Resource Management in Mobile Edge Computing: A Comprehensive Survey

XIAOJIE ZHANG and SAPTARSHI DEBROY, City University of New York, USA

With the evolution of 5G and Internet of Things technologies, Mobile Edge Computing (MEC) has emerged as a major computing paradigm. Compared to cloud computing, MEC integrates network control, computing, and storage to customizable, fast, reliable, and secure distributed services that are closer to the user and data site. Although a popular research topic, MEC resource management comes in many forms due to its emerging nature and there exists little consensus in the community. In this survey, we present a comprehensive review of existing research problems and relevant solutions within MEC resource management. We first describe the major problems in MEC resource allocation when the user applications have diverse performance requirements. We discuss the unique challenges caused by the dynamic nature of the environments and use cases where MEC is adopted. We also explore and categorize existing solutions that address such challenges. We particularly explore traditional optimization based methods and deep learning based approaches. In addition, we take a deeper dive into the most popular applications and uses cases that adopt MEC paradigm and how MEC provides customized solutions for each use cases, in particular, video analytics applications. Finally, we outline the open research challenges and future directions. [1]

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Cloud computing**; **Heterogeneous (hybrid) systems**.

Additional Key Words and Phrases: Mobile edge computing, resource management, task offloading, machine learning, data-intensive applications.

1 INTRODUCTION

With the rapid development of data science, data-intensive smart applications such as, smart transportation, smart healthcare, AR/VR/MR, and real-time gaming are becoming increasingly popular. These applications often require massive data computations/processing and have strict low/ultra-low latency requirement. Nevertheless, deploying such applications on mobile devices is still a challenging problem, primarily due to their hardware limitations. Particularly: i) most mobile devices do not have powerful central processing units (CPU) and thus cannot host computation-intensive applications; ii) the battery capacity of mobile devices is greatly restricted by the small physical size, i.e., intensive computation drains the battery quickly, which limits computation-intensive application hosting; and iii) smart applications require considerable memory space and may cause device memory shortages. In particular, machine learning and artificial intelligence (ML/AI) based applications can easily occupy the entire memory making the mobile devices significantly slow. The most obvious solution to address such issues is the use of remote computation that can provide elastic and on-demand resources.

[1]This material is based upon work supported by the National Science Foundation under Award Number: CNS-1943338. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Authors' address: Xiaojie Zhang, xzhang6@gradcenter.cuny.edu; Saptarshi Debroy, saptarshi.debroy@hunter.cuny.edu, City University of New York, 695 park Avenue, New York, New York, USA, 10065.

0360-0300/2023/3-ART $15.00
https://doi.org/10.1145/3589639

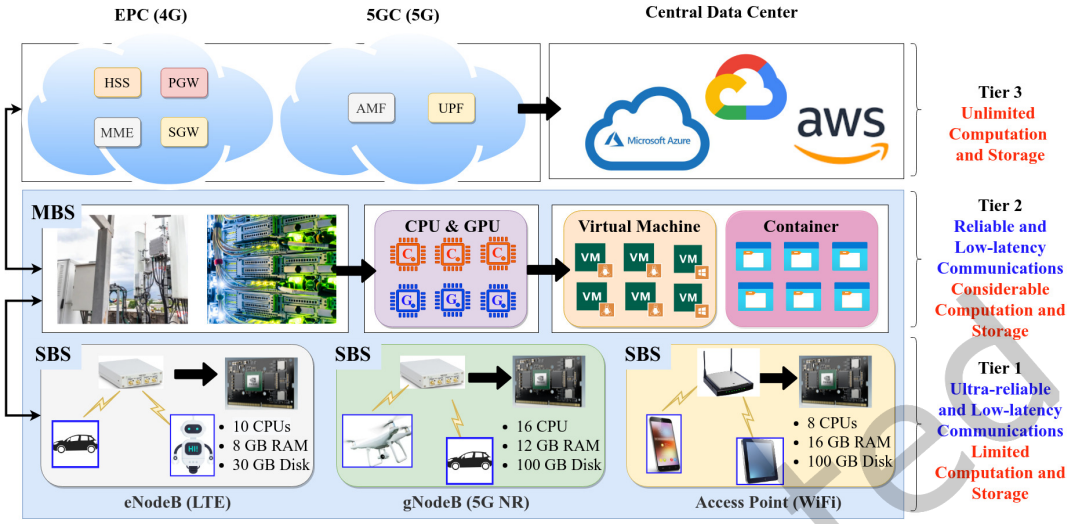


Fig. 1. A three tiered computing architecture and involved components.

1.1 Tiered Computing Architecture: Cloud, Edge and Device

A tiered computing architecture (as shown in Fig. 1) that spans across cloud data centers, edge servers, and local computation capabilities of mobile devices is the de-facto paradigm for such remote computation. The tiers are typically classified by reliability, latency, and computation capacity. Cloud resources in Tier 3 provides users with powerful computing resources (e.g., CPU, GPU) with a choice of different operating systems. It also provides pre-installed software and libraries (e.g., ML/AI, visual computing) that in turn enable mobile services to support more sophisticated and complex end-to-end applications (e.g., 3D reconstruction) with better user experience. However, due to the long network distance between the mobile user and the cloud data center (i.e., switching, routing, and congestion), the application suffers a higher end-to-end delay. This makes cloud computing unsuitable for many mission-critical mobile applications (e.g., AR/VR/MR, autonomous driving) that have strict low end-to-end latency requirements. In recent times, MEC has emerged as the alternative approach which is a combinations of the bottom two tiers in Fig. 1. The bottom tier consists of multiple heterogeneous small base stations (SBS), including eNodeB (LTE 4G), eNodeB (5G NR), and WiFi access points (AP) that together represent different wireless technologies. In addition, users can operate different types of mobile devices (e.g., smartphones, drones, and robots) with diverse computation and energy capacities. At the same time, computation capacity can be augmented by the use of small scale edge nodes with CPU/GPU capabilities such as, Jetson nano and TX2. This tier brings computation resources closer to the mobile devices, therefore is able to achieve ultra-reliable (network reliability $\geq$ 99.999%) and low-latency communications (usually wireless). However, the computation resources available at this tier are very limited due to the of limited computation capacity of the mobile and edge devices. This makes it difficult to host sophisticated applications entirely within this tier. Tier 2 consists of several Macro base stations (MBS) that typically host city-level data centers (e.g., COSMOS [1]) containing considerable computation and storage resources that are able to host multiple virtual machines (VMs) and containers simultaneously. In general, the MBS are connected to SBS via high-speed optical fiber and thus can provide reliable and low-latency communications.

1.1.1 Edge Computing vs Fog Computing. In recent times, fog computing as a concept has become popular. As both edge computing and fog computing bring computational resources closer to the mobile devices, there is often debate about their similarities and differences. The concept of fog computing is first defined in [2] which states that the characteristics of fog computing include *low latency and location awareness, wide-spread geographical distribution, mobility, very large number of nodes, predominant role of wireless access, strong presence of streaming and real time applications, and heterogeneity*. Although edge computing shares most of these characteristics, it is more resource-limited and closer to data generation site than fog computing. Another way to distinguish edge computing and fog computing is mentioned in [3, 4] which state that fog computing is designed to provide computing, networking, storage, and control services anywhere from the cloud to mobile devices while the edge computing utilizes resources only located at the edge of the network. The scope of this survey is resource management in edge computing, but the collection of papers discussed here is not limited to 'edge-only' computing as long as their proposed systems benefit from collaborative computing paradigms across edge, fog, and cloud.

1.2 Applications supported by MEC

In order to capture the challenges of resource management in MEC, it is necessary to understand the classification and characteristics of complex applications. According to the International Telecommunication Union (ITU), current and future 5G mobile applications are classified into three categories, viz., Enhanced Mobile Broadband (eMBB), Massive Machine Type Communications (mMTC), and Ultra-reliable and Low-latency Communications (uRLLC) [5]. As shown in Fig. 2, eMBB aims to support stable communication that have high bandwidth requirements (e.g., 4K video and Virtual Reality). Whereas, mMTC serves a massive number of IoT devices which are active intermittently (e.g., smart city with high connection density). Finally, uRLLC accommodates services with low-latency and high reliability requirements (e.g., connected autonomous vehicles). Therefore, given such diverse set of requirements (viz., latency, scalability, availability and reliability), a combination of customized scheduling and resource management strategies are needed to allow heterogeneous applications running on the same system stack (includes computation and network). To this end, the concept of *network slicing* or *virtual networks* is being used to allocate customized end-to-end resources to diverse applications with different requirements to ensure performance satisfaction and mutual isolation. Network slices are defined as end-to-end logical/virtual networks on top of a common shared physical network infrastructure. The design and composition of network slices are driven by the need to fulfill the latency, scalability, availability, and reliability requirements for vertical-specific applications, such as eMBB, uRLLC, or mMTC as shown in Fig. 2.

1.3 Challenges in MEC Resource Allocation

Given this tiered computing architecture and characteristics of complex applications, we can see that the optimization problems to achieve efficient resource allocation and application management is more complicated in MEC than traditional distributed and cloud systems due to the inherent heterogeneity of MEC resources. Specifically the challenges are:

- As explained before, the diversity of applications and a variety of user specified requirements add additional complications to the system and application optimization problems in terms of resource allocation decision making. This makes things especially complicated for joint optimization problems where resource provisioning and placement problems involve both computation and network/radio resources. Since most of the joint optimization problems are NP-hard, conventional approaches are unable to solve such problems efficiently.
- MEC environments are more dynamic and prone to faults/fluctuations than traditional cloud environments. These fluctuations include and are not limited to: 1) unpredictable changes in user requirements and mobility, 2) dynamic network connecting devices and edge servers with fluctuations in wireless channel quality, 3) edge servers with fluctuating computation resource availability, and 4) unpredictable changes in device energy

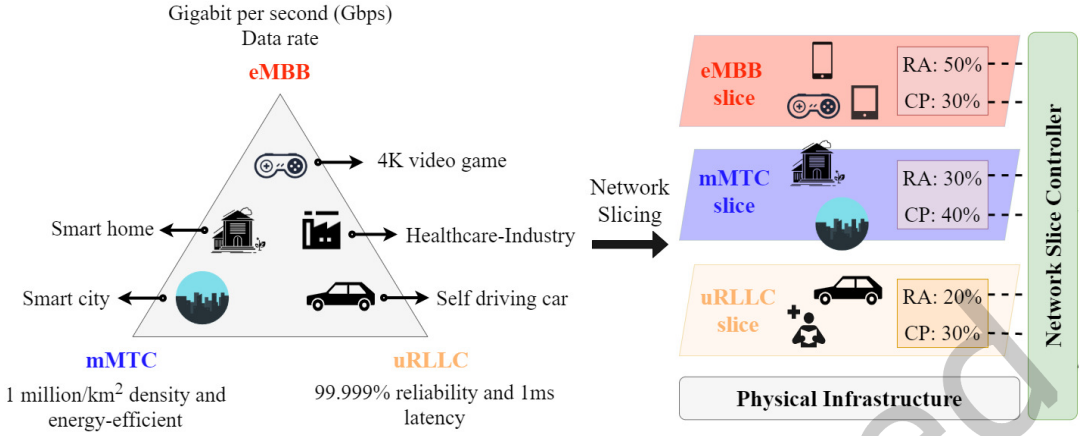


Fig. 2. Applications and requirements of 5G Wireless Network and network slicing [6].

levels. Inability to handle such faults/fluctuations efficiently and promptly may cause serious performance degradation of mission-critical applications. At the same time, continuous handling of the fluctuations though efficient resource adaptations in order to optimize long-term system performance is a non-trivial problem.

- Real-time or close to real-time video analytics is being touted as *the killer* application in MEC in recent years. Unlike generic applications, in video analytics the users may select different configurations (e.g., frame rate, frame resolution, and number of video sources) in order to balance the trade-off between application quality of service (QoS) such as latency and energy consumption, and application quality of experience (QoE) such as quality of analysis outcome. This makes efficient resource optimization even more challenging.

1.4 Related Surveys

Please refer to Appendix A in the Supplementary File for a summary of the existing surveys [7–12] and their comparison to this survey.

1.5 Our Contributions

In this survey, we focus on both system-side and application-side optimization problems within MEC systems, viz., computation offloading decision-making, resource allocation, and configuration adaptation in handling dynamism in edge systems. In particular, the main contributions of this survey can be summarized as follows:

(1) We offer a comprehensive discussion on the optimization of computation offloading and resource allocation at the edge. In particular, we discuss these problems from different performance objectives' perspective ranging from simple (e.g., minimizing energy consumption and latency) to more complex (e.g., minimizing energy consumption subjected to latency constraints).
(2) In this survey we also take a look at works that address the dynamic nature of the environments and systems where MEC is adopted. We particularly focus on the impact of stochastic computation offloading that includes dynamic task arrival, fluctuations in channel quality, and changes in computation availability. We discuss the existing works that propose service migration strategies within the edge architecture triggered by the aforementioned system dynamism. We also investigate the role of MEC in vehicular networks, especially in handling user mobility.

(3) Most optimization problems in MEC are joint optimization problems and are Mixed-Integer Nonlinear Programming (MINLP) based. It is difficult to obtain closed-form solutions for MINLP problems subject to non-convex constraints and integer (or binary) parameters. In this survey, we discuss existing decentralized frameworks for solving complex decision-making problems (e.g., user-server association, channel allocation, computation offloading model selection). Game theory based frameworks are also introduced such as, potential game, Stackelberg game, and matching game. We also discuss existing task queue management and reinforcement learning (RL) techniques that make a sequence of decisions based on the changes in the edge environment. We then show how recent works in federated learning use RL techniques and optimization.
(4) Finally, we highlight the challenges and solutions of deploying and optimizing video analytics workflows on MEC. Typically, video analytics require considerable data transmission (e.g., live streaming video) and computation resources to run sophisticated processing algorithms (e.g., larger and complex neural networks). Compared to generic non-video applications, tasks in video analytics are both computation-intensive and bandwidth-hungry. On the other hand, such tasks warrant performance guarantees (e.g., accuracy, completeness) in addition to energy efficiency and latency sensitivity requirements. Therefore, simply optimizing computation offloading and resource allocation strategies from the system point of view does not yield the best performance. Thus for video analytics, the system optimizations to be application-aware is of paramount importance. In this survey, we mainly consider three types of optimizations from a video analytics application point of view: i) balance between performance of video analytics and cost for mobile devices, ii) task placement for different video pipelines (i.e., sequential, parallel and hybrid) based on directed acyclic graph (DAG), and iii) deep neural network (DNN)-level optimization, and distributed and collaborative model inference.

The rest of the paper is organized as follows. Section 2 presents the existing work in computation offloading and resource allocation. Section 3 discusses the dynamic optimization problems in MEC. Section 4 presents the edge optimization solution approaches. Section 5 discusses the challenges and solutions on video analytic in MEC. Section 6 presents the open research challenges and future directions. Section 7 concludes the survey. A list of important acronyms (in the order they appear in the survey) are summarized in Table 2 of the Supplementary File.

1.6 Survey Methodology

Please refer to Appendix B in the Supplementary File.

2 COMPUTATION OFFLOADING AND RESOURCE ALLOCATION

As we discussed in the previous section, most modern mobile devices are limited by their battery and computing capacities. Therefore, offloading their data to nearby edge servers for the purpose of energy preservation and latency reduction makes a lot of sense. In this section, we describe and compare the common computation offloading models, viz., 1) full offloading (or edge-only), and 2) partial offloading. We also discuss some common optimization objectives (towards such computation offloading) in terms of latency and energy consumption minimization. The classification of related works (i.e., papers) in this space and their objectives and optimized metric are summarized in Appendix F (particularly, Table 3 and Table 4) of the Supplementary File. Fig. 2 in the Supplementary File presents their relative comparison.

2.1 Full Offloading/Edge-only Computation

In full offloading [13–22], mobile devices transfer the whole application data to the edge server(s). Since the application only runs on the edge server, only the energy consumption of data transmission need to be considered. The end-to-end application latency in such cases includes the data transmission time (from devices to edge servers) and the computation time at the edge server. The former depends on the transfer data rate, while the

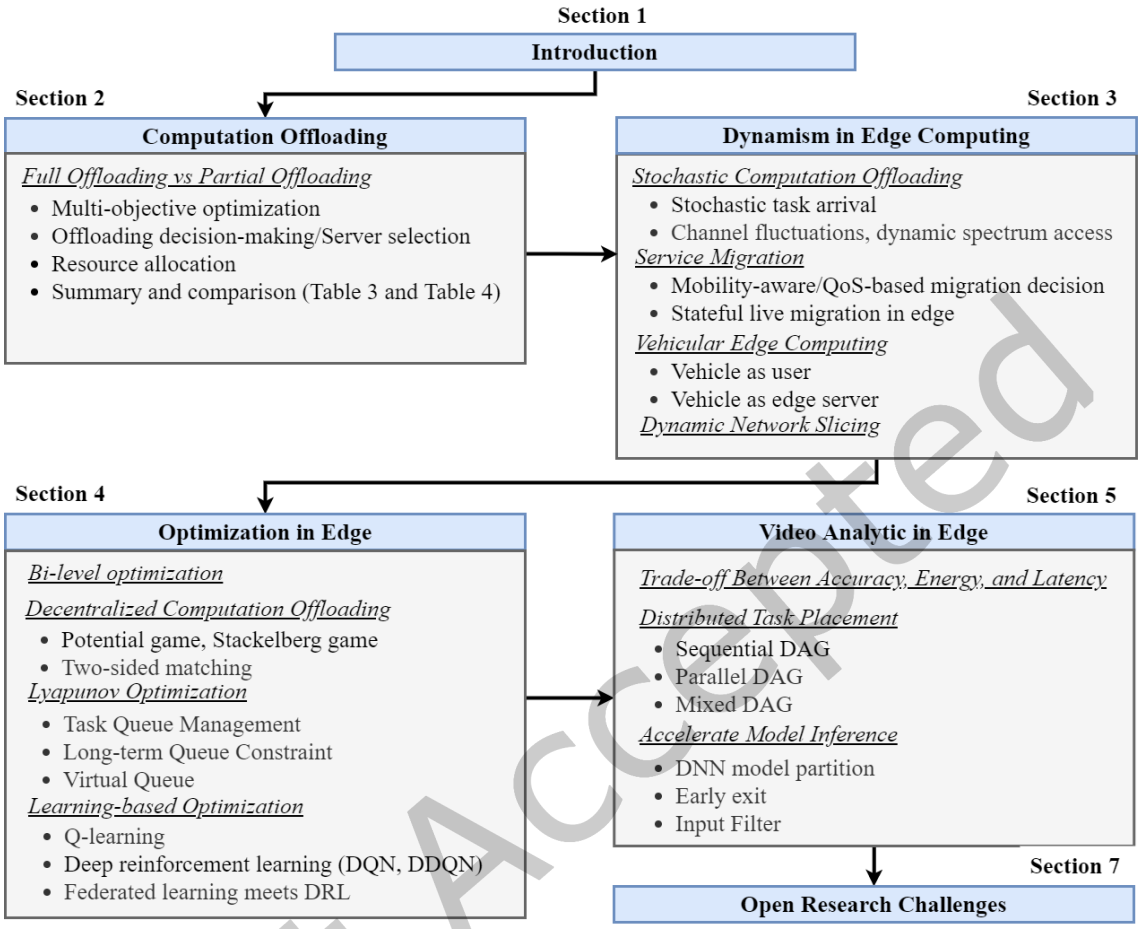


Fig. 3. Survey organization. It presents papers that solve full offloading and partial offloading problems in generic communication and computation environment (Section 2). Then several dynamic factors (e.g., tasks, channels and mobility) and their impacts on the edge system are presented. In particular, papers on service migration, vehicular network, and dynamic network slicing are presented in Section 3. This follows the summary and comparison of existing mathematical tools and techniques that are widely used to solve the problems of computation offloading and resource allocation in Section 4. Finally, video analytics as an exemplar application is discussed to show how application-side optimizations can improve the overall performance in resource-constrained edge computing systems (Section 5).

latter depends on the allocated computing resources (such as CPU cycle frequency). In case of multiple edge servers, the mobile device not only needs to determine whether the task should be offloaded, but also need to select the best edge server. However, when more and more mobile devices choose the same server, resource competition among mobile devices becomes more intense and the system performance starts to decline. Therefore, both resource allocation and edge server selection are important factors in performance optimization of full offloading strategies. Overall, the strategies where mobile devices benefit from full offloading can be classified into: 1) ones where the energy consumption can be reduced by sending the task data to the edge while satisfying the latency constraint and 2) ones where the task can not meet the latency constraint by running it locally.

The key to a successful (or beneficial) full offloading is to allocate sufficient radio and computation resources such that both the transmission time and energy can be minimized. However, such application-specific minimization can be defined in many different ways. For example, [14] studies full offloading strategy for a specific area with multiple IoT devices and multiple edge servers to *minimize the end-to-end latency*. The proposed sample average approximation (SAA) based method achieves 20% of global cost reduction on a true base station dataset. Authors in [22] and [15] propose offloading frameworks which aim to *minimize the weighted energy consumption and latency*. In order to *minimize the energy consumption while satisfying the end-to-end latency constraint*, the offloading decisions in [18] are determined by sub-carrier power and sub-carrier allocation strategy while [20] performs a device classification and priority determination strategy to make offloading decisions based on communication channels and computation requirements. The summary and comparison of works proposing full offloading strategies are described in Appendix F (particularly, Table 3) of the Supplementary File.

2.2 Partial Offloading

Under ideal conditions, all application data should be sent to the edge server to be processed there. Nevertheless, when mobile devices have to transmit a large amount of data (e.g., training data for machine learning, videos processing) to the edge server, the transmission cost (to edge servers) may overcome the offloading benefits due to the consequent increase in energy expenditure (due to data transmission) of the mobile devices. In order to solve this problem, the partial offloading model [23–32] allows mobile devices to offload part of the data to the edge server, while the rest of the data is processed/executed locally. In such cases, mobile devices and edge servers can process different parts of data, i.e., processing tasks concurrently, thereby reducing the end-to-end processing latency. However, the energy consumption at mobile devices may increase compared to full offloading since local computation at the devices can consume significant energy. Therefore, partial offloading frameworks typically use a configurable offloading ratio to control the trade-off between communication cost and computation cost making such offloading models more flexible.

Related works in this space include [24] that proposes a partial offloading based video compression framework. The authors formulate a latency-minimization problem for multi-user video compression with joint communication and computation resource allocation. Their simulation results show that as the number of devices increases, the performance gap between the full offloading and partial offloading models becomes more evident. It indicates that the partial offloading model can greatly reduce the system delay by introducing local computations. In order to minimize the energy consumption, [31] proposes a Deep Reinforcement Learning (DRL)-based approach to partially offload parts of tasks to the edge server based on the current queue length and the reward obtained from cooperative spectrum sensing. This work allows devices to adjust their computation and transmission speeds to perform different ratio of task offloading. Authors in [32] propose a multi-user and multi-server partial task offloading and resource allocation framework. They use a many-to-one matching game to perform user-server association and then solve the resource allocation and task partition problems at the user side with the queue length bounds. In [29], the authors propose a novel three-node edge system that the data is divided into three parts, i.e.: 1) device processing, 2) helper node processing and 3) edge node processing. The helper node acts as a small edge server and a decode-and-forward relay for cooperative communication to help the user device offload some of the data to the edge node. Such data partition is based on minimizing the total energy consumption of the user and helper with a latency constraint.

Since the major motivation for partial offloading is energy saving, such works are widely used in *energy harvesting* applications where mobile devices use harvested energy to perform local processing as well as data transmission. Works such as [25] proposes a UAV-enabled wireless powered MEC system to maximize the sum of computation rate of all the users. The UAV is equipped with a radio-frequency energy transmitter that can charge multiple mobile users. It shows that the partial offloading mode achieves largest weighted sum computation

bits compared to local computing and binary mode (i.e., a flexible selection between full offloading and local). Authors in [33] integrate simultaneous wireless information and power transfer (SWIPT) technologies into MEC system. Enabled by SWIPT and power splitting receiver, the user device can perform energy harvesting and information decoding simultaneously. The authors propose an energy-efficiency problem to determine the relay beamforming, device CPU frequency, transmission rate of uplink and downlink channels, and the task partition. The goal is to minimize the system energy consumption while satisfying the latency constraint. Compared to [25] and [33], authors in [30] make partial offloading decisions based on the backup power supply of an edge device rather than that of users. The authors propose a renewable-powered edge-cloud computing system. Based on whether the existing battery level can support the basic computation and transmission in each time slot, the edge device can offload all the workload or part of the workload to the cloud, and select the number of active servers. A summary and comparison of all such works that propose partial offloading frameworks are described in Appendix F (particularly, Table 4) of the Supplementary File.

3 DYNAMISM IN EDGE COMPUTING

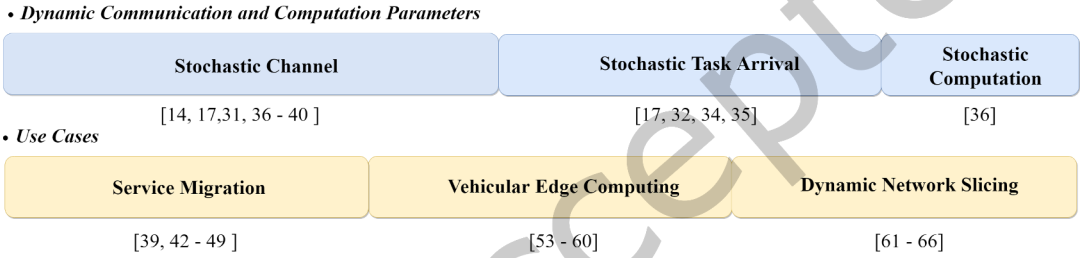


Fig. 4. Dynamism in Edge Computing.

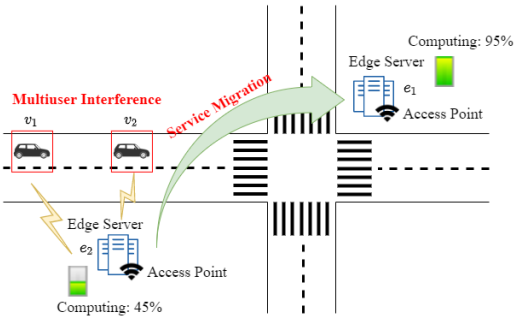
Computation offloading and resource allocation optimization can be effectively performed when the task information within the application model is known in advance and MEC environment is stable. However, for real-time and complex applications, task models are not deterministic. Moreover, MEC environments are more dynamic and prone to faults/fluctuations than traditional cloud environments. Thus, especially for mission critical applications, the optimization is unlikely to satisfy strict user requirements when such task dynamism and environmental fluctuations are not addressed. In this section, we discuss how the current literature address the dynamic changes in MEC environments. To that end, related works in this space are primarily grouped into stochastic computation offloading and several use cases such as service migration, vehicular edge computing, and dynamic network slicing.

3.1 Stochastic Computation Offloading

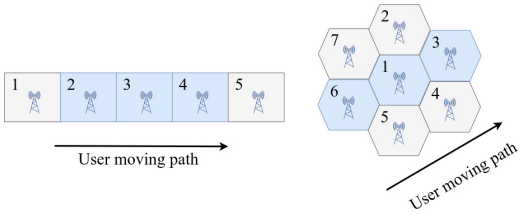
In this subsection, along with different aspects of stochastic computation offloading, we describe the list of factors that leads to such model adoption. These include unpredictable changes in user requirements [17, 32, 34, 35] and fluctuating wireless [14, 17, 31, 36–41] and computation resource availability [36]. Details can be found in Appendix D in the Supplementary File.

3.2 Service Migration

Service migration from current edge server to a new edge server (from the user/device point of view) may occur in the following situations as shown in Fig. 5a: i) the user is moving away from the current edge server and ii) the



(a) An example of service migration for vehicular network.



(b) Mobility-aware migration: a mobile device is moving from location 6 to location 3.

Fig. 5. Examples of service migration

resource availability at the current edge server has diminished and iii) the wireless interference caused by other users. In those situations, the basic motivation for service migration is to find a new edge server to maintain service continuity and QoS/QoE. Fig. 5a shows an illustrative example of such migration where vehicle v_1 is driving towards the edge server e_1 initiating service migration from e_2 to e_1, while vehicle v_2 also heading towards e_1 and thus initiating similar migration. The cost of migrating a stateless application is the migration delay in the network handover, which is usually negligible. However, stateful services such as VM and/or container based systems need to move the running states (i.e., CPU states, memory) from current (i.e., source) server to the new (i.e., destination) server. In such cases, the service may be suspended during the migration of those run-time states and such migration cost cannot be ignored. Typically, service migration in edge computing is based on the following two considerations: i) firstly, since service migration requires expensive operations and resources, such as I/O, CPU, bandwidth, the QoS/QoE will be degraded during the process of migration. The trade-off between service migration cost and migration benefit is crucial and ii) secondly, it is also very important to find a suitable strategy to determine how the memory of stateful services will be transmitted. Especially in a resource-constrained system, this strategy has a great impact on the overall migration performance.

3.2.1 Taking Migration Decision. Here, we describe a list of strategies used to take migration decisions in an edge computing environment. The papers are classified into two types, viz., Mobility-aware migration [42–46] and QoE/QoS-based migration [39, 47, 48].

Fig. 5b shows an example of mobility-aware migration, where the network topology consists of a set of possible service locations and it is assumed that each location is associated with an edge server. The state of the system is defined as the distance between the user location and the service location. When mobile users are moving across different geographical areas, we can predict the user mobility pattern to find the optimal migration decision (i.e., when and where to migrate). In [42, 43], the migration cost and the transmission cost are assumed to be functions of distance. In [42], the authors propose a mobility-driven service migration for 1D mobility patterns. In uniform 1D random walk, the mobile device moves to the left and right with equal probability r or stay in the same location with probability $1 - 2r$. Authors in [43] propose a distance-based migration framework to handle 2D mobility. While in 2D random walk (e.g., Fig. 5b), the mobile device steps to one of its six neighboring cells with equal probability r or stays in the same location with probability $1 - 6r$. In this work, the cost is modeled as a constant-plus-exponential function with respect to the distance. The authors propose a distance-based MDP and use a modified policy-iteration approach to find the optimal migration policy. Their numerical evaluations

show that the proposed migration approach reduces the cost 9 – 54% compared with the never/always migrate or myopic policies. This improvement largely depends on the total amount of available resources.

The mobility patterns of the above papers are either too simplistic or they assume that users' mobility patterns known in advance (e.g., the transition probabilities) - both scenarios are inadequate to deal with real world mobility patterns. By contrast, [44] predicts the target area of the user's movement by using an idealized geometric model which is based on a 6-tuple information, viz., position, position error, speed, speed error, direction, and direction error. The authors use T-pattern tree to mine local motion trajectory information, which can complement location prediction in case of incomplete tuple information. However, this work only considers linear mobility pattern. Works such as [46, 49] use ML-based techniques to implement mobility prediction. Authors in [49] propose *glimpse*, a seq2seq model customized for predicting a sequence of future locations. The model consists of an encoder and a decoder with both using LSTM neural networks. The authors use mobility data from New York City CRAWDAD NCSU data set [45] to train the prediction model. The authors in [46] propose a DQN-based task migration framework in MEC system which supports arbitrary moving patterns and network structures.

Instead, distance-based migration strategies use distance differences as the only reference that cannot capture the impact of other environmental fluctuations on QoE/QoS performance. While in QoE/QoS (Performance)-based Migration, the system jointly considers the impact of wireless condition and computation availability when making migration decision - a more practical solution for real-world use-cases. In [47], a dynamic service-migration mechanism based the user's QoE is proposed. This mechanism considers both the user mobility and the dynamic network resources (e.g., storage and bandwidth). The migration objective is to minimize the service cost while improving the QoE by offering different service resolutions. In [47], the migration decision is based on a score function S and a cost function C. The migration cost function depends on the data size of the task. The score function reflects the relationship between the user's QoE and the acquired service resolution. The authors in [39] use a balance factor to characterize the trade-off between too frequent and too infrequent migrations in consideration of the job completion rate, transmission cost and migration overhead. With the use of dynamic spectrum access communication, frequent migration will adversely affect short-term performance, which may be detrimental to the success of the tasks. However, too infrequent migration may leave the task vulnerable to upcoming spectrum fluctuations. A migration strategy is carried out by the optimizations of the trade-off between transmission cost improvement and migration overhead. The strategy tries to find the optimal 'application to edge server' mapping which maximizes the benefit of migration. In [48], the authors use threshold-based mechanisms to control the balance between too frequent and too infrequent migrations based on prediction measurements. The authors study the service migration problem that considers network state and server response time in making migration decisions. This work proposes a QoS-aware migration strategy that determines when to migrate the service. The system monitors and predicts the QoS either when the user moves or the server and/or network workload changes. When the predicted QoS violates the QoS threshold, the service migration is triggered. Although Performance-based migration can capture realistic performance of applications to some extent, it needs to collect huge amount state information to get accurate prediction. This inevitably adds more network and computational overhead.

3.2.2 Stateful Live Migration in Edge. Although application performance is improved upon service migration, the process is expensive that consumes extra network and computation resources. This in turn affects the performance of the running application. To handle this issue, live migration strategies are introduced to reduce the service downtime and the overall migration time. There are mainly two types of migration strategies, namely pre-copy and post-copy.

In pre-copy migration [50], the whole memory of the source machine is transmitted to the destination machine while the source machine is still hosting the application. In the meantime, the dirty memory pages will be re-copied to the destination machine. This process terminates when the re-copied rate is greater than the page

'dirty'-ing rate. Then the source stops the service and transmits the rest of dirty pages to the destination (i.e., stop and copy phase). Afterwards, the destination machine resumes the service for the mobile device. The key factor affecting the performance of pre-copy is the dirtying rate. This can be considered as a synchronization process between the source and destination machines. This synchronization also impacts the service downtime. For example, under a low dirtying rate, the service can be kept running on the source until the whole memory is copied. Therefore, the service downtime is simply the startup time of the new service instance on the destination. But for a high dirtying rate, the service has to stop before the whole memory is copied, thereby causing longer service downtime. To characterize the migration cost in pre-copy strategy, the authors in [51] propose a profit maximization framework to optimize the trade-off between the migration gain (reduction of the delay between the mobile device and the service machine) and the migration cost. The authors perform the pre-copy live migration and estimate the total migration time based on the network bandwidth between the source and the destination, the size of the memory, and the page dirtying rate of the application. Since the same migration time may cause varying performance degradation for different applications (e.g., I/O intensive or CPU intensive), the migration cost is determined by the total migration time, and a weighted sum of utilization of different resources.

While in post-copy [52], the source machine immediately suspends the service and only sends a minimal set of the state data to the destination (e.g., CPU state, register). The destination machine resumes the service and the source machine actively pushes remaining memory pages to the destination. For example, when page faults occur at the destination, those pages will be pulled from the source machine. Compared to post-copy, the pre-copy migration reduces the service downtime but adds more network overhead (i.e., transferring dirty papers). While the post-copy migration transfers less data, but it subjects the service to considerable delays when calling the missed memory pages - this lasts until the whole memory is transferred. Therefore, post-copy is not ideal for mission-critical applications where low-latency is required. In the rest of this subsection, we will discuss several edge frameworks based on pre-copy migration.

3.3 Vehicular Edge Computing

Mobility as the primary requirement for vehicular systems brings many unique challenges such as routing and forwarding, content caching, trust and authentication, and flexible network management [53, 54]. However, the scope of this survey is resource allocation and computation offloading aspects within edge systems, which are two fundamental problems for vehicular edge computing (VEC). In recent years, edge computing combined with Internet of Vehicle (IoV) techniques have provided a reliable and low/ultra-low latency computation and communication environment for VEC, thus playing an important role in handling real-time vehicle traffic and latency-sensitive computation tasks. Vehicles can be considered as users/devices with unique abilities within the MEC space as their fast-moving characteristics generate many stochastic resource allocation and computation offloading problems. Authors in [55] propose a road-segment based network infrastructure where the system aims to ensure that a task can be completed before the vehicles leaves the connecting roadside unit (RSU) under current segment. A game-theoretic approach is used to determine whether a task should be processed locally or remotely, followed by a Lagrange multiplier method to find the optimal solution for resource allocation. In comparison, [56] proposes a predictive offloading scheme which allows the vehicles to offload their tasks to the RSU in front along the direction in which the vehicle is moving. With accurate prediction of communication and computation latencies, the transmission cost for task output is greatly reduced since the output data can be transmitted directly from RSU to vehicles. However, this work also requires additional information to determine the future location of the vehicle. Instead of improving the end-to-end latency on average, the authors in [57] propose a risk-sensitive task fetching and offloading framework. The proposed system formulates a risk minimization problem by considering a set of reliability measures e.g., mean, variance, skewness, and other higher-order statistics. The problem is solved by a 'joint utility and policy estimation'-based learning algorithm. However,

based on frequently changing vehicle density and computation requirements, the RSU operators usually need to continuously adjust their computation offloading and resource allocation strategies. In this context, VEC systems, such as [58–60] that use DRL methods for computation offloading and resource allocation are further discussed in subsection 4.3.3.

3.4 Dynamic Network Slicing

As discussed in Subsection 1.2, network slicing is a promising solution to solve resource management issues in complex communication and computation environments where various applications with different requirements are involved. Similarly, one of the key design goals of dynamic network slicing is to fulfill the diverse slice performance requirements and provide multi-dimensional resources on demand. For example, a dynamic network slicing system should ideally be able to change its resource allocation policy according to actual user requirements and resource availability, which may change periodically or randomly in both spatial and temporal dimensions. Works such as [61–63] focus on dynamic inter-slice and intra-slice resource management, which is an inherently complicated problem. The authors in [61] jointly optimize dynamic assignment of tasks to slices, inter-slice resource management, and intra-slice resource management. A mixed-integer problem is proposed whose objective is to minimize the completion time of tasks. The problem is solved by an approximation algorithm with bounded approximation ratio obtained from a game theoretic treatment. Authors in [62] propose *EdgeSlice*, a decentralized resource orchestration system for dynamic network slicing. A performance coordinator and multiple orchestration agents are deployed to handle inter-slice and intra-slice resource allocations, respectively. The orchestration agents use DRL methods to manage resources for their own network slices under the supervision of a coordinator (i.e., connecting the two sub-problems by introducing a set of auxiliary variables). Work such as, [63] formulates a non-cooperative stochastic game in which tenants (i.e., service providers) aim to selfishly maximize their own long-term payoff. To address the network dynamics, a DRL algorithm is proposed to make joint communication and computation resource allocation policy.

On the other hand, efficient admission control is required to schedule incoming slice requests while guaranteeing existing performance requirements. Works such as, [64–66] address the admission control process in dynamic network slicing. Authors in [64] propose a stochastic optimization to jointly optimize slice request admission and resource allocation. The authors use Lyapunov optimization to handle unknown channel information and traffic distributions. A heuristic algorithm is applied to obtain the dynamic slice request admission decision with the purpose of maximizing the operator's revenue. Work such as [65] proposes a global service provisioning component which aims to provide admission control for incoming slice requests. A blockchain-based bidding system is applied to map user requests to appropriate network slices. Based on the prediction of network slices' traffic and user mobility patterns, a learning and forecasting based admission control is proposed in [66]. In this work, the admission control problems are considered as two-dimensional geometric knapsack problems, where slice requests are sorted in non-increasing order according to their profits and traffic classes. The problem is solved by a simulated annealing based algorithm.

4 OPTIMIZATION IN EDGE COMPUTING

Typically, joint optimization of computation offloading and resource allocation is defined as a Non-Convex Mixed-Integer Nonlinear Programming (MINLP) problem (as shown in Fig. 6) that is NP-hard in general. It is expensive and time-consuming to find the optimal solution (e.g., closed-form expressions) to such problems.

One of the most widely used methods in solving joint task offloading and resource allocation problems is bi-level optimization [18, 28]. This technique is also known as decomposition optimization. Here, the original NP-hard problem is divided into two sub-problems and the solution is obtained by alternatively and iteratively solving the sub-problems. This method typically used to separate computation offloading from resource allocation,

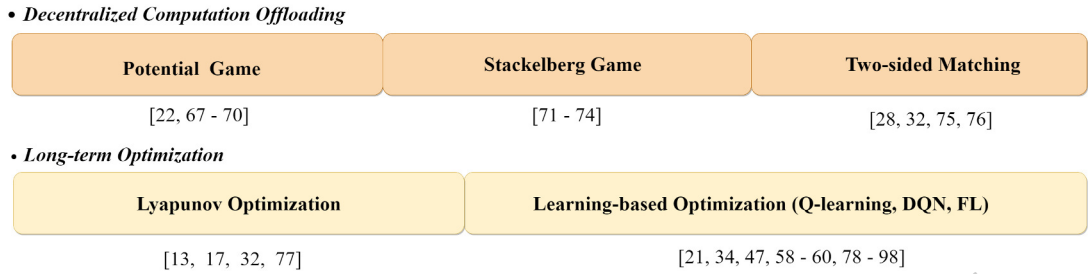


Fig. 6. Joint optimization of computation offloading and resource allocation in dynamic edge computing

i.e., the integer variables are relaxed (e.g., LP relaxation or Lagrangian relaxation) to turn the resource allocation into convex optimization problems that can be easily solved by classic convex optimization algorithms. However, when the optimal solution of the relaxed problem does not have all variables either 0 or 1 (i.e, some of them have fractional values), the solution only gives an upper bound. That is, the quality of the integer solution after variables are rounding back to integers may not at as good as that of the relaxed solution. Although methods such as Branch and Bound (B&B) can be used to effectively solve such discrete and combinatorial optimization problems when facing fractional values, they may cause huge computational complexity when the MEC contains a large number of users and servers. In this survey, we discuss an alternative bi-level optimization, which turns the 0-1 integer programming problem into a distributed problem. We then describe several decentralized frameworks such as game theory and matching techniques that are used to obtain a high quality solution with less computational overhead. On the other hand, to address user mobility and resource availability, it is necessary to continuously compute such optimal decision. Thus, we also review the most widely used long-term resource allocation and computation offloading techniques.

4.1 Decentralized Computation Offloading

Since in real-world scenarios mobile devices running data-intensive applications are managed by individual stakeholders (teams, agencies, users) who are selfish in nature, most centralized solutions ignore the satisfaction and willingness of users to agree to centralized server scheduling and resource allocation techniques. Therefore, in real-world scenarios users and their applications may behave in a non-cooperative manner when facing resource competition.

Let $\mathcal{N}$ be a set of players (i.e., mobile devices) and $\mathcal{A}$ be a set of feasible actions, an example of transformation from centralized optimization into decentralized optimization can be stated as:

$$\min_{a_n \in \mathcal{A}} u_n(a_n, \mathbf{a}_{-n}),\ \forall n \in \mathcal{N} \leftarrow \min_{a_n \in \mathcal{A}} \sum_{n=1}^{N} u_n(a_n) \tag{1}$$

where $\mathbf{a}_{-n}$ is the set of strategies made by all other mobile devices except for device n and u_n is the cost function of device n. This decentralized framework provides a solution that is close to the performance of optimal centralized solution but with less computational overhead and at the same time ensures that all players are mutually satisfied.

4.1.1 Game Theory. Game theory is one of the powerful tools to solve distributed optimization problems with rational and selfish actors, such as, multi-device and multi-server edge environments. Through a game theoretic framework, mobile devices are considered as a set of players that distributedly select their offloading decisions from a feasible strategy space $\mathcal{A}$. As shown in Eq. (1), the original centralized optimization problem is transformed

into multiple identical and distributed strategy making problems, i.e, scenarios where each mobile device aims to find a strategy $a_n \in \mathcal{A}$ to minimize its own cost (i.e., best response strategy). The most common way to define a game solution is Nash equilibrium (NE). At NE (definition (1)), all mobile devices satisfy their final offloading strategies and have no incentive to deviate from their strategies. Therefore, if a mobile device at NE chooses full or partial offloading, then it must indicate that the local computation leads to higher cost [67]. Otherwise, the mobile device can just unilaterally switch to local computation without consulting other mobile devices.

Definition 1. *A strategy profile $\mathcal{A}^* = \{a_1^*, a_2^*, ..., a_N^*\}$ is a NE of a strategic offloading game, if at the equilibrium $\mathcal{S}^*$, no player (mobile device) can further reduce its cost by unilaterally altering its strategy, i.e.,*

$$u_n(a_n^*, \boldsymbol{a}_{-n}^*) \leq u_n(a_n', \boldsymbol{a}_{-n}^*), \forall a_n' \in \mathcal{A}, n \in \mathcal{N}$$

Potential game [22, 67–70] is one of the useful tools to find the NE for game-based optimization frameworks where players have the same interests and share a global potential function defined in Eq. (2)). The potential function indicates the incentives of all mobile devices to change their strategies. For example, when a mobile device updates its strategy, the same behavior occurs in both the cost function of the mobile device and the potential function of all other mobile devices. The beauty of potential game is in its finite improvement property (FIP) which ensures that any algorithm that asynchronously updates the player's strategies is guaranteed to reach a NE within finite strategy update iterations (e.g., linear time complexity [68], quadratic convergence time [67]).

Definition 2. *A game is called a potential game if there exists a potential function (global cost function) $p(\boldsymbol{a})$ such that $\forall n \in \mathcal{N}$ and $a_n, a_n' \in \mathcal{A}$, if*

$$u_n(a_n', \boldsymbol{a}_{-n}) < u_n(a_n, \boldsymbol{a}_{-n})$$

we have

$$p(a_n', \boldsymbol{a}_{-n}) < p(a_n, \boldsymbol{a}_{-n})$$

Work such as, [69] utilizes a potential game approach to allow IoT users to maximize their own QoE. The authors propose a near optimal ϵ−NE resource allocation mechanism to reduce the time complexity of the best response algorithm to $O(N/\epsilon)$ i.e., the player can increase its QoE by no more than ϵ at ϵ−NE. In order to customize it for edge computing, one needs to study the property of the original centralized problem and produce a customized potential function. Works, such as [22, 67, 70] propose a multi-device computation offloading game which seeks to minimize a weighted sum of computational time and energy of mobile devices in a multi-channel wireless interference environment. Stackelberg game, a two-stage game model is also used to provide incentive mechanism for computation offloading in edge computing [71–73]. Here the players are labeled as a leader (e.g., edge servers) and multiple followers (e.g., mobile devices). In Stackelberg game, the leader chooses the best response strategy to maximize its payoff, while the followers react rationally to the leader's action to minimize their game cost functions [74]. The solution of Stackelberg game is Stackelberg equilibrium which is a little different from NE where the follower's optimal strategy is also the optimal strategy for the leader.

4.1.2 Two-sided Matching. Two-sided matching approaches can also help to solve decision making problems for edge computing in a decentralized manner, especially for sub-channel allocation and user-server association as defined in *Definition 3.* A two-sided matching game for user-server association are typically modeled with two sets of player i.e., mobile devices $\mathcal{U}$ and edge servers $\mathcal{S}$. Since one mobile device is only connected to at most one edge server but one server can accept multiple mobile devices, the user-server association is defined as many-to-one matching. Every mobile device $u \in \mathcal{U}$ has strict preference order $\succ_u$ over $\mathcal{S} \cup \{\theta^*\}$ where θ^* denote local computation (no offloading), while every edge server $s \in \mathcal{S}$ has strict preference order $\succ_s$ over

$\mathcal{U}$. The preference order are defined by user-defined performance metric (e.g., latency, energy consumption). Authors in [28] use energy consumption as its matching preference and formulate a two-sided matching game for sub-channel access. In [32], the matching preference is defined as the weighted transmission rates in a descending order, which is affected by the other UEs (matched to the same server) via co-channel interference. In [75], the mobile user prefers to select the edge server that provides the higher offloading rate and computation resource, while the edge server prefers the users with lower computation overhead.

Definition 3. *A matching μ is a collection of pairs in $\mathcal{U} \times \mathcal{S}$ s.t. every $u \in \mathcal{U}$ is a member of at most one pair, and every $s \in \mathcal{S}$ is a member of at most $|\mathcal{U}|$ pairs.*

The matching game should also consider the externalities in which the matching preference dynamically changes with the matching states of the other mobile devices in the same set [28, 32, 75, 76]. According to *Definition 4*, the goal of many-to-one matching is to obtain a stable matching μ such that both mobile devices and edge servers have no incentive to exchange their matched pairs. A simple approach is to initialize a temporary matching μ based on Deferred Acceptance (DA) method [76] i.e., mobile devices propose to their most preferable edge servers and edge servers accept or reject mobile devices from its applicant list based on their applicant preferences. The remaining mobile devices remain unmatched and repeat this process until all mobile devices are matched. Once an initialized μ is ready, all the pairs are traversed to find a block pair, then the matching process is updated and repeated until a stable matching is achieved. The complexity of this approach is $O(|\mathcal{U}|^2)$ [32].

Definition 4. *A matching μ is blocked by a pair $(u, s) \in \mathcal{U} \times \mathcal{S}$ iff $s \succ_u \mu(u)$ and $\exists s' \in \mu(s)$ s.t. $u \succ_s u'$. Also, a matching is stable iff it is not blocked by any pair.*

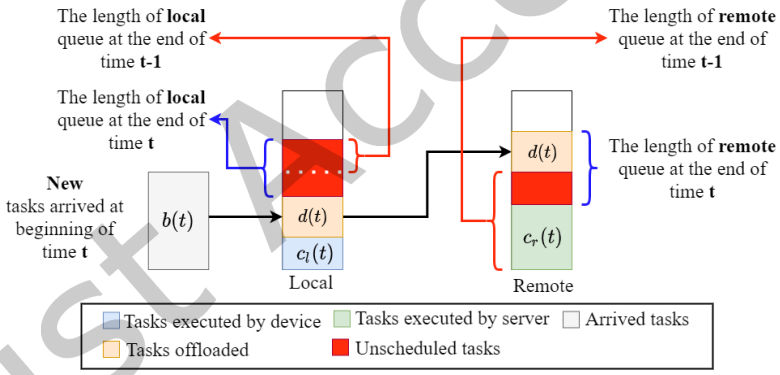


Fig. 7. The queue length of task buffer

4.2 Lyapunov Optimization

In order to address the temporal and spatial differences in edge environments caused by user mobility and resource availability, one can consider computation offloading and resource allocation as dynamic programming problems. We study related Lyapunov optimization-based online algorithms that seek to make a sequence of computation offloading and resource allocation decisions based on the changes in the edge environment.

4.2.1 Task Queue Management. Task queue management is a classic dynamic programming problem within edge computing environments that is usually subjected to long-term queue constraints. We also discuss how Lyapunov optimization balances the trade-off between queue stability and other user-defined performance metrics.

When partial offloading is enabled, mobile devices need to maintain a local queue and a remote queue i.e., the arrived/offloaded but not yet executed tasks will be queued in the task buffer of the device and the edge server. The queue length changes dynamically, triggering the need for online optimization. As shown in Fig. 7, the evolution of the local and remote queue lengths can be stated as follows: 1) The local queue length at the end of time slot t is determined by the number arrived tasks at the beginning of time slot t, the number of tasks executed by the mobile device (which can be controlled by the device's CPU-cycles frequency), and the number of tasks offloaded to the edge server (which is controlled by the data rate) during the time slot t and 2) The length of the remote queue at the edge server is determined by the number of unscheduled tasks executed by the server (which is controlled by the computation resource allocated to the mobile device) and the number of offloaded tasks from the mobile device during time slot t.

4.2.2 Long-term Queue Constraint. According to the Little's law [17, 32], it is assumed that the average delay is proportional to the sum queue length of the local queue and remote queue. Many works use long-term queue constraints to describe the latency requirement.

For example, the authors in [17] propose an online joint radio and computational resource management algorithm to minimize the 'long-term weighted average' power consumption of the mobile devices and the edge server. In order to guarantee that the tasks can be completed within finite delay, the authors add the following constraints to enforce the task buffers to be mean rate stable:

$$\lim_{T\to\infty} \frac{\mathbb{E}[|Q(t)|]}{T} = 0, \ \lim_{T\to\infty} \frac{\mathbb{E}[|T(t)|]}{T} = 0$$

where $Q(t)$ and $T(t)$ are the queue lengths of local and remote queues respectively. An extension of this type of constraint is to further ensure the reliability of the queue by adding probabilistic constraints. In [32], the authors propose a ultra-reliable and low latency communication (URLLC) task offloading and resource allocation framework. The statistics of the queue length is studied and the authors impose probabilistic constraints to characterize the queue length. This work uses two queue buffers to store the split tasks for local computation $Q^L(t)$ and offloading $Q^O(t)$. The queue length constraints are expressed as follows:

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} Pr(Q^L(t) > d^L) \leq \epsilon^L, \ \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} Pr(Q^O(t) > d^O) \leq \epsilon^O$$

where d^L and d^O are the the queue length bounds, ϵ^L and ϵ^O are the tolerable bound violation probabilities (which are small values less than 1).

4.2.3 Lyapunov Optimization. Once the queue constraints are given, the problem seeks to find ways to adjust decisions according to the most recent queue length. Here, we discuss the Lyapunov Optimization that tackle such problems. Many Lyapunov optimization frameworks [13, 17, 32, 77] have been proposed as solutions to deal with task queue management problems that aim to stabilize the queue while optimizing a user-defined long-term average performance objectives. The basic idea is to transform the original dynamic programming problem into individual deterministic per-time slot optimization problems. Let us consider a typical queue-based edge computing optimization system (e.g., Fig. 7) where $Q(t)$ and $T(t)$ indicate the lengths of local and remote queues respectively. In time slot t, a Lyapunov function will be defined to drive the current optimal decision, which is a quadratic equation that can be expressed as:

$$L\Big(\Theta(t)\Big) = \frac{1}{2}\Big(Q(t)^2 + T(t)^2\Big)$$

Then, the Lyapunov drift in computed to indicate the stability of the queue by measuring the difference in function $L(\Theta(t))$ between two adjacent time slots (i.e., $t+1$ and t). It can stated as

$$\Delta(\Theta(t)) = \mathbb{E}[L\Big(\Theta(t+1)\Big) - L\Big(\Theta(t)\Big)|\Theta(t)]$$

In $\Delta(\Theta(t))$, the difference can be computed as:

$$L\Big(\Theta(t+1)\Big) - L\Big(\Theta(t)\Big) = \frac{1}{2}\Big(Q(t+1)^2 - Q(t)^2\Big) + \frac{1}{2}\Big(T(t+1)^2 + T(t)^2\Big)$$

Afterwards, the conditional Lyapunov drift-plus-penalty is applied to balance the trade-off between latency and objective (i.e., energy minimization) optimization, which is stated as:

$$\Delta(\Theta(t)) + V \times \mathbb{E}[X(t)|\Theta(t)]$$

where $X(t)$ is the per-time slot objective function of the original problem and V is a control parameter that adjusts the trade-off between queue stability and objective function. In [17], it shows that there exists an $[O(1/V), O(V)]$ trade-off between objective function and the queue stability.

4.2.4 Virtual Queue. The Lyapunov optimization method can also be used to solve optimization problems with generic long-term constraint by introducing the virtual queue. In [13], the authors create a virtual queue as a historical measurement of the exceeded latency for real-time video analytics. The virtual queue and the equivalent long-term constraint are defined as:

$$q(t+1) = [q(t) + l(t) - L_{max}]^+ \leftarrow \lim_{T\to+\infty} \frac{1}{T}\sum_{t=1}^{T} l_t \leq L_{max}$$

where $l(t)$ is the average latency of video streams at time slot t and L_{max} is an average latency threshold of a long period (i.e, $1 \rightarrow T$). The stability of the queue ensures that the average latency does not exceed L_{max}. Similarly, [77] constructs a virtual queue to transform the long-term data transmission constraint into the following equivalent long-term constraint, which is state as:

$$q(t+1) = [q(t) - s(t) + \frac{L}{T}]^+ \leftarrow \sum_{t=1}^{T} s(t) \geq L$$

where $s(t)$ is the amount of data transmitted in time slot t and the system needs to transmit L bits of data within the deadline T. Although Lyapunov optimization method reduces the complexity of the original problem by transforming the long-term problem into individual time slot problems, the transformed objective is still a weighted function. For works with strict requirements, it is time consuming to find the optimal V such that the requirement can be satisfied.

4.3 Learning-based Optimization

Papers mentioned above solve computation offloading and resource allocation problems using conventional mathematical tools, such as Sample Average Approximation (SAA) [14], Lagrange dual decomposition and subgradient projection [15], alternating direction method of multipliers (ADMM) [16], piecewise optimization [24], successive convex approximation (SCA) method [25], and interior point method [29]. To formulate a realistic optimization problem, these methods need to create precise mathematical models which are usually obtained by performing extensive experimental measurements or drawing from historical experience. Therefore, conventional mathematical tools may only be successful in some specific use cases, as they rely entirely on pre-defined mathematical models. In contrast, learning-based methods such as Reinforcement Learning (RL) allow the system to estimate mathematical models from interactions between learning agents and edge environment, resulting

Table 1. The comparison of individual papers working on RL.

Ref	Actions	Objective	RL approach
[21]	Channel allocation, User-association	Maximize long-term downlink reward	DDQN
[34]	Computation offloading, Energy allocation	Minimize weighted sum of delay and computation task dropping	DQN
[47]	Service location, service resolution	Maximize Quality of Experience (QoE)	Q-learning
[58]	Server selection, Data transmission mode selection	Maximize utility	DQN
[59]	Spectrum, computing, and storing resource allocation	Maximize the numbers of offloaded tasks that are completed with satisfied QoS requirements	DDPG
[60]	Channel prediction	Maximize utility	DQN
[78]	Transmit power, Offloading decision-making	Minimize weighted energy consumption and latency	Q-learning
[79]	Computing capability, Ratio of task computed locally	Minimize execution time (subjects to energy constraint)	Cooperative Q-learning
[80]	Computation offloading, Energy harvest time	Maximize computation rate	DQN
[81]	Device classification	Minimize energy consumption	DQN
[82]	Computation offloading, Energy allocation	Minimize experienced delay	DDQN

in a more efficient and adaptable solution. Furthermore, learning-based methods are able to capture the spatial and temporal correlations between environmental states, making them more suitable for long-term/online optimization. In particular, we focus on Markov Decision Process (MDP) and RL based techniques for edge resource management. The RL frameworks in the current literature are classified into the following two types, viz., Q-learning based (i.e., a Q-table based iterative method) and Deep Q-learning based (i.e., deep neural network based iterative method). We also present Federated learning (FL) as a novel distributed training scheme for learning-based methods.

4.3.1 Markov Decision Process. Many learning based resource management problems in dynamic edge environment are modeled as Markov Decision Process (MDP). Although conventional optimization methods (e.g., value or policy iteration) can be applied to obtain optimal long-term strategies for problems with known transition probabilities, they are not always suitable for many dynamic edge systems. In many real-world use cases, the value or the policy-based iteration mechanisms suffer from exponential computation complexity due to the multi-dimensional state and action spaces. On the other hand, the system may also lack prior information i.e., transition probabilities. To address these issues of stochastic task offloading and resource allocation problems, RL methods are being used in edge computing systems. RL is a major branch of artificial intelligence and machine learning (AI/ML) where the basic model can be defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(s_{t+1}|s_t, a_t)$ is the transition probability (unknown) from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ after the agent takes the action $a_t \in \mathcal{A}$, the reward $R(s_t, a_t)$ represents the immediate utility obtained by taking action a_t at state s_t. $\gamma \in [0, 1)$ is the discount factor which determines the importance of future rewards compared to the recently rewards. In RL, the agent (decision maker) defines the control policy $\pi(s_t) = a_t$ as a mapping from a state to an action and the goal of the agent is to learn an optimal policy π that maximizes the expected long-term discounted reward which can be expressed as

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{+\infty} \gamma^t R(s_t, \pi(s_t))|s_0, a_0], \ \forall s \in \mathcal{S}$$

The agent should study the trade-off between exploration and exploitation and to find an optimal balance among the two. Exploration aims to find better new action that may yield a higher reward in the future, while exploitation using the action that has the highest cumulative reward tried in the past. A well-known solution is called ϵ-greedy, which acts randomly with probability ϵ (exploring) and acts greedily with probability 1-ϵ (exploiting). In edge

systems, the utilities are usually measured as computational and communication costs such as processing latency, energy consumption or a combination of both. The key to obtain an effective learning policy is to ensure that minimizing the expected cumulative discounted cost is equal to minimizing the objective function of the original optimization problem. Table 1 describes the list of related papers that use RL based frameworks to optimize the computation offloading and resource allocation in edge environments.

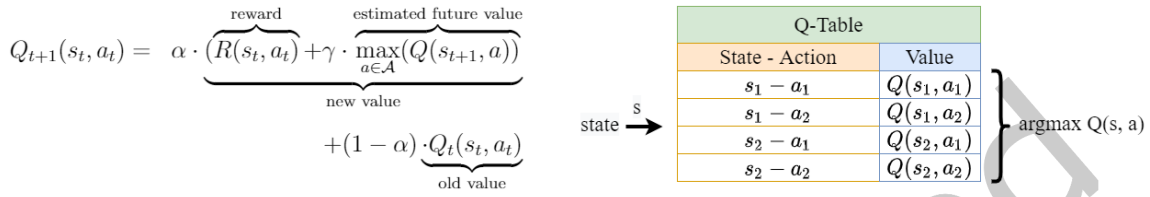


Fig. 8. Bellman equation and corresponding Q-table.

4.3.2 Q-learning. It is a model-free scheme to learn the quality of actions in unknown environments without knowing any dynamic statistics. In Q-learning, the quality of action a_t is defined given certain state s_t as a Q-value $Q(s_t, a_t)$. The optimal Q-value represents the maximum expected reward by following a policy π after receiving the state s_t and taking the action a_t. Even though the state transition probabilities are unknown, the optimal policy can be found in a recursive manner based on historical records. The update of Q-value can be obtained via the Bellman equation (as shown in Fig. 8), where α is the learning rate which impacts the updating speed of Q-value learning from the new value. The new value depends on two elements: 1) The immediate reward $R(s_t, a_t)$ by taking action a_t in given state s_t. Then, the next state s_{t+1} can be observed and 2) The update scheme adds a weighted future value to the immediate reward. This weighted future value is computed by searching the maximum Q-value of all the actions in state s_{t+1}, that is $\gamma \cdot \max_{a \in \mathcal{A}}(Q(s_{t+1}, a))$. Conventional Q-learning methods store and update the values of all the state-action pairs in a Q-table as shown in Fig. 8. The size of the table entries equals to the size of state space times the size of action space. At each decision epoch, the agent is going to search the Q-table by the given state and find the optimal action based on state-action values. Also, the corresponding value will be updated based on the Q-function.

Authors in [78] propose resource allocation for edge computing in IoT networks, where end devices adopt the time division multiple access TDMA scheme to transmit their data to a gateway with edge server. In order to minimize the 'long-term weighted sum' of power consumption and task execution latency in time-varying channel gain condition, a ϵ-greedy Q-learning is used to make computation task offloading decision and select the transmit power level from a set of discrete variables. Software defined edge cloudlet (SDEC) based RL optimization framework is proposed in [79] to tackle the task offloading and resource allocation in wireless MEC. The authors use a cooperative Q-learning technique to further enhance the search speed in Q-learning method. The basic ideal is that the agents search different choices in parallel by sharing their information decreases the search time greatly. In cooperative Q-learning, the Q-tables of agents that are located in the same vicinity are shared with one another. Afterward, each agent takes the weighted average of the others Q-tables and uses the resulted table as its new Q-table, where the weights are assigned based on agent's expertness (rewards). In the simulation result, they found that the proposed cooperative scheme achieves around 31.39% and 62.10% sum delay reduction when compared to traditional Q-learning with random algorithm and Q-learning with epsilon greedy, respectively. Authors in [47] use Q-learning to solve the service migration problem caused by user mobility and dynamic

network resources. The objective is to find the best service locations and service resolution that maximize the system reward for a sequence of batch requests.

4.3.3 Deep Reinforcement Learning. Deep Q-network (DQN) complements Q-table to handle environments with high-dimensional action-state spaces. To this end, the DQN method integrates the deep neural network (DNN) into the RL framework by adding an online DNN, a target DNN, and an experience replay [83]. The online DNN is used to approximate the Q-function, which is denoted as $Q(s, a) \approx Q(s, a; \theta)$ where θ stands for the weights of the online DNN. The target DNN is used to stabilize and improve the performance of the network and its weights θ^- are copied from the online DNN in every few iterations. The online DNN is trained to minimize a sequence of the loss functions that are defined as MSEs between the current predicted Q-value and target Q-value. The loss function at time-step t is is expressed as:

$$L_t(\theta) = \mathbb{E}[(\underbrace{R(s_t, a_t) + \gamma \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-)}_{\text{target Q-value}} - \underbrace{Q(s_t, a_t; \theta)}_{\text{predicted Q-value}})^2]$$

The experience replay strategy is applied to address the instability during the training procedure. In particular, the user experiences (s_t, a_t, r_t, s_{t+}) are stored into a replay memory of a finite size. At each learning epoch, the DNN is trained by a random mini-batch of experiences from the replay memory. The use of experience replay strategy helps to breaks the correlation of learning data. The framework of DNQ is illustrated in Fig. 9.

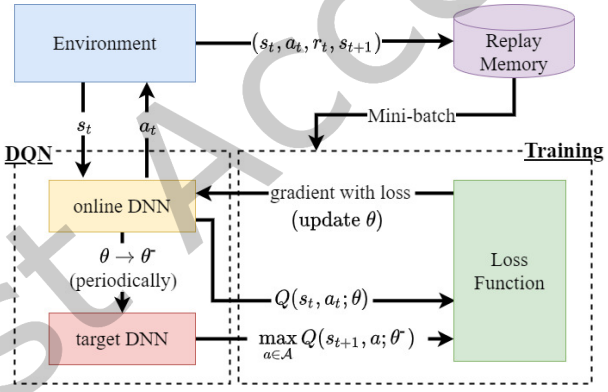


Fig. 9. An exemplary schematic of a framework with Deep Q-learning network

Works such as, [34] and [80] use DQN to optimally adapt task offloading decisions and resource allocation in time-varying wireless channel conditions. Authors in [34] propose a DQN-based algorithm for a single-user MEC system to jointly decide computation offloading and energy allocation. The states of the environment include the channel qualities between the mobile user and the base stations, the energy queue state, and the task queue state. In their simulation, they found that wider (with bigger number of neurons) DQN can better approximate the Q-function compared to deeper (with more hidden layers) DQN. Their algorithm achieves up to 56% in performance improvement. However, the convergence of the proposed algorithm is slow and the whole process leads to huge computational complexity. The reason behind this is the fact that DRL agents cannot handle high-dimensional discrete states (i.e., the channel gain). In order to avoid the dimensionality problem and reduce

complexity, DNN can be used to only solve some sub-problems [80]. In order to maximize the weighted sum computation rate in a wireless powered system, [80] proposes a learning-based framework that decomposes the original optimization problem (mixed integer programming non-convex problem) into an offloading decision sub-problem and a resource allocation sub-problem. The framework first uses DNN with an order-preserving quantization method to obtain the binary offloading actions. Once the offloading actions are fixed, the original problem reduces to a convex problem which can be easily solved using bisection search with $O(N)$ complexity. Authors in [81] addresses the long-term energy efficiency problem of an IoT-based network structure designed for green energy management systems (in smart cities). They propose energy management architecture and software model along with the DRL process. Moreover, the authors introduce a collaborative learning method in which the edge server offloads the DNN training to the cloud to reduce computing costs and the edge server deploys the online DNN based on the trained weights sent from the cloud.

Since Q-learning and DQN methods use the same values to select and evaluate an action, the Q-function may be over-optimistically estimated. In order to address this problem, Double Deep Q-learning (DDQN) methods decouple the selection from the evaluation [84]. Here, the target is as

$$y_t^{DDQN} = R(s_t, a_t) + \gamma Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^-)$$

In DDQN, the online DNN and its weight θ are used to determine the greedy policy while the target DNN and weight are selected for value evaluation. Works, such as [21, 82] apply DDQN approaches to learn the optimal computation offloading policy without apriori knowledge of network dynamic. Work such as, [21] presents a multi-user deep reinforcement learning algorithm for user association and resource allocation in heterogeneous networks. They use DDQN approach to solve a joint optimization problem with non-convex and combinatorial features. The authors in [82] consider a representative mobile user in ultra-dense networks where computational tasks are offloaded to the edge server via different base stations. They use a DDQN framework to obtain the optimal offloading decision as well as energy unit allocation. Those works show that the DDQN method outperforms Q-learning and DQN on both the learning speed and system performance.

DRL techniques are widely used in VEC to design learning policies which handle problems of stochastic vehicular traffic management. Authors in [58] adopt a deep Q-learning based approach for designing optimal offloading schemes that jointly consider selection of target server and data transmission path (i.e., vehicle to base station, vehicle to RSU, and vehicle to vehicle). The authors analyze the impact of real traffic on the obtained average utilities of a task with different offloading schemes. The proposed deep Q-learning scheme yields higher offloading utility compared to game theoretic approaches. Authors in [59] propose a DRL-based framework that optimizes spectrum, computing, and storage resource allocation jointly for moving vehicles and their dynamically changing computing tasks. Since the convergence time of the DRL networks becomes longer as the number of vehicles under the service area increases, the authors develop a hierarchical learning architecture that decomposes the original problem into sub-problems of spectrum allocation and computing/storage resource allocation. Subsequently, two deep deterministic policy gradient algorithms (i.e., DDPG, an improved actor-critic algorithm that combines the advantages of policy gradient and DQN algorithms) are applied to solve the sub-problems, respectively. Authors in [60] propose a model-assisted DRL framework where the DRL agent adaptively selects the appropriate transition data to update the weights of an online DNN based on their learning complexities (compared to other works that choose data randomly from experience). Such modifications improve the performance of the DRL framework in handling time-varying transition memory created by fast-moving vehicles.

4.3.4 Federated learning. With the increasing popularity of artificial intelligence applications and the volume of IoT data, traditional centralized data training on high-performance cloud data center or server is becoming unsuitable as centralized training leads to considerable communication overhead along with other issues such

as reliability and data privacy (discussions of such issues are beyond the scope of this survey) [85]. To solve these issues, the concept of FL is developed by Google in 2016 [86] where edge devices download and train the models using local data and only send the learnt parameters to the server for aggregation and model update. The local training and aggregation are repeated until the loss function reaches convergence. Such exchange of model parameters instead of raw data greatly improves data privacy and network resource conservation. Below, major research on FL for edge resource allocation and FL driving edge resource allocation are discussed.

a. Resource optimization for FL: Given that edge devices are usually constrained in terms of computing power and energy budget, implementing distributed data training on resource-constrained edge environment remains a challenging problem. For example, the time and the energy consumption of FL caused by local training and parameter transmission are two conflicting metrics. Therefore, striking a balance between the two is non-trivial. In [87, 88], the authors minimize the weighted sum of training latency and total device energy consumption by finding the optimal device CPU frequency, transmission latency, and local accuracy. In [87], a TDMA-based communication time allocation scheme is adopted, while [88] uses an OFDMA-based communication scheme. Problem decomposition and iterative algorithmic solution are proposed to obtain the optimal resource allocation strategy under a given latency constraint.

Considering that the processing latency of local training and the transmission time of parameters (to server) can vary significantly between devices, optimal client and global aggregation frequency selection strategies are crucial to the convergence speed of FL. Work such as, [89] proposes a control algorithm that determines the frequency of global aggregation under a given resource budget. To this end, the authors analyze the convergence bound of distributed gradient descent from a theoretical perspective and study the trade-off between local update and global parameter aggregation in terms of minimizing the loss function. Afterward, the control algorithm uses non-i.i.d. data distribution, system dynamics, and model characteristics to make the frequency of global aggregation dynamically adapt in real time. Authors in [90] propose a joint device scheduling and resource allocation algorithm to maximize the model accuracy under a given training time budget. Similar to [89], this work theoretically bounds the impact of the number of rounds and the number of scheduled devices to the convergence bound of FL. Then, the trade-off between latency per round and number of required rounds to achieve a fixed accuracy is investigated. The proposed accuracy maximization problem is solved by decoupling resource allocation from device scheduling. A binary search algorithm is designed to obtain optimal bandwidth allocation such that devices with worse channel conditions and/or weaker computation capabilities get more bandwidth. In terms of device scheduling, a greedy algorithm is introduced to select devices with the least update time, one after the other.

Authors in [91] bring FL into vehicular edge computing and propose a FL framework that jointly optimizes the on-board computation capability (CPU frequency), transmission power, and local model accuracy to minimize the maximum energy consumption (of vehicles) and computation latency. Since training devices are moving vehicles in this work, new vehicles entering the current service area are added to the participating vehicles list if the energy and latency can be minimized by selecting these new vehicles instead of the existing ones.

b. FL using DRL: In one of group among works on FL using DRL, DRL-based techniques (e.g., DQN, DDQN) typically provide a model-free solution for optimizing FL frameworks, helping them deal with learning problems under dynamic edge environments [92–95]. Authors in [92] propose a DRL-assisted FL framework where DRL is used to select Industrial Internet of Things (IIoT) devices with high quality data for local training. The proposed framework aims to increase the model aggregation rate and reduce communication costs. The authors in [93] use DRL to adjust the CPU-cycle frequency of mobile devices with the purpose of minimizing the weighted sum of training time and energy consumption. To achieve update synchronization, devices reduce their CPU-cycle frequency if they are faster in the training group. Work such as [94] uses DRL for channel selection and energy decision in a mobility-aware FL network. This work aims to maximize the number of successful transmissions while minimizing the energy and channel costs. The authors in [95] focus on the inter-client correlation across

the clients as they propose a novel FL model that utilizes DRL to determine the weights of local parameters in the aggregation process.

In the other group, DRL-based edge resource allocation and offloading decision making solutions are proposed where FL helps to train DRL agents in a distributed manner, leading to higher resource efficiency (i.e., FL-assisted DRL). In [96], authors compare the pros and cons of several training schemes of a DRL agent. In centralized DRL, training data from all participants are uploaded to central servers and DRL agents are trained at servers. Although centralized DRL gives the best performance, it suffers from massive redundant data transmission and privacy risk. While distributed DRL is designed to train DRL agents individually on the participants, the additional energy consumption and weak computing power of participating devices make them impractical in the real world. By contrast, FL replaces raw data transmission with model parameter upload to reduce communication cost. Secondly, FL selectively lets part of the devices train locally to achieve higher energy savings. Therefore, FL-assisted DRL serves as an intermediate design between centralized and distributed DRL and represents the best of both worlds. In [97], DRL agents are responsible for making offloading and energy allocation decisions for IoT devices within an edge environment. To address the issues of non-I.I.D data, IoT devices which have sufficient computational and energy resources are selected to join the local training and global model aggregation process. Authors in [98] propose a two-timescale DRL approach to make real-time computation offloading decisions and resource allocation strategies in the ultra-dense 5G network scenarios. The authors leverage FL to train the DRL model in a distributed manner, aiming to obtain faster and more robust training.

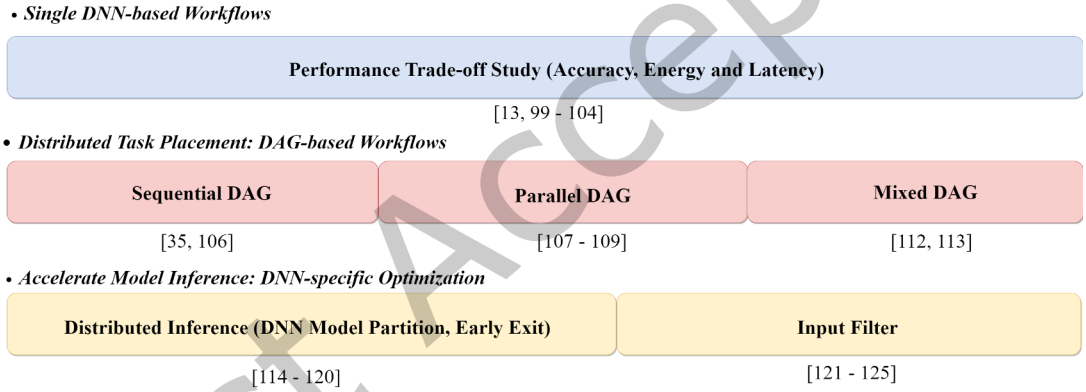


Fig. 10. Optimization for DAG-based application and single DNN-based application.

5 VIDEO ANALYTIC IN EDGE COMPUTING

Video analytics is one of the driving applications in edge computing that brings new resource allocation and computing offloading challenges. Many critical, next generation use cases (e.g., disaster response, tactical scenarios, industrial IoT, vehicular systems) that are increasingly relying on edge computing are driven by the need of performing video analytic at scale and in real-time, thus making it one of the most adopted application domains supported by edge systems. Compared to optimization for generic applications, optimizing real-time video streaming at the edge requires customized solutions and frameworks. As shown in Fig. 10, the current literature mainly considers two types of optimizations from a model point of view: 1) optimizing DAG-based workflows with mixed generic tasks and DNNs and 2) optimizing single DNN-based workflows. In order to optimize the DAG-based workflows, finding optimal placement of distributed tasks across edge and end devices is paramount. While for single DNN-based workflows, optimizing DNN model partition enables the edge system to run the target DNN in a

distributed manner, either vertically or horizontally. Apart from these workflow-specific optimizations, achieving performance trade-off by balancing data configurations and desired application performance is something that is equally important for both optimization scenarios.

5.1 Performance Trade-off Study: Accuracy, Energy, and Latency

The performance of video analytics (often measured in terms of accuracy, energy efficiency, and latency) depends on a variety of application data configuration parameters such as, frame resolution, frame rate etc. The selection of configuration has a great impact on different performance metrics as increasing configuration parameters often result in higher quality, but at the cost of higher latency.

In works such as [13, 99], the authors use extensive measurements to establish the relationship between different configuration parameters and performance metrics. The authors in [99] study the trade-off between latency and accuracy. They design and implement an edge-based orchestrator for Mobile Augmented Reality (MAR). The orchestrator aims to find the optimal server assignment and frame resolution based on empirical modeling. From the measurement data with YOLO [100] and SSD [101], the relationship between computational complexity, analytics accuracy, and video frame resolution is obtained and then fitted into the optimization problems. With respect to the video frame resolution, the authors model the computational complexity as a convex function (e.g., quadratic or cubic), while the accuracy is fitted into a concave function. The proposed problem is solved by the block coordinate descent method that iteratively optimizes the video frame resolution and server assignment. The edge-based orchestrator achieves 25% latency reduction at the cost of less than 1% accuracy loss. However, this paper considers a simplified latency model where the achievable data rate is considered as constant for MAR users. Such strong assumptions might be impractical in the real-world wireless scenarios. Compared to [99], the authors in [13] propose a framework for both configuration adaptation and bandwidth allocation for edge-assisted video analytics. This work aims to optimize the trade-off between accuracy and energy consumption for a given service latency constraint. They consider a practical scenario in which multiple video streams connect to the same edge server sharing a narrow uplink channel. Similar to the experiment in [99], the results show that the frame resolution and frame sampling rate independently impact the accuracy. In order to handle the problems caused by bandwidth variation and intrinsic dynamics of video contents, an online algorithm based on Lyapunov Optimization is used to select the optimal CNN model, sampling frame rate, and uplink bandwidth for each time slot. The proposed algorithm achieves a 44% reduction in energy consumption at the cost of 4% accuracy loss. The authors in [102] develop a data-driven optimization framework which introduces a complex interaction between accuracy, video bit rate, battery constraints, network parameters. The goal is to find an optimal offloading strategy for AR devices. This work formulates the configuration adaptation problem caused by time dynamic as a multiple-choice, multiple-constraint knapsack program and solves it with an improved brute-force search.

In VideoStorm [103] and VideoEdge [104], the trade-off between query accuracy and resource demand is extensively studied. VideoStorm [103] optimizes query scheduling by exploring utility-based resource management in terms of query accuracy and delay; while VideoEdge extends the problem to query placement across a hierarchy of clusters. More specifically, VideoEdge proposes a 3-tier hierarchical architecture, including cameras, clusters, and the cloud for video analytics. For every video query, VideoEdge searches the components implementation, knobs, and placement and finds a configuration to balance the accuracy and resource demands using an efficient heuristic.

5.2 Distributed Task Placement

Typically the pipeline of many complex video analytics applications are modeled as multi-stage tasks, varying from sequential batch processing to branched pipelined tasks. This subsection discusses optimization techniques

for DAG-based video pipelines. A directed acyclic graph $G = (\mathbf{V}, \mathbf{E})$ can be used to describe the video pipeline and intermediate result passing between different stages. In graph G, each vertex $v \in \mathbf{V}$ can be described as a task (e.g., encoding, decoding and object recognition). The edge $e \in \mathbf{E}$ represents the data passing between tasks. As shown in Fig. 11, [68] shows the pipeline of face recognition. It contains three tasks which are executed sequentially: 1) face detection, 2) feature encoding, and 3) face matching. The intermediate results are the face images and lists of face encoding. The end-to-end performance of the video pipeline is determined by the critical path of the DAG, that is the largest weighted path from source task to the sink task. In resource-constrained edge environments, effectively placing these tasks in optimal computation locations is paramount. Nevertheless, when tasks are placed on different edge servers, the critical path determination and performance estimation become challenging problems due the heterogeneous processing capacity (e.g., CPU and GPU) and dynamically changing wireless conditions at the edge servers.

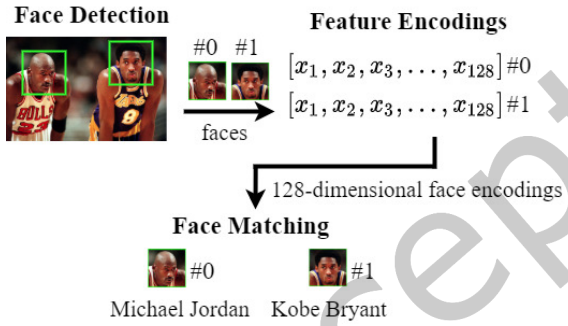


Fig. 11. An exemplar task graph for face recognition [68]

As shown in Fig. 12, video pipeline graphs can be classified into three types based on the dependencies among tasks: 1) Sequential DAG, 2) Parallel DAG and 3) Mixed (arbitrary) DAG. Here, we describe the task placement problems of different types of DAG. In general, the task placement for a DAG-based video pipeline can be stated as a $M \times K$ matrix:

$$A_{M\times K} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & \ldots & K \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 1 & \ldots & 0 \end{pmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ \ldots \\ M \end{matrix}$$

where $a_{m,k} = 1$ indicates that the task m is placed on edger server k, otherwise $a_{m,k} = 0$. Since a task can only be placed on edge servers, the placement follows $\sum_{\forall k \in K} a_{m,k} = 1$.

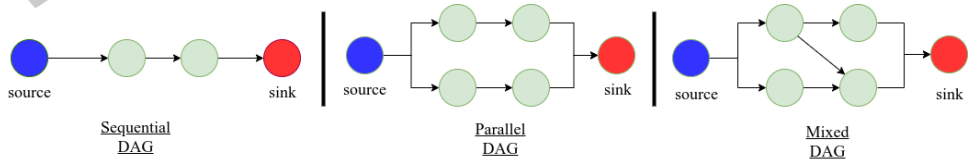


Fig. 12. The DAG classification.

a. Sequential DAG: Here, the tasks are executed one at a time. A task can only have one predecessor and one successor. The pipelines of some photogrammetry-based applications (e.g., openMVG [105]) and well-known image processing CNNs or DNNs (e.g., LeNet, AlexNet, and ResNet) can be described as sequential DAGs. The basic idea of task placement for sequential DAG is task partition, i.e., tasks are partitioned into local-processing tasks and remote-processing tasks to reduce the communication cost (latency and energy consumption). However, this partitioning problem is usually integrated with other problems such as resource allocation and server selection, making it challenging to solve. In [35], the authors propose a distributed offloading architecture and formulate a strategic game to find the optimal server selection and resource allocation, DAG partition and the optimal configuration for transmission power and CPU frequency at the mobile devices. The experimental results show that the proposed offloading schemes saves 40% (in sufficient network resource scenarios) to 60% (in limited network resource scenarios) on the total energy consumption. Authors in [106] classify application components into local-only phases and offloadable phases. The offloadable phases can be executed locally or to be transmitted to the server for processing. In order to ensure predictable performance (e.g., response time) while minimizing energy consumption, they propose a 'Suspension-and Energy-Aware' offloading algorithm where the tasks are executed following the earliest-deadline-first (EDF) task scheduling policy. For DNN partitioning, the discussions are made in subsection 5.3.

b. Parallel DAG: A parallel DAG contains multiple sequential sub-DAGs that can be executed independently. An example is given in Distream [107], where the proposed DAG contains multiple branches and each branch is dedicated to process a specific type of the detected object. The authors propose a stochastic partitioning scheme by profiling the accumulated inference cost of all the possible execution paths in the DAG. Based on these costs, a workload adaptation controller is applied to determine the probabilities for partitioning at certain vertices. Authors in [108] propose Hetero-Edge, an edge computing platform designed to minimize the end-to-end latency for real-time vision applications on heterogeneous edge clouds. This work uses a 3D scene reconstruction as a driving application example for the evaluation of resource allocation and orchestration. Two practical topologies, viz., Serial Topology (serial-DAG) and Parallel Topology (para-DAG) are considered for the 3D scene reconstruction application. In para-DAG, the framework partitions the images into multiple sections during the generation of disparity map. This partition creates a data-parallel bolt that runs the same disparity calculation function to accelerate the computation. The platform is built on Apache Storm, and consists of multiple edge servers and a distributed resource orchestration framework. The servers have heterogeneous computation and networking resources. The orchestration framework stores edge application as Storm tasks which are defined by a DAG and maps these tasks onto heterogeneous edge servers for efficient execution. The implemented testbed can achieve 40% latency reduction with an average per-frame latency of 32 ms. In [109], a collaborative mobile edge environment is established to split the pipeline of 3D reconstruction into high frequency and low frequency tasks. The evaluation on a hardware testbed using publicly available datasets shows upto $\sim$ 54% reduction in latency with negligible loss ($\sim 4-7\%$) in reconstruction quality. These works [107–109] use available edge resources intelligently by enabling extreme task parallelization, which proves to be a highly effective approach.

c. Mixed DAG: This type of DAG has no dependency restrictions and a task can have multiple predecessors and multiple successors. Many deep learning-based video processing applications have arbitrary DAGs, such as MV3D [110] and MVSNet [111] in the field of 3D object detection and reconstruction. For mixed DAGs, task placement is more complicated as most of optimization problems are NP-hard. The authors in [112] present Latency-Aware Video Edge Analytics (LAVEA), a low-latency video analytics edge computing platform that can serve multiple clients at the same time. They assume that DAG tasks are only offloaded to the nearest edge node (the edge-front node) via access points and formulate a mixed integer non-linear programming problem (MINLP) to determine the offloading decision and connection rate assignment. The problem is solved by integer relaxation and branch and bound (B&B) method. Since the edge-front node receives a large number of offloaded

tasks from the clients at each time epoch, LAVEA uses a task queue prioritizer to minimize the makespan for the task scheduling and proposes inter-edge collaboration to avoid workload overload on the edge-front node. When the edge front-end node is full of requests, it starts to coordinate with nearby edge nodes by placing some tasks on the less busy edge nodes so that all tasks can be scheduled within a reasonable time. In order to reduce computation complexity of dependent task offloading, the authors in [113] consider each DAG as a set of 'co-subtask' stages (tasks with the same depth). A flow scheduling heuristic is proposed to determine the task scheduling priority based on the release time and maximum computation load of a task within a co-subtask stage. Most of the work in this subsection only considers task placement without optimizing other aspects. It is crucial that an edge framework can jointly optimize task placement, configuration searching and resource allocation, which together provide a highly efficient and adaptive system dedicated to video analytics.

5.3 Accelerate Model Inference

Previous subsections discuss video analytics from application-level perspective, here we list a set of optimization approaches at the DNN-level. For a given task workflow and a set of configuration parameters, video analytics performance can be further improved by looking at the inner structure of the DNN model and the property of the video content that will quicken the model inference. In this way, unnecessary communication and computation under resource constraints can be avoided, thereby improving the overall inference efficiency. Here, we discuss three useful methods, namely model partition, early exit, and input filters.

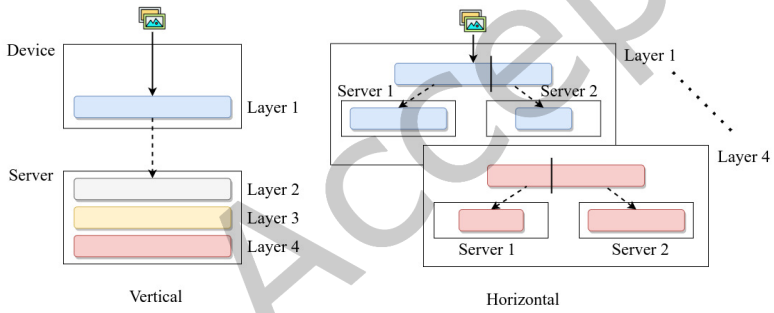


Fig. 13. Schematic of vertical and horizontal DNN model partitioning

5.3.1 DNN model partition. A layered DNN model can be partitioned either vertically or horizontally across mobile devices and edge servers to accelerate the computation and reduce energy consumption. Examples of vertical and horizontal model partition for a 4 layer DNN are shown in Fig. 13.

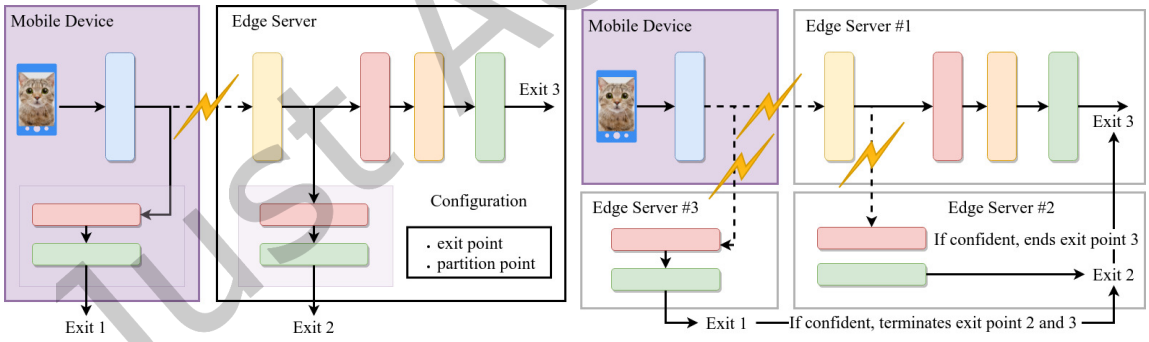
a. Vertical partitioning: Most state-to-art DNN models contain a sequence of layers (e.g., convolution layer, fully connected layer) that are executed sequentially by passing intermediate data between layers. Thus conceptually, this method is quite similar to sequential DAG partitioning. The intuitive idea is to find the best partition point that offloads computation-intensive layers to the edge server with little data transmission. Given that the DNN layers can vary significantly in both computation requirement and data size, the selection of partition point can significantly affect the overall latency and energy consumption of mobile devices. In addition, the application performance requirement as well as the edge environment may change frequently (as discussed before). Thus it is necessary to build a layered-based prediction model. This can be done by deploying a real-time monitor that periodically collects the availability of communication and computation resources in the system.

Neurosurgeon [114], Edgent [115] and SPINN [116] are cutting edge frameworks that provide automated, adaptive and collaborative DNN inference at the edge. Driven by DNN model partitioning, Neurosurgeon [114] and Edgent [115] establish a regression model to estimate the layer-wise performance (e.g., end-to-end latency

or mobile energy consumption) and a dynamic mechanism to find the optimal partition point that meets the user-defined requirements during the run-time. The authors in Edgent [115] consider DNN inference under dynamic network environment and build a reward-based configuration map constructor to customize their framework. SPINN [116] measures the inference latency using a 2-staged linear model based on the scaling factor between the actual time and the offline latency estimation.

b. Horizontal partitioning: Among works that propose horizontal DNN partition, MoDNN [117] and [118] are notable. MoDNN [117] is a distributed mobile computing system in Wireless Local Area Network (WLAN) where the DNN models are partitioned by layers and mapped onto mobile devices to accelerate the computation. Based on the computation time and the memory usage of the layers along with the worker's resource availability, each worker is mapped with a part of the layer inputs to increase the overall parallelism. Authors in [118] propose a scalable Fused Tile Partitioning (FTP) of convolutional layers to minimize memory footprint and enable parallelism. It also develops a novel work scheduling process to reduce overall execution latency.

5.3.2 Early exit. Confidence is an important factor in DNN inference. It indicates the likelihood that an anchor box contains an object. Early exit method explores the relationship between the layer's computational overhead and confidence score. This method gives a layer-wise selection to further accelerate the inference at the edge by terminating the ongoing processing at customized exit points. BranchyNet [119] proposes an open-source framework to obtain 'Branchy' (i.e., branched) DNN with multiple early exit points. When the classifier at a particular exit point is confident in the prediction (i.e., its confidence is above a threshold), the inference can directly terminate. BranchyNet applies the weighted sum of the loss functions of each exit branch as its optimization objective function. The results show that when using some well-known CNNs such as LeNet, AlexNet, and ResNet for testing, BranchyNet can provide 2X to 6X acceleration on the CPU and GPU. Early exit is also an enabler for distributed computing in edge when DNNs have more than one exit points. Authors in [120] propose a 3-layer framework across mobile devices, edge, and cloud to execute different exit points of a given DNN in a distributed manner. This work allows distributed devices and edge servers to jointly perform classification and aggregates the outputs.



(a) Early-Exit policy and DNN model partition based on [116]

(b) Early-Exit policy across multiple edge server.

Fig. 14. Examples of Early-Exit policy

Furthermore, the DNN model inference at edge can be further optimized by combining early exit policy with DNN model partition as proposed in [115, 116]. Compared to monotonous optimization of early exit strategies or DNN model partitioning, this combined method can deal with diverse user requirements and dynamic edge environments more flexibly. The secret lies in finding a combinatorial configuration of exit points and partition points which jointly optimize the overall performance. An example of early exit with DNN model partition is

shown in Fig. 14a with 3 exit points. Authors in [115] maps system states (e.g., latency, bandwidth) to exit points and partition points, then perform model partitioning on the selected exit point. While in [116], the on-device inference continues even after the partition point. The advantage of this method is that when the mobile device reaches an early exit point and the classifier gives a sufficient confidence (above the predefined threshold), the inference on the edge server can be stopped. This approach may also reduce the unnecessary data transmission if the server-side inference has not yet started. In the illustrative Fig. 14a, if the first exit point outputs a confident prediction that the image contains a 'cat', then the execution of exit point 2 and 3 can be terminated and thus computation resources can be saved. Otherwise, server-side inference will continue until a qualified early exit point is encountered. Based on this feature, SPINN achieves a speedup of up to 83% and 52% when compared to Neurosurgeon [114] and Edgent [115].

However, works such as, [116] will significantly increase the energy consumption of mobile devices and therefore it is not suitable for energy-constrained scenario. In fact, when there are multiple edge servers, the inference can be accelerated by sending the layers of different early exit branches to different edge servers. For example, as shown in Fig. 14b, we can let the exit point 2 and 3 run on two servers to improve parallelism. Similar to [116], when exit point 2 is completed, it can notify the server who is running exit point 3 based on its prediction confidence. We can also introduce another edge server to run exit point 1 to reduce the on-device computation overhead. But this will incur extra communication overhead since the intermediate data generated by the first layer have to be transmitted to both edge servers, which in turn generates a more changeling optimization problem.

5.3.3 Input Filter. Input filter is a key accelerator for running DNN model inference in edge computing environment with limited bandwidth and computation resources. Its objective is to remove redundant transmission and computation without compromising the accuracy. The basic idea is to deploy a lightweight pre-processing algorithm to determine the Region-of-Interest (RoI) within individual frames or filter non-target-object across consecutive frames. Then compute the difference among those areas or calculate the motion behaviors of the target objects to further make inference decision. This approach significantly reduces the latency and energy consumption, especially for video analytics.

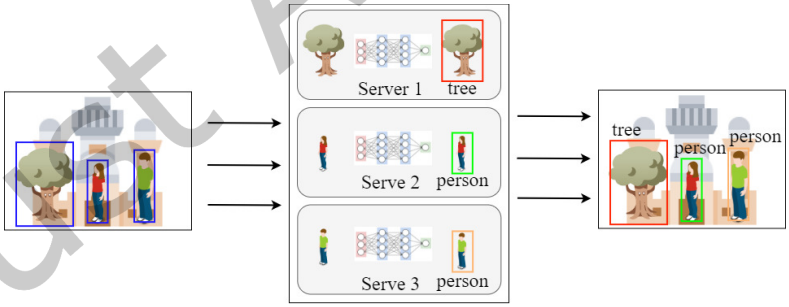


Fig. 15. An example of input filter based on RoI and parallelism inference with 3 edge servers. The detected results are fused back to the original frame.

Authors in [121] propose a Dynamic RoI Encoding technique to decrease the encoding quality of uninteresting areas to reduce the transmission latency and bandwidth consumption. More specifically, this work uses candidate RoIs generated by previous processed frame and slightly expands each RoI by one macroblock. A Parallel Streaming and Inference pipeline is introduced to further reduce the latency. This work solves the problem caused by object movement by shifting the bounding box based on the motion vectors from the current encoded

frame. Similarly, [122] uses a linear velocity model to approximate the ground truth boxes based on temporal correlation. While in [123], the authors apply an attention-based LSTM network to predict the region proposals (RP) of a new frame based on detected historical frames. The bounding boxes of RPs are dynamically expanded to address the trade-off between accuracy and latency. In addition, the authors propose a content-ware frame partitioning and offloading pipeline that effectively assign RP-Boxes to the edge servers (as shown in Fig. 15). Instead of targeting the RoIs or RPs, an alternate approach is to simply skip the non-target-object frames [124] or consecutive frames which have little change [125]. The authors in [124] consider the bandwidth-efficiency problem of real-time drone video analytics. The authors introduce the EarlyDiscard strategy, which is based on on-board processing to identify and filter useless frames, thereby reducing the number of frames required for transmission. More specifically, a weak detector such as image classification (e.g., MobileNet) is deployed to filter useless frames. Authors in NoScope [125] use a difference detector to compute the MSE between a labeled reference frame and an unlabeled frame in order to skip the frames whose MSE is lower than the predefined threshold. The authors also optimize the trade-off between accuracy and reference frame update speed. However, both works require specialized pre-trained model for a small set of target object classes (e.g, trees, cars, boats). Moreover, the weak detector [124] and difference detector [125] must have considerable small inference latency and energy consumption compared to original model inference.

6 OPEN RESEARCH CHALLENGES AND FUTURE DIRECTIONS

The open research challenges and future directions on edge resource management can be found in Appendix E in the Supplementary File.

7 DISCUSSION AND CONCLUSIONS

MEC is an emerging distributed computing paradigm that brings low/ultra-low latency, intelligent network and compute capabilities to the users and helps preserve mobile devices' energy. Due to the heterogeneity of edge resources, application structures, and user's requirements, most of optimization problems in MEC are NP-hard. On the other hand, the dynamic nature of environments and use cases where MEC are deployed brings unique challenges and thus adds novel design dimensions. This survey presented a state-of-the-art literature review of research thrusts in MEC from various design, deployment, and application perspectives. In this survey, we categorized and discussed the existing MEC research works based on their computation offloading models and resource allocation strategies as well as performance metrics. We also discussed the DAG based task modeling and placement problems for multi-stage computations. We then outlined the dynamic issues in MEC such as, stochastic task arrival and channels and service migration followed by existing research efforts that address such issues including MDP, Q-learning, and Deep Q-learning. Finally, we discussed the challenges of video analytics in MEC from joint system-side and application-site optimization considerations. We believe that the outcomes of this survey research will benefit the future distributed computing, smart applications, and data-intensive science communities to build effective, efficient, and robust MEC environments.

REFERENCES

[1] *Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment - https://www.cosmos-lab.org/.*

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[3] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[4] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.

[5] Petar Popovski, Kasper Fløe Trillingsgaard, Osvaldo Simeone, and Giuseppe Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *Ieee Access*, 6:55765–55779, 2018.

[6] Thomas Fehrenbach, Rohit Datta, Bariş Göktepe, Thomas Wirth, and Cornelius Hellge. Urllc services in 5g low latency enhancements for lte. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6. IEEE, 2018.

[7] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[8] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[9] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[10] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. Survey on multi-access edge computing for internet of things realization. *IEEE Communications Surveys & Tutorials*, 20(4):2961–2991, 2018.

[11] Najmul Hassan, Kok-Lim Alvin Yau, and Celimuge Wu. Edge computing in 5g: A review. *IEEE Access*, 7:127276–127289, 2019.

[12] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

[13] Can Wang, Sheng Zhang, Yu Chen, Zhuzhong Qian, Jie Wu, and Mingjun Xiao. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *Proc. IEEE INFOCOM*, pages 1–10, 2020.

[14] Cheng Zhang, Hailiang Zhao, and Shuiguang Deng. A density-based offloading strategy for iot devices in edge computing systems. *IEEE Access*, 6:73520–73530, 2018.

[15] Jianbo Du, Liqiang Zhao, Jie Feng, and Xiaoli Chu. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Transactions on Communications*, 66(4):1594–1608, 2018.

[16] Mengting Liu, F Richard Yu, Yinglei Teng, Victor CM Leung, and Mei Song. Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing. *IEEE Transactions on Wireless Communications*, 18(1):695–708, 2018.

[17] Yuyi Mao, Jun Zhang, SH Song, and Khaled B Letaief. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, 16(9):5994–6009, 2017.

[18] Kang Cheng, Yinglei Teng, Weiqi Sun, An Liu, and Xianbin Wang. Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems. In *2018 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2018.

[19] Yinghao Yu, Jun Zhang, and Khaled B Letaief. Joint subcarrier and cpu time allocation for mobile edge computing. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[20] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE access*, 4:5896–5907, 2016.

[21] Nan Zhao, Ying-Chang Liang, Dusit Niyato, Yiyang Pei, and Yunhao Jiang. Deep reinforcement learning for user association and resource allocation in heterogeneous networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[22] Jing Zhang, Weiwei Xia, Feng Yan, and Lianfeng Shen. Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing. *IEEE Access*, 6:19324–19337, 2018.

[23] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung-Hoon Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2016.

[24] Jinke Ren, Guanding Yu, Yunlong Cai, and Yinghui He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[25] Fuhui Zhou, Yongpeng Wu, Rose Qingyang Hu, and Yi Qian. Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems. *IEEE Journal on Selected Areas in Communications*, 36(9):1927–1941, 2018.

[26] Feng Wang, Jie Xu, Xin Wang, and Shuguang Cui. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, 17(3):1784–1797, 2017.

[27] Mengyuan Li, Shuo Yang, Zhenduo Zhang, Jinke Ren, and Guanding Yu. Joint subcarrier and power allocation for ofdma based mobile edge computing system. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[28] Y. Chen, B. Ai, Y. Niu, Z. Zhong, and Z. Han. Energy efficient resource allocation and computation offloading in millimeter-wave based fog radio access networks. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[29] Xiaowen Cao, Feng Wang, Jie Xu, Rui Zhang, and Shuguang Cui. Joint computation and communication cooperation for mobile edge computing. In *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 1–6. IEEE, 2018.

[30] Jie Xu, Lixing Chen, and Shaolei Ren. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. *IEEE Transactions on Cognitive Communications and Networking*, 3(3):361–373, 2017.

[31] Xiaojie Zhang, Amitangshu Pal, and Saptarshi Debroy. Deep reinforcement learning based energy-efficient task offloading for secondary mobile edge systems. In *2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, pages 48–59. IEEE, 2020.

[32] Chen-Feng Liu, Mehdi Bennis, Merouane Debbah, and H Vincent Poor. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Transactions on Communications*, 67(6):4132–4150, 2019.
[33] Zhigang Wen, Kaixi Yang, Xiaoqing Liu, Shan Li, and Junwei Zou. Joint offloading and computing design in wireless powered mobile-edge computing systems with full-duplex relaying. *IEEE Access*, 6:72786–72795, 2018.
[34] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis. Performance optimization in mobile-edge computing via deep reinforcement learning. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6. IEEE, 2018.
[35] Xiaojie Zhang, Amitangshu Pal, and Saptarshi Debroy. Effect: Energy-efficient fog computing framework for real-time video processing. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 493–503. IEEE, 2021.
[36] Yunzheng Tao, Changsheng You, Ping Zhang, and Kaibin Huang. Stochastic control of computation offloading to a helper with a dynamically loaded cpu. *IEEE Transactions on Wireless Communications*, 18(2):1247–1262, 2019.
[37] Arvin Hekmati, Peyvand Teymoori, Terence D Todd, Dongmei Zhao, and George Karakostas. Optimal mobile computation offloading with hard deadline constraints. *IEEE Transactions on Mobile Computing*, 19(9):2160–2173, 2019.
[38] Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Transactions on Wireless Communications*, 14(1):81–93, 2014.
[39] Xiaojie Zhang and Saptarshi Debroy. Migration-driven resilient disaster response edge-cloud deployments. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2019.
[40] Jordi Navarrette, Subash Shankar, Xiaojie Zhang, and Saptarshi Debroy. Formal modeling and analysis of multi-rogue backoff manipulation attacks in unlicensed networks. In *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*, pages 1–7, 2020.
[41] Boyang Liu, Jin Wang, Shuai Ma, Fuhui Zhou, Yujiao Ma, and Guangyue Lu. Energy-efficient cooperation in mobile edge computing-enabled cognitive radio networks. *IEEE Access*, 7:45382–45394, 2019.
[42] Adlen Ksentini, Tarik Taleb, and Min Chen. A markov decision process-based service migration procedure for follow me cloud. In *2014 IEEE International Conference on Communications (ICC)*, pages 1350–1354, 2014.
[43] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic service migration in mobile edge-clouds. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2015.
[44] Hua Wei, Hong Luo, and Yan Sun. Mobility-aware service caching in mobile edge computing for internet of things. *Sensors*, 20(3):610, 2020.
[45] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seongjoon Kim, and Song Chong. CRAWDAD dataset ncsu/mobilitymodels (v. 2009-07-23). Downloaded from https://crawdad.org/ncsu/mobilitymodels/20090723, July 2009.
[46] Cheng Zhang and Zixuan Zheng. Task migration for mobile edge computing using deep reinforcement learning. *Future Generation Computer Systems*, 96:111–118, 2019.
[47] Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. A dynamic service migration mechanism in edge cognitive computing. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–15, 2019.
[48] Wuyang Zhang, Yi Hu, Yanyong Zhang, and Dipankar Raychaudhuri. Segue: Quality of service aware edge cloud service migration. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 344–351. IEEE, 2016.
[49] Chao-Lun Wu, Te-Chuan Chiu, Chih-Yu Wang, and Ai-Chun Pang. Mobility-aware deep reinforcement learning with glimpse mobility prediction in edge computing. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.
[50] Fei Ma, Feng Liu, and Zhen Liu. Live virtual machine migration based on improved pre-copy approach. In *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 230–233. IEEE, 2010.
[51] Xiang Sun and Nirwan Ansari. Primal: Profit maximization avatar placement for mobile edge computing. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.
[52] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 43(3):14–26, 2009.
[53] Salman Raza, Shangguang Wang, Manzoor Ahmed, and Muhammad Rizwan Anwar. A survey on vehicular edge computing: architecture, applications, technical issues, and future directions. *Wireless Communications and Mobile Computing*, 2019, 2019.
[54] Lei Liu, Chen Chen, Qingqi Pei, Sabita Maharjan, and Yan Zhang. Vehicular edge computing and networking: A survey. *Mobile networks and applications*, 26(3):1145–1168, 2021.
[55] Junhui Zhao, Qiuping Li, Yi Gong, and Ke Zhang. Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(8):7944–7956, 2019.
[56] Ke Zhang, Yuming Mao, Supeng Leng, Yejun He, and Yan ZHANG. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology Magazine*, 12(2):36–44, 2017.
[57] Sadeep Batewela, Chen-Feng Liu, Mehdi Bennis, Himal A. Suraweera, and Choong Seon Hong. Risk-sensitive task fetching and offloading for vehicular edge computing. *IEEE Communications Letters*, 24(3):617–621, 2020.
[58] Ke Zhang, Yongxu Zhu, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. Deep learning empowered task offloading for mobile edge computing in urban informatics. *IEEE Internet of Things Journal*, 6(5):7635–7647, 2019.

[59] Haixia Peng and Xuemin Shen. Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. *IEEE Transactions on Network Science and Engineering*, 7(4):2416–2428, 2020.
[60] Yi Liu, Huimin Yu, Shengli Xie, and Yan Zhang. Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, 68(11):11158–11168, 2019.
[61] Sladana Jošilo and György Dán. Joint wireless and edge computing resource management with dynamic network slice selection. *IEEE/ACM Transactions on Networking*, 2022.
[62] Qiang Liu, Tao Han, and Ephraim Moges. Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 234–244. IEEE, 2020.
[63] Xianfu Chen, Zhifeng Zhao, Celimuge Wu, Mehdi Bennis, Hang Liu, Yusheng Ji, and Honggang Zhang. Multi-tenant cross-slice resource orchestration: A deep reinforcement learning approach. *IEEE Journal on Selected Areas in Communications*, 37(10):2377–2392, 2019.
[64] Jie Feng, Qingqi Pei, F Richard Yu, Xiaoli Chu, Jianbo Du, and Li Zhu. Dynamic network slicing and resource allocation in mobile edge computing systems. *IEEE Transactions on Vehicular Technology*, 69(7):7863–7878, 2020.
[65] Mohammed Amine Togou, Ting Bi, Kapal Dev, Kevin McDonnell, Aleksandar Milenovic, Hitesh Tewari, and Gabriel-Miro Muntean. Dbns: A distributed blockchain-enabled network slicing framework for 5g networks. *IEEE Communications Magazine*, 58(11):90–96, 2020.
[66] Vincenzo Sciancalepore, Xavier Costa-Perez, and Albert Banchs. Rl-nsb: Reinforcement learning-based 5g network slice broker. *IEEE/ACM Transactions on Networking*, 27(4):1543–1557, 2019.
[67] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.
[68] X. Zhang and S. Debroy. Energy efficient task offloading for compute-intensive mobile edge applications. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.
[69] Hamed Shah-Mansouri and Vincent WS Wong. Hierarchical fog-cloud computing for iot systems: A computation offloading game. *IEEE Internet of Things Journal*, 5(4):3246–3257, 2018.
[70] Jun Guo, Heli Zhang, Lichao Yang, Hong Ji, and Xi Li. Decentralized computation offloading in mobile edge computing empowered small-cell networks. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2017.
[71] Yang Liu, Changqiao Xu, Yufeng Zhan, Zhixin Liu, Jianfeng Guan, and Hongke Zhang. Incentive mechanism for computation offloading using edge computing: A stackelberg game approach. *Computer Networks*, 129:399–409, 2017.
[72] Zehui Xiong, Jiawen Kang, Dusit Niyato, Ping Wang, and H Vincent Poor. Cloud/edge computing service management in blockchain networks: Multi-leader multi-follower game-based admm for pricing. *IEEE Transactions on Services computing*, 13(2):356–367, 2019.
[73] Zehui Xiong, Shaohan Feng, Dusit Niyato, Ping Wang, and Zhu Han. Optimal pricing-based edge computing resource management in mobile blockchain. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.
[74] Andrzej Wilczyński, Agnieszka Jakóbik, and Joanna Kołodziej. Stackelberg security games: Models, applications and computational aspects. *Journal of Telecommunications and Information Technology*, 2016.
[75] Quoc-Viet Pham, Tuan Leanh, Nguyen H Tran, Bang Ju Park, and Choong Seon Hong. Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach. *IEEE Access*, 6:75868–75885, 2018.
[76] Xiaojie Zhang and Saptarshi Debroy. Adaptive task offloading over wireless in mobile edge computing. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 323–325, 2019.
[77] Su Pan and Yuqing Chen. Energy-optimal scheduling of mobile cloud computing based on a modified lyapunov optimization method. *IEEE Transactions on Green Communications and Networking*, 3(1):227–235, 2018.
[78] Xiaolan Liu, Zhijin Qin, and Yue Gao. Resource allocation for edge computing in iot networks via reinforcement learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.
[79] Nahida Kiran, Chunyu Pan, Sihua Wang, and Changchuan Yin. Joint resource allocation and computation offloading in mobile edge computing for sdn based wireless networks. *Journal of Communications and Networks*, 22(1):1–11, 2019.
[80] Liang Huang, Suzhi Bi, and Ying-Jun Angela Zhang. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing*, 19(11):2581–2593, 2019.
[81] Yi Liu, Chao Yang, Li Jiang, Shengli Xie, and Yan Zhang. Intelligent edge computing for iot-based energy management in smart cities. *IEEE Network*, 33(2):111–117, 2019.
[82] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.
[83] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.
[84] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[85] Dinh C Nguyen, Ming Ding, Quoc-Viet Pham, Pubudu N Pathirana, Long Bao Le, Aruna Seneviratne, Jun Li, Dusit Niyato, and H Vincent Poor. Federated learning meets blockchain in edge computing: Opportunities and challenges. *IEEE Internet of Things Journal*, 8(16):12806–12825, 2021.
[86] Haftay Gebreslasie Abreha, Mohammad Hayajneh, and Mohamed Adel Serhani. Federated learning in edge computing: a systematic survey. *Sensors*, 22(2):450, 2022.
[87] Nguyen H Tran, Wei Bao, Albert Zomaya, Minh NH Nguyen, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019-IEEE conference on computer communications*, pages 1387–1395. IEEE, 2019.
[88] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. Energy efficient federated learning over wireless communication networks. *IEEE Transactions on Wireless Communications*, 20(3):1935–1949, 2020.
[89] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.
[90] Wenqi Shi, Sheng Zhou, Zhisheng Niu, Miao Jiang, and Lu Geng. Joint device scheduling and resource allocation for latency constrained wireless federated learning. *IEEE Transactions on Wireless Communications*, 20(1):453–467, 2020.
[91] Huizi Xiao, Jun Zhao, Qingqi Pei, Jie Feng, Lei Liu, and Weisong Shi. Vehicle selection and resource optimization for federated learning in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 2021.
[92] Peiying Zhang, Chao Wang, Chunxiao Jiang, and Zhu Han. Deep reinforcement learning assisted federated learning algorithm for data management of iiot. *IEEE Transactions on Industrial Informatics*, 17(12):8475–8484, 2021.
[93] Yufeng Zhan, Peng Li, and Song Guo. Experience-driven computational resource allocation of federated learning by deep reinforcement learning. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 234–243. IEEE, 2020.
[94] Huy T Nguyen, Nguyen Cong Luong, Jun Zhao, Chau Yuen, and Dusit Niyato. Resource allocation in mobility-aware federated learning networks: A deep reinforcement learning approach. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE, 2020.
[95] Nang Hung Nguyen, Phi Le Nguyen, Duc Long Nguyen, Trung Thanh Nguyen, Thuy Dung Nguyen, Huy Hieu Pham, and Truong Thao Nguyen. Feddrl: Deep reinforcement learning-based adaptive aggregation for non-iid data in federated learning. *arXiv preprint arXiv:2208.02442*, 2022.
[96] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.
[97] Jianji Ren, Haichao Wang, Tingting Hou, Shuai Zheng, and Chaosheng Tang. Federated learning-based computation offloading optimization in edge computing-supported internet of things. *IEEE Access*, 7:69194–69201, 2019.
[98] Shuai Yu, Xu Chen, Zhi Zhou, Xiaowen Gong, and Di Wu. When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network. *IEEE Internet of Things Journal*, 8(4):2238–2251, 2020.
[99] Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han. An edge network orchestrator for mobile augmented reality. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 756–764. IEEE, 2018.
[100] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
[101] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
[102] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1421–1429. IEEE, 2018.
[103] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 377–392, 2017.
[104] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131, 2018.
[105] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. Openmvg: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 60–74. Springer, 2016.
[106] Zheng Dong, Yuchuan Liu, Husheng Zhou, Xusheng Xiao, Yu Gu, Lingming Zhang, and Cong Liu. An energy-efficient offloading framework with predictable temporal correctness. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–12, 2017.
[107] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 409–421, 2020.
[108] Wuyang Zhang, Sugang Li, Luyang Liu, Zhenhua Jia, Yanyong Zhang, and Dipankar Raychaudhuri. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1270–1278. IEEE, 2019.

[109] Xiaojie Zhang, Mingjun Li, Andrew Hilton, Amitangshu Pal, Soumyabrata Dey, and Saptarshi Debroy. End-to-end latency optimization of multi-view 3d reconstruction for disaster response. In *2022 10th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 17–24. IEEE, 2022.

[110] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[111] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 767–783, 2018.

[112] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

[113] Yuvraj Sahni, Jiannong Cao, Lei Yang, and Yusheng Ji. Multihop offloading of multiple dag tasks in collaborative edge computing. *IEEE Internet of Things Journal*, 8(6):4893–4905, 2020.

[114] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[115] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[116] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2020.

[117] Jiachen Mao, Xiang Chen, Kent W Nixon, Christopher Krieger, and Yiran Chen. Modnn: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1396–1401. IEEE, 2017.

[118] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.

[119] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[120] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339. IEEE, 2017.

[121] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[122] Woo-Joong Kim and Chan-Hyun Youn. Lightweight online profiling-based configuration adaptation for video analytics system in edge computing. *IEEE Access*, 8:116881–116899, 2020.

[123] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 201–214, 2021.

[124] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 159–173. IEEE, 2018.

[125] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *arXiv preprint arXiv:1703.02529*, 2017.