A Survey of Computational Offloading in Cloud/Edge-based Architectures: Strategies, Optimization Models and Challenges

Manal M. Alqarni[1,2*], Asma Cherif[1], and Entisar Alkayal[3]
[1] King Abdulaziz University, Faculty of Computing and Information Technology
Department of Information Technology
Jeddah 8030, Saudi Arabia
[e-mail: malqarni0323@stu.kau.edu.sa, acherif@.kau.edu.sa]
[2] Taif University, Faculty of Computing and Information Technology
Department of Information Technology
Taif, Saudi Arabia
[3] King Abdulaziz University, Faculty of Computing and Information Technology,
Department of Information Technology
Rabigh, Saudi Arabia
[e-mail: ealkayyal@kau.edu.sa]
[*]Corresponding author: Manal M. Alqarni

Received October 28, 2020; revised January 12, 2021; accepted March 3, 2021; published March 31, 2021

Abstract

In recent years, mobile devices have become an essential part of daily life. More and more applications are being supported by mobile devices thanks to edge computing, which represents an emergent architecture that provides computing, storage, and networking capabilities for mobile devices. In edge computing, heavy tasks are offloaded to edge nodes to alleviate the computations on the mobile side. However, offloading computational tasks may incur extra energy consumption and delays due to network congestion and server queues. Therefore, it is necessary to optimize offloading decisions to minimize time, energy, and payment costs.

In this article, different offloading models are examined to identify the offloading parameters that need to be optimized. The paper investigates and compares several optimization techniques used to optimize offloading decisions, specifically Swarm Intelligence (SI) models, since they are best suited to the distributed aspect of edge computing. Furthermore, based on the literature review, this study concludes that a Cuckoo Search Algorithm (CSA) in an edge-based architecture is a good solution for balancing energy consumption, time, and cost.

Keywords: Offloading, Optimization, Swarm Intelligence, MEC, Edge, Cloud Computing

1. Introduction

Over the last decade, mobile computing has been growing exponentially. According to [1], 78.1 billion people were using mobile devices in 2017. It is expected that this number will reach 258.2 billion by 2022 [1]. Indeed, mobile devices as smartphones, tablets, or even sensors provide a wide range of services and applications to end-users [2]. For example, real-time applications (such as e-commerce applications, gaming applications, and healthcare applications) that require significant computing resources for a high level of responsiveness, are becoming increasingly managed by these mobile devices [2]. However, mobile devices face a significant challenge because they lack certain crucial resources, particularly those related to storage capacity, computation power, bandwidth, and battery.

The Mobile Cloud Computing (MCC) paradigm was introduced as a way to overcome these limitations through the integration of mobile computing and cloud computing [2]. The paradigm migrates the computation and data of mobile applications into cloud data centers, which reduces mobile terminal overhead [3]. The cloud infrastructure in MCC is based on the idea of a pool of resources made available to end-users through several deployment models such as public and private clouds and services models. Advantages of the cloud include its position as a pay per use model that requires minimal management effort, etc. [4].

As mobile applications have become more ubiquitous in daily life, the requirement for high computation and short response time has also increased. However, the centralization of MCC increases the delay of computation responses, which harms the functionality of real-time applications. To address this issue, the Mobile Edge Computing (MEC) paradigm was proposed in September 2013 by the European Telecommunication Standard Institute (ETSI). MEC has a decentralized architecture based on three layers, which are the mobile device layer, the edge layer, and the cloud layer, as shown in **Fig. 1**. According to this paradigm, mobile devices can offload their operations and data into the edge layer through wireless communication to minimize the delays incurred by MCC.

MEC architecture offers MCC services through a Radio Access Network (RAN) at the edge layer of the mobile network, which is closer to mobile devices. MEC servers make the computations at MEC servers in base stations of a RAN, which reduces the occurrence of bottlenecks at data centers [2].

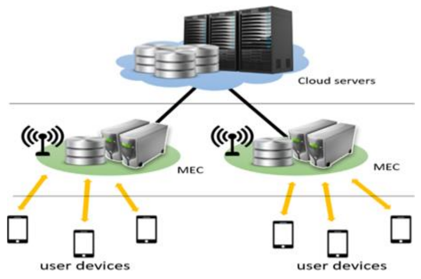


Fig. 1. MEC architecture [5]

Computational offloading has been proposed as a way of overcoming the resource limitations of mobile devices. Computational offloading is defined as the process of sending a task and related data to remote rich resources like cloud servers or edge servers to be processed as needed [2]. Several factors affect the offloading decision, including local device resources such as CPU, storage, and battery consumption, as well as delay in transformation through the network and the magnitude of the transferred data [3].

The MEC paradigm has emerged as the most popular model for mobile application offloading due to its capacity to reduce execution time, energy consumption, and delays. However, offloaded tasks may still face some issues such as waiting a long time in queues and network congestion. Therefore, it is necessary to optimize the offloading decisions to save time, energy, and cost [3].

Several studies have discussed MCC from several perspectives in order to review computation offloading. For instance, Gu et al. [4] reviewed enabling techniques and different computational resources used for partitioning and offloading. Besides, Azam [6] examined the technologies used for offloading in fog computing; Wang et al. [7] investigated computation offloading in MEC without considering optimization techniques, and Satria et al. [8] examined the optimization techniques applied in MCC.

Though many approaches have been proposed for computation offloading in the MCC architecture, there is a lack of a comparative study between these solutions. Moreover, none of the previous reviews has discussed the optimization techniques for computation offloading in MEC. Meanwhile, it is required to investigate AI-based models to fill the gap and propose a model that suits the characteristics of distributed edge-based architectures which serves distributed architectures besides QoS of real-time applications.

For these reasons, this paper investigates both offloading in general and its optimization models, specifically AI-based ones, with an emphasis on architectural design to identify the main parameters that should be taken into account for an optimized offloading model that fits edge computing. Thus, the paper offers an in-depth study of offloading and valuable support for researchers to investigate the optimization of computational offloading.

The main contributions of this paper are:

1) The presentation of an overview of existing literature and research related to computational offloading strategies including a comparison of offloading strategies in terms of architecture, application partitioning, resource allocation decisions, researchers' contributions, advantages, and disadvantages.

2) The provision of a comprehensive taxonomy of optimized offloading strategies.

3) A comparison of several AI algorithms used in the computational offloading process in terms of payment cost, time, energy, scalability, resource utilization, queue congestion, and the number of iterations.

The remainder of this article is organized as follows: A general overview of offloading is presented in Section 2. In Section 3, the optimization offloading models are surveyed and discussed. In Section 4, the main challenges and some future directions are presented. Finally, Section 4 concludes the paper.

2. General Overview on Offloading

The following section presents the main offloading types, strategies, and partitioning processes.

2.1 Offloading Types

In general, there are two main types of offloading: data offloading and computation offloading. Data offloading is used to transfer data from a mobile device, which has limited storage and capacity, into a repository located in the cloud. Zhang [9] investigated the process of data offloading from multi-mobile devices to multi-MEC servers. He proposed a game-centric pricing scheme in order to prioritize the data offloading according to mobile device and resource allocation through MEC. In addition, Xu et al. [10] investigated both the privacy and time consumption of data offloading in the edge system. They proposed a time-efficient offloading method (TEO) that included privacy conservation. According to their proposed method, they used an improved Strength Pareto Evolutionary Algorithm (SPEA2) to minimize time consumption and maximize the privacy of data. The experimental results demonstrated the reliability and efficiency of the TEO model. Similarly, Liu et al. [11] sought to minimize the cost of data offloading by proposing hybrid mobile offloading approaches according to the used network (D2D or WIFI). They based their proposal on the Finite Horizon Markov Decision Process (FHMDP). They also proposed an offloading algorithm for several time requirements (i.e. wide or tight). Furthermore, they used a monotone offloading algorithm to minimize the complexity of the offloading process.

Computation offloading is used to process intensive applications remotely in order to obtain benefits from powerful resources in cloud (or edge) servers that overcome the CPU and battery limitations on the mobile side [12]. In addition, the computation offloading can be applied in different infrastructures, such as MCC, MEC, and fog, as shown in **Fig. 2**.

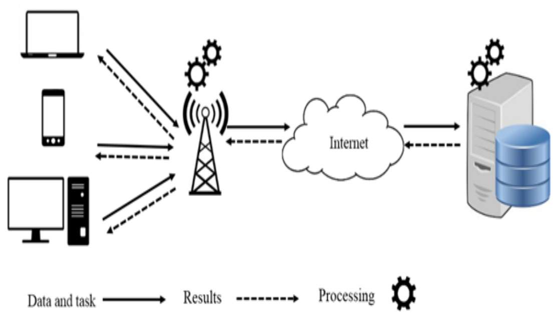
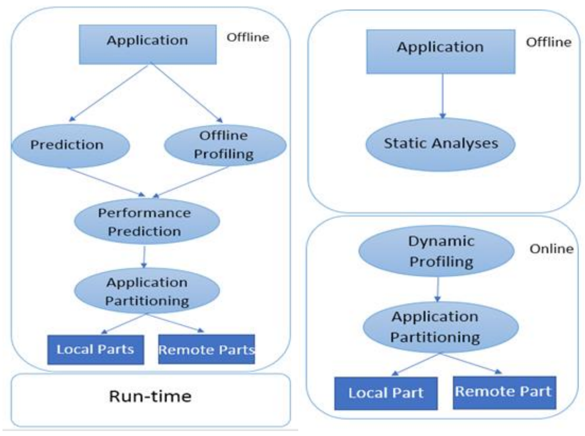


Fig. 2. Offloading architecture

Computation offloading can be static or dynamic. Static offloading first estimates offloading performance using offline profiling or models of performance estimation. Then, the application is partitioned into two main parts (locally and remotely executed), as presented in **Fig. 3(a)** [12]. The dynamic offloading (see **Fig. 3(b)**) starts with static analysis of the application's code and the required resources. Then, at run-time, dynamic profiling is conducted to partition the application into two main parts (locally and remotely executed) [12].



(a) Static offloading (b) Dynamic offloading
Fig. 3. Static and dynamic offloading

2.2 Offloading Strategies

The main offloading modes are binary and partial, as shown in **Fig. 4**. In the binary mode, the task is offloaded without partitioning [13-18]. On the other hand, the partial offloading mode includes task partitioning. So, some parts are processed locally, and some are offloaded remotely [19-21]. The partial offloading strategy can be classified into three categories: static, dynamic, or hybrid, as discussed in the next section.

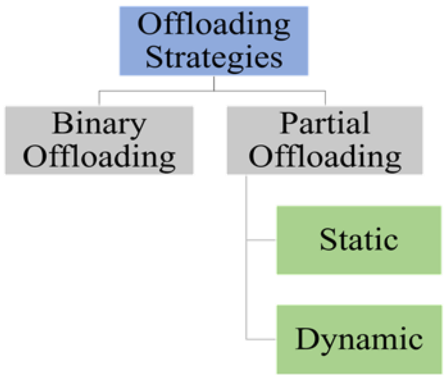


Fig. 4. Offloading strategies

It is important to note that partial offloading is more efficient. It has become the most used strategy in current frameworks because it reduces energy consumption, queue overhead, and transformation delay [4].

To achieve partial offloading, the application should be partitioned into different parts based on a specific granularity. This can be done statically, dynamically, or both. Static application partitioning consists of dividing the application into parts. This should be carried out by the developer at the design phase. Static partitioning improves the resource allocation process in terms of time because it reduces the partitioning time. However, it cannot be adapted easily to run-time changes such as environmental changes (bandwidth, resource conditions, etc.) [4].

Unlike static partitioning, dynamic partitioning splits the application into tasks at run time. It can be easily adapted to environmental changes automatically. Nevertheless, this partitioning technique is more challenging and time-consuming. An example of a framework that uses dynamic partitioning is the one suggested by Kovachev et al. [13]: a Mobile Augmentation Cloud Services (MACS) middleware that can be integrated with an existing Android application model to provide an adaptive model that allows for elastic offloading from mobile devices into the cloud. According to this model, a service manager is used to register each service. When an application invokes a service, the Android platform will create a proxy in that service. Another example is Clonecloud proposed by Chun et al. [14], which integrates both static application analyses and dynamic application partitioning in order to achieve the advantages of both.

2.3 Computational Offloading Challenges and Issues

This section presents the main challenges that appear in MEC environments through the computational offloading process.

- **Resource allocation:** One of the main challenges related to determining the number of resources needed to execute a task. If the resources are few, then the offload process will be

most encouraged. On the other hand, when the available resources are more than what the service requires, then the system will be under-utilized [15].
- **Scalability:** Real-time applications are often executed by various users at the same time, which is handled by different algorithms responsible for many requests from users. However, to allow the application to scale up without harm, the offloading process must be optimized [16].
- **Security:** Through the offloading process, tasks and users' data are moved through the network, which increases the risk of data theft and misuse. To overcome this issue, it is necessary to involve a trustworthy entity when the offloading decision is made [17].
- **The trade-off of energy consumption:** The offloading process itself consumes energy and bandwidth, so the offloading decision must be made according to this trade-off [16].
- **Decision making:** It is a challenge to decide whether to offload a task due to several parameters, such as delay, energy, or payment costs [18].
- **Availability:** Mobile devices have to connect to the edge/cloud all the time, which is challenging due to lack of network coverage, network congestion, low bandwidth, and any other network-related failures [16].
- **Mobility:** This presents a significant challenge when connecting to the edge [19].
- **Load balancing:** The edge node or datacentre may be overloaded at any time, which leads to delays and, in some situations, incorrect results. To avoid this, load-balance is used to share tasks with other edge nodes that are underloaded [15].
- **Resource utilization:** The number of resources on the edge is limited compared to the cloud infrastructure. Thus, resource utilization management is required in order to gain the most benefits from limited resources without causing system overhead [20].

The next sections present the main offloading frameworks proposed in the existing literature.

2.4 Main Computational Offloading Frameworks

This section investigates the main computational offloading frameworks proposed by researchers to enhance different parameters such as time, energy, scalability, etc. Some of them are investigated in relation to edge and others are investigated in relation to MCC.

Shu et al. [21] focused on the energy and delay trade-off of data offloading scheduling. They proposed the eTime — a strategy of data transmission between mobile devices and the cloud in which the offloading decision was conducted online in order to improve scheduling and make it time adaptive. To reduce the energy consumed by the communication, the eTime used online data traffic information to decide when it would send data remotely. This model was based on the energy-delay trade-off algorithm.

Zhang et al. [22] proposed a theoretical framework to reduce energy consumption based on wireless condition parameters for application offloading decisions. They used a threshold policy based on the wireless transmission model and the ratio of energy coefficients of the computation in both the mobile and cloud environments.

Kosta et al. [23] proposed ThinkAir, an offloading framework that supported dynamic resource allocation. The framework was based on a Virtual Machine (VM) and operated the virtualization on the cloud. Overall, their ThinkAir architecture was composed of three components: profilers, an application server, and an execution controller. At the mobile device, there were profilers (hardware, software, and network) that collected data and sent them to the energy prediction model, which dynamically assessed the local run of the task and transmitted the results to the execution controller. Then, the offloading decision was made based on data such as energy consumption and execution time of the previous tasks. ThinkAir demonstrated many benefits such as reduced energy consumption, reduced execution due to the parallelism of the VMs, and improved scalability. However, the authors noted that sensitive applications require more security arrangements.

Khanna et al. [24] proposed a Mobile Computation Offloading (MCO) model that offloaded application tasks into the cloud. The MCO was composed of four components, some of which worked on the mobile and others on the cloud. On the mobile side, the device profiler divided the application into several tasks and categorized them into tasks carried out locally or offloaded. Furthermore, the application analyzer component, which worked on the cloud, identified the energy and time consumption of the task. Then, it decided whether the task should be executed locally or remotely. Also, a network profiler component computed the transmission cost of the task. Then, the decision engine in the cloud took input from the application analyzer and network profiler to decide whether the task should be offloaded or not. According to the simulation results, the offloaded tasks consumed less execution time than the local tasks. Besides, with more offloaded tasks the execution time eventually increased, and energy consumption decreased. However, the model was not scalable because it was affected by the increasing number of tasks.

Orisini et al. [25] proposed the CloudAware framework for offloading tasks into mobile edge computing. They posited that the CloudAware framework would support ad-hoc and real-time communication and integrates computation offloading with context adaptation using the knowledge of developers as well as the component scheme to make the offloading decision. However, the authors presented their architecture without implementation or testing.

Meurisch et al. [26] investigated the problem of how to make an offloading decision for unknown systems. To address this issue, they sent a micro-task to the computing system to evaluate the computing system and the existing network. Their calculation was based on different parameters such as completion time, data size, network bandwidth, and energy consumption. Then, to offload larger tasks, they used a polynomial regression model to predict the cost and time. The experiments revealed accuracy of up to 85.5% when sending two micro tasks to predict the cost and performance.

Habak et al. [27] created the Femtocloud system, which leveraged from nearby mobile devices' capabilities to produce a collaborative computation cluster in stable environments like classrooms and coffee shops where people commonly gather. This framework demonstrated the advantage of being more scalable than the cloudlet. In addition, it was based on cooperative, unfixed services at the edge layer of the network. The authors' evaluation of their proposed system was based on both experiments and simulations and proved that the system was efficient in terms of energy and computation and also reduced latency.

Liu et al. [28] investigated vehicles as mobile edge servers that could serve mobile devices as well as fixed edge servers and proposed a vehicle edge computing operating architecture. They formulated an optimization problem to increase the utilization of the system, which they converted into a semi-Markov problem in order to manage the dynamic nature of the vehicles, communication time, and offloading requests. Furthermore, they used Q-Learning and Deep Reinforcement Learning (DRL) to identify the most appropriate policies of resource allocation and computation offloading procedures. The evaluation results indicated that their system may achieve greater performance than fixed and vehicle systems.

Huang et al. [29] investigated wireless powered MEC networks. They created a deep learning based online offloading algorithm — DROO, which split the optimization problem into two sub-problems: resource allocation and offloading decision. Their evaluation results demonstrated the successful performance of the proposed algorithm.

A summary of these main offloading frameworks is presented in **Table 1**.

Table 1. Summary of main offloading frameworks

Ref	**Arch.**	**Offloading Strategy**	**Partitioning Type**	**Resource alloc. decision**	**Parameters**	**Pros**	**Cons**	**Testing**
[21]	MCC	Binary	-	-	Energy + time	Offloading decision depends on network+ queues conditions	Centralized	Simulation/experiments
[22]	MCC	Binary	-	-	Energy	Reduces energy	Theoretical only	-
[13]	MCC	Partial	Dynamic	Online	Energy	Lightweight offloading + Low latency and energy + Programmer assistance	Not generic (depends on android application model) + high profiler overhead + needs application developer support	Experiments
[14]	MCC	Partial	Hybrid	-	Energy	Low energy	Centralized	Experiments
[23]	MCC	Partial	Dynamic	Online	Time + Energy	Low energy + scalability + high efficiency	Synchronization overheads	Experiments
[24]	MCC	Partial	Dynamic	-	Time + Energy	Low execution time + energy + support programmers	Not scalable (sensitive to high number tasks)	Simulation (cloudSim)
[25]	MEC	Partial	Static / dynamic	Online	Time + Energy	Low time + improve offloading decision	Static architecture (based on the well-defined hierarchy between fog and cloud)	
[26]	Cloud / cloudlet	Partial	-	Online	Time+ Energy+ Bandwidth	85% of accuracy prediction + flexibility + low cost overhead	More tasks lead to more cost consumption	Experiments
[27]	MEC	Partial	-	Online	Time+ Energy+ Computing utilization	Low latency and energy + improved computational efficiency	Not flexible with changing the connection time + security issues	Simulation/ experiments

[30]	MEC	Binary	-	Online	Time+ Energy	Task delay reduced by 20% + the energy is saved by 30%	Not consider the payment cost or bandwidth	Simulation
[28]	MEC	Binary	-	Online	Resource utilization + resource allocation	Low delay of allocating resource + higher utility	Higher payment cost than local computation	Simulation
[29]	MEC	Binary	-	Online	Dynamic offloading decision + resource allocation based on time variation of wireless channels	Low time + resource allocation	Not flexible to implement on dynamic mobile devices	Simulation

3. Optimization Models for Offloading

The following section presents a general overview of existing optimization techniques, classifications, and applications. This is followed by an investigation of some relevant research focused on offloading optimization and a comparison of the proposed frameworks.

3.1 Overview of Optimization Techniques

Optimization can be defined as the performance of an action, which can be a formula or a model, to make something, like a system or a decision, as perfect as possible. It can also be referred to as "a mathematical discipline that concerns the finding of the extreme (minima and maxima) of numbers, functions, or systems" [31]. That is, it involves the maximization and minimization of functions. Optimization techniques are used to solve an optimization problem which can be single- or multi-objective. Single-objective optimization is used to solve a single objective function to give one optimal solution; multi-objective optimization is used to solve many objectives at the same time and gives many solutions rather than a single optimal solution [32, 33].

Optimization techniques can be classified as linear and nonlinear programming optimization. Linear programming optimization is a linear objective function that seeks to maximize or minimize objectives to linear constraints. On the other hand, nonlinear programming aims to maximize or minimize nonlinear objective functions [33]. Furthermore, nonlinear programming can be classified into deterministic and stochastic, as presented in **Fig. 5** [32, 33].

Deterministic optimization methods follow strict mathematical models, which are based on linear programming that seeks to guarantee the most accurate and frequent solution, i.e., with the same input, it will achieve the same output every time. However, in this case, the optimal solution is mostly local and cannot be global. Thus, deterministic optimization methods better fit single-objective problems than multi-objective problems [33].

On the other hand, stochastic optimization methods do not produce a guaranteed solution, because they are based on searches with random variables. Indeed, for every iteration, different outputs may be produced for the same input. Nevertheless, although they don't provide an exact solution, stochastic optimization methods have the advantage of being adaptive, i.e., they can be used for any optimization problem either in local or global search. Moreover, they produce an approximated solution in an acceptable time [31].

Stochastic optimization can be classified into two main categories: heuristic and meta-heuristic algorithms. The evolution of the heuristic category has produced meta-heuristic algorithms that combine random processing and global optimization [34]. The meta-heuristic category can be further classified into Evolutionary Algorithms (EA) and Swarm Intelligence (SI) algorithms [31]. EA are inspired by biological evolution to find the nearest optimal solution. They are used to solve local optimization problems using random searches [33]. Many EA have been proposed by researchers, including evolution strategies; genetic programming, which uses computers to produce solutions as programs; and genetic algorithms, which are inspired by biological genetics and their solutions represented as series of strings [35].

In particular, SI is an artificial intelligence scheme based on self-organized and distributed algorithms. It was inspired by the behavior of social animals like birds, fish, and insects, such as Particle Swarm Optimization (PSO), Cuckoo Search Algorithm (CSA), Monkey Algorithm (MA), etc. [36].

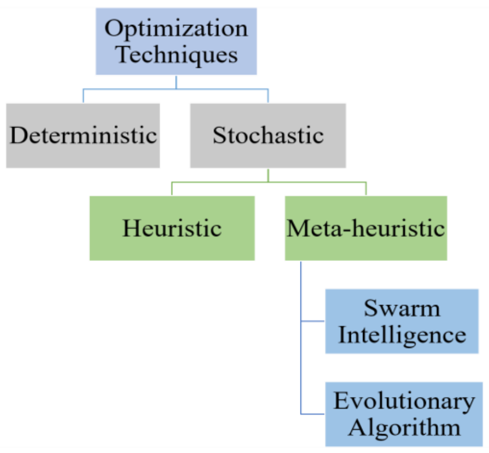


Fig. 5. Optimization techniques

3.2 Proposed Optimization Models for Offloading Decision

The following section presents several models that researchers have proposed for optimizing the offloading decision. Here, they are classified according to the optimization technique used.

3.2.1 Deterministic Optimization Models

Pinherio et al. [2] discussed the cost of accurate offloading decisions for cloud service providers. They proposed using a Stochastic Petri Net (SPN) framework, which an extension of Petri Net (PN). The role of an SPN is to predict the performance of an application, data traffic through the offloading process, and finally the offloading cost. This prediction is made at method-call-level, which generates highly accurate estimations. Moreover, an SPN considers the network bandwidth used to send and receive methods. It therefore helps developers at the design phase to develop their applications with accurate information about application performance and cost prediction.

Wang et al. [7] focused on two main problems: minimizing energy consumption and latency. For latency, they proposed offloading tasks from smartphones to Femtoclouds at the edge layer. In addition, they argued that tasks could be executed in parallel to reduce time.

Moreover, they proposed using Latency-optimal Partial Computation Offloading (LPCO) to reduce latency in many cloud server cases. For energy consumption, they used Dynamic Voltage Scaling (DVS) technology that adapted smartphone computation speed based on the computation load of the device. Moreover, they proposed using an Energy-optimal Partial Computation Offloading (EPCO) algorithm to minimize energy consumption.

Liu et al. [37] investigated the fog layer in MCC. They discussed an offloading multi-objective optimization problem that emphasized three parameters: energy consumption, execution delay, and payment cost. Queuing theory was utilized to solve the weighted optimization problem. The problem was formulated to minimize the three parameters mentioned above. A scalarization method was used to convert the optimization problem from multi- to single-objective. Also, the offloading probability and transmission power were reconfigured in order to minimize energy consumption, execution delay, and payment cost. Moreover, they used an Interior Point Method (IPM) algorithm in iteration processes to increase accuracy. The simulation results showed that the performance of the proposed solution was extremely high. However, some results showed that at a certain point, when the number of offloading requests increased, the energy consumption and delay increased.

Zhao et al. [38] examined the computational offloading of mobile devices. They proposed using an offloading algorithm for energy consumption oriented to reduce energy besides constraints like transmission power and time. In their proposed algorithm, the mobile device calculated the energy consumption of offloading to both cloud and fog. Then, it compared these to identify the process with the lowest energy consumption. Their algorithm was based on an architecture of three layers (mobile device, fog, and cloud). The simulation results showed the proposed algorithm achieved higher performance and lower energy consumption for one user, and the results for multiple users were left for future research.

Chen and Hao [30] investigated the optimization of task offloading in an ultra-dense network. First, they proposed a system model of a Software Defined Ultra-Dense Network (UT-UDN). They aimed to minimize both the task delay and energy consumption by formulating a mixed-integer nonlinear programming optimization problem. They proposed using a scheme called a Software Defined Task Offloading (SDTO) to break down the optimization problem into two sub-problems. The first problem was a resource allocation problem which was solved using Karush–Kuhn–Tucker (KKT) conditions. The second one was a task placement problem, which was solved by a task placement algorithm. The simulation results showed that task delay was reduced by 20% and 30% more energy was conserved.

3.2.2 Heuristic Optimization Models

Du et al. [39] focused on the optimization of resource allocation at edge servers in order to minimize task service costs and maximize the number of clients served per edge. To solve this multi-optimization problem, they modified it into a deterministic optimization problem. Then, they split it into sub-problems using Lyapunov optimization. They proposed an Online Joint Task Offloading and Resource Allocation Algorithm (OJTORA) to solve these sub-problems. The experimental results proved that OJTORA obtained greater performance than other baseline strategies. However, they didn't consider the bandwidth condition and mobility of client services.

Thai et al. [40] proposed an approach for using a cooperative mobile edge computing network to reduce energy and delay consumption. First, they formulated mixed resource allocation and task offloading in MEC. Then, they proposed a relaxed solution with an Improved Branch and Bound Algorithm (IBBA) to solve the mixed-integer nonlinear programming problem. They developed two solutions — an Interior Point Method (IPM) and an Improved Branch and Bound Algorithm (IBBA) — to identify the optimal solution for edge nodes and mobile users. The experiments showed the efficiency of both solutions regarding time and energy consumption.

Xu et al. [41] proposed a system architecture and formulated an optimization problem to minimize both energy and time consumption. The proposed system model involved many mobile devices, one edge, and one cloud server. They assumed that the bandwidth between the three layers was large enough to reject the bottleneck condition. The optimization problem was a nonlinear mixed-integer programming problem. To solve it, they proposed two algorithms — an Enumeration Algorithm and a Branch and Bound algorithm. The simulation showed that the Branch and Bound Algorithm produced better results than the Enumeration Algorithm.

3.2.3 Meta-Heuristic Optimization Models

Yang et al. [42] focused on minimizing the energy consumption and queue congestion of task offloading to MEC. First, they proposed a mobile device classification algorithm to solve the offloading decision problem. To solve the queuing congestion problem, they proposed using the Promoted by Probability (PBP) mechanism, which organizes the priority of tasks in order to reduce energy consumption. Then, they formulated a mixed-integer optimization problem to minimize energy consumption and packet delay. They applied the krill herd meta-heuristic optimization algorithm to solve the NP-hard optimization algorithm by optimizing the task offloading decision while minimizing queuing congestion. The simulation results demonstrated the high performance of their solution.

Dai et al. [43] investigated the resource allocation problem in wireless communication technology at a MEC. At their system, the Access Node (AN) and edge-computing server schedules carriers and corresponding computation resources. Then, results were sent back to the device, which then decided either to offload or not based on these results. However, the system model used an Orthogonal Frequency Division Multiplexing (OFDM) system that split the existing channel into many sub-carriers to reduce offloaded data stream. The problem was presented in a mathematical model and then broken into two sub-problems. The first aimed to maximize the difference between a task’s completion time locally and remotely at the MEC, and the second sought to compute the maximum uploading rate of the task. To address these two problems, the authors proposed a Hybrid Quantum Behavior Particle Swarm Optimization (HQPSO) that used the water-filling algorithm in sub-problem two to reduce the dimension of the QPSO equation, which enforce the accuracy and speed of the solution. Simulations showed that the accuracy and performance of the model were high. However, the model was still outperformed by traditional binary searches by 5% regarding saved completion time and 10% regarding accuracy.

Rashidi and Sharifain [44] proposed a model based on Ant Colony Optimization (ACO) and Queue Decision Maker (QDM) — known as ACOQDM — for task assignment optimization. They aimed to reduce completion time, communication time, power consumption, and task drop rate, and improve load balancing through two layers of cloud computing (cloudlets and cloud). When the tasks were offloaded to the cloudlet, they were put in the proxy’s buffer to be sent to the dispatcher unit, which used a Decision Maker (DM) to decide whether to send tasks to cloudlet or cloud serves. The DM used the information generated by the repository and QDM and ACO algorithms to build its decision. First, the QDM goal was to minimize the response time by computing the probability of assigning a task into either a cloudlet or cloud. Then, the ACO used the task assignment probabilities and the communication time between the user and the specific cloudlet as an input to minimize the communication time of the whole system. The ACOQDM successfully reduced the response time, completion time, transition time, power consumption, and the drop rate of tasks.

Xu et al. [45] examined the task offloading of workflow applications in fog-cloud environments in order to reduce the cost and the time of all tasks. They proposed an algorithm for workflow scheduling. The scheduling method was based on an Improved Particle Swarm Optimization (IPSO) algorithm and was a PSO algorithm integrated with inertia weight. Inertia weight was used to enforce the searching ability of particles from the original PSO. It designs the problem as a nonlinear decreasing function of both time and cost in global and local practical searchability. The experiments showed that the reduction of cost and time was greater than with the original PSO.

Ramezani et al. [46] proposed a task scheduling model using multi-objective optimization. Their solution was based on minimizing execution time, transmission time, and execution cost using Multi-Objective PSO (MOPSO), which is suitable for distributed systems. The system fulfilled the required service requirements (mainly QoS); however, the study did not investigate energy consumption or task prioritization.

Alexander and Joseph [47] examined computation offloading into cloud data centers. They aimed to minimize cost and time and maximize resource utilization. To address this optimization problem, they proposed a load-aware resource allocation based on the Cuckoo Search Algorithm (CSA). The simulation results proved that the Cuckoo Algorithm reduced time and cost and improved resource utilization compared with PSO.

Kaur and Mehta [48] applied Grey Wolf Optimization (GWO) to optimize the offloading plan in order to increase performance and decrease time, cost, and energy. They focused on the centralized architecture of cloud datacenters.

Goudarzi et al. [49] proposed using Fast Hybrid Multi-site Computation Offloading (FHMCO) to identify the best application partitioning based on the size of the application in a short time. First, the weighted cost model was used to reduce the energy and time consumption of the process. Moreover, the authors used two decision algorithms: Optimized Multi-site Offloading Problem (OMB&B) and Optimized Multi-site Particle Swarm Optimization (OMPSO). OMB&B is usually used to identify the optimal solution for small scale applications in a short time, mobile applications, whereas OMPSO is used to search in large spaces and produce near-optimal solutions in a reasonable time. The simulation and experiments proved that the FHMCO achieved better performance than alternative frameworks. However, it was based on a centralized architecture.

Guo et al. [50] investigated the problem of offloading decision making as well as resource and channel allocation. To minimize energy consumption, they used a Genetic Algorithm based on a Computation Algorithm (GACA) to solve the mixed-integer non-linear programming problem. Their simulation indicated that their solution was slow.

Huynh et al. [51] focused on minimizing time and energy consumption using resource allocation and offloading decision making. Their model environment involved the use of multi-user and multi-edge servers in heterogeneous networks. They formulated a mixed-integer non-linear programming problem of resource allocation and offloading decision making. To solve it, they divided it into two subproblems and proposed using a PSO-based algorithm (JROPSO) to optimize resource allocation and computation offloading decisions jointly. Their simulation results demonstrated the efficiency of the proposed algorithm.

Li et al. [52] examined green computation in ultra-dense networks using computation offloading. They aimed to minimize energy and time consumption by using edge-based architecture. Their proposed system involved multi-mobile devices, multi-small base stations

(SBS), and one macro-base station (MBS). Each mobile device owned just one task. The authors proposed using a computation offloading mechanism based on CSA to solve the non-linear programming problem. The simulation results showed that the proposed CSA reduced both time and energy consumption.

Min-Allah et al. [53] investigated implementing task scheduling and resource allocation in MCC in order to minimize cost and time consumption in the offloading system. The authors formulated an optimization problem to reduce both time and execution cost consumption. Furthermore, they proposed a Hybrid Genetic and Cuckoo Search algorithm (HGCS) to solve the optimization problem. They aimed to find an optimal schedule for a group of real-time tasks in an optimal VM in the cloud. The simulation results proved the efficiency of the HGCS compared with GA and CSA alone.

Finally, Arun and Prabu [54] considered job-sharing and load-balance in VMs of MCC. They formulated an optimization problem, which is NP-hard, to minimize both time and cost. To solve the optimization problem, they proposed using an accelerated CSA to identify a task with minimum time and cost. The simulation results proved that the proposed solution was stronger than other algorithms in terms of minimizing execution time, job-sharing value, used bandwidth, transmission speed, and buffering overhead.

Table 2. Summary of offloading optimization techniques

Ref.	**Year**	**Arch.**	**Optimization Tech.**	**Optimization Algo.**	**Parameters**	**Method**	**Limitation**
[2]	2018	MCC	Deterministic	SPN	Time (communication + completion)	Estimate the application performance	It is not a context-aware offloading application + not use energy as a metric
[7]	2015	MEC	Deterministic	EPCO	Time + energy	DVS technology to optimize the computation offloading	It focuses on improving the mobile device without considering computational resources
[43]	2017	MEC	AI	HQPSO	Time + number of iterations	Resource allocation	5% less than binary search inaccuracy assumed a single task for advice + single thread for CPU
[37]	2017	Fog	Deterministic	Weight method	Time (execution) + energy + cost	Queue theory	It does not consider the network dynamic conditions as data traffic and no resource allocation considered
[36]	2017	Cloudlet/ Cloud	AI	ACO-GA	Time + energy + improving queue drop rate + load balance	Queue theory	Centralized architecture
[44]	2019	Cloud/ Fog	AI	Workflow scheduling based on IPSO	Time + cost	Task scheduling	Centralized architecture
[46]	2013	Cloud	AI	MOPSO	Time + cost	Task scheduling	Centralized architecture
[47]	2016	Cloud	AI	CSA	Time + cost + resource utilization	Load aware resource allocation	They only work in one data centre
[48]	2019	MCC	AI	GWO	Time + energy + cost	Minimize execution cost	The cost and time still higher than the exhaustive approach
[49]	2017	MCC	AI	PSO	Time + energy	Offloading partitioning based on the size of the mobile application	Centralized architecture
[38]	2017	Cloud/ Fog	Deterministic	Optimal energy consumption algorithm	Time + energy	Resource allocation	The model is a single user only

[39]	2019	MEC	Heuristic	Lyapunov	Time + energy + scalability	Resource allocation	Based on static bandwidth allocation
[40]	2018	MEC	Heuristic	IBBA	Time + energy	Resource allocation	Not considered the scalability
[41]	2019	MEC/Cloud	Heuristic	Enumerating and Branch + Bound Algorithm	Time + energy	Task offloading decision	Low speed of operation
[42]	2018	MEC	Meta-heuristic	krill herd algorithm	Time + energy +minimize queue congestion	Resource allocation + binary offloading	Not consider network resources + heterogeneous of servers in the system
[30]	2018	MEC	Deterministic	KKT + task placement algorithm	Time + energy	Resource allocation	Not consider the mobility of the users
[50]	2018	MEC	AI	GACA	Energy	Resource allocation	The solution is slow
[51]	2019	MEC	AI	JROPSO	Time + energy	Resource allocation + offloading decision making	Dismiss the waiting and response time
[52]	2019	MEC	AI	CSA	Time + energy	Green computation + computation offloading	Did not focus on the factors of SDN ultra-dense network
[53]	2019	MCC	AI	HGCS	Time + cost	Task scheduling + resource allocation	Centralized architecture
[54]	2019	MCC	AI	CSA	Time and cost	Job-sharing + load-balance	Centralized architecture

3.3 Comparing Existing Solutions

The aforementioned solutions are summarized in **Table 2**, which shows that some proposed optimization techniques for offloading optimization were deterministic and others were AI-based. AI-based solutions are usually based on the use of an SI algorithm to optimize the offloading process. Indeed, SI supports heterogeneous, global, and distributed environments as MEC. Moreover, SI produced automatically, adaptability, and self-organization techniques. For this, we investigate these solutions in greater detail in order to select the most appropriate one to be applied in the research. To do so, the relevant criteria were defined as follows:

1) Algorithm: The SI algorithm is used to optimize offloading.
2) Distribution: Whether the architecture is centralized in a cloud or distributed at the edge layer.
3) Optimization parameters: The parameters considered in the optimization.
   (1) Time: The total execution and transmission time for each task.
   (2) Energy: The energy consumption for each task (execution and transmission).
   (3) Cost: The payment cost of the application execution.
   (4) Scalability: The ability of the application to scale up without harm.
   (5) Resource utilization: The utilizing of limited resources without causing an overhead of the system.
   (6) Load balance: The ability to share tasks with other edge nodes that are underloaded.
   (7) Queue congestion: The avoiding of queue congestion when the arrival rate of the tasks succeeds the service rate, which causes system overhead.
   (8) No. of iterations: This is related to the completion time of the task.

Based on the criteria, a comparison between SI-based models is presented in **Table 3**.

Table 3. Offloading techniques applied SI

Requirements/ Ref	Algorithm	Distribution	Payment cost	Time	Energy	Scalability	Resource utilization	Load Balance	Queue congestion	No of iterations
[42]	krill herd algorithm	√	X	√	√	X	√	X	√	X
[43]	HQPSO	√	X	√	√	X	X	X	X	√
[44]	ACO-GA	√	X	√	√	X	X	√	√	X
[45]	IPSO	√	√	√	X	X	X	X	X	X
[46]	MOPSO	√	√	√	X	X	X	X	X	X
[48]	CSA	X	√	√	X	X	√	X	X	X
[48]	GWO	X	X	√	√	X	X	X	X	X
[49]	PSO	X	X	√	√	X	X	X	X	X
[50]	GACA	√	X	X	√	X	X	X	X	X
[51]	JROPSO	√	X	√	√	X	X	X	X	X
[52]	CSA	√	X	√	√	X	X	X	X	X
[53]	HGCS	X	√	√	X	X	X	X	X	X
[54]	CSA	X	√	√	X	X	X	X	X	X

Table 3 shows that most researchers have investigated the MEC environment due to its decentralized infrastructure, which facilitates the avoidance of bottleneck issues. Researchers have also mainly focused on time rather than energy and payment cost. The mathematical equations of these parameters can be presented as follows:

1) Time: Includes processing time, transmission time, waiting time, and response time, which is calculated as

$$T_p = \frac{S}{f} \tag{1}$$

where T_p is the task's processing time, S is the task size, and f is the processing computation time;

$$T_t = \frac{S_{in}}{B} \tag{2}$$

where T_t is the transmission delay, S_{in} is the size of the input data, and B is the bandwidth between a mobile device and the edge node;

$$T_r = \frac{S_{out}}{B} \tag{3}$$

where T_r is the response delay and S_{out} is the size of output data; and

$$T_{wait} = \frac{L^e}{\lambda^e} \tag{4}$$

where T_{wait} is the waiting time, L^e is the average number of waiting tasks, and λ^e is arrival rate of tasks.

2) Energy: Includes transmission energy, idle energy, and response energy, which is calculated as:

$$E_t = T_t \times p_t \tag{5}$$

where E_t is the transmission energy and p_t is the transmission power of a mobile device in watts;

$$E_i = T_p \times p_i \tag{6}$$

where E_i is the idle energy of the edge node and p_i is the idle power of the edge node in watts; and

$$E_r = T_r \times p_r \tag{7}$$

where E_r is the response energy and p_r is the response power of the edge node in watts.

3) Cost:

$$C = T_p \times ResCost \tag{8}$$

where C is the payment cost of the offloading and ResCost is the payment cost of the processor in a second of time.

On the other hand, other parameters such as load balancing, number of iterations, queue congestion, or resource utilization have rarely been investigated, and scalability has never been considered in AI solution research.

4. Discussion, Open Issues, and Future Directions

This section highlights the challenges of computation offloading in cloud/edge-based architectures, discusses the open research issues, and explores future research directions.

It has been shown in this research that in recent years, optimization techniques have been proposed to optimize offloading decisions, specifically Swarm Intelligence (SI) models since they provide a better fit for the distributed and dynamic aspect of edge computing. Indeed, SI produces a better solution in the shortest time, which satisfies the requirements of real-time applications. The analyses conducted in this paper showed that many researchers have investigated offloading, specifically in MCC. These models should be adapted from the current centralized architecture into a more distributed architecture using edge layers. Moreover, most studies have focused on cost, latency, or energy reduction but have not investigated them all in the same research. Additionally, different models have generally focused on different objectives. However, it is crucial to provide a solution that optimizes as many objectives as possible. Based on the in-depth study of the main proposed models, the current researches suggest focusing on many parameters, particularly energy, payment cost, time, dynamic network congestion, etc.

It is important to note that there are still several open issues in the offloading process that need to be investigated by the research community. These issues include resource allocation for the offloaded tasks even in edge servers or virtual machines. Indeed, it is a challenge to allocate resources at a lower cost in terms of time and energy. Furthermore, mobile devices may lose their connection because of their mobility while sending /receiving data. Therefore, offloading models should produce fault-tolerant mechanisms to resend the lost data, which also allows minimizing the response time and energy consumption. As a result, local and global convergence between mobile devices and edge nodes and load balancing should be investigated. Moreover, offloading models need to be further automated to discover network areas, new nodes, lost components, etc. This will make the offloading process more efficient.

One of the most leading research directions is to combine the SI-based offloading theory with edge computing to enhance current centralized solutions and adapt them to the distributed schema while considering multiple objectives. As discussed earlier, cuckoo search appears to be a suitable algorithm for solving multi-objective optimization in offloading since it achieves good performance in this area compared with other AI algorithms as shown in [47] and [52]. As a result, applying Cuckoo to an edge-based architecture would enhance the offloading process and allow for more robust solutions to support mobile devices in executing highly intensive applications. Moreover, the computational offloading to the edge can be improved by combining CSA with parallel computing between edges, which maximize the computation capacity and minimize time as well.

It is also important to consider the mobility aspect of nodes. Thus, considering historical movement data may enhance the offloading decisions. Such information can be stored at edge nodes and used to predict mobile locations thus enhancing the offloading decisions. Prediction models may be used along with the optimization theory to enhance the offloading process.

Finally, security and privacy are still challenging since the offloaded tasks will go through the network. AI-based models can be used also to predict certain attacks and change the offloading decision accordingly.

5. Conclusion

The number of mobile device users is increasing. Indeed, mobile device usage has been growing exponentially in recent years. Mobile devices should be able to support real-time applications such as gaming, e-commerce, healthcare, etc. Furthermore, mobile devices users expect as high a Quality of Service (QoS) as desktop-level applications. However, real-time applications require additional resources including storage capacity, computation power, and battery. In order to overcome the limitations of these resources in mobile devices, offloading is used to alleviate mobile tasks by sending all/some of them into rich resources such as cloud or edge servers to be processed there and then returned to the mobile devices. However, computational offloading also consumes time and energy, which are critical to the success of real-time applications.

This paper investigated different computational offloading models and compared them in order to identify the offloading parameters that need to be optimized. Based on these analyses, a taxonomy of optimization offloading strategies was proposed. Moreover, this study conducted a comparison of several optimization techniques used to optimize the offloading decision, as well as a comparison of several AI algorithms used in the computational offloading process in terms of payment cost, time, energy, etc.

References

[1] Mobile Action Team, "2018 App Industry Report & Trends to Watch for 2019," *Mobile Action Blog*, Dec. 2018. Article (CrossRef Link)

[2] T. F. da Silva Pinheiro, F. A. Silva, I. Fé, S. Kosta, and P. Maciel, "Performance prediction for supporting mobile applications' offloading," *Journal Supercomputing*, vol. 74, no. 8, pp. 4060-4103, Aug. 2018. Article (CrossRef Link)
[3] S. E. Mahmoodi, K. Subbalakshmi, and R. N. Uma, Spectrum-Aware Mobile Computing: Convergence of Cloud Computing and Cognitive Networking, Springer International Publishing, 2019. Article (CrossRef Link)
[4] F. Gu, J. Niu, Z. Qi, and M. Atiquzzaman, "Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions," *Journal of Network and Computer Applications*, vol. 119, pp. 83-96, Oct. 2018. Article (CrossRef Link)
[5] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computing Systems*, vol. 70, pp. 138-147, May 2017. Article (CrossRef Link)
[6] A. Mohammad, S. Zeadally and K. A. Harris, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computing Systems*, vol. 87, pp. 278-289, 2018. Article (CrossRef Link)
[7] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-Edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268-4282, Oct. 2016. Article (CrossRef Link)
[8] E. Ahmed, A. Gani, M. Sookhak, S. H. Ab Hamid, and F. Xia, "Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges," *Journal of Network Computer Applications*, vol. 52, pp. 52-68. Article (CrossRef Link)
[9] T. Zhang, "Data Offloading in Mobile Edge Computing: A Coalition and Pricing Based Approach," *IEEE Access*, vol. 6, pp. 2760-2767, 2018. Article (CrossRef Link)
[10] Z. Xu, X. Liu, G. Jiang, and B. Tang, "A time-efficient data offloading method with privacy preservation for intelligent sensors in edge computing," *EURASIP Journal Wireless Commununications Networking*, vol. 2019, no. 1, p. 236, Oct. 2019. Article (CrossRef Link)
[11] D. Liu, L. Khoukhi, and A. Hafid, "Prediction-Based Mobile Data Offloading in Mobile Cloud Computing," *IEEE Transactions Wireless Communications*, vol. 17, no. 7, pp. 4660-4673, July 2018. Article (CrossRef Link)
[12] M. A. Khan, "A survey of computation offloading strategies for performance improvement of applications running on mobile devices," *Journal of Network Computer Applications*, vol. 56, pp. 28-40, 2015. Article (CrossRef Link)
[13] D. Kovachev, T. Yu, and R. Klamma, "Adaptive Computation Offloading from Mobile Devices into the Cloud," in *Proc. of 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 784-791, July 2012. Article (CrossRef Link)
[14] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in *Proc. of the 6th Conference on Computer*, pp. 301-314, 2011. Article (CrossRef Link)
[15] P. D. Nguyen, V. N. Ha, and L. B. Le, "Computation Offloading and Resource Allocation for Backhaul Limited Cooperative MEC Systems," in *Proc. of the 90th Vehicular Technology Conference(VTC2019-Fall)*, pp. 1-6, Sep. 2019. Article (CrossRef Link)
[16] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computing Systems*, vol. 87, pp. 278-289, Oct. 2018. Article (CrossRef Link)
[17] F. Wang, B. Diao, T. Sun, and Y. Xu, "Data Security and Privacy Challenges of Computing Offloading in FINs," *IEEE Network*, vol. 34, no. 2, pp. 14-20, Mar. 2020. Article (CrossRef Link)
[18] S. R. Behera, N. Panigrahi, S. Bhoi, A. Sahani, J. Mohanty, D. Sahoo, A. Maharana, L. P. Kanta, and P. Mishra, "A Novel Decision Making Strategy for Computation Offloading in Mobile Edge Computing," in *Proc. of 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, Mar. 2020, pp. 1-5. Article (CrossRef Link)
[19] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Applied Computing Informatics*, vol. 14, no. 1, pp. 1-16, Jan. 2018. Article (CrossRef Link)

[20] K. Peng, B. Zhao, S. Xue, and Q. Huang, "Energy- and Resource-Aware Computation Offloading for Complex Tasks in Edge Environment," *Complexity*, Mar. 26, 2020. Article (CrossRef Link)

[21] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "eTime: Energy-efficient transmission between cloud and mobile devices," in *Proc. of IEEE Annual Joint Conference: INFOCOM*, pp. 195-199, 2013. Article (CrossRef Link)

[22] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel," *IEEE Transitions on Wireless Communications*, vol. 12, no. 9, pp. 4569-4581, Sep. 2013. Article (CrossRef Link)

[23] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of Annual Joint Conference: IEEE INFOCOM*, pp. 945-953, 2012. Article (CrossRef Link)

[24] A. Khanna, A. Kero, and D. Kumar, "Mobile cloud computing architecture for computation offloading," in *Proc. of the 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Oct. 2016, pp. 639-643, 2016. Article (CrossRef Link)

[25] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading," in *Proc. of the 8th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 112-119, 2015. Article (CrossRef Link)

[26] C. Meurisch, J. Gedeon, T. A. B. Nguyen, F. Kaup, and M. Muhlhauser, "Decision Support for Computational Offloading by Probing Unknown Services," in *Proc. of the 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1-9, 2017. Article (CrossRef Link)

[27] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge," in *Proc. of IEEE 8th International Conference on Cloud Computing*, pp. 9-16, 2015. Article (CrossRef Link)

[28] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep Reinforcement Learning for Offloading and Resource Allocation in Vehicle Edge Computing and Networks," *IEEE Transitions on Vehicular Technology*, vol. 68, no. 11, pp. 11158-11168, Nov. 2019. Article (CrossRef Link)

[29] L. Huang, S. Bi, and Y. J. A. Zhang, "Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks," *IEEE Transitions on Mobile Computing*, pp. 2581-2593, 2020. Article (CrossRef Link)

[30] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal of Selected Areas in Communications*, vol. 36, no. 3, pp. 587-597, 2018. Article (CrossRef Link)

[31] S. Kiranyaz, T. Ince, and M. Gabbouj, "Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition," Springer-Verlag Berlin Heidelberg, vol. 15, 2014. Article (CrossRef Link)

[32] J. Branke, K. Deb, K. Miettinen, and R. Słowiński, "Multiobjective optimization: Interactive and evolutionary approaches", Springer-Verlag Berlin Heidelberg, vol. 5252, 2008. Article (CrossRef Link)

[33] M. Cavazzuti, "Deterministic optimization," in Optimization Methods, Springer-Verlag Berlin Heidelberg, pp. 77-102, 2013. Article (CrossRef Link)

[34] P. Kunche and K. V. V. S. Reddy, "Heuristic and Meta-Heuristic Optimization," in Metaheuristic Applications to Speech Enhancement, Springer International Publishing, pp. 17-24, 2016. Article (CrossRef Link)

[35] Sastry K., Goldberg D.E., Kendall G, "Genetic Algorithms," in Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Boston, MA, USA: Springer US, pp. 97-125, 2005. Article (CrossRef Link)

[36] O. Zedadra, A. Guerrieri, N. Jouandeau, G. Spezzano, H. Seridi, and G. Fortino, "Swarm intelligence-based algorithms within IoT-based systems: A review," *Journal of Parallel Distributed Computing*, vol. 122, pp. 173-187, Dec. 2018. Article (CrossRef Link)

[37] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective Optimization for Computation Offloading in Fog Computing," *IEEE Internet Things of Journal*, vol. 5, no. 1, pp. 283-294, Feb. 2018. Article (CrossRef Link)

[38] X. Zhao, L. Zhao, and K. Liang, “An Energy Consumption Oriented Offloading Algorithm for Fog Computing,” in *Proc. of International Conference on Heterogeneous Networks for Quality, Reliability, Security and Robustness*, pp. 293-30, 2017. Article (CrossRef Link)

[39] W. Du, T. Lei, Q. He, W. Liu, Q. Lei, H. Zhao, and W. Wang, “Service Capacity Enhanced Task Offloading and Resource Allocation in Multi-Server Edge Computing Environment,” *ArXiv Prepr. ArXiv190304709*, 2019. Article (CrossRef Link)

[40] T. T. Vu, N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, “Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks,” in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, 2018. Article (CrossRef Link)

[41] J. Xu, Z. Hao, and X. Sun, “Optimal Offloading Decision Strategies and Their Influence Analysis of Mobile Edge Computing,” *Sensors*, vol. 19, no. 14, Jan. 2019. Article (CrossRef Link)

[42] Y. Yang, Y. Ma, W. Xiang, X. Gu, and H. Zhao, “Joint optimization of energy consumption and packet scheduling for mobile edge computing in cyber-physical networks,” *IEEE Access*, vol. 6, pp. 15576-15586, 2018. Aricle (CrossRef Link)

[43] S. Dai, M. Liwang, Y. Liu, Z. Gao, L. Huang, and X. Du, “Hybrid Quantum-Behaved Particle Swarm Optimization for Mobile-Edge Computation Offloading in Internet of Things,” in *Proc. of International Conference on Mobile Ad-hoc and Sensor Networks*, pp. 350-364, 2018. Article (CrossRef Link)

[44] S. Rashidi and S. Sharifian, “A hybrid heuristic queue based algorithm for task assignment in mobile cloud,” *Future Generation Computing Systems*, vol. 68, pp. 331-345, Mar. 2017. Article (CrossRef Link)

[45] R. Xu, Y. Wang, Y. Chen, Y. Zhy, Y. Xie, A. S. Sani, and D. Yuan, “Improved Particle Swarm Optimization Based Workflow Scheduling in Cloud-Fog Environment,” in *Proc. of International Conference on Business Process Management Workshops*, vol. 342, pp. 337-347, 2019. Article (CrossRef Link)

[46] F. Ramezani, J. Lu, and F. Hussain, “Task Scheduling Optimization in Cloud Computing Applying Multi-Objective Particle Swarm Optimization,” in *Proc. of Service-Oriented Computing*, vol. 6470, pp. 237-251, 2013. Article (CrossRef Link)

[47] A. A. Alexander and D. L. Joseph, “An Efficient Resource Management for Prioritized Users in Cloud Environment Using Cuckoo Search Algorithm,” *Procedia Technol.*, vol. 25, pp. 341-348, 2016. Article (CrossRef Link)

[48] P. Kaur and S. Mehta, “Efficient computation offloading using grey wolf optimization algorithm,” in *Proc. of AIP Conference Proceedings*, 2019. Article (CrossRef Link)

[49] M. Goudarzi, M. Zamani, and A. T. Haghighat, “A fast hybrid multi-site computation offloading for mobile cloud computing,” *Journal of Network and Computer Applications*, vol. 80, pp. 219-231, Feb. 2017. Article (CrossRef Link)

[50] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, “Energy Efficient Computation Offloading for Multi-Access MEC Enabled Small Cell Networks,” in *Proc. of IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1-6, 2019. Article (CrossRef Link)

[51] L. N. T. Huynh, Q. V. Pham, X-Q. Pham, T. D. T. Nguyen, M. D. Hossain, and E. N. Huh, “Efficient Computation Offloading in Multi-Tier Multi-Access Edge Computing Systems: A Particle Swarm Optimization Approach,” *Applied Science*, vol. 10, no. 1, Dec. 2019. Article (CrossRef Link)

[52] F. Li, H. Yao, J. Du, C. Jiang, and F. R. Yu, “Green Communication and Computation Offloading in Ultra-Dense Networks,” in *Proc. of 2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, Dec. 2019. Article (CrossRef Link)

[53] N. Min-Allah, M. B. Qureshi, S. Alrashed, and O. F. Rana, “Cost efficient resource allocation for real-time tasks in embedded systems,” *Sustainable Cities Society*, vol. 48, July 2019. Article (CrossRef Link)

[54] C. Arun and K. Prabu, “An efficient job sharing strategy for prioritized tasks in mobile cloud computing environment using ASC-JS Algorithm,” *Journal of Theoretical and Applied Information Technolgoy*, vol. 97, no. 4, pp. 1-15, 2005. Article (CrossRef Link)

Manal Alqarni received the bachelor's degree in information technology from King Abdulaziz University, Jeddah, Saudi Arabia. She is currently a Master student at King Abdulaziz University. She is teaching assistant in Taif University. Her research interests include networks, artificial intelligence and offloading.

Asma Cherif received her MS. and Ph.D. degrees in computer science from Lorraine University, France in 2008 and 2012 respectively. She conducted her research at INRIA-LORIA research center of Nancy, France. She is currently associate professor in the Faculty of Computing and Information Technology at King Abdulaziz University (Saudi Arabia). Her current research interests include distributed, collaborative, and real time systems, security, cloud/edge computing, and AI.

Entisar Alkayal received the Ph.D. degree from Southampton University, UK, in 2018. She is currently the supervisor of the Information Technology department in the faculty of Computing and information technology at Rabigh branch. Her research interests include key technologies in Internet of Things (IoT) and Cloud Computing.