**Offloading using Traditional Optimization and Machine Learning in Federated Cloud-Edge-Fog Systems: A Survey**

Binayak Kar, Member, IEEE, Widhi Yahya, Ying-Dar Lin, Fellow, IEEE, and Asad Ali, Student Member, IEEE

**Abstract**—The huge amount of data generated by the Internet of things (IoT) devices needs the computational power and storage capacity provided by cloud, edge, and fog computing paradigms. Each of these computing paradigms has its own pros and cons. Cloud computing provides enhanced data storage and computing power but causes high communication latency. Edge and fog computing provide similar services with lower latency but limited capacity, capability, and coverage. A single computing paradigm cannot fulfill all the requirements of IoT devices and a federation between them is needed to extend their capacity, capability, and services. This federation is beneficial to both subscribers and providers and also reveals research issues in traffic offloading between clouds, edges, and fogs. Optimization has traditionally been used to solve the problem of traffic offloading. However, in such a complex federated system, traditional optimization cannot keep up with the strict latency requirements of decision-making, ranging from milliseconds to sub-seconds. Machine learning approaches, especially reinforcement learning, are consequently becoming popular because they could quickly solve offloading problems in dynamic environments with some unknown information. This study provides a novel federal classification between cloud, edge, and fog and presents a comprehensive research roadmap on offloading for different federated scenarios. We survey the relevant literature on the various optimization approaches used to solve this offloading problem and compare their salient features. We then provide a comprehensive survey on offloading in federated systems with machine learning approaches and the lessons learned as a result of these surveys. Finally, we outline several directions for future research and challenges that have to be faced in order to achieve such a federation.

**Index Terms**—Offloading, cloud computing, edge computing, fog computing, federation, optimization, machine learning

**I. INTRODUCTION**

There are many computing paradigms which provide computational power and storage services for the huge amounts of data generated by an ever-increasing number of heterogeneous devices. Three of the most well-known and widely adopted computing paradigms are cloud, edge, and fog computing. The terms cloud, edge, and fog computing. The terms cloud, edge, and fog computing refer to the three computing tiers of cloud, edge, and fog computing systems. Each of these computing paradigms has its own advantages and disadvantages. Cloud computing provides enhanced data storage and computing power but causes high communication latency. Edge and fog computing provide similar services with lower latency but limited capacity, capability, and coverage. A single computing paradigm cannot fulfill all the requirements of IoT devices and a federation between them is needed to extend their capacity, capability, and services. This federation is beneficial to both subscribers and providers and also reveals research issues in traffic offloading between clouds, edges, and fogs. Optimization has traditionally been used to solve the problem of traffic offloading. However, in such a complex federated system, traditional optimization cannot keep up with the strict latency requirements of decision-making, ranging from milliseconds to sub-seconds. Machine learning approaches, especially reinforcement learning, are consequently becoming popular because they could quickly solve offloading problems in dynamic environments with some unknown information. This study provides a novel federal classification between cloud, edge, and fog and presents a comprehensive research roadmap on offloading for different federated scenarios. We survey the relevant literature on the various optimization approaches used to solve this offloading problem and compare their salient features. We then provide a comprehensive survey on offloading in federated systems with machine learning approaches and the lessons learned as a result of these surveys. Finally, we outline several directions for future research and challenges that have to be faced in order to achieve such a federation.

**II. OFFLOADING USING TRADITIONAL OPTIMIZATION**

Offloading is the process of shifting computational tasks from one device to another. In the context of federated systems, offloading can be achieved using traditional optimization techniques. Optimization has its origins in the European Telecommunication Standards Institute (ETSI), which proposed virtualizing the computing resources of cloud computing into mobile network operators. An edge server can be deployed behind a cellular system's base station and central office, which re-architects the central office as a data center (CORD) to provide more computing resources close to the user equipment (UE). This tier includes different service providers such as Verizon, T-Mobile, AT&T, and Chunghwa telecom, and so on, and consists of local network assets, micro data centers, central offices, base stations, etc. (c) Fog tier: the bottom tier is a fog system [3] or IoT...
system consisting of mobile users (e.g., smartphones, tablets, and laptops), smart vehicles forming vehicular fogs, and IoT devices, such as industrial actuators, wearable devices, and smart sensors. Fog nodes are located close to the data source and extend the infrastructures like cloud and edge. The fog computing resource aggregates some fog nodes at a given time and location. The computational node capacity can change over time [4]. It covers real-time data processing on industrial PCs, process-specific applications and autonomous equipment, a group of local computing devices, electronic vehicles, etc. Since most vehicles are now equipped with sensors and computing to ensure safety and comfort, vehicular fog is gaining popularity [5].

A. Cloud, Edge, and Fog Computing Paradigms

The Internet of Things (IoT) devices, that have taken the world by storm, need computational power and storage capacity for the huge amounts of data generated by them, to provide services to their subscribers [6]. Cloud, edge and fog computing are the potential paradigms that could fulfill the demand of subscribers [7]. Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without the need for direct active management by a user [1]. However, cloud computing introduces high communication latency in the hundreds of milliseconds because its servers are far from end-users or subscribers. A cloud computing paradigm is not suitable for some applications with stringent communication latency limitations, such as Ultra Reliable Low Latency Communications (URLLC) and enhanced Mobile Broadband (eMBB) services, which have a unit millisecond latency constraint. This is where edge and fog computing models play a crucial role in providing similar services with lower latency [8] [9].

Again, all these computing paradigms, i.e., cloud, edge, and fog, have limitations regarding capacity, capability, coverage, storage, and latency. A single computing paradigm cannot, by itself, fulfill the diverse requirements of a vast number of traditional and heterogeneous IoT devices. For example, a user might need to use two different applications at the same time, and one of them is latency sensitive, while the other is computation sensitive. In this case, the user would require the services provided by both cloud and edge or fog [10]. Also, if a cloud customer needs some extra service that is not available in that cloud, then the cloud must try to arrange that service for the customer without a delay to provide satisfaction. The cloud may otherwise lose the trust of the customer and, in some cases, may lose the customer, which may affect its business financially and reputation. This is where a federation between multiple computing paradigms can play a key role in resolving these issues. Such a federation is not only suitable for subscribers but also for providers. A subscriber will be able to access the services provided by different computing paradigms without having to buy multiple subscriptions. On the other hand, providers would be able to extend their capacity, capability, and coverage without having to lose subscribers to other providers.

B. Offloading in Federated Fog-Edge-Cloud Systems

A federation between multiple computing paradigms gives rise to many opportunities and challenges such as authentication, access control, resource sharing, and traffic offloading. Traffic offloading is an important challenge as the federation among multiple computing paradigms is a distributed system with dispersed computing capacity. In this work, we focus on offloading in a federated environment where cloud, edge, and fog offload traffic to each other. Such offloading is basically a transfer of tasks that are resource intensive to a separate platform in order to perform a task in a better way. Such offloading becomes necessary when a task assigned to a service provider exceeds its computing resources and has to be offloaded to another service provider that can provide the required computing power. Thus, offloading is required in order to fulfill different constraints under different situations. Some important constraints are latency, load balancing, privacy, storage constraints, guarantees, SLAs, etc.

There can be two types of offloading In a federated system, intra-domain and inter-domain offloading, as there are multiple domains in such a federation. Intra-domain offloading involves the traffic offloaded between the entities belonging to the same tier i.e., cloud-to-cloud, edge-to-edge, or fog-to-fog, while inter-domain offloading involves the entities belonging to different tiers, such as cloud-to-edge, cloud-to-fog, or edge-to-fog, etc. Intra-domain and inter-domain offloading can occur between two providers.

Optimization has traditionally been used effectively to offload traffic in single networks [11] or in a federation, because a single network provides the optimal offloading ratio that reduces the overall cost of the network. Although traditional optimization has been used for years, it takes much time to generate decisions because of a network’s complexity and the large number of variables involved. The non-convex algorithms in traditional optimization perform an exhaustive search to find an optimal solution, which takes much time to converge [12]. Modern applications are latency sensitive and cannot afford such delays in offloading decisions, as the control and data planes need a decision in milliseconds to seconds. In the current era, optimization solutions for quick offloading decisions are becoming obsolete and machine learning approaches are taking the place of traditional optimization in complex network systems because of their faster response times.

The machine learning approach has an advantage over the traditional optimization approach in such complicated federated systems, because machine learning does not require complete knowledge of the system compared to the traditional optimization, and it can quickly solve offloading problems with various bits of unknown information. In the various machine learning approaches, reinforcement learning (RL) is the most suitable for offloading decisions because RL does not need a well-labeled dataset, and can learn directly from the environment [13]. This makes RL suitable for offloading decisions in a dynamic environment with much unknown information. This also shows that the RL is better than the traditional optimization approach because, in such complex
systems, traditional optimization may not be able to converge to an optimal solution and may preferably rely on heuristics. Traditional optimization would take much more time for decision-making compared to the RL because of exhaustive searching. When we consider offloading in a complex federated environment together with traditional optimization, machine learning, and reinforcement learning, many research opportunities and challenges arise.

We summarize these various research opportunities such as V2X, fog-fog federation, mobility in vehicular-fog, scaling, resource allocation, centralized vs. distributed federation, etc. We also provide some insight into the important challenges that will be faced by the operators to deploy this kind of federation, such as redundancy, fault tolerance, service-level agreement (SLA), reliability, geo-diversity, performance, security, and interoperability between entities of the different domains in a federated environment.

C. Survey on Surveys

In this section, we discuss some recent studies that survey offloading in federated systems, as well as the importance of our survey. Table I compares offloading surveys which are divided into coverage, federation models, optimization approaches, and what the focus of that survey is.

The authors of [14]–[18] discussed traffic and data offloading between cellular, Wi-Fi, and opportunistic networks, but did not consider the computation offloading in a federated system such as an edge-cloud system. Rebecchi et al. [14] reviewed data offloading approaches in cellular systems with Wi-Fi environments and categorized them based on their latency requirements. Maallawi et al. [15] surveyed offloading and management approaches in wireless access and in core networks. Their objective was to address providers’ problems such as radio access scheduling, revenue per user decrease, and coverage. Chen et al. [18] surveyed traffic offloading in heterogeneous cellular networks, including small cells, Wi-Fi networks, and opportunistic networks, and [16] focused on the algorithm for selecting the optimal subset of offloading nodes in an opportunistic network, which would allow a node to offload traffic and computation tasks to another node. This kind of D2D offloading is beneficial to cellular operators and users in terms of monetary cost. Huan et al. [17] surveyed mobile data offloading, which involves small cells, Wi-Fi networks, opportunistic networks, and heterogeneous networks. The pros and cons of each of these networks are also detailed.

Computation offloading between mobile devices and the cloud is discussed in [19]–[21]. Kumar et al. [19] categorized offloading techniques based on the decision characteristics and applications. The security and privacy challenges in mobile cloud computing are discussed in [20]. The offloading techniques with environmental variation which included applications, networks, execution platforms, and cloud management, are summarized in [21].

Edge-cloud system offloading was surveyed in [10], [13], [22]–[28]. Mach et al. [22] discussed mobile edge-cloud system architectures and considered computation offloading resources allocation, and mobility management. Wu et al. [24] discussed multi-objective offloading, which was initiated by a large heterogeneous system such as mobile edge computing. Response time and energy consumption were their two main objectives. Offloading criteria were categorized into what, when, where, and how to offload. The taxonomy of edge-cloud offloading was categorized in [26], based on the task type, offloading scheme, objectives, device mobility, and multi-hop cooperation. De et al. [27] presented a classified taxonomy of V2X system offloading, based on a communication standard, problem, and experiment.

Fog-edge-cloud offloading was discussed in [10], [13], [23], [25], [28]. Jiang et al. [10] surveyed and discussed state-of-the-art computational offloading in mobile edge computing. Aazam et al. [23] discussed the offloading technologies in fog computing for IoT. The survey of Zhou et al. [25] focused on vehicular offloading, which included vehicle-to-vehicle, vehicle-to-infrastructure, and vehicle-to-everything, with a brief discussion of the architecture design, algorithm, and problem formulation. Lin et al. [28] focused on offloading modeling, which included communication, computation, energy harvesting, and channel modeling. Shakarami et al. [13] classified machine learning-based offloading into approaches such as supervised ML, unsupervised ML, and reinforcement learning (RL).

None of these surveys details a federation between cloud, edge, fog, and vehicular-fog. Each combination of such a federation has different characteristics and offloading directions which leads to complex issues. In most of the surveys, traditional optimization was used to optimize the offloading decision in a federation. Traditional optimization takes a long time to converge in such a complex federation system. By contrast, an offloading decision must be rapidly determined by the control plane. ML-based approaches have recently become popular to solve offloading optimization problems in such a complex federation system with fast response times. This survey focuses on edge-cloud federation offloading and covers state-of-the-art offloading approaches that use ML.

D. Survey Methodology

The survey methodology is outlined in this Section. Fog, edge, cloud, V2X, and vehicular-fog were the terms we searched for in articles published between 2012 and 2022 that pertain to offloading optimization for these edge systems. The cited publications study one or more computing paradigms, constituting a federated system. The majority of the selected papers are from IEEE journals and conferences. Some other journals are published in Science Direct, ACM, Springer, and MDPI. We selected articles based on their relevance to the survey topic and their citation count.

E. Contributions

The major contributions in this paper are as follows. First, we discuss the classification of federation between cloud, edge, and fog systems. Each federation scheme has different characteristics which affect the offloading directions. Second, we discuss the current research status of different federated architectures and offloading techniques and classify them.
Third, we survey offloading based on traditional optimization and machine learning approaches and make a comparative study of both approaches. Finally, we discuss some key research challenges associated with the task offloading and point to possible future research directions. To the best of our knowledge, this is the first work to discuss computing system federation, offloading optimization, and the offloading optimization approaches, i.e., traditional optimization and machine learning.

The rest of this paper is organized as follows. Section II describes a federation of cloud-edge-fog systems and the classification of such a federation. Section III presents the offloading, classification of the offloading, and the current research status of federated architecture and offloading. The survey on offloading is detailed in Section IV, which also classifies the approaches into traditional optimization and machine learning. Lessons learned from the survey are discussed in Section V. The research opportunities and challenges are presented in Section VI and the conclusions of this survey are discussed in Section VII.

### II. A Federation

A federation can be defined as the collection of clouds that cooperate to provide resources requested by users [29]. Stated another way, a cloud can provide computing resources wholesale or rent to another cloud provider [30]. A federation can render the cloud a user and resource provider at the same time [33]. A customer’s request submitted to one cloud can be fulfilled by another. A cloud provides capacity and coverage, but for latency reduction and fault tolerance, a cloud needs edge and fog. Likewise, fog and edge need a cloud service to increase their capacity and coverage. The federation could be realized if an agreement is established between service providers to develop cooperation that benefits them and their customers. All federation possibilities are depicted in Fig. 3, and federation agreements can be made between 1-to-1, 1-to-many, and many-to-many.

**TABLE I: Survey on Surveys on Offloading in the Federated Systems**

<table>
<thead>
<tr>
<th>References</th>
<th>Coverage</th>
<th>Federation model</th>
<th>Optimization</th>
<th>Focus of the survey</th>
</tr>
</thead>
<tbody>
<tr>
<td>[14]</td>
<td>✓</td>
<td></td>
<td>T</td>
<td>Data offloading techniques in cellular networks</td>
</tr>
<tr>
<td>[15]</td>
<td></td>
<td></td>
<td></td>
<td>Offload techniques and management in wireless access and core networks</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td></td>
<td></td>
<td>Traffic offloading in heterogeneous cellular network</td>
</tr>
<tr>
<td>[16]</td>
<td></td>
<td></td>
<td></td>
<td>Opportunistic Offloading</td>
</tr>
<tr>
<td>[17]</td>
<td></td>
<td></td>
<td></td>
<td>Mobile data offloading technologies</td>
</tr>
<tr>
<td>[19]</td>
<td></td>
<td></td>
<td>Single</td>
<td>Computation offloading for mobile systems</td>
</tr>
<tr>
<td>[20]</td>
<td></td>
<td></td>
<td></td>
<td>Security and privacy challenges in mobile cloud computing</td>
</tr>
<tr>
<td>[21]</td>
<td></td>
<td></td>
<td>T</td>
<td>Adaptation techniques in computation offloading</td>
</tr>
<tr>
<td>[22]</td>
<td></td>
<td></td>
<td></td>
<td>Architecture and Computation Offloading</td>
</tr>
<tr>
<td>[24]</td>
<td></td>
<td></td>
<td>T</td>
<td>Multi-objective decision-making for mobile cloud offloading</td>
</tr>
<tr>
<td>[26]</td>
<td></td>
<td></td>
<td></td>
<td>A Survey and taxonomy on task offloading for edge-cloud computing</td>
</tr>
<tr>
<td>[27]</td>
<td></td>
<td></td>
<td></td>
<td>Computation offloading for vehicular environments</td>
</tr>
<tr>
<td>[10]</td>
<td></td>
<td></td>
<td>T</td>
<td>Offloading in edge-cloud systems</td>
</tr>
<tr>
<td>[23]</td>
<td></td>
<td></td>
<td></td>
<td>Offloading in fog computing: Enabling technologies</td>
</tr>
<tr>
<td>[25]</td>
<td></td>
<td></td>
<td>T</td>
<td>Data offloading techniques in V2X networks</td>
</tr>
<tr>
<td>[28]</td>
<td></td>
<td></td>
<td>T</td>
<td>Computation offloading modeling for edge computing</td>
</tr>
<tr>
<td>[13]</td>
<td></td>
<td></td>
<td>ML</td>
<td>Machine learning-based approaches in mobile edge computing</td>
</tr>
<tr>
<td>Ours</td>
<td>✓</td>
<td>✓</td>
<td>T/ML</td>
<td>Offloading in federated cloud-edge-fog systems</td>
</tr>
</tbody>
</table>

T: Traditional; ML: Machine Learning

**T-system** is an example of the federation of clouds that covers cloud services supported by a number of cloud providers, including Open Telecom Cloud, AWS, Azure, and Google Cloud. They also federate edge and the mentioned public clouds to support rich services [31]. They deliver industry-specific, end-to-end IT systems and digital transformation solutions for all businesses and the public sector at the lowest possible cost and with the least amount of complexity. They provide extensive coverage in over twenty countries. Another federation example is Google’s Fi project, which allows federation across Wi-Fi and cellular networks. This project includes T-mobile and U.S. cellular service provider federation. This federation is, nonetheless, a type of communication service federation [32].

Cloud, edge, and fog computing paradigms provide different services to users or subscribers, depending on their limitations and capacity [37]. Since subscribers have different demands and service requirements, each paradigm may not have all kinds of services to fulfill all users’ needs because each computing paradigm has its limitations [38]. Thus there is a need for a “federation” between different service providers to cope...
TABLE II: Federation Framework

<table>
<thead>
<tr>
<th>Federation type</th>
<th>Framework</th>
<th>Features</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E-C</td>
<td>Zadara [34]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C-C</td>
<td>BEACON [35]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F-C</td>
<td>Kubefed [36]</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

with the users’ heterogeneous requirements and to increase the capacity, coverage, capability, and fault tolerance of the service providers. For example, in a smart city environment, different users or IoT devices have different requirements that a single service provider may not address. A federation then comes into the picture to fulfill the various demands. The benefits of the federation are twofold, i.e., from both a subscriber’s and provider’s perspectives. A subscriber would not have to subscribe to the services of all providers but will get the services of all by just subscribing to one of them. Subscribers do not have to keep multiple accounts and do not have to pay multiple providers. On the other hand, a provider will not lose a customer just because it cannot provide a particular service.

Table II presents several federation frameworks developed by enterprises or academic institutions. The federated program of Zadara [34] enables hosting providers or managed service providers to manage dispersed clouds and enable edge computing fully. Involving cloud and edge providers from around the world to deliver computing resources in proximity to users with low propagation latency. BEACON [35] manages cross-site virtual networks for federated cloud infrastructures in order to facilitate the automatic deployment of applications and services across various clouds and data centers. The overlay layer-2 and layer-3 network concepts are employed to interconnect services hosted by multiple cloud providers. These overlay networks are developed using OpenDOVE, which provides network virtualization. Kubefed [36] facilitates the management of several Kubernetes clusters via a centralized API. The goal is to facilitate the deployment of multi-geo applications. Kubernetes is an open-source container management, deployment, and scalability platform that may also be used to create fog, edge, and cloud. None of these frameworks specifically explain the offloading mechanisms that are used. Only Zadara provides load balancing to distribute the traffic in some instances but does not specify the action taken for incoming traffic in case an edge/cloud site is overloaded.

A. Federation vs. Non-federation

A non-federated scenario is one where a service provider cannot share its resources with other service providers, and it can neither lend nor rent its surplus resources to others. In such a scenario, it is difficult to handle the dynamic demands of users, and the service provider may face issues like Lock-in [39] [40] [41] and single point failure [42] [43]. Lock-in is one of the most cited and controversial obstacles to widespread cloud computing adopted by enterprises [44]. It is also risky for a customer to be tied to a single vendor because that vendor might raise prices, go out of business, become unreliable, or fail to keep up with technological progress.

Different service providers, such as cloud, edge, or fog, provide different services to their subscribers depending on their limitations and capacity [45]. Again, a subscriber of a service provider may have different demands at different times, which may not be fulfilled by the service provider always. This can also be understood with the help of an example of a perfect smart city where there would be different types of IoT devices, and each type would have its own requirements that a single IoT service provider cannot fulfill. In such a scenario, the provider will be able to provide all sorts of IoT services after federating with other providers. When acquiring IoT deployment, a federated environment is thus more beneficial than a non-federated scenario.

Federated fog-edge-cloud encounters several challenges due to the varied characteristics and administration complexities of fog-edge-cloud systems. Fig. 2 depicts the differences across computing paradigms regarding network locations, computing resource capacity, dispersion, dynamism, and ownership. The cloud has more computing power with less dispersion and volatility. Low dispersion indicates that the cloud has a centralized location. Edge and fog are both examples of distributed systems. Since fog comprises user equipment, which includes vehicular-fog, it has the highest computing resource dynamism.

In [46], Cominardi et al. listed some challenges of the federated system that also pertain to the challenges of federated offloading, including trust management, dynamic resource discovery, multi-tenancy, multi-virtualization technology co-existence, application placement, dynamic service placement and migration, dynamic resource management, and security. We introduce offloading as a federation challenge because, in a distributed system, incoming traffic must be spread across available resources. The challenges mentioned are the costs that must be considered during the federation. Maintaining trust between entities is also not trivial. The federated systems
address the aforementioned challenges by outsourcing to a trusted company (broker) or maintaining them distributively by the entities (peer-to-peer). This paper focuses on multi-direction offloading optimization in federated systems, which include a large distributed and dynamic system.

B. Classification of Federation

With federation technology, different users or subscribers of different service providers get different benefits. With this technology, different service providers can federate with each other to provide a better service to their users. A federation between these service providers can be divided into three categories, horizontal, vertical, and hybrid federations. These federations are all based on the cloud, edge, and fog integrated architecture; the classification of all possible federation scenarios is shown in Fig. 3. To the best of our knowledge, such a classification has not been dealt with in any of the studies we reviewed.

1) Horizontal Federation. A horizontal federation consists of two federated entities in the same tier, such as a cloud-cloud federation [47]. A horizontal federation can be cloud-cloud \((C-C)\) or \(C_H\), edge-edge \((E-E)\) or \(E_H\), or Fog-Fog \((F-F)\) or \(F_H\).

2) Vertical Federation. A vertical federation is a federation between entities in different tiers [33] as in a cloud-edge federation. Since a cloud-edge-fog system is a three-tier system, we can classify a vertical federation into two and three-tier federations, such as cloud-edge \((C-E)\), edge-fog \((E-F)\), and cloud-fog \((C-F)\) federations, or a cloud-edge-fog \((C-E-F)\) federation.

3) Hybrid Federation. A hybrid federation is a federation that combines both horizontal and vertical combinations [61], where entities can simultaneously federate horizontally with another entity in the same tier, as well as vertically with an entity in another tier. For example, in an edge-edge-cloud \((E_1-C)\) federation, an edge is federated with another edge in tier-2 and also federated with a cloud in tier-3. Such a hybrid federation can be classified into two-tier and three-tier federations.

1) A two-tier hybrid federation consists of all possible combinations of horizontal and two-tier vertical federations. For example, in \((C_H-E)\) federation, one cloud \((C_1)\) will federate with another cloud \((C_2)\) horizontally, and with an edge \((E_1)\) vertically. Similarly, in \((C_H-E_H)\), two clouds \((C_1\) and \(C_2)\) become federated with each other, two edges \((E_1\) and \(E_2)\) are federated with each other horizontally, and are simultaneously also federated vertically \((E_1\) and \(C_1)\). All nine possible two-tier hybrid federation combinations are shown in Fig. 3.

2) A three-tier hybrid federation consists of all possible combinations of a horizontal federation and three-tier vertical federation; all seven possible federation combinations are also shown in Fig. 3. For example, in \((C_H-E_H-F_H)\), two clouds \((C_1\) and \(C_2)\) in tier-3, are federated with each other, two edges \((E_1\) and \(E_2)\) in tier-2 are federated with each other, and two fogs \((F_1\) and \(F_2)\) in tier-2 are federated with each other. At the same time, \(C_1\) with \(E_1\), and \(E_1\) with \(F_1\) also become federated vertically.

The three federation schemes discussed above give new computing systems with varying characteristics, objectives, offloading directions, and complexity. Section III describes the concept of offloading, and classifies some offloading schemes required by some federated systems. This offloading scheme is
associated with the federation scheme. For example, horizontal offloading is employed between entities in the same tier, whereas vertical offloading is employed during federation between entities in different tiers.

III. OFFLOADING

When an entity or service provider (say SP1) with a federated architecture receives requests from its subscribers or customers, and needs another entity (say SP2) to execute tasks on behalf of the SP1 and return the results. This is called offloading [48]. Again, there are various criteria that are used when deciding whether to offload certain tasks. A few examples of this are as follows. To meet a resource constraint: when a task requires more computing resources than the local system’s available capacity, it must be offloaded to another system with the required capacity [49]. To address latency: as distance affects time-sensitive applications, the node closest to the receiving node must be involved in the task of offloading to provide the services faster [50]. Load balancing: when a server has reached its capacity for executing tasks, additional tasks need to be distributed between other entities in the service provider’s ecosystem [51]. Storage: small computing devices with limited storage facilities may require offloading to another that has a large storage capacity [52]. To maintain privacy, confidentiality, and security: depending on the sensitivity of data, they may be offloaded to more secure cloud storage [53].

A. Renting vs. Scaling vs. Offloading

In an offloading scenario, resources are used based on the requests from customers. These may vary from time to time, based on demand. Here, the use of resources can be scaled up or down based on the demand, and a customer will pay according to the use. This is called autoscaling. However, in renting, a customer will reserve the required resources for a predetermined duration for the future use. The customer may or may not utilize the entire resources that were reserved, but will pay according to the reservation. Offloading is a method where a service provider passes the request to another service provider to provide the service to its own customer. For example, a client of Amazon sends a request to Amazon, but Amazon passes the request to Google, and Google provides the service, provided there is a federation agreement between the two service providers.

B. Classification of Offloading

Based on the federation agreement between entities, one entity can offload its tasks to another entity for service. This offloading can be classified into Horizontal, Vertical, or Hybrid offloading, based on different federation agreements. Our offloading classification focuses on the computation capacity and communication time perspective. However, an offloading classification can be also applied to other criteria such as storage, security, etc.

1) Horizontal Offloading. Horizontal offloading always takes place between two entities in the same tier with a horizontal federation agreement. As with a horizontal federation, horizontal offloading also comes in three types, shown as #1 to #3 in Fig. 4.

- 1) In cloud-to-cloud (C ↔ C) horizontal offloading, two federated clouds can offload to each other [54]. Google can offload to Amazon or vice versa.
- 2) In edge-to-edge (E ↔ E) horizontal offloading, two service providers in edge tiers can offload to each other [55].
- 3) In fog-to-fog (F ↔ F) horizontal offloading, two computing resources in two different fogs can offload to each other [55].

2) Vertical Offloading. Vertical offloading always takes place between two entities in different tiers, for example, edge-to-cloud. There are fifteen different vertical offloading combinations from #4 to #18 in Fig. 4, which can be classified into four different categories: upward (#4 to #6), downward (#7 to #9), reverse (#7 to #9), and bi-directional (#10 and #11).

- 1) Vertical offloading occurs upward from the lower to the higher tier, which is more centralized, covers a bigger area, and has a greater computing capacity than the lower tier [78]. The possible upward offloading scenarios are edge-to-cloud (E ↩ C), fog-to-edge (F ↩ E), and fog-to-cloud (F ↩ C) offloading.
- 2) When an upper tier offloads its task to a lower layer entity that is closer to the user and has lower network latency than the upper tier, it is known as downward vertical offloading. [63], [96]. The possible downward offloading scenarios are cloud-to-edge (C ↩ E), cloud-to-fog (C ↩ F), and edge-to-fog (E ↩ F) offloading. These scenarios are triangular, i.e., the user requests are given to an upper-tier entity and then offloaded to a lower-tier entity. For example, in cloud-to-edge offloading, the cloud user gives its request to the cloud, then the cloud will offload the task to the edge with which it has a federation agreement.
- 3) Reverse offloading is a special type of downward vertical offloading, where the distance between two entities is relatively far, and to overcome latency and data transfer costs associated with highly time-sensitive tasks, an entity in the upper tier can reverse offload its task to a lower-tier entity [58]. These are non-triangular offloading scenarios, i.e., if there is a federation between two entities in two different tiers, and if a subscriber of an entity in an upper tier is closer to an entity in a lower tier, then a user’s requests are given directly to the lower tier entity, instead of to the entity in the upper tier. For example, cloud-to-edge reverse offloading: if there is a federation between cloud and edge, the subscriber to the cloud is closer than to an edge, it can directly send the request to the edge instead of to the cloud. Cloud-to-edge (C ~ E) and cloud-to-fog (C ~ F) are the two reverse offloading scenarios for our system. Since edge and fog are very close to each other, we do not consider the reverse offloading scenario between them.
- 4) Bidirectional offloading is a combination of all possible scenarios of upward with downward offloading, upward reverse offloading, or a combination of all three, i.e.,
upward with both downward and reverse offloading. For example, the offloading scenario #12 in Fig. 4 is a combination of offloading scenario #4 and #8; similarly, #14 is a combination of #4, #8 and #10. All possible bidirectional vertical offloading scenarios are shown in Fig. 4.

3) Omni-directional Offloading. Omni-directional offloading is the combination of all possible horizontal and bidirectional offloading scenarios. For example, the #22 offloading scenario in Fig. 4 is the combination of offloading scenarios #1 and #13 (a combination of #4 and #10). There are twenty-one different omni-directional offloading scenarios from #19 to #39 as shown in Fig. 4.

However, these scenarios are only limited to two-tier architectures. They can be further extended to three-tier architecture by combining two two-tier architectures. To the best of our knowledge, such classification of the offloading scenarios has not previously been considered and is here set out.

C. Current Research Status of Federated Architectures and Offloading

Before doing the survey, we consider the current status of different federated architectures and offloading scenarios, which are divided into three categories, as shown in Table III. Fig. 4 shows 39 different offloading scenarios. However, out of these scenarios, 11 are core offloading scenarios that are considered for this categorization based on a one-to-one federation and offloading. Some scenarios have been addressed in many papers, which we consider as a saturated scenario—for example, a $C \rightarrow C$ federation. In [59], Mashayekhy et al. proposed a game-theoretical model to reshape the business structure between cloud providers, which could improve their dynamic resource scaling capabilities by establishing cooperation with the federation method. They proposed a cloud federation mechanism to maximize the profit of cloud providers, by reducing the utilization of computing resources. Hassan et al. [60] presented a capacity-sharing mechanism using game theory in a federated cloud environment. This mechanism may lead to a global energy sustainability policy for federated systems and can encourage such systems to cooperate. The main goal of the paper is to minimize the overall energy cost by means of a capacity sharing technique, that will promote the long-term individual profit of cloud providers.

The integration of vertical and horizontal cloud federations is discussed in [61]. In this integration, private clouds are known as secondary clouds, and are federated with each other horizontally, which become federated vertically with the public clouds, termed primary clouds. The objective of [61] is to establish stable cooperative partnerships for the federation to improve efficiency. In [62], a distributed resource allocation problem is discussed in a horizontally dynamic

<table>
<thead>
<tr>
<th>Saturation Level</th>
<th>Federation Types</th>
<th>Offloading Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturated</td>
<td>$C \rightarrow C$</td>
<td>$C \leftrightarrow C$</td>
</tr>
<tr>
<td>Semi-saturated</td>
<td>$C \rightarrow F$, $E \rightarrow F$, $E \rightarrow E$</td>
<td>$E \downarrow C$, $F \uparrow C$, $C \uparrow E$, $E \leftrightarrow E$</td>
</tr>
<tr>
<td>New</td>
<td>$F \rightarrow F$</td>
<td>$F \leftrightarrow F$, $C \downarrow E$, $C \uparrow F$, $E \downarrow F$, $C \sim E$, $C \sim F$</td>
</tr>
</tbody>
</table>

Fig. 4: Classification of offloading based on federation

TABLE III: Saturation Level of Federated Systems and Different Offloading Scenarios
cloud federation (HDCF) platform. These authors used a game theoretical solution to address this problem, to ensure mutual benefits to encourage cloud providers (CPs) to form an HDCF platform.

Similarly, cloud-to-cloud offloading is very rare as the clouds lack capacity, capability, etc. One cloud may not have something that another one can cover, and it is then considered saturated. There are some federation architectures and offloading scenarios which have been addressed by some researchers, but there is still much to address. These scenarios are termed semi-saturated; the rest are called new scenarios, in which hardly any research has been done. These three categories are shown in Table III. Note that the fog used in this paper includes any static or dynamic fog, including vehicular-fog that may have mobility.

Fig. 5 illustrates offloading in a three-tier fog-edge-cloud federation. The fog system comprises a variety of devices, such as smartphones, laptops, automobiles, and roadside units (RSUs), all of which interact with one another and can even collaborate on some tasks. Between fog and cloud lies a two-tier MEC system with computing capacity behind the base stations (AN-MEC) and in a central office with core network functions (CN-MEC). Cloud computing is the top tier, with massive computing capacity but is geographically remote from UEs or data sources.

Fig. 5 shows three different offloading scenarios based on task sources. The first scenario (1) involves a heavy task or hotspot traffic at a stadium that is hosting a sporting event or music concert. The task will be offloaded from the UEs to the nearest AN-MEC. Because of the AN-MEC’s limited computational capabilities, the task can be offloaded to a less loaded AN-MEC or CN-MEC, and computing delay can thus be minimized. In the second scenario (2), the vehicle generates tasks from its sensors or multimedia applications for safety and comfort. Some vehicle tasks are latency-sensitive that are part of the navigation, autopilot, accident, or alert systems. A nearby server must serve those kinds of tasks with low propagation and computing latencies. The tasks can be offloaded either horizontally to other vehicles or vertically to an RSU. If the RSU is overloaded, it will vertically offload the tasks to an AN-MEC, and the overloaded AN-MEC can offload the tasks downward to vehicular fog. The third scenario (3) describes the traffic generated by industrial IoT sensors, with some operations requiring low latency, such as robotic process automation, danger alerts, and suspicious activity alerts, and can be vertically offloaded to AN-MEC. Large amounts of sensor data from industrial IoT can be offloaded to a cloud for future analysis.

IV. SURVEY ON OFFLOADING

This section provides a summary of the literature that deals with the federated environment with different offloading scenarios. Some papers discuss current surveys on cloud federation [56] [57] with cloud-to-cloud offloading, some edge federation [95], some edge-to-cloud offloading [23], some edge-to-vehicular-fog offloading [63], and some cloud to edge reverse offloading [58]. The major purpose of a federation is to enhance storage and processing capabilities. Many factors influence offloading strategies, such as the location [64], energy [67], and different optimization objectives. We classify this work on offloading into two categories, (a) traditional optimization techniques, that mostly focus on management plane decisions, and (b) machine learning techniques that focus on control plane decisions.

A. Traditional Optimization

Table IV lists the earlier research on traditional optimization-based offloading, according to the direction
TABLE IV: Analysis of Recent Research on Offloading in the Federated Systems with Traditional Optimization

<table>
<thead>
<tr>
<th>References</th>
<th>Offloading Types</th>
<th>Metrics</th>
<th>Approach</th>
<th>Method</th>
<th>Evaluation</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>[64]</td>
<td>\textit{D2D}</td>
<td>✓ ✓</td>
<td>Exact</td>
<td>Convex optimization</td>
<td>Simulation</td>
<td>Offloading in realistic human mobility scenario</td>
</tr>
<tr>
<td>[65]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Lyapunov optimization</td>
<td>Simulation</td>
<td>Fogging framework</td>
<td></td>
</tr>
<tr>
<td>[66]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>Tree search algorithm</td>
<td>Simulation</td>
<td>Face recognition</td>
<td></td>
</tr>
<tr>
<td>[67]</td>
<td>✓ ✓</td>
<td>No approach</td>
<td>Real test-bed</td>
<td>Experimental</td>
<td>MCC application</td>
<td></td>
</tr>
<tr>
<td>[68]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Lyapunov optimization</td>
<td>Simulation</td>
<td>Face recognition</td>
<td></td>
</tr>
<tr>
<td>[69]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>ODLOO</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
</tr>
<tr>
<td>[70]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Branch and bound</td>
<td>Simulation</td>
<td>Smart mobile device (SMD)</td>
<td></td>
</tr>
<tr>
<td>[71]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>EPCO algorithm, LPCO algorithm</td>
<td>Simulation</td>
<td>Data partitioned in SMD</td>
<td></td>
</tr>
<tr>
<td>[72]</td>
<td>✓ ✓</td>
<td>Policy</td>
<td>One-dimensional search algorithm</td>
<td>Simulation</td>
<td>MEC systems</td>
<td></td>
</tr>
<tr>
<td>[73]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Lyapunov optimization</td>
<td>Simulation</td>
<td>Energy harvesting for devices</td>
<td></td>
</tr>
<tr>
<td>[74]</td>
<td>✓ ✓</td>
<td>Policy</td>
<td>Lyapunov optimization</td>
<td>Simulation</td>
<td>Multi-user multi-tasking</td>
<td></td>
</tr>
<tr>
<td>[75]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>Aura architecture</td>
<td>Experimental</td>
<td>Prototype design</td>
<td></td>
</tr>
<tr>
<td>[76]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>EEPCO scheme</td>
<td>Simulation</td>
<td>5G heterogeneous networks</td>
<td></td>
</tr>
<tr>
<td>[77]</td>
<td>✓ ✓</td>
<td>Policy</td>
<td>KKT conditions</td>
<td>Simulation</td>
<td>Augmented reality</td>
<td></td>
</tr>
<tr>
<td>[78]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>Iterative greedy</td>
<td>Simulation</td>
<td>Intelligent transportation systems</td>
<td></td>
</tr>
<tr>
<td>[79]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>Iterative greedy, DOCP</td>
<td>Simulation</td>
<td>Intelligent transportation systems</td>
<td></td>
</tr>
<tr>
<td>[80]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>EEPCO scheme</td>
<td>Simulation</td>
<td>Industrial IoT</td>
<td></td>
</tr>
<tr>
<td>[81]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>LTS-AQW</td>
<td>Simulation</td>
<td>Real-time applications</td>
<td></td>
</tr>
<tr>
<td>[82]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>DPTO</td>
<td>Simulation</td>
<td>IoT applications</td>
<td></td>
</tr>
<tr>
<td>[83]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Iterative greedy</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
</tr>
<tr>
<td>[84]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Subgradient iterative method</td>
<td>Simulation</td>
<td>MCC applications and ITS</td>
<td></td>
</tr>
<tr>
<td>[85]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>Iterative search algorithm</td>
<td>Simulation</td>
<td>Multi-cell MEC networks</td>
<td></td>
</tr>
<tr>
<td>[86]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>Iterative searching</td>
<td>Simulation</td>
<td>Multimedia applications</td>
<td></td>
</tr>
<tr>
<td>[87]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Branch-and-bound and Edmonds–Karp</td>
<td>Simulation</td>
<td>Traffic management system</td>
<td></td>
</tr>
<tr>
<td>[88]</td>
<td>✓ ✓</td>
<td>Heuristic</td>
<td>Bisecion method</td>
<td>Simulation</td>
<td>Multi-cell wireless network</td>
<td></td>
</tr>
<tr>
<td>[89]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Branch and bound</td>
<td>Simulation</td>
<td>Traffic engineering</td>
<td></td>
</tr>
<tr>
<td>[90]</td>
<td>✓ ✓</td>
<td>Scheme</td>
<td>Dynamic algorithm</td>
<td>Experimental</td>
<td>Traffic engineering</td>
<td></td>
</tr>
<tr>
<td>[91]</td>
<td>✓ ✓</td>
<td>No approach</td>
<td>Architecture</td>
<td>Osmotic computing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[92]</td>
<td>✓ ✓</td>
<td>Analysis</td>
<td>Simulated annealing</td>
<td>Simulation</td>
<td>Traffic engineering</td>
<td></td>
</tr>
</tbody>
</table>

\textit{D}: Device; \textit{D2D}: Device to Device; \textit{MCC}: Mobile Cloud Computing; \textit{MEC}: Multi-access Edge Computing; \textit{IoT}: Internet of Things; \textit{EPCO}: Energy-Optimal Partial Computation Offloading; \textit{LPCO}: Latency-Optimal Partial Computation Offloading; \textit{LEAD}: Latency-Aware workloAd offloaDing; \textit{EEPCO}: Energy-Efficient Computing Offloading; \textit{KKT}: Karush-Kuhn-Tucker; \textit{DOCP}: Decentralized Offloading Configuration Protocol
and destination of offloading.

1) Device-to-Device (D2D) Offloading. Some research papers focus on device-to-device (D2D) offloading [64]–[66]. Wang et al. [64] investigated the mobility-assisted opportunistic computation offloading problem focusing on the patterns of contacts between mobile devices. They used the convex optimization method to determine the amount of computation tasks that can be offloaded from one device to another. Pu et al. [65] proposed a device-to-device (D2D) fogging framework, where mobile users can dynamically and beneficially share computation and communication resources between themselves. The objective of D2D fogging is to achieve optimal energy conservation for executing the tasks of network-wide users. Yu et al. [66] proposed a hybrid multicast-based task execution framework for multi-access edge computing (MEC). In this framework, multiple devices can collaborate at the edge of a network for wireless distributed computing (MDC) and outcome sharing. Such a framework is socially aware of building effective D2D links with the objective of achieving an energy-efficient task assignment policy for mobile users. They used the Monte-Carlo search tree-based algorithm to achieve their objective.

2) Device-to-Fog (D2F), Device-to-Edge (D2E) and Device-to-Cloud (D2C) Offloading. Two papers, [67] and [75], focused on device-to-cloud and device-to-fog offloading, respectively, while device-to-edge offloading was discussed in [68]–[74]. Barbera et al. [67] tested the feasibility of mobile computation offloading in real-life scenarios. They considered an architecture where each real device is associated with a software clone on the cloud. Huang et al. [68] proposed a dynamic offloading algorithm based on the Lyapunov optimization that maximizes energy efficiency while preserving the required latency with face recognition applications. Zhang et al. [69] investigated the trade-off between energy consumption and latency for an MEC system with energy harvesting technology. They formulated the weighted sum of energy consumption and computation latency minimization of mobile device with the stability of queues and battery level, and used the Lyapunov function to ensure system stability.

Zhao et al. [70] proposed a multi-mobile-user MEC system, where multiple smart mobile devices (SMDs) can offload their tasks to an MEC server, with the objective of minimizing the energy consumption of SMDs. To optimize this, they coordinated the offloading selection, radio resource allocation, computational resource allocation, and used the branch and bound method to solve the optimization problem. Wang et al. [71] investigated partial computation offloading with dynamic voltage scaling (DVS) technology, in mobile edge computing, where devices can partially offload their tasks. They formulated an optimization problem with two objectives: energy consumption of SMD minimization (ECM) and latency minimization of application execution (LM). They proposed two optimal algorithms named Energy Optimal Partial Computation Offloading (EPCO), and Latency Optimal Partial Computation Offloading (LPCO) to solve the ECM, and LM problems, respectively.

To achieve minimum average delay, Liu et al. [72] adopted the Markov decision model for computational task scheduling. They proposed a searching algorithm to determine optimal scheduling. Such task scheduling is unique, as the computation tasks are scheduled based on the queuing state of the task buffer, the execution state of the local processing unit, and the state of the transmission unit. Mao et al. [73] developed a Lyapunov Optimization-based Dynamic Computation Offloading (LODCO) algorithm to minimize the execution delay and addressed task failure as the performance metric. This algorithm determines the offloading decision, the CPU-cycle frequencies for mobile execution, and the transmission power for computation offloading. However, without requiring distribution information such as computation task requests, wireless channel, energy harvesting (EH) processes, etc., these decisions depend only on the system’s current state.

Chen et al. [74] formulated a multi-user, multi-task computation offloading problem for green Mobile Edge Cloud Computing (MECC), and used the Lyapunov Optimization approach to determine an energy harvesting policy. This policy determines how much energy is harvested from each wireless device (WD) in the task offloading schedule – the set of computation offloading requests that can be admitted into the mobile edge cloud, the set of WDs that can be assigned to each accepted offloading request, and the amount of workload that can be processed at the assigned WDs. In [75], Hasan et al. present the Aura architecture, a highly localized and mobile ad-hoc cloud computing model using IoT devices present in the ubiquitous environment for task offloading schemes and enhancing applications. They implemented the Aura on the Contiki platform and a simplified Map-Reduce port, which demonstrates such architecture’s feasibility.

3) Device-Fog-Cloud and Device-Edge-Cloud Vertical Upward Offloading. The offloading scenarios adopted in papers [76]–[78], [80]–[83] were vertical upward, which included from device to any entity offloading, and one entity to another entity offloading. Gou et al. [76] presented an architecture for collaborative computation offloading over FiWi networks. They addressed the problem of cloud-MEC collaborative computation offloading to minimize the energy consumption of all the MDs, while satisfying the computation execution time constraint. They proposed a distributed collaborative computation offloading scheme by adopting game theory and analyzing the Nash equilibrium.

Sun et al. [77] addressed the latency-aware workload offloading (LEAD) problem, where they formulated a task offloading problem to minimize the average response time for mobile users. They designed the LEAD strategy, and offloaded the workloads to suitable cloudlets to reduce average response times. Tong et al. [78] proposed a hierarchical edge cloud architecture to improve the performance of mobile computing by leveraging cloud computing, and offloading mobile workloads for remote execution at the cloud. For the efficient utilization of resources and workload placement, they used simulated annealing (SA) [79] to determine which programs are placed on which edge cloud servers, and how much computational capacity is available to execute that program. They implemented the proposed architecture in small-scale, conducted a simulation experiment over a larger topology, and evaluated the performance of a proposed workload placement.
can an edge offload its tasks to the cloud, but the cloud can develop the concept of reverse offloading, where not only the objective of this architecture was to minimize the communication and data processing delays in IOT systems. Resource allocation and offloading optimization for heterogeneous real-time tasks were carried out by means of an adaptive queueing weight (AQW) resource allocation policy in [82]. A trade-off between throughput and task completion ratio optimization was also achieved by taking laxity and completion times into account when designing the offloading policy. Adhikari et al. [83] designed a novel delay-dependent Priority-Aware Task Offloading (DPTO) algorithm for scheduling and handling IoT device tasks in an appropriate computing server. The computing locations were chosen based on the types of task deadlines, which were classified as soft and hard-deadline tasks.

4) Device-Edge-Cloud Hybrid Offloading. Hybrid offloading was discussed in [84]–[86] which included device-edge vertical offloading. Tran and Pompili [84] formulated a mathematical model for the joint optimization of task offloading and resource allocation in MEC. In this work, they did not only account for the allocation of computing resources but also for the allocation of the transmission power of mobile users.

The two-tier MEC architecture proposed by yahya et al. [85] comprises an access network MEC (AN-MEC) and a core network MEC (CN-MEC). CN-MEC has greater capacity, but is less wide spread than AN-MEC. Two-phase optimization was used to achieve capacity optimization by modifying the offloading ratio and capacity iteratively. For hot-spot traffic, offloading and scaling were merged into short-term and long-term solutions. They considered both vertical, device-edge, and horizontal offloading between edges. In a comparison between pre-CORD and CORD, shown in Fig. 7, a trade-off between computing and communication latency was introduced for different distances of the CN-MEC, which affected the task processing distribution. Thai et al. [86] proposed workload and capacity optimization to minimize computation and communication costs for cloud-edge federated systems, by taking into consideration vertical and horizontal offloading. They designed a branch and bound algorithm with parallel multi-start search points to solve this problem.

Villar et al. [58] introduced osmotic computing, a new paradigm for edge and cloud integration. In their research, they developed the concept of reverse offloading, where not only can an edge offload its tasks to the cloud, but the cloud can also reverse offload time-sensitive tasks to edges. A two-tier cloud-edge federated architecture was proposed by Kar et al. [97], who considered edge-to-edge horizontal offloading and edge-to-cloud vertical offloading, together with cloud-to-edge reverse offloading. They formulated an optimization problem with the objective of minimizing costs where latency was the key constraint, and used simulated annealing to solve it. As shown in Fig. 6, the simulated annealing technique gathers system information and carries out an exhaustive search into acquiring the best offloading decision.

5) Fog-Edge-Cloud Vertical Upward Offloading. Some papers [87], [89], [91]–[92] focus on entity to entity upward offloading, and some adopt hybrid offloading scenarios [94]–[95]. Fantacci and Picano [87] carried out queuing analysis of cloud-fog-edge computing infrastructure, and proposed a heuristic to determine offloading ratios and computing capacities at fog, edge, and cloud. Kar et al. [88] considered a federated architecture with mobile device, edge, cloud, and vehicular-fog together. They used the queuing theory to analyze the performance to minimize QoS violation probability, and used a subgradient searching algorithm to determine the optimal probabilities.

An intelligent offloading method (IOM) for smart cities, conserving privacy, improving offloading efficiency, and promoting edge utility, was proposed to address the privacy disclosure in Xu et al. [89]. The authors used the ant colony optimization (ACO) [90] method to achieve the trade-offs between minimizing service response time, energy optimization, and maintain load balance, while ensuring privacy preservation during service offloading. An energy-efficient computation offloading mechanism for MEC in 5G heterogeneous networks was proposed in [91]. They formulated the energy minimization problem of an offloading system, where both task computing and file transmission energy costs were considered.

Lu et al. [92] addressed the problem of computation offloading by using edge computing. They formulated the problem...
as a two-stage Stackelberg game problem and show that it achieves a Nash equilibrium. Their objective was to maximize cloud service operators’ and edge server owners’ utilities by obtaining optimal payment and computation offloading strategies with low delay. Ma et al. [93] proposed a cloud-assisted framework in MEC, termed Cloud Assisted Mobile Edge computing (CAME), to minimize resource costs by combing queuing network and convex optimization theories. They solved the convex problem by using Karush-Kuhn-Tucker (KKT) conditions, and augmented reality to represent delay-sensitive and computation-intensive mobile applications.

Jiao et al. [94] presented an integrated framework for computation offloading and resource allocation in MEC networks, where both single and multi-cell networks were taken into consideration. To minimize energy consumption and delay, they proposed an energy-aware offloading scheme that considers both computation and communication resource allocation. In [95], a horizontal edge federation was proposed together with UE to edge, and edge to cloud vertical offloading scenarios. They experimentally showed that an edge federation model improves the quality of experience (QoE) of end-users and saves on the costs of edge infrastructure providers (EIPs).

6) Vehicular-Fog and V2X Offloading. The single edge to vehicular-fog task offloading problem was addressed in [63], where an iterative greedy algorithm was used to solve the optimization problem. Yen et al. [96] proposed a decentralized offloading configuration protocol (DOCP) for single edge to vehicular-fog offloading, with a matching protocol between multiple edge systems to resolve the resource contention, when resources from the same vehicular-fog were requested simultaneously.

Offloading optimization for vehicular-to-everything (V2X) systems was addressed in [98], [99]. Zhang et al. [98] considered hybrid offloading between vehicles and fogs, and formulated a mixed-integer, nonlinear programming (MINLP) solution for optimizing both user association, and radio resource allocation in vehicular networks (VNET). To obtain a globally optimal solution, this MINLP problem was transformed by applying norm theory to non-convex nonlinear fraction optimization, and then showed to be equivalent to convex optimization using weighted minimum mean square error (WMMSE) and Perron-Frobenius theory. Wang et al. [99] proposed a real-time traffic management algorithm for fog-based Internet-of-Vehicle (IoV) systems. This consisted of a three-tier architecture of fog, cloudlet, and cloud for providing computing resources to traffic management systems. They also looked into vertical offloading optimization between fog, cloudlet, and cloud. Some modeled the offloading optimization problem into convex optimization and used exhaustive searching to find the optimal solution. The global optimum is easier to get in such a convex problem than in a non-convex problem.

D2D, V2X, and vehicular-fog offloading optimization incorporate dynamic resource capacity and network topology, which are difficult to handle using traditional optimization techniques. A heuristic approach such as the previously stated Monte-Carlo search tree could be utilized. The heuristic techniques are the most commonly utilized algorithm in offloading optimization, as shown in Table IV. This heuristic, however, may get stuck in local minima/optima. Another method for obtaining optimal offloading decisions, is simplifying the problem, for example, determining how much traffic to be offloaded or where to offload (one dimension), and creating a convex problem. Choosing where and how much to offload introduces a large continuous action space that is challenging to solve with classical optimization.

A summary of the above-discussed literature is given in Table IV. The organization of the comparison table is as follows. We discussed different core offloading methods used in the papers, including device-to-device (D2D) and device to other entities. Four standard metrics, i.e., cost, energy, capacity, and latency, are considered that are commonly used in most literature. Although, there are other factors such as QoS, load balance, intensive, etc., that are not presented in the
table, but are already addressed in descriptions. Each paper has a different approach such as exact, analysis, scheme, policy, heuristic, and evaluation method presented in the table.

### B. Machine Learning

In federation architecture, an offloading module which distributes tasks from one entity to other entities or tiers, is part of the control plane. The decision of task offloading in an extensive federated system must be carried out quickly, usually in seconds. Traditional optimization, such as a non-convex algorithm, carries out an exhaustive search that takes a long time to converge, and violates the delay requirements of tasks [123]. Furthermore, a traditional optimization algorithm needs complete system information to determine offloading, which some federations may not provide. Intensive system monitoring, that provides complete information for determining offloading action in a federation, is not trivial because each provider uses different devices, protocols, and operating systems. Some applications provided by federation, may also have different requirements [100], [101], [103], [109]. Machine learning is a suitable approach to address such offloading problems in a highly dynamic system with some unknown information.

Machine learning-based (ML) offloading can automatically improve its actions by learning from the collected data (dataset), or interacting with the environment. Some ML approaches are compared in Table V. Supervised ML and Deep Learning (DL) update their model’s weight in order to execute the best offloading decision by learning from previous data, which is categorized as offline learning. A well-labelled dataset has first to be constructed before being provided to the ML algorithms. Gathering comprehensive data and labelling the data in such a dynamic network system are costly. Even more, some providers may restrict the details of their datasets because of security. Another way to train an offloading model is through online interaction between a learning agent and the environment, which is termed Reinforcement Learning (RL). The learning agent observes an environment’s conditions to determine an offloading action. An environment will then give positive and negative feedback on the taken action, termed reward and punishment. In essence, an agent memorizes this interaction in the form of a table to decide the best action to take in the future. In a large system, such as a federation, maintaining agent interactions in a table leads to a scalability problem. Deep Reinforcement Learning exchanges the table with a neural network which can predict the reward of an action for given environment’s state. RL basically uses a random action at the beginning of the iteration. Even it can produce the offloading decision quickly, but relying on the random action in the beginning induces long convergence times.

These concepts are classified into 11 types in Table VI, depending on their offloading direction and destination in the federated fog, edge, and cloud.

1) **Device-Edge-Cloud Offloading.** Junior et al. [100] considered cloud capacity to provide an external computation capacity to UE applications, such as image editors, face detection, and online games. They proposed a device application architecture that consists of middle-ware, a profiler, and a decision engine to determine offloading policy. The decision engine employs multiple classifiers to decide the offloading action for each incoming task, with the objectives of minimizing latency and energy efficiency. This classification relies on the characteristics of applications and the system conditions. When the number of applications increases and the system conditions change dynamically, creating labels for this classification will be expensive.

There are some studies [101]–[112] that dealt with the device to edge offloading. Other than offloading policy, Saguil and Azim [101] also considered caching strategy to locate the application codes and data. Q-learning and DQN-based algorithms solved this joint optimization problem. Li et al. [102] considered task deadline time in determining task offloading policy. They proposed an E2D DRL to derive the best offloading policy and solve the scalability problem of DQN action space. Wang et al. [103] optimized a UAV trajectory and offloading decision, which included discrete and continuous variables, by using multi-agent reinforcement learning. The previously mentioned DQN approach is only applicable to discrete action space problems. DDPG technique was chosen because it solved the overestimation problem of RL and worked in high-dimensional action spaces (continuous action space). Fig. 8 shows the DDPG algorithm overview. This is implemented on an agent that determines the optimal

<table>
<thead>
<tr>
<th>ML Approaches</th>
<th>Paper</th>
<th>Online learning</th>
<th>Supervisor</th>
<th>Learning object</th>
<th>Model dependence</th>
<th>Learning direction</th>
<th>Performance</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised ML</td>
<td>[100], [102], [103]</td>
<td>Yes</td>
<td>Dataset</td>
<td>✓</td>
<td>✓</td>
<td>Depend on data and learning algorithm</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>DL</td>
<td>[108], [112], [116], [117]</td>
<td>Yes</td>
<td>Dataset</td>
<td>✓</td>
<td>✓</td>
<td>Through exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAB</td>
<td>[121]</td>
<td>✓</td>
<td>No</td>
<td>Environmental</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RL)Q-Learning</td>
<td>[107], [111]</td>
<td>✓</td>
<td>No</td>
<td>Environmental</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(DRL)DQN</td>
<td>[104], [110], [114], [118], [133]</td>
<td>✓</td>
<td>No</td>
<td>Environmental</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(DRL)E2D</td>
<td>[105]</td>
<td>✓</td>
<td>No</td>
<td>Environmental</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(DRL)DDPG</td>
<td>[101], [103], [106], [109], [115], [120]</td>
<td>✓</td>
<td>No</td>
<td>Environmental</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
offloading decision based on federated system data, such as channel status, arrival traffic information, computation, and networking capacity. DDPG makes use of Actor and Critic neural networks. Actor networks predict the optimal action for a given state, whereas Critic networks predict the value of state-action pairs. The Q-value provides the discounted total future reward for the current state-action pair. By satisfying Bellman’s equation, the critic network learns this value.

Joint offloading and resources allocation optimization was carried out by Yang et al. [104], who applied single and multi-agent reinforcement learning to optimize caching and offloading decisions, and LSTM to predict task popularity in pre-processing. Ale et al. [105] addressed the computation offloading problem of a multi-server MEC system by using DRL. Reformattting the features and storing in a tree-like data structure were carried out to accelerate the DRL’s convergence time.

DRL was used in [106] to group NOMA’s UEs to minimize offloading energy, by minimizing multiple access interference.

Chen et al. [107] extended DDPG with a temporal feature extraction network (TFEN), and a rank-based Prioritized Experience Replay (rPER) to achieve training stability, and reduce the convergence time. Guo et al. [108] used a binary-tree-based supervised ML to construct an intelligent offloading task with high accuracy and low complexity. Multichannel access problems arise in multi-user offloading when some mobile users utilize the same channel, which then results in longer transmission latency due to interference. Cao et al. [109] used multi-agent reinforcement learning to derive the best offloading policy. The user device plays the role of an agent that observes channels condition to determine the offloading policy. Yang et al. [110] combined offline learning based on a feed-forward neural network and online inference to derive an offloading strategy in near real-time. Zhang et al. [113] enhanced the DQN algorithm with a heuristic offloading technique, in order to reduce both latency and energy consumption. The objective of using a heuristic algorithm was to minimize convergence time. DQN and DDPG

### TABLE VI: Analysis of Recent Research on Offloading in the Federated Systems with Machine Learning

<table>
<thead>
<tr>
<th>References</th>
<th>Offloading Types</th>
<th>Metrics</th>
<th>Method</th>
<th>Agent</th>
<th>Evaluation</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>[100]</td>
<td>D1C</td>
<td>✓ ✓</td>
<td>ML supervised</td>
<td>I</td>
<td>Experimental</td>
<td>Multimedia apps.</td>
</tr>
<tr>
<td>[101]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Real-time video analytic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[102]</td>
<td>✓ ✓</td>
<td>(DRL)E2D</td>
<td>Simulation</td>
<td>Video, smart home, and AI apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[103]</td>
<td>✓ ✓</td>
<td>(MARL)DDPG</td>
<td>Simulation</td>
<td>UAV based application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[104]</td>
<td>✓ ✓</td>
<td>(MARL)Q-Learning</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[105]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Generic user application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[106]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[107]</td>
<td>✓ ✓ ✓</td>
<td>(DRL)DDPG+ TADPG</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[109]</td>
<td>✓ ✓</td>
<td>(MARL)DDPG</td>
<td>Simulation</td>
<td>IoT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[110]</td>
<td>✓ ✓</td>
<td>DL</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[111]</td>
<td>✓ ✓</td>
<td>(DRL)DQN &amp; DDPG</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[112]</td>
<td>✓ ✓</td>
<td>(DRL)DDPG+ Optimization</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[113]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[114]</td>
<td>✓ ✓</td>
<td>DL</td>
<td>Simulation</td>
<td>IoT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[115]</td>
<td>✓ ✓</td>
<td>(MARL)DDPG</td>
<td>Simulation</td>
<td>IoT applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[116]</td>
<td>✓ ✓</td>
<td>(MARL)Q-Learning</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[117]</td>
<td>✓ ✓</td>
<td>ML supervised</td>
<td>I</td>
<td>Experimental</td>
<td>IoT with ML jobs</td>
<td></td>
</tr>
<tr>
<td>[118]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[119]</td>
<td>✓ ✓</td>
<td>(MARL)DQN</td>
<td>Simulation</td>
<td>IoT applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[120]</td>
<td>✓ ✓</td>
<td>DL-unsupervised</td>
<td>I</td>
<td>Simulation</td>
<td>Mobile applications</td>
<td></td>
</tr>
<tr>
<td>[121]</td>
<td>✓ ✓</td>
<td>DL + Pareto optimization</td>
<td>I</td>
<td>Simulation</td>
<td>V2X applications</td>
<td></td>
</tr>
<tr>
<td>[122]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>V2X applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[123]</td>
<td>✓ ✓</td>
<td>ML, MAB</td>
<td>Emulation</td>
<td>V2X applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[124]</td>
<td>✓ ✓</td>
<td>MAB</td>
<td>Simulation</td>
<td>IoT application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[125]</td>
<td>✓ ✓</td>
<td>(DRL)DQN</td>
<td>Simulation</td>
<td>Generic user applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[126]</td>
<td>✓ ✓</td>
<td>(MARL)DDPG+LSTM</td>
<td>Simulation</td>
<td>Payment application</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
were compared in optimizing offloading decisions by [111].
DDPG outperforms DQN in terms of convergence time and
performance in minimizing system latency. Li et al. [112]
integrated a Lyapunov optimization with DDPG to achieve
a long-term objective in online offloading.

Other than vertical offloading from UE devices to edge,
He et al. [113] also considered horizontal offloading between
UE devices. They applied QoE in determining offloading
policy, and defined task priority assignment, redundant task
elimination, and defined task scheduling to achieve optimum
QoE. Since the offloading decision involved continuous action
space, the DDPG-based method of DRL was used.

The edge and cloud federation was considered in [114]–
[116]. Sun et al. [114] proposed a machine learning model
that cooperatively trained the two-tier edge-cloud architecture.
The Industrial Internet of Things (IIoT) devices were used to
determine whether to offload their tasks to an edge or cloud,
depending on which would satisfy the tasks in terms of latency.
This is categorized as upward vertical offloading. If both edge
and cloud could not meet the latency requirements, the IIoT
device processed the task locally. Hou et al. [115] applied
Cybertwin to coordinate resources between end-edge-cloud.
Cybertwin functions as an intelligent agent that makes the
offloading decisions necessary to accomplish the objectives
of maximizing processing efficiency and task completion rate.
They classified IoT applications into either delay-sensitive or
delay-tolerant. To maximize processing efficiency, a joint
optimization of hierarchical task offloading and resource allo-
cation based on MADDPG was proposed. The offloading agent
was trained in a federated fashion. These approaches share
only a trained model during the training process, avoiding the
sharing of local data, which could jeopardize privacy. Zhang et
al. [116] discussed downward vertical offloading, which was
carried out by multi-cloud systems to edge servers or mobile
devices. Multiple clouds compete with each other to access
network and MEC resources. A distributed offloading problem
arises in a system with no centralized control, such as a multi-
cloud system. They also proposed multi-agent Q-learning to
determine the optimum offloading policy, which minimizes the
system latency.

2) Device-Fog-Cloud Offloading. Devices-to-fog offloading
was discussed in [117]–[119]. Saguil and Azim [117] con-
sidered worst-case execution time in determining offloading
policy to fog nodes. Their objective was to minimize the
execution time of time-consuming ML tasks generated by
an embedded system. Li et al. [118] considered time-varying
task characteristics and fog node capability in determining
the offloading policy of a DQN-based algorithm. Alelaiwi et al.
[120] also considered a fog and cloud federation, particularly
horizontal offloading between fogs. DL was used to predict
the response times at multi-tier fog, edge, and cloud, which
were task-offloading destinations. They applied Deep Belief
Network (DBN) and logistic regression layer, which accepted
processing, memory and link capacity as inputs. Ren et al.
[119] used MADRL-based DQN to determine the best fog
access point (F-AP) to serve as an IIoT node request. Because
of the capacity constraints of the F-AP, some IIoT device
requests have to be offloaded to the cloud, a decision made

![DRL-based offloading in the fog-edge-cloud federation](image)

Fig. 8: DRL-based offloading in the fog-edge-cloud federation

using a low-complexity greedy algorithm.

3) V2X Offloading. A federation which included a V2X
system were considered in [121]–[126]. The papers [121]–
[123] optimized vertical offloading from vehicles to edge
servers. Online and offline learning were used by Fan et
al. [121], to maximize user and access network throughput.
Pareto optimization mapped the vehicles and access points,
and the optimal results were used to construct a data set for
DNN model training. An online stage used the output of the
trained DNN model, to predict the optimal association between
vehicles and access points.

Ning et al. [122] optimized offloading decisions and re-
source allocations jointly in a vehicular edge system, with
the objective of maximizing QoE. DQN-based offloading task
scheduling, which also considers user mobility, was proposed.
Sonmez et al. [123] proposed an ML-based task orchestrator
for vehicular edge systems, including LAN, MAN, and WAN
networks. An ML-based task orchestrator guarantees a task
being served successfully (in time) and in the lowest service
time, and Xie et al. [124] considered not only vertical offload-
ing between vehicles and edge, but also considered horizontal
offloading between vehicles. Vehicles, which have tasks to
offload, learned the environment with the multi-armed bandit
(MAB) method to determine offloading policy, which resulted
in lower average latency than the Greedy algorithm.

The papers [125], [126] considered a fog and cloud federa-
tion to accommodate offloading tasks from vehicles. Khayyat
et al. [125] used deep-Q learning, which has multiple DNN
that can work in parallel to obtain the optimal offloading
decision. In their environment, five DNNs would outperform
a single DNN. Gao et al. [126] addressed the task dependency
offloading problem by using multi-agent reinforcement
learning. Their objective was minimizing energy and latency of the
offloading task. LSTM was integrated into an RL to alleviate
an incomplete environment’s state.

A summary of ML-based offloading literature is shown
in Table VI. The comparisons are classified based on the offload-
The objective of heuristic searching is to minimize energy and latency. Heuristic searching is used to achieve good policies through adversarial generated models. GAIL aims to learn the expert’s distribution of states and actions. GAIL can only be collected in a few samples, so GAIL was used to model the expert’s distribution of states and actions. GAIL aims to learn good policies through adversarial generated training. Zeng et al. [132] considered the trustworthiness of devices. Offloaded traffic to untrusted devices leads to task failure. The designed offloading scheme evaluated the trust level of devices using a heuristic trust evaluation algorithm. The agents then mimicked the heuristic to evaluate the trust of the device, and for offloading action, the agents with the A3C algorithm mimicked the heuristic algorithm.

C. Imitation Learning: Online ML Imitating Offline Traditional Optimization

Supervised and unsupervised machine learning-based offloading rely on labeled, and unlabeled datasets, respectively. Offloading actions can be used as data labels in a federated system offloading. Given the system information, an expert or conventional optimization algorithm can be used to determine the optimal offloading action (label), which leads to an objective such as minimizing latency experienced by the arrival traffic. In RL, the agent derives the label through interaction with the environment. RL approaches initially rely on random actions. This slows the RL’s convergence to the optimal value.

Some of the researchers create labeled datasets by combining heuristic and conventional optimization, shown in Table VII. This dataset is then used to train the ML/RL model offline, before applying it to the real system. This method is called imitation learning, in which the agent model imitates an expert’s execution of a task to rapidly converge on the optimal solution. In addition to accelerating the convergence time, imitation learning can be used to reduce the training cost by creating an artificial dataset, labeled by conventional optimization, and applying it to a live testbed.

VEC networks are distributed systems characterized by expansive state and action spaces. Applying RL to optimize the offloading decisions in such large and dynamic systems, results in a slow convergence time, due to RL’s reliance on random actions at the beginning. Wang et al. [127] utilized imitation learning, which trains the DL network with the dataset created using branch-and-bound, to achieve the optimal result (offline), and then applies the model to the running system (online). Through this mechanism, the online model converges more quickly than the model without pre-training. Imitation learning can be used to overcome the difficulties brought on by RL exploration, when deciding to offload actions with vast input and action spaces. Nei et al. [128] observed that the initial 1000 iterations of DRL performance were poor. The DRL model was pre-trained using a heuristic search-obtained dataset as an expert, in order to achieve good performance at system startup.

Initially, Ning et al. [129] optimized offloading and caching decisions with the branch-and-bound (B&B) algorithm. Due to the complexity of the B&B algorithm, they minimized the B&B decision tree by labeling each node prune or preserve. The nodes that do not lead to the optimal solution will be pruned, while those that do, will be maintained. For a number of iterations with various states, pruning behaviors were recorded and used to train a binary classifier (SVM). This classifier was utilized to optimize (prune or maintain) the B&B branches to determine the optimal offloading and caching decisions.

Yu et al. in [130] solved offloading optimization offline using several traditional optimization problems such as exhaustive search, CPLEX, and optimal approximation algorithms. The state and optimal offloading action were recorded and used to train a neural network (NN) that was set up to perform multi-class classification. As a result, the trained model used in the online system had the lowest offloading cost and execution time, compared to other ML- and RL-based offloading approaches. The authors did not investigate the effect of the dynamic environment on offloading performance in this study.

Wang et al. [131] optimized the decision for decentralized offloading for pervasive edge computing. In offline learning, multiple agents use global system information to obtain optimal offloading decisions using natural gradient policies, such as ACKTR. Experts train policy and value networks, which are then used for online training. Because expert demonstrations can only be collected in a few samples, GAIL was used to model the expert’s distribution of states and actions. GAIL aims to learn good policies through adversarial generated training. Zeng et al. [132] considered the trustworthiness of devices. Offloaded traffic to untrusted devices leads to task failure. The designed offloading scheme evaluated the trust level of devices using a heuristic trust evaluation algorithm. The agents then mimicked the heuristic to evaluate the trust of the device, and for offloading action, the agents with the A3C algorithm mimicked the heuristic algorithm.

D. Traditional Optimization vs. Machine Learning

Three reasons why machine learning is required for offloading federated MEC systems are summarized in Table VIII. First, a control plane module must make an immediate choice about offloading. Traditional optimization, with its high computational complexity and exhaustive searching, is not capable of meeting a control plane’s latency requirement.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Target network</th>
<th>Objective</th>
<th>ML-based</th>
<th>Heuristic/optimization</th>
<th>Reason of heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[127]</td>
<td>V ⊥ E</td>
<td>Minimize energy and latency</td>
<td>DL</td>
<td>Branch and bound</td>
<td>Minimize convergence time</td>
</tr>
<tr>
<td>[128]</td>
<td>V ↔ V</td>
<td>Minimize average age critical information</td>
<td>DRL</td>
<td>Heuristic searching</td>
<td>Minimize convergence time</td>
</tr>
<tr>
<td>[129]</td>
<td>V ⊥ R</td>
<td>Minimize latency</td>
<td>ML</td>
<td>Branch and bound</td>
<td>Minimize training samples</td>
</tr>
<tr>
<td>[130]</td>
<td>D ⊥ E</td>
<td>Minimize tasks execution time</td>
<td>DL</td>
<td>Exhaustive search</td>
<td>Minimize offloading cost</td>
</tr>
<tr>
<td>[131]</td>
<td>D ↔ D</td>
<td>Minimize task completion time</td>
<td>DL</td>
<td>ACKTR</td>
<td>Minimize convergence time</td>
</tr>
<tr>
<td>[132]</td>
<td>D ⊥ E</td>
<td>Minimize task completion time</td>
<td>A3C</td>
<td>Heuristic</td>
<td>Minimize convergence time</td>
</tr>
</tbody>
</table>
Second, monitoring dynamic MEC environments is not trivial and can introduce unknown information into the control plane module, that is responsible for determining offloading policy. Third, modelling a heterogeneous MEC system precisely is challenging. Some researchers carried out traditional optimization in federated offloading using a system snapshot.

There are some studies [110], [112], [115], [133] that employed ML to achieve fast offloading decisions in a complex federated system. These offloading decisions and resource allocations were modelled as mixed-integer nonlinear programming (MINLP), that would take a long time to solve by conventional optimization. Yang et al. [110] used DL approaches that solved the MINLP problem in near-real-time. DL also outperforms a conventional branch-and-bound algorithm, in terms of system costs. A mobile device in a MEC system should take an online offloading decision in a complex and dynamic system which makes relaxation-based and local-search-based approaches to rerun in every change to the environment. These traditional optimization algorithms carry out exhaustive searching, which is not suitable for online decisions. Zhang et al. [133] extended a heuristic algorithm to the DQN, resulting in a fast-convergence algorithm suitable for real-time application offloading. However, Fan et al. [112] proposed a Lyapunov-aided DRL framework to determine the offloading policy in near-real-time with a near-optimum result compared to the exhaustive searching approaches.

Offloading in dynamic federated systems with unknown information was considered by proposing ML-based approaches in the papers [105], [109], [121], [122], [126]. Fan et al. [121] extended an SDN-controller with DL to learn a dynamic V2X system and carried out optimum offloading. This approach outperformed conventional traffic offloading (CTO), which uses heuristic algorithms, in terms of network throughput. Gao et al. [126] modelled offloading problem of V2X systems into Multi-Armed Bandit (MAB) and solved it by Probability-Based V2X Communication (PBVC), and adaptive learning-based task offloading (ALTO). Guo et al. [109] proposed DRL to address dynamic MEC systems for IoT. The current optimization techniques only take a snapshot of a system and cannot address the dynamic environment. In their previous work, Ale et al. [134] predicted traffic conditions and updated the cache by using DL. However, DL needs a large, labeled dataset to train models.

Channel conditions, available communication, and computation resources change dynamically over time. Such changes may render some information unknown to the IIoT agents, which determines the offloading policy. Guo et al. [109] used a multi-agent DDPG approach to tackle an offloading problem with some unknown or incomplete information. To ensure that a conventional algorithm, such as Greedy, works in this scenario, assumptions such as requiring agents to be aware of the channel and resource conditions in real-time were made. In terms of the success rate in utilizing available channels, the results showed that MADDPG outperforms the Greedy algorithm. Zhaolong et al. [122] addressed offloading and resource allocation problems by using a DRL approach. The proposed DRL approach had higher system utilities than a Greedy algorithm and a little lower than the Brute-force. However, Brute-force carried out exhaustive searching, which is not suitable for a control plane.

A heterogeneous federated system is difficult to model precisely, which makes the traditional offloading optimization difficult to implement. The papers [113], [123] used ML to carry out offloading in such a heterogeneous system. In Sonmez et al. [123], the ML-based approach outperformed the Game-theory-based optimization in terms of the success of tasks. Quality of experience (QoE)-aware task offloading in a Mobile Edge Network (MEN), which has heterogeneous computation and communication resources, is difficult to model for conventional optimization. He et al. [113], therefore, proposed Double DDPG with which, its learning agents could automatically update its model according to its experiences in interacting with the environment. This proposed method outperformed Greedy in terms of latency.

The references in Table VIII do not specifically compare the traditional optimizations with the ML-based approaches. Most of them used model-free reinforcement learning approaches, such as DQN and DDPG, because these can directly adopt a

<table>
<thead>
<tr>
<th>References</th>
<th>ML Approach</th>
<th>Traditional Optimization</th>
<th>Reason of using ML</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>[110]</td>
<td>DL</td>
<td>Branch and bound</td>
<td>Computation complexity</td>
<td>ML-based offloading has lower latency</td>
</tr>
<tr>
<td>[133]</td>
<td>DRL</td>
<td>Greedy and Heuristic</td>
<td></td>
<td>ML-based offloading has the lowest convergence time with better latency and energy usage</td>
</tr>
<tr>
<td>[112]</td>
<td>DRL</td>
<td>Relaxation-based, and local-search-based approaches</td>
<td></td>
<td>ML-based offloading has the lowest convergence time with near-optimum result in terms of computation rate</td>
</tr>
<tr>
<td>[121]</td>
<td>DL</td>
<td>Heuristic CTO</td>
<td>Unknown information in dynamic Environment</td>
<td>ML-based offloading higher throughput</td>
</tr>
<tr>
<td>[126]</td>
<td>MAB</td>
<td>Greedy</td>
<td></td>
<td>ML-based offloading has lower latency</td>
</tr>
<tr>
<td>[105]</td>
<td>DRL</td>
<td>Greedy</td>
<td></td>
<td>ML-based offloading has more completed task</td>
</tr>
<tr>
<td>[109]</td>
<td>MADDPG</td>
<td>Greedy</td>
<td></td>
<td>ML-based has lower latency and has higher channel access success rate</td>
</tr>
<tr>
<td>[122]</td>
<td>DQN</td>
<td>Greedy, Brute force</td>
<td></td>
<td>ML-based offloading has higher system utilities than greedy and a little bit lower than Brute force</td>
</tr>
<tr>
<td>[123]</td>
<td>ML, MAB</td>
<td>Game theory optimization</td>
<td>Heterogenous environment</td>
<td>ML-based offloading has fewer failed task</td>
</tr>
<tr>
<td>[113]</td>
<td>Double DDPG</td>
<td>Greedy</td>
<td></td>
<td>ML-based offloading has lower latency</td>
</tr>
</tbody>
</table>
model from the environment and do not need to provide the environment’s model to the learning agent. The Greedy algorithm is the preferred traditional algorithm because, with incomplete information from the environment, Greedy can still converge, although it may become stuck in local optima/minima. ML-based approaches can converge faster than traditional optimization with near-optimum results. Nonetheless, traditional optimization, which applies an extensive procedure, can ensure that the global optimum is found because it visits all feasible solutions. In contrast, machine learning is dependent on the data utilized in the training phase.

V. Lessons Learned

We categorize the lessons learned from this survey on the approaches that were used in the survey, such as traditional optimization and machine learning.

A. Traditional Optimization-Based Offloading

Some understanding comes from the survey of papers on traditional optimization-based offloading which explore the basic idea of carrying out offloading in a cloud-edge-fog system.

1) Traffic offloading is a short-term solution to the dynamic arrival traffic rate, while capacity allocation is a long-term solution. Traffic or task offloading in a MEC system is a part of control plane problems, for which a quick decision must be taken in response to traffic fluctuations. The control plane determines the offloading policy upon the arrival of traffic or a task, leading to an objective such as minimizing latency. The control plane reacts to incoming traffic within seconds. On the other hand, the management plane forecasts future traffic or task arrival rates based on historical data. The system’s capacity is then scaled to accommodate the predicted offloaded traffic. By integrating the control and management plane modules, it is possible to meet the arrival traffic or task’s latency requirements while allocating the fewest possible resources.

2) There are two offloading decisions to be made—where to offload, and how much to offload. An offloading decision could be a binary decision, which is a decision to offload or not, or a ratio-based offloading decision, which determines how much and where to offload tasks or traffic. Binary offloading is usually carried out by UEs, as UEs lack complete knowledge of external system resources. Each UE measures its capacity to compute a task locally or to offload to external resources. Ratio-based offloading is carried out by network devices controlled by an orchestrator, which has global information to determine where and how much to offload. To reduce computation latency, this offloading is performed from an overloaded federation entity to an idle federation entity. The ratio-based offloading is better suited for the network control plane than per-packet/task offloading, as the latter requires large control plane computations due to millions of network packets.

3) Hierarchical offloading—application offloading by UE and traffic/task offloading by the federation network control plane. UEs carry out offloading to extend their computation capacity and extend their battery life since UEs are equipped with limited computation and battery capacity. UEs sense environmental conditions such as signal strength, battery level, and resource utilization (local information) to determine where a task is to be executed. Network devices offload their task to another network device with the least load to avoid overloading and minimize latency. These network devices could be a router, traffic dispatchers, MEC servers, or fog servers with a data plane function. Network device offloading is determined by an orchestrator which has access to global system information.

4) Infrastructure capacity expands UE capacity, while non-infrastructure capacity extends UE connections. Traffic or tasks can be offloaded to infrastructure and/or non-infrastructure. Infrastructure comprises all entities that belong to providers or organizations, such as base stations, MEC servers, fog nodes, and the cloud. Such infrastructure entities are used to extend a mobile device and UEs’ capacity. Some areas may, however, not be covered by infrastructure entities. UEs offload traffic or tasks to another UE or mobile device (non-infrastructure) in such an area, called an opportunistic network. A UE or mobile device can share its computing capacity as a server or share its communication capability as a relay to infrastructure in an opportunistic network.

5) Horizontal offloading boosts east-west traffic while keeping traffic at the lower tier with low-latency services. Vertical offloading, on the other hand, minimizes capacity allocation and simplifies management by centralizing the upper tier. A federation of cloud-edge-fog is a hierarchical system in which the cloud is at the top, the edge is in the middle, and the fog is at the bottom. Furthermore, each tier may include some providers. This system has two offloading directions, which can be bottom-up (vertical) or east-west (horizontal).

Vertical offloading occurs between the customer and provider or between tiers within a provider, such as offloading in two-tier MEC architecture. In customer-provider (upward) vertical offloading, a customer will be charged for each resource used, and minimizing costs will be of concern in such cases. On the other hand, an upper-tier provider may offload a service to a lower-tier provider in order to meet the required latency (downward offloading). The upper tier provider pays an incentive to a lower tier provider for every served task. The upper-tier provider minimizes offloading costs while maintaining the required latency. Vertical offloading can also be carried out within a provider and is typically used to move tasks from a lower, more dispersed, tier to a higher, more centralized, tier with larger capacity. The upper tier provides greater coverage and shares its capacity with a couple of bottom-tier sites, in order to handle high arrival traffic or task rates that would overwhelm some bottom-tier sites.

Horizontal offloading is carried out to distribute traffic or tasks to the same tier first, rather than offloading them to a higher tier. Keeping tasks on the bottom tier, such as fog or edge, might help reduce communication latency due to their proximity to the UE or MD. While horizontal offloading keeps traffic on the bottom tier, the decision to offload must be made on the neighboring side, otherwise, the traffic will encounter prolonged communication latency, and a trade-off between minimizing computing and communication latency.
takes place.

6) Service chaining offloading is unavoidable, given that modern services are composed of multiple microservices. In task offloading, a task can be a single task or consists of several sub-tasks. Full offloading is carried out for a single task which can be processed at the local device (UE) or remotely (cloud, edge, or fog). While in partial offloading, some sub-tasks can be processed locally, and others are offloaded to a remote server. Sub-task synchronization may be needed if some sub-tasks depend on other sub-task outputs.

7) Different providers have different offloading objectives, such as minimizing cost or delay, and they might compete with each other. Optimizing offloading decisions in a federated system with some providers is challenging because each provider has its own objective. A collaborative offloading approach by adopting game theory and analyzing Nash equilibrium is carried out. In the Nash equilibrium, the offloading strategy of each provider is optimal when taking another providers’ decisions into account. Each provider ends up winning since everyone gets the result they expect.

B. Machine Learning-Based Offloading

Some insights were gained from a survey of ML-based offloading in a cloud-edge-fog federation. An offloading decision in a cloud-edge-fog federation is made by the control plane and applied in the data plane of the networking devices. This control plane decision must be carried out quickly (fast response time). Making a quick decision in a federated system with high complexity is very challenging. Traditional optimization, which uses exhaustive searching, may violate the latency requirements of a control plane decision. ML-based offloading is a promising method that automatically maps a given system settings to arrive at the best offloading decision.

1) Offline and online learning of ML-based offloading optimization approaches. Unsupervised-based ML is used to predict future system conditions by using earlier/older data. The label of the data is not essential in unsupervised learning. The predicted results are used by the control plane to make the best offloading decisions. Unlike unsupervised learning, supervised learning trains the model directly, based on collected data that an expert has labelled. The label is an offloading decision that leads to an objective such as minimizing delay or costs for a given current system condition.

While supervised learning can derive an optimal solution, deriving a well-labelled dataset is not easy; extensive monitoring is required. Some monitoring mechanisms that carry out costly broadcast data are required to obtain information in distributed edge-fog systems. A label of data must be updated frequently in a dynamic system because an offloading decision must be carried out quickly.

Both unsupervised and supervised learning are categorized as offline learning because they learn from previous data, and not directly from the environment. Reinforcement learning is one of the ML-based approaches which learns directly from the environment, to determine the best action. In a cloud-edge-fog federation, an offloading decision is produced by a learning agent. In the beginning, the agent will perform poorly; it will remember previous successful actions taken in a given environment state condition and forget failure action. With such trial-and-error attempts, the agent will improve after several attempts.

2) Machine learning approaches are a panacea for optimizing offloading decisions with some missing information. Some providers in a federation may hide some information from others. However, traditional optimization requires all information to calculate optimal offloading. Some researchers made assumptions about such hidden information because the calculation could otherwise not have been carried out. By contrast, ML-based offloading will map any given input that may also be incomplete to determine the best offloading decision. This learning process can be carried out with incomplete information. However, in order to converge, ML relies on datasets obtained from the environment. Experts are needed to label the data so that it can converge on an optimal solution. The RL algorithm begins with trial and error and has a slow convergence time.

3) Retraining is more efficient than recalculation for control plane problems, such as offloading optimization. In terms of the cost to obtain an optimal decision, traditional optimization recalculates the decision for each new input before deriving a new offloading decision. The ML-based solution will retrain the model to obtain an optimal decision. Without the retraining process, the model could still come up with a sub-optimal result. ML-based approaches can thus decide quickly, without waiting for the training process to be completed; and ML models can also be reused and transferred.

4) Federated systems are a type of multi-agent environment. A learning agent, a control plane module, can be a single agent placed in a central location or a multi-agent distributed over some areas. A single agent determines the offloading decisions for all devices in a federation. An agent’s model is trained by a centralized dataset. The size of the federation system will affect the dataset’s dimensions and raise a scalability problem because of the very large dimensions of the observations and decisions. Single-agent learning is unrealistic because a federation consists of many providers who have different offloading policies. Multi-agent learning is suitable in a federated system for two reasons. The first reason is scalability. A provider may have agents in some areas which produce offloading decisions based on local observations. Having multi-agents, which calculate offloading decisions in parallel, can reduce convergence times. The second reason is the federation itself because the federation is a kind of multi-agent environment where each agent belongs to each provider.

5) Federated learning sharing model weights is a promising approach to obtain global optimum offloading decisions with low communication cost. A cloud-edge-fog federation may consist of millions of devices scattered over a large area. A learning agent that produces offloading decisions can be trained by a centralized dataset with global information, or trained by a local dataset consisting of local information; being trained by local dataset results in a local optimum offloading decision. Although, using a centralized dataset with global information results in a globally optimum offloading decision, but generates extensive communication costs.
C. Imitation Learning: Online ML Imitating Offline Traditional Optimization

1) Traditional optimization as an expert on the ML model. A well-labeled dataset is required for machine learning to reach the best decision. Given environmental information, a human expert could not directly create the label for offloading actions. To properly label the data, they require formulas or experiments. By using simulations and predefined scenarios, traditional optimization could be utilized to determine the optimal offloading decision (label). The collected data from this simulation could then be used to train the ML model prior to its application to the live system. Using this method, the ML model mimics the performance of conventional optimization. Even though the traditional optimization, during the creation of datasets, could not account for every possible scenario, RL algorithms can retrain the model using direct feedback and interaction data from the environment. In contrast, supervised learning requires additional offline training to cover new scenarios.

2) Traditional optimization accelerates machine learning convergence, and vice versa. ML has a faster decision time than traditional optimization because, unlike traditional optimization, it does not wait for convergence before deciding on an action. The convergence of these RL algorithms, which rely on random action and direct environment feedback, requires numerous iterations. Using conventional optimization to generate a preliminary dataset for training a machine learning (ML) model, could accelerate the ML model’s convergence, as the offloading action obtained by conventional optimization could serve as initial labels for the optimal solution.

3) Offline learning could minimize the training cost of online learning. Applying an untrained model to a real-world system will require numerous iterations to arrive at the optimal solution. Users of the system will experience significant performance degradation. Model transferability is one of the benefits of offloading based on machine learning. Before applying to the existing system, the model could be trained offline with various potential scenarios. Consequently, when the model is applied to the system, users may still experience optimal or suboptimal offloading decisions.

D. Traditional vs. Machine Learning Based Offloading

1) While traditional optimization determines the optimum offloading action based on a snapshot of the system, machine learning-based offloading would make use of continuous system information. Because a system snapshot may not accurately represent future system behavior, offloading becomes obsolete in traditional optimization. ML-based techniques, particularly RL-based offtload, can be trained on batches of collected data without waiting for complete or a large amount of collected data from the environment.

2) Traditional optimization techniques are suitable for management plane problems, while machine learning-based approaches are best suited for control plane problems. A management plane, which is in control of resource allocation in a federated system, makes decisions in minutes or hours, and so generates long-term solutions for hot-spot traffic. A management plane problem is well-suited to traditional optimization, which has a long decision time. While the short-term solution to hot-spot traffic is offloading, which is part of the control plane problem. Offloading using machine learning has a short decision time, since it produces a sub-optimal solution after only a few training processes, making it appropriate for the control plane problem.

3) Transitioning from traditional optimization to machine learning-based approaches can minimize the assumptions of unknown information in modeling federated systems. Channel, network, and server settings are difficult for a learning agent to obtain completely. Certain pieces of information, such as the relationship between tasks and the processing capacity required to do those tasks, may be unknown to the learning agent. To deal with unknown information in traditional optimization, some researchers make assumptions such as the computing and networking capacity is homogeneous and able to collect system information completely. Because machine learning-based techniques can map any input to a desired output, they can be utilized to minimize the assumptions of unknown information.

VI. RESEARCH OPPORTUNITIES AND CHALLENGES

A. Research Opportunities

1) Fog-Fog Federation. The development of fog computing, creates several benefits for application developers, applications, and different industries by distributing functions [135]. A fog-fog federation helps to monitor, process, analyze, react, and distribute computation, communication, storage, control, and decision-making closer to the users. However, such a federation also results in challenges for individual fogs. When fogs are closer to each other than to edge or cloud, the federation between fogs allows them to enhance their data aggregation, processing, and storage capabilities, and requires cooperation between these fogs to ensure the proper coordination for the necessary interactions. However, because the fog is a collection of mobile devices, it poses a significant difficulty regarding resource management, capacity discovery, authentication, etc.

2) V2X. In the past few years, Internet usage has continued to increase with the development of advanced technologies. The gradual increase of smart vehicle applications has produced computation-intensive tasks for vehicles, and thus, the internet of vehicles (IoV) improves traffic conditions [136]. However, these vehicles are independently unable to meet the demands of their limited computing resources. Vehicle-to-everything (V2X) communication is an emerging technology, that supports vehicles to offload their tasks across vehicles [137]. With vehicle-to-infrastructure communication (V2I), a vehicle can offload to infrastructures such as RSU, edge, or cloud, and with vehicle-to-vehicle communication (V2V), one vehicle can offload its tasks to other vehicles. As an alternative, this technology also facilitates multiple vehicles forming a fog by sharing their resources, popularly known as a vehicular-fog, to provide services to others. Furthermore, the dynamic
3) Mobility of a Vehicular-Fog. Intelligent transport systems (ITS) [138] exchange information for safe V2V and V2I communication. In a V2V environment, vehicles communicate directly with each other, and with services that support safe driving and provide information. However, in a dynamic environment, some applications of the moving nodes require high computing power, and the computational resources of each vehicle may not be able to meet such a requirement. We address this issue of V2I communication, by using MECs or RSUs that are closer to the vehicles. However, in a certain dynamic environment, vehicles can move out of communication range during task offloading. In such a case, either the task cannot be offloaded to the infrastructure, or, if offloaded, the vehicle cannot receive the results. In an environment where vehicles are on the move, the rate of movement of vehicles is usually fast, and the change of the topology is intense, and it is here, where the V2V task offloading is a matter that needs to be investigated. Recent vehicular-fog research has focused mostly on the static vehicle scenario [63], whereas in a vehicular-fog set-up, vehicles are mostly managed by a fog manager like RSUs, which is part of the infrastructure. In a dynamic vehicular-fog with mobility, management of the federation is a matter of concern.

4) Scaling. Auto-scaling [139] can be classified into different categories. First, manual scaling, where we specify only the maximum, minimum, or desired capacity changes to auto-scaling groups, and auto-scaling maintains the instances with updated capacity. Second, scaling is based on a schedule, where one can scale an application ahead of known load changes. For example, on some particular day, in peak loads or on a limited offer, one can scale an application based on scheduled scaling in such cases. Third, for scaling based on demand or dynamic or reactive scaling, resources are adjusted in real-time based on the number of incoming requests. Finally, predictive scaling predicts future arrival rates by learning past arrival traffic information, and the learning outcomes are then used to make scaling decisions. In a federated system, service providers can scale resources up or down by adopting different scaling methods. It is also essential for a service provider to make decisions based on different performance metrics, whether to scale up its resources to accommodate more incoming requests or offload the request to others. So, it remains challenging to decide when the resources need to be scaled up to avoid offloading and when to be encouraged to offload to avoid scaling [140]. Scaling in a container/virtual machine-based system could be fine-grained, such as increasing the capacity of the VM (vertical scaling) or adding more containers/VM to the available servers (horizontal scaling). The offloading controller must account for the time required for scaling to avoid packet loss due to incomplete computing resources scaling.

5) Centralized vs. Distributed Federation. Various factors affect a federation, such as the services available from service providers, the type of services, their capacity and capabilities, their geographical location, number of customers, type of customers, etc. A federation between the service provider can be centralized or distributed. A centralized federation has a single federation manager between multiple federated entities, and that manager manages the federation. In such cases, there is a joint federation agreement between all the entities, based on which offloading decisions will be taken [141]. In a distributed scenario, a federation is formed between two individual entities or a group of entities of a system. In such cases, a separate agreement is made based on what communications take place. Following the establishment of the agreement, decisions on service discovery, authentication, and service migration can be made centrally or distributively.

6) Resource Allocation. When the number of resources in a system is large, some may remain underutilized. If the number of resources is small, offloading may be triggered too often. Hence, one of the key challenges of offloading is to determine the right amount of resources required at the location where the tasks will be executed, otherwise, after offloading, if there is any shortage of resources, tasks will be offloaded further away [142]. This may increase the communication latency, and an increase in the number of offloading hops may also trigger a breach of data privacy [143]. Again, most applications and services in the system that require intensive computation and high processing are incompatible with devices because of their limited resources.

7) Energy Consumption. Although task offloading is largely inevitable in a federated system, it is still a highly energy-consuming process. One of the challenges is to estimate the energy consumed in communication activities of task offloading to make task offloading efficient [144]. As a result, it is sometimes a challenge whether to offload or not. An efficient energy estimation model would help to decide whether to perform task offloading, based on the energy cost of the communication activities.

8) Task Offloading in Different Application Scenarios. Task offloading can take place at different levels of different federated systems, depending on the type of service required, and based on different criteria. This section is an overview of different application scenarios where task offloading has recently played a key role. In ITS, automatic traffic monitoring and management systems [145], edges, and RSUs can assist drivers by providing traffic updates, emergency alerts, etc. In fog-to-fog offloading scenarios, one vehicle can assist another by caching data if required [136]. Emergency help alert mobile cloud (E-HAMC) can provide a quick way of notifying the relevant emergency authorities by utilizing the services of fog for offloading and pre-processing purposes [146]. When an alert message is sent, these services can automatically transmit the location of an incident and the emergency contact information.

New offloading schemes can improve privacy levels, reduce computation latency, and save the energy of healthcare IoT devices [147]. Exploiting fog and cloud computing paradigms in healthcare reduces the hospital’s capital expenditures for patient data processing and storage. The fog can be used for simple data analytics, whereas the cloud is used for big data analytics [148]. Edge Computing also offers intriguing possibilities for smart agriculture [150]. For example, sustainable water management is a common issue at the farm...
level. By offloading sensible data from the sensor devices to the edge server, appropriate action can be taken. Edge and fog computing enables intelligent surveillance with real-time video stream processing. Processing large amounts of data to the cloud will result in long communication latency, consume massive network resources, and jeopardize video privacy [149].

B. Research Challenges

1) Interoperability. Interoperability is closely related to both standards and lock-in [151]. Internet service providers use multiple networks so that the failure of a single provider will not disrupt communications entirely. Here we will focus on interoperability between cloud, edge, and fog providers. In a federated system, an application’s execution can be carried out with its components spread over different service providers. From an architectural perspective, appropriate signaling, data, and control interfaces are needed to ensure interoperability at the architectural level, or more precisely, to support an application’s life-cycle, the control interfaces are needed for interactions between the different domains. A provider should design control interfaces to allow other providers to federate and utilize their resources safely, without exposing the architecture’s details. Furthermore, a federation interface standard could be developed to support federation and make task offloading between providers easier and safer.

2) Service Level Agreement. Every user wants assurances that their service provider will remain reliable because service interruptions can cause significant financial harm. Service Level Agreements (SLAs) [152] are contractual agreements for certain levels of reliability, which would then be compensated in various ways, if there was any breach of the contract. The same kind of agreement is applicable between the service providers who federate with each other to provide services to their respective subscribers. There must be a contract, a federation agreement, to provide a certain level of reliability. Such provisions may include monetary compensation, if the level of service offered is below the contractually specified level.

3) Redundancy. Redundancy is crucial in numerous scenarios to ensure the system’s high availability. Obviously, redundancy is more critical at the cloud level as compared to edge and fog, as a cloud has multiple data centers, redundant networking, backup power, data backup plans, and other redundancy resources.

4) Fault Tolerance. In a federated system, offloading gradually becomes automated, and where heterogeneous entities are involved, the risks of failure increases. Some common examples are connectivity failure, use of faulty devices, communication delays, etc. Offloading processes must be robust, and capable of not only detecting, but also handling faults on time. The accuracy and timeliness of the fault detection algorithm to detect the faults are thus of significant importance.

5) Security. In a federated architecture, multiple systems communicate with each other, and when offloading occurs, there is a risk of data theft and misuse. The misuse of data can be a serious threat to security systems such as of the military, healthcare, etc. It may also compromise the privacy of individuals. Hence, efficient and robust data security measures would be required so that offloading decisions are precise, because security breaches are something that may not be publicly disclosed by service providers, unless compelled to do so by particular regulations [153].

Cloud, edge, and fog can be federated in several different ways in which a subscriber can move from using the services of one (cloud, edge, or fog) to using the services of another. There is a need for authentication when a subscriber moves between two entities that are federated, directly or indirectly via some hops in between. This leads to 3rd-party [154] and 4th-party authentication.

6) Geo-Diversity. The location and geographical diversity of a service provider might be of concern to some users. Compared to centralized cloud systems, widely spread geographical distribution of fogs and edges can be considered as one of the key enablers of the Internet of Things (IoT), and big data applications [155]. These offer low latency and location awareness due to the proximity of the computing devices. Controlling a distributed area’s forwarding devices with a centralized network control plane causes control traffic to travel a long distance, and has scalability and high availability issues. The use of multiple network control planes in edge computing should be investigated to reduce latency and achieve high availability.

7) Reliability. The reliability track record of a service provider is just as crucial as contractual guarantees. Big cloud providers are likely to have significantly better reliability than relatively small, self-maintained IT infrastructures, as they have massive computing capabilities. Edge and fog computing systems are closer to a user and improve user experience by providing low latency and highly efficient computing. When computationally intensive components are offloaded to edge servers or distributed to fog nodes, various constraints such as power limitations, limited computing resources, inevitable server failure, etc. come into play. In such a scenario, how is the reliability of offloaded computing [156] to be guaranteed? How then does one find an appropriate offloading point that can guarantee completing a task at a low cost, with minimal energy consumption for communication? What is achievable minimal latency for the completion of the task?

8) Performance. In federated systems, many customers may share common physical computer hardware and network infrastructure. However, sharing can also cause performance problems. As providers use statistical multiplexing, excessive levels of over-subscription may degrade services. Poor resource scheduling and poor management could also degrade performance, even if there is no over-subscription. If any service provider in a federated system misrepresents their available capacity or capability, it may cause performance degradation. A real-time benchmarking service for federated edge resources is required in this case.

VII. Conclusion

Network communication relies on the coexistence of a variety of architectures of different services. The coexistence
of such distinct architectures and complementary technologies opens up new issues in resource, latency and storage limitation, type of services, etc., which can emerge with federation architectures and cannot be addressed individually.

We have discussed various federation architectures for cloud, edge, and fog systems. These each have their own tier within a federation, as top, middle, and bottom tiers. Such a federation can be classified as vertical, horizontal, or hybrid. Horizontal federation is between providers of the same tier, vertical federation is a federation at different levels that can result in a 2-tier or 3-tier architecture, and a hybrid federation is a term used to describe the combination of vertical and horizontal federations.

We also give an overview of the various offloading techniques in such cloud-edge-fog federated systems and classify them according to the federation relationship and direction of offloading, i.e., horizontal offloading occurs in a horizontal federation, and so on. Most of the works we have considered have focused exclusively on vertical offloading within a federation. Horizontal offloading should also be considered, as there are multiple providers within the same federation tier that can provide resources.

We also reviewed the literature on various recently proposed offloading approaches, categorizing them as traditional optimization and machine learning-based approaches. In a federated system, the high dimension and dynamic input with unknown input parameters complicate the calculation of offloading decisions. Offloading decisions in the control plane module must be made quickly. The traditional optimization approach, which relies on exhaustive searching, may violate a control plane’s latency requirements. Machine learning-based approaches that map any input parameters (even with unknown ones) to a desired output, have emerged as a solution to the limitations of traditional optimization. The machine learning model can also be used to generate immediate offloading decisions without waiting for all training processes to be finalized, which would result in an optimal solution. Because reinforcement learning approaches can derive an offloading decision directly from the environment without requiring a well-labelled data set, they become the most preferred approach. Finally, we discussed some future research directions for such offloading scenarios and highlighted some key challenges associated with task offloading.

REFERENCES


Binayak Kar is an Assistant Professor of computer science and information engineering at National Taiwan University of Science and Technology (NTUST), Taiwan. He received his Ph.D. degree in computer science and information engineering from the National Central University (NCU), Taiwan in 2018. He was a post-doctoral research fellow in computer science with National Chiao Tung University (NCTU), Taiwan from 2018 to 2019. His research interests include network softwarization, cloud/edge/fog computing, optimization, queuing theory, machine learning, and network security.

Widhi Yahya received his M.S. degree in Computer Science, National Central University (NCU), Taiwan in 2014. He is a lecturer and a researcher in computer science at University of Brawijaya, Indonesia. He is currently pursuing Ph.D. degree at Electrical Engineering and Computer Science (EECS) International Graduate Program of National Yang Ming Chiao Tung (NYCU). His research interest are in network programming, software-defined networking, and multi-access edge computing (MEC) optimization.

Ying-Dar Lin is a Chair Professor of computer science at National National Yang Ming Chiao Tung University (NYCU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose during 2007–2008, CEO at Telecom Technology Center, Taiwan, during 2010-2011, and Vice President of National Applied Research Labs (NARLabs), Taiwan, during 2017-2018. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and automated tools, and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, later acquired by D-Link Corp, and O’Prueba Inc. in 2018. His research interests include network security, wireless communications, and network softwarization. His work on multi-hop cellular was the first along this line, and has been cited over 1000 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (2014–2017), ONF Research Associate, and received in 2017 Research Excellence Award and K. T. Li Breakthrough Award. He has served or is serving on the editorial boards of several IEEE journals and magazines, and was the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST), during 2016–2020. He published a textbook, Computer Networks: An Open Source Approach (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Asad Ali is currently a project assistant researcher with the Cybersecurity Center of Excellence, where he works on the research problems related to the Cyber Threat Intelligence. He received the B.S. degree in electrical engineering and the M.S. degree in electrical engineering (telecommunications & computer networks) from the University of Engineering and Technology (UET) and National University of Science & Technology (NUST), Pakistan, in 2012 and 2015 respectively. He received his Ph.D. degree in electrical engineering and computer sciences from the National Yang Ming Chiao Tung University (NYCU), Taiwan in 2022. His research interests include cyber threat intelligence, 4G/5G cellular networks, network security, network protocols, wireless communications, artificial intelligence wireless, network design, and optimization.