017(2),* 1–9. DOI 10.1155/2017/3437854.
9. Wang, F., Zhu, H., Srivastava, G., Li, S., Khosravi, M. R. et al. (2021). Robust collaborative filtering recommendation with user-item-trust records. *IEEE Transactions on Computational Social Systems, 3,* 1–11. DOI 10.1109/TCSS.2021.3064213.
10. Shakarami, A., Ghobaei-Arani, M., Masdari, M., Hosseinzadeh, M. (2020). A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective. *Journal of Grid Computing, 18(4),* 639–671. DOI 10.1007/s10723-020-09530-2.
11. Lin, H., Zeadally, S., Chen, Z., Labiod, H., Wang, L. (2020). A survey on computation offloading modeling for edge computing. *Journal of Network and Computer Applications, 169,* 102781. DOI 10.1016/j.jnca.2020.102781.
12. Xu, X., Li, H., Xu, W., Liu, Z., Yao, L. et al. (2022). Artificial intelligence for edge service optimization in internet of vehicles: A survey. *Tsinghua Science and Technology, 27(2),* 270–287. DOI 10.26599/TST.2020.9010025.

13. Ahmed, E., Gani, A., Sookhak, M., Hamid, S. H. A., Xia, F. (2015). Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications, 52(11),* 52–68. DOI 10.1016/j.jnca.2015.02.003.
14. Shakarami, A., Ghobaei-Arani, M., Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks, 182(2),* 107496. DOI 10.1016/j.comnet.2020.107496.
15. Bhattacharya, A., De, P. (2017). A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications, 78,* 97–115. DOI 10.1016/j.jnca.2016.10.023.
16. Carvalho, G., Cabral, B., Pereira, V., Bernardino, J. (2020). Computation offloading in edge computing environments using artificial intelligence techniques. *Engineering Applications of Artificial Intelligence, 95,* 103840. DOI 10.1016/j.engappai.2020.103840.
17. Guevara, J. C., Torres, R., da, S., da Fonseca, N. L. S. (2020). On the classification of fog computing applications: A machine learning perspective. *Journal of Network and Computer Applications, 159(8),* 102596. DOI 10.1016/j.jnca.2020.102596.
18. Shan, X., Zhi, H., Li, P., Han, Z. (2018). A Survey on computation offloading for mobile edge Computing information. *Proceedings of 4th IEEE International Conference on Big Data Security on Cloud, BigDataSecurity 2018, 4th IEEE International Conference on High Performance and Smart Computing, HPSC, 2018 and 3rd IEEE International Conference on Intelligent Data and Securit*, pp. 248–251. Omaha, NE, USA.
19. Phan, L. A., Nguyen, D. T., Lee, M., Park, D. H., Kim, T. (2021). Dynamic fog-to-fog offloading in SDN-based fog computing systems. *Future Generation Computer Systems, 117(1),* 486–497. DOI 10.1016/j.future.2020.12.021.
20. Zhang, Y., Li, J., Li, Y., Xu, D., Ahmed, M. et al. (2019). Cellular traffic offloading via link prediction in opportunistic networks. *IEEE Access, 7,* 39244–39252. DOI 10.1109/ACCESS.2019.2891642.
21. Liu, H., Kou, H., Yan, C., Qi, L. (2019). Link prediction in paper citation network to construct paper correlation graph. *Eurasip Journal on Wireless Communications and Networking, 2019(1),* 233. DOI 10.1186/s13638-019-1561-7.
22. Zhang Y., Niyato D., Wang P. (2015). Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing, 14(12),* 2516–2529. DOI 10.1109/TMC.2015.2405539.
23. Wang, Z., Liang, W., Huang, M., Ma, Y. (2018). Delay-energy joint optimization for task offloading in mobile edge computing. http://arxiv.org/abs/1804.10416.
24. Malan, D. J. (2006). Getting started with amazon EC2 self-service, prorated super computing fun! https://cs.harvard.edu/malan/publications/ec2.pdf.
25. Polak, A. (2012). The beginner's guide to microsoft azure. www.connection.com/∼/media/pdfs/content/cloudtechnology/beginnersguidetomicrosoftazure.pdf?la=en.
26. Rego, P. A. L., Trinta, F. A. M., Hasan, M. Z., de Souza, J. N. (2019). Enhancing offloading systems with smart decisions, adaptive monitoring, and mobility support. *Wireless Communications and Mobile Computing, 2019,* 1–18. DOI 10.1155/2019/1975312.
27. Liu, F., Huang, Z., Wang, L. (2019). Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for IoT sensors. *Sensors, 19(5),* 1105. DOI 10.3390/s19051105.
28. Huang, L., Feng, X., Feng, A., Huang, Y., Qian, L. P. (2018). Distributed deep learning-based offloading for mobile edge computing networks. *Mobile Networks and Applications, 3(6),* 1123–1130. DOI 10.1007/s11036-018-1177-x.
29. Wang X., Ning Z., Wang L. (2018). Offloading in internet of vehicles: A fog-enabled real-time traffic management system. *IEEE Transactions on Industrial Informatics, 14(10),* 4568–4578. DOI 10.1109/TII.2018.2816590.
30. Huang, D., Wang, P., Niyato, D. (2012). A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications, 11(6),* 1991–1995. DOI 10.1109/TWC.2012.041912.110912.

31. Yang, L., Cao, J., Tang, S., Li, T., Chan, A. T. S. (2012). A framework for partitioning and execution of data stream applications in mobile cloud computing. *Proceedings of 2012 IEEE 5th International Conference on Cloud Computing*, pp. 794–802. Honolulu, HI, USA.
32. Huang, L., Feng, X., Zhang, C., Qian, L., Wu, Y. (2019a). Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks, 5(1),* 10–17. DOI 10.1016/j.dcan.2018.10.003.
33. Tse, D., Viswanath, P. (2012). The wireless channel. *Fundamentals of Wireless Communication*, pp. 10–48. Cambridge, UK, Cambridge University Press.
34. Rabbachin, A., Quek, T. Q. S., Shin, H., Win, M. Z. (2011). Cognitive network interference. *IEEE Journal on Selected Areas in Communications, 29(2),* 480–493. DOI 10.1109/JSAC.2011.110219.
35. Yang, C., Liu, Y., Chen, X., Zhong, W., Xie, S. (2019). Efficient mobility-aware task offloading for vehicular edge computing networks. *IEEE Access, 7,* 26652–26664. DOI 10.1109/ACCESS.2019.2900530.
36. Zhao, P., Tian, H., Qin, C., Nie, G. (2017). Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing. *IEEE Access, 5,* 11255–11268. DOI 10.1109/ACCESS.2017.2710056.
37. Education, I. cloud (2020). Machine learning. https://www.ibm.com/cloud/learn/machine-learning.
38. Ai, E. (2020). What is machine learning? A definition. https://www.expert.ai/blog/machine-learning-definition/.
39. Liao, X., Zheng, D., Cao, X. (2021). Coronavirus pandemic analysis through tripartite graph clustering in online social networks. *Big Data Mining and Analytics, 4(4),* 242–251. DOI 10.26599/BDMA.2021.9020010.
40. Berwick, R. (2003). An idiot's guide to support vector machines (SVMS): A new generation of learning algorithms key ideas. *Village Idiot*, pp. 1–28. http://www.cs.ucf.edu/courses/cap6412/fall2009/papers/Berwick2003.pdf.
41. Moore, A. W. (2003). Support vector machines why maximum margin? https://people.idsia.ch/~juergen/mooresvm.pdf.
42. Wu, S., Xia, W., Cui, W., Chao, Q., Lan, Z. et al. (2018). An efficient offloading algorithm based on support vector machine for mobile edge computing in vehicular networks. *2018 10th International Conference on Wireless Communications and Signal Processing*, pp. 1–6. Hangzhou, China.
43. Majeed, A. A., Khan, A. U. R., Ulamin, R., Muhammad, J., Ayub, S. (2017). Code offloading using support vector machine. *6th International Conference on Innovative Computing Technology*, pp. 98–103. Dublin, Ireland.
44. Wang, S., Chen, M., Saad, W., Yin, C. (2020). Federated learning for energy-efficient task computing in wireless networks. *IEEE International Conference on Communications*, pp. 20–25. Dublin, Ireland.
45. Wang, W., Wang, Z., Zhou, Z., Deng, H., Zhao, W. et al. (2022). Anomaly detection of industrial control systems based on transfer learning. *Applied Mathematics and Computation, 412(6),* 821–832. DOI 10.26599/TST.2020.9010041.
46. Chen, H., Zhang, Y., Cao, Y., Xie, J. (2021). Security issues and defensive approaches in deep learning frameworks. *Tsinghua Science and Technology, 26(6),* 894–905. DOI 10.26599/TST.2020.9010050.
47. Hou, C., Wu, J., Cao, B., Fan, J. (2021). A deep-learning prediction model for imbalanced time series data forecasting. *Big Data Mining and Analytics, 4(4),* 266–278. DOI 10.26599/BDMA.2021.9020011.
48. Zhang, S., Liu, H., He, J., Han, S., Du, X. (2021). Deep sequential model for anchor recommendation on live streaming platforms. *Big Data Mining and Analytics, 4(3),* 173–182. DOI 10.26599/BDMA.2021.9020002.
49. Ali, Z., Jiao, L., Baker, T., Abbas, G., Abbas, Z. H. et al. (2019). A deep learning approach for energy efficient computational offloading in mobile edge computing. *IEEE Access, 7,* 149623–149633. DOI 10.1109/ACCESS.2019.2947053.

50. Yu, S., Wang, X., Langar, R. (2018). Computation offloading for mobile edge computing: A deep learning approach. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–6. Montreal, QC, Canada.
51. Zhao, M., Zhou, K. (2019). Selective offloading by exploiting ARIMA-BP for energy optimization in mobile edge computing networks. *Algorithms, 12(2),* 48. DOI 10.3390/a12020048.
52. Rego, P. A. L., Cheong, E., Coutinho, E. F., Trinta, F. A. M., Hasany, M. Z. et al. (2017). Decision tree-based approaches for handling offloading decisions and performing adaptive monitoring in MCC systems. *Proceedings of 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 74–81. San Francisco, CA, USA.
53. Crutcher, A., Koch, C., Coleman, K., Patman, J., Esposito, F. et al. (2017). Hyperprofile-Based computation offloading for mobile edge networks. *Proceedings-14th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 525–529. Orlando, FL, USA.
54. Sheng, J., Hu, J., Teng, X., Wang, B., Pan, X. (2019). Computation offloading strategy in mobile edge computing. *Information, 10(6),* 1–20. DOI 10.3390/info10060191.
55. Kiran, N., Pan, C., Wang, S., Yin, C. (2020). Joint resource allocation and computation offloading in mobile edge computing for SDN based wireless networks. *Journal of Communications and Networks, 22(1),* 1–11. DOI 10.1109/JCN.2019.000046.
56. Hossain, M. S., Nwakanma, C. I., Lee, J. M., Kim, D. S. (2020). Edge computational task offloading scheme using reinforcement learning for IIoT scenario. *ICT Express, 6(4),* 291–299. DOI 10.1016/j.icte.2020.06.002.
57. Chen X., Zhang, H., Wu, C., Mao, S., Ji, Y. et al. (2018). Performance optimization in mobile-edge computing via deep reinforcement learning. *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, Chicago, IL, USA.
58. Zhao, R., Wang, X., Xia, J., Fan, L. (2020). Deep reinforcement learning based mobile edge computing for intelligent Internet of Things. *Physical Communication, 43(9),* 101184. DOI 10.1016/j.phycom.2020.101184.
59. Tong, Z., Deng, X., Ye, F., Basodi, S., Xiao, X. et al. (2020). Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Information Sciences, 537(2),* 116–131. DOI 10.1016/j.ins.2020.05.057.
60. Xu, Z., Wang, Y., Tang, J., Wang, J., Gursoy, M. C. (2017). A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6. Paris, France.
61. Shan, N., Cui, X., Gao, Z. (2020). "DRL + FL": An intelligent resource allocation model based on deep reinforcement learning for mobile edge computing. *Computer Communications, 160,* 14–24. DOI 10.1016/j.comcom.2020.05.037.
62. Ho, T. M., Nguyen, K. (2020). Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network : A deep reinforcement learning approach. *IEEE Transactions on Mobile Computing, 21(7),* 2421–2435. DOI 10.1109/TMC.2020.3043736.
63. Chen, C., Zhang, Y., Wang, Z., Wan, S., Pei, Q. (2021). Distributed computation offloading method based on deep reinforcement learning in ICV. *Applied Soft Computing, 103,* 107108. DOI 10.1016/j.asoc.2021.107108.
64. Lin, B., Lin, K., Lin, C., Lu, Y., Huang, Z. et al. (2021). Computation offloading strategy based on deep reinforcement learning for connected and autonomous vehicle in vehicular edge computing. *Journal of Cloud Computing: Advances, Systems and Applications, 10(1),* 33. DOI 10.1186/s13677-021-00246-6.
65. Alam, M. G. R., Hassan, M. M., Uddin, M. Z., Almogren, A., Fortino, G. (2019). Autonomic computation offloading in mobile edge for IoT applications. *Future Generation Computer Systems, 90(1),* 149–157. DOI 10.1016/j.future.2018.07.050.

66. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y. et al. (2019). Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal, 6(3),* 4005–4018. DOI 10.1109/JIOT.2018.2876279.

67. Ke, H., Wang, J., Deng, L., Ge, Y., Wang, H. (2020). Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks. *IEEE Transactions on Vehicular Technology, 69(7),* 7916–7929. DOI 10.1109/TVT.2020.2993849.

68. Huang, L., Bi, S., Zhang, Y. J. A. (2020). Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing, 19(11),* 2581–2593. DOI 10.1109/TMC.2019.2928811.

69. Liu Y., Cui, Q., Zhang, J., Chen, Y., Hou, Y. (2019). An actor-critic deep reinforcement learning based computation offloading for three-tier mobile computing networks. *11th International Conference on Wireless Communications and Signal Processing*, Xi'an, China.

70. Wang, Y., Fang, W., Ding, Y., Xiong, N. (2021). Computation offloading optimization for UAV-assisted mobile edge computing: A deep deterministic policy gradient approach. *Wireless Networks, 27(4),* 2991–3006. DOI 10.1007/s11276-021-02632-z.

71. Chen, Z., Wang, X. (2020). Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach. *Eurasip Journal on Wireless Communications and Networking, 2020(1),* 188. DOI 10.1186/s13638-020-01801-6.

72. Ko, H., Lee, J., Pack, S. (2018). Spatial and temporal computation offloading decision algorithm in edge cloud-enabled heterogeneous networks. *IEEE Access, 6,* 18920–18932. DOI 10.1109/ACCESS.2018.2818111.

73. Wei, Z., Zhao, B., Su, J., Lu, X. (2019). Dynamic edge computation offloading for Internet of Things with energy harvesting: A learning method. *IEEE Internet of Things Journal, 6(3),* 4436–4447. DOI 10.1109/JIOT.2018.2882783.

74. Zhang, X., Zhang, J., Liu, Z., Cui, Q., Tao, X. et al. (2020). MDP-based task offloading for vehicular edge computing under certain and uncertain transition probabilities. *IEEE Transactions on Vehicular Technology, 69(3),* 3296–3309. DOI 10.1109/TVT.2020.2965159.

75. Eom, H., Juste, P. S., Figueiredo, R., Tickoo, O., Illikkal, R. et al. (2013). Machine learning-based runtime scheduler for mobile offloading framework. *Proceedings of 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pp. 17–25. Dresden, Germany.

76. Lu, H., Gu, C., Luo, F., Ding, W., Liu, X. (2020). Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Generation Computer Systems, 102(5),* 847–861. DOI 10.1016/j.future.2019.07.019.

77. Huang, L., Feng, X., Zhang, L., Qian, L., Wu, Y. (2019). Multi-server multi-user multi-task computation offloading for mobile edge computing networks. *Sensors, 19(6),* 1446. DOI 10.3390/s19061446.

78. Darbanian, E., Rahbari, D., Ghanizadeh, R., Nickray, M. (2020). Improving response time time of task offloading by random forest, extra-trees and adaboost classifiers in mobile fog computing. *Jordanian Journal of Computers and Information Technology, 6(4),* 486–498. DOI 10.5455/jjcit.71-1590557276.

79. Cui, Y., Zhang, D., Zhang, T., Chen, L., Piao, M. et al. (2020). Novel method of mobile edge computation offloading based on evolutionary game strategy for IoT devices. *AEU-International Journal of Electronics and Communications, 118,* 153134. DOI 10.1016/j.aeue.2020.153134.

80. Mao, Y., Zhang, J., Song, S. H., Letaief, K. B. (2016). Power-delay tradeoff in multi-user mobile-edge computing systems. *2016 IEEE Global Communications Conference*, Washington DC, USA.

81. Liu, C. F., Bennis, M., Poor, H. V. (2018). Latency and reliability-aware task offloading and resource allocation for mobile edge computing. *2017 IEEE Globecom Workshops*, pp. 1–7. Singapore.

82. de Haan, L., Ferreira, A. (2006). Extreme value theory: An introduction. In: *Springer series in operations research and financial engineering,* pp. 1–413. Secaucus, NJ, USA. Springer Science & Business Media.

83. Chen, X, Jiao, L., Li, W. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking, 24(5),* 2795–2808. DOI 10.1109/TNET.2015.2487344.
84. Zhou, S., Jadoon, W. (2020). The partial computation offloading strategy based on game theory for multi-user in mobile edge computing environment. *Computer Networks, 178(2020),* 107334. DOI 10.1016/j.comnet.2020.107334.
85. Alioua, A., Djeghri, H. E., Cherif M. E. T., Senouci, S. M., Sedjelmaci, H. (2020). UAVs for traffic monitoring: A sequential game-based computation offloading/sharing approach. *Computer Networks, 177,* 107273. DOI 10.1016/j.comnet.2020.107273.
86. Liu, Y, Xu, C., Zhan, Y., Liu, Z., Guan, J. et al. (2017). Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach. *Computer Networks, 129(2),* 399–409. DOI 10.1016/j.comnet.2017.03.015.
87. Anbalagan, S., Kumar, D., Raja, G., Balaji, A. (2019). SDN assisted Stackelberg game model for LTE-WiFi offloading in 5G networks. *Digital Communications and Networks, 5(4),* 268–275. DOI 10.1016/j.dcan.2019.10.006.
88. Kim, S. H., Park, S., Chen, M., Youn, C. H. (2018). An optimal pricing scheme for the energy-efficient mobile edge computation offloading with OFDMA. *IEEE Communications Letters, 22(9),* 1922–1925. DOI 10.1109/LCOMM.2018.2849401.
89. Wang, C., Liang, C., Yu, F. R., Chen, Q., Tang, L. (2017). Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications, 16(8),* 4924–4938. DOI 10.1109/TWC.2017.2703901.
90. Bi, S., Zhang, Y. J. (2017). Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Transactions on Wireless Communications, 17(6),* 4177–4190. DOI 10.1109/TWC.2018.2821664.
91. Tang, Q., Lyu, H., Han, G., Wang, J., Wang, K. (2020). Partial offloading strategy for mobile edge computing considering mixed overhead of time and energy. *Neural Computing and Applications, 32(19),* 15383–15397. DOI 10.1007/s00521-019-04401-8.
92. Baidas, M. W. (2021). Resource allocation for offloading-efficiency maximization in clustered NOMA-enabled mobile edge computing networks. *Computer Networks, 189,* 107919. DOI 10.1016/j.comnet.2021.107919.
93. Yang, S. (2020). A joint optimization scheme for task offloading and resource allocation based on edge computing in 5G communication networks. *Computer Communications, 160(1),* 759–768. DOI 10.1016/j.comcom.2020.07.008.
94. Parrilo, P. A. (2005). Semidefinite programming relaxations for semi algebraic problems. *Functional Ecology, 19(2),* 594–601. DOI 10.1007/s10107-003-0387-5.
95. Freund, R. (2004). Introduction to Semidefinite Programming (SDP). Massachusetts Institute of Technology. https://ocw.mit.edu/courses/15-084j-nonlinear-programming-spring-2004/a632b565602fd2eb3be574c537eea095_lec23_semidef_opt.pdf.
96. Overton, M., Wolkowicz, H. (1997). Semidefinite programming. *Mathematical Programming, 77(1),* 105–109. DOI 10.1007/BF02614431.
97. Chen, M. H., Liang, B., Dong, M. (2016). Joint offloading decision and resource allocation for multi-user multi-task mobile cloud. *2016 IEEE International Conference on Communications*, vol. 67, pp. 2–7. Kuala Lumpur, Malaysia.
98. Liu, J., Zhang, Q. (2019). Code-partitioning offloading schemes in mobile edge computing for augmented reality. *IEEE Access, 7,* 11222–11236. DOI 10.1109/ACCESS.2019.2891113.
99. Li, S., Ge, H., Chen, X., Liu, L., Gong, H. et al. (2021). Computation offloading strategy for improved particle swarm optimization in mobile edge computing. *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics*, pp. 375–381. Chengdu, China.

100. Zhang, N., Guo, S., Dong, Y., Liu, D. (2020). Joint task offloading and data caching in mobile edge computing networks. *Computer Networks, 182,* 107446. DOI 10.1016/j.comnet.2020.107446.

101. Kuang, L., Gong, T., Ouyang, S., Gao, H., Deng, S. (2020). Offloading decision methods for multiple users with structured tasks in edge computing for smart cities. *Future Generation Computer Systems, 105(6),* 717–729. DOI 10.1016/j.future.2019.12.039.

102. Zhao, H., Du, W., Liu, W., Lei, T., Lei, Q. (2018). QoE aware and cell capacity enhanced computation offloading for multi-server mobile edge computing systems with energy harvesting devices. *Proceedings of 2018 IEEE Smart World, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovations, SmartWorld/UIC/ATC/ScalCom/CBDCo*, pp. 671–678, Guangzhou, China.

103. Shahryari, O. K., Pedram, H., Khajehvand, V., TakhtFooladi, M. D. (2021). Energy and task completion time trade-off for task offloading in fog-enabled IoT networks. *Pervasive and Mobile Computing, 74(5),* 101395. DOI 10.1016/j.pmcj.2021.101395.

104. Hussain, M. M., Beg, M. M. S. (2021). CODE-V: Multi-hop computation offloading in vehicular fog computing. *Future Generation Computer Systems, 116(3),* 86–102. DOI 10.1016/j.future.2020.09.039.

105. Wang, X., Xu, W., Jin, Z. (2017). A hidden markov model based dynamic scheduling approach for mobile cloud telemonitoring. *2017 IEEE EMBS International Conference on Biomedical and Health Informatics, BHI 2017*, pp. 273–276. Orlando, FL, USA.

106. Jazayeri, F., Shahidinejad, A., Ghobaei-Arani, M. (2021). A latency-aware and energy-efficient computation offloading in mobile fog computing: A hidden Markov model-based approach. *Journal of Supercomputing, 77(5),* 4887–4916. DOI 10.1007/s11227-020-03476-8.

107. Liao, Y., Shou, L., Yu, Q., Ai, Q., Liu, Q. (2020). Joint offloading decision and resource allocation for mobile edge computing enabled networks. *Computer Communications, 154(1),* 361–369. DOI 10.1016/j.comcom.2020.02.071.

108. Ren, J., Yu, G., Cai, Y., He, Y. (2018). Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications, 17(8),* 5506–5519. DOI 10.1109/TWC.2018.2845360.

109. Xing, H., Liu, L., Xu, J., Nallanathan, A. (2019). Joint task assignment and resource allocation for D2D-enabled mobile-edge computing. *IEEE Transactions on Communications, 67(6),* 4193–4207. DOI 10.1109/TCOMM.2019.2903088.

110. You, C., Huang, K., Chae, H., Kim, B. H. (2017). Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications, 16(3),* 1397–1411. DOI 10.1109/TWC.2016.2633522.

111. Xu, Y., Wang, Y., Yang, J. (2020). Meta-heuristic search based model for task offloading and time allocation in mobile edge computing. *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, pp. 117–122. Tianjin, China.

112. Wang, Y., Min, S., Wang, X., Liang, W., Li, J. (2016). Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications, 64(10),* 4268–4282. DOI 10.1109/TCOMM.2016.2599530.

113. Guo, S., Xiao, B., Yang, Y., Yang, Y. (2016). Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. *35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA.

114. Yan, J., Bi, S., Zhang, Y. J., Tao, M. (2020). Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. *IEEE Transactions on Wireless Communications, 19(1),* 235–250. DOI 10.1109/TWC.2019.2943563.

115. Zhong, S., Guo, S., Yu, H., Wang, Q. (2021). Cooperative service caching and computation offloading in multi-access edge computing. *Computer Networks, 189,* 107916. DOI 10.1016/j.comnet.2021.107916.

116. Barbarossa, S., di Lorenzo, P., Sardellitti, S. (2014). Computation offloading strategies based on energy minimization under computational rate constraints. *European Conference on Networks and Communications*, pp. 16–20. Bologna, Italy.
117. Zhang, Y., Fu, J. (2021). Energy-efficient computation offloading strategy with tasks scheduling in edge computing. *Wireless Networks, 27(1),* 609–620. DOI 10.1007/s11276-020-02474-1.
118. Nath, S., Wu, J. (2020). Dynamic computation offloading and resource allocation for multi-user mobile edge computing. *2020 IEEE Global Communications Conference*. Taipei, Taiwan.
119. Feng, S., Chen, Y., Zhai, Q., Huang, M., Shu, F. (2021). Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms. *Eurasip Journal on Advances in Signal Processing, 36(1),* 462. DOI 10.1186/s13634-021-00751-5.
120. Li, N., Yang, S., Wang, Z., Hao, W., Zhu, Y. (2020). Multi-tier MEC offloading strategy based on dynamic channel characteristics. *IET Communications, 14(22),* 4029–4037. DOI 10.1049/iet-com.2020.0371.
121. Xu, R., Wang, Q. (2019). DPDK-accelerated partial offload for fine-grained HQoS. https://www.dpdk.org/wp-content/uploads/sites/35/2019/10/AcceleratedPartialOffload.pdf.
122. Kuang, Z., Li, L., Gao, J., Zhao, L., Liu, A. (2019). Partial offloading scheduling and power allocation for mobile edge computing systems. *IEEE Internet of Things Journal, 6(4),* 6774–6785. DOI 10.1109/JIOT.2019.2911455.
123. Dat, V. T., Truong, T. P., Nguyena, T. V., Noh, W., Cho, S. (2021). Partial computation offloading in noma-assisted mobile-edge computing systems using deep reinforcement learning. *IEEE Internet of Things Journal, 8(17),* 13196–13208. DOI 10.1109/JIOT.2021.3064995.
124. Wang, Z., Hao, W., Yan, L., Han, Z., Yang, S. (2020). Cooperative scheduling of multi-core and cloud resources: Fine-grained offloading strategy for multithreaded applications. *IET Communications, 14(10),* 1632–1641. DOI 10.1049/iet-com.2019.1060.
125. Qi, L., He, Q., Chen, F., Zhang, X., Dou, W. et al. (2020). Data-driven web apis recommendation for building web applications. *IEEE Transactions on Big Data,* 116, 1–15. DOI 10.1109/TBDATA.2020.2975587.
126. Zhang, J., Xu, Q. (2021). Attention-aware heterogeneous graph neural network. *Big Data Mining and Analytics, 4(4),* 233–241. DOI 10.26599/BDMA.2021.9020008.
127. Qi, L., Song, H., Zhang, X., Srivastava, G., Xu, X. et al. (2021). Compatibility-aware web API recommendation for mashup creation via textual description mining. *ACM Transactions on Multimedia Computing, Communications and Applications, 17,* 1–19. DOI 10.1145/3417293.
128. Saab, S. A., Chehab, A., Kayssi, A. (2013). Energy efficiency in mobile cloud computing: Total offloading selectively works. does selective offloading totally work. *2013 4th Annual International Conference on Energy Aware Computing Systems and Applications*, pp. 165–168. Istanbul, Turkey.
129. Messous, M. A., Senouci, S. M., Sedjelmaci, H., Cherkaoui, S. (2015). A game theory based efficient computation offloading in an UAV network. *IEEE Transactions on Vehicular Technology, 68(5),* 4964–4974. DOI 10.1109/TVT.2019.2902318.
130. Liu, Y., Wang, S., Huang, J., Yang, F. (2018). A computation offloading algorithm based on game theory for vehicular edge networks. *IEEE International Conference on Communications*, pp. 1–6. Kansas City, MO, USA.
131. Tang, L., He, S. (2018). Multi-user computation offloading in mobile edge computing: A behavioral perspective. *IEEE Network, 32(1),* 48–53. DOI 10.1109/MNET.2018.1700119.
132. You, C., Huang, K., Chae, H. (2016). Energy efficient mobile cloud computing powered by wireless energy transfer. *IEEE Journal on Selected Areas in Communications, 34(5),* 1757–1771. DOI 10.1109/JSAC.2016.2545382.
133. Tran, T. X., Pompili, D. (2019). Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology, 68(1),* 856–868. DOI 10.1109/TVT.2018.2881191.

134. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T. et al. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices, 52(4),* 615–629. DOI 10.1145/3093336.3037698.
135. Sun, W., Liu, J., Yue, Y. (2019). AI-enhanced offloading in edge computing: When machine learning meets industrial IoT. *IEEE Network, 33(5),* 68–74. DOI 10.1109/MNET.001.1800510.
136. Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L. et al. (2016). DeepX: A software accelerator for low-power deep learning inference on mobile devices. *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks*, Vienna, Austria.
137. Ran, X., Chen, H., Liu, Z., Chen, J. (2017). Delivering deep learning to mobile devices via offloading. *Proceedings of the 2017 Workshop on Virtual Reality and Augmented Reality Network, Part of SIGCOMM 2017*, pp. 42–47. Los Angeles CA USA.
138. Xu, M., Qian, F., Zhu, M., Huang, F., Pushp, S. et al. (2020). DeepWear: Adaptive local offloading for on-wearable deep learning. *IEEE Transactions on Mobile Computing, 19(2),* 314–330. DOI 10.1109/TMC.2019.2893250.
139. Ayav, T. (2021). Embedded computer systems lecture notes fault-tolerance. http://web.iyte.edu.tr/~tolgaayav/courses/ceng314/Fault-Tolerance.pdf.
140. Notes, L. (2009). Faults and fault-tolerance. https://homepage.divms.uiowa.edu/~ghosh/16612.week10.pdf.
141. Yang, T., Hu, Y., Gursoy, M. C., Schmeink, A., Mathar, R. (2018). Deep reinforcement learning based resource allocation in low latency edge computing networks. *Proceedings of the International Symposium on Wireless Communication Systems*, pp. 1–5. Lisbon, Portugal.
142. Qiao, G., Leng, S., Zhang, Y. (2019). Online learning and optimization for computation offloading in D2D edge computing and networks. *Mobile Networks and Applications, 29(7),* 1–9. DOI 10.1007/s11036-018-1176-y.
143. Zhang, Y. W., Pan, J., Qi, L., He, Q. (2021). Privacy-preserving quality prediction for edge-based IoT services. *Future Generation Computer Systems, 114(6),* 336–348. DOI 10.1016/j.future.2020.08.014.
144. Qi, L., Hu, C., Zhang, X., Khosravi, M. R., Sharma, S. et al. (2021). Privacy-aware data fusion and prediction with spatial-temporal context for smart city industrial environment. *IEEE Transactions on Industrial Informatics, 17(6),* 4159–4167. DOI 10.1109/TII.2020.3012157.
145. Hsu, R. H., Lee, J., Quek, T. Q. S., Chen, J. C. (2018). Reconfigurable security: Edge-computing-based framework for IoT. *IEEE Network, 32(5),* 92–99. DOI 10.1109/MNET.2018.1700284.