# EFFICIENT ORCHESTRATION METHODS IN AIR COMPUTING USING DEEP REINFORCEMENT LEARNING

by

Barış Yamansavaşçılar

B.S., Computer Engineering, Yıldız Technical University, 2015

M.S., Computer Engineering, Boğaziçi University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2025

# ACKNOWLEDGEMENTS

In 2021, in the PhD thesis defense of one of my colleagues, Ahmet Cihat Baktır, an essential but unexpected question was asked about the philosophical part of his thesis. This was an essential and perfect question since a PhD is a doctor of philosophy in the end. However, it was unexpected as each PhD candidate focuses on the technical part of their studies. Therefore, throughout my PhD journey, I regularly thought about that considering my thesis. Now, I can say that just like the task orchestration in my technical work, the main philosophy is that there are always good alternative policies and paths in life to enhance quality. Therefore, I would like to thank all the events during my PhD journey, including good and bad ones, that have led me to this conclusion. They completed my technical work and thus made me a doctor.

I would like to thank my advisor, Prof. Cem Ersoy, and co-advisor, Prof. Atay Özgövde, for their guidance, wisdom, and especially friendship over the years. Without their technical and emotional support, I would never achieve my accomplishments.

Moreover, I want to thank my thesis committee members, Prof. Tuna Tuğcu and Prof. Gülfem Işıklar Alptekin. Through their invaluable and constructive contributions, my thesis could reach this quality. I would also like to thank my jury members, Prof. Tolga Ovatman and Prof. Can Özturan, for their time, participation, and suggestions.

Furthermore, I would like to thank the NETLAB team and CmpE members, who have always helped and supported me under different circumstances and for various reasons.

Finally, I want to thank my sister and my mother for their endless support. Especially my mother, Nuray Özyavaş, who has always been on my side.

The visuals produced within the scope of this thesis and whose copyright has been transferred to the publisher have been used in the thesis book under the publisher's publication policy on the reuse of texts and graphics created by the author, as stated on the publisher's official website.

# ABSTRACT

## EFFICIENT ORCHESTRATION METHODS IN AIR COMPUTING USING DEEP REINFORCEMENT LEARNING

After the proliferation of smart devices and diverse Internet of Things (IoT) requirements, we observe the dominance of cutting-edge applications with ever-increased user expectations in terms of mobility, pervasiveness, and real-time response. Over the years, to meet the requirements of those applications, cloud computing provides the necessary capacity for computation, while edge computing ensures low latency. However, these two essential solutions would be insufficient for next-generation applications due to the highly dynamic load. Therefore, we put forward a novel, next-generation paradigm called air computing that presents a dynamic, responsive, and high-resolution 3D computation environment for all spectrum of applications. To this end, we first investigate efficient task orchestration through deep reinforcement learning (DRL) considering terrestrial resources via edge computing. Afterwards, to evaluate the corresponding air computing environments correctly, we design and develop our novel discrete event simulator, AirCompSim. Subsequently, we focus on the unknown user locations in an infrastructure-less environment in which users cannot connect to any communication device or computation-providing server, which is essential to task offloading in order to achieve the required quality of service. Thus, we propose a novel DRL-based scheme, DeepAir, which uses four main phases including sensing, localization, resource allocation, and multi-access edge computing. Performance evaluation results show that our DRL-based methods outperform the benchmark methods including heuristics and fuzzy logic. Thus, problems related to dynamic capacity enhancement and orchestration can be overcome in air computing environments.

# ÖZET

## HAVADA HESAPLAMADA DERİN PEKİŞTİRMELİ ÖĞRENME KULLANARAK ETKİLİ ORKESTRASYON YÖNTEMLERİ

Akıllı cihazların yaygınlaşması ve çeşitli Nesnelerin İnterneti (IoT) gereksinimlerinin ortaya çıkmasıyla birlikte, hareketlilik, yaygınlık ve gerçek zamanlı yanıt açısından kullanıcı beklentilerinin giderek arttığı, ileri düzey uygulamaların hakimiyetine tanık olmaktayız. Geçtiğimiz yıllarda, bu uygulamaların gereksinimlerini karşılamak amacıyla, bulut hesaplama gerekli hesaplama kapasitesini sağlarken, uçta hesaplama ise düşük gecikmeyi temin etmiştir. Ancak, yüksek dinamik yük nedeniyle ortaya çıkan hesaplama ve iletişim darboğazları, bu iki temel çözümün yeni nesil uygulamalar için yetersiz kalmasına sebep olabilmektedir. Bu nedenle, tüm uygulama çeşitleri için dinamik, duyarlı ve üç boyutlu bir hesaplama ortamı sunan yeni nesil bir paradigma olan havada hesaplamayı ileri sürmekteyiz. Bu doğrultuda, öncelikle, karasal kaynakları kullanarak uçta hesaplama aracılığıyla derin pekiştirmeli öğrenme (DRL) ile verimli görev orkestrasyonunu araştırdık. Ardından, havada hesaplama ortamlarını doğru bir şekilde değerlendirebilmek amacıyla AirCompSim adınıda yenilikçi ayrık olay simülatörümüzü geliştirdik. Sonrasında, kullanıcıların herhangi sunucuya bağlanamadığı altyapısız bir ortamda, kaliteli görev aktarımını sağlamak için bilinmeyen kullanıcı konumlarına odaklandık. Sonuç olarak, algılama, konumlandırma, kaynak ayırma ve çoklu erişimli uçta hesaplama olmak üzere dört temel aşamadan oluşan DeepAir adında, DRL tabanlı yeni bir yöntem önerdik. Başarım değerlendirme sonuçları, DRL tabanlı yöntemlerimizin, sezgisel ve bulanık mantık yaklaşımları dahil olmak üzere ilgili karşılaştırma yöntemlerinden üstün olduğunu göstermektedir. Sonuç olarak, havada hesaplama ortamlarında dinamik kapasite artırımı ve orkestrasyon ile ilgili sorunlar aşılabilmektedir.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| $C_m$ | Required CPU cycle for task $W_m$ |
| $d_{m,n_d}$ | Distance between user $m$ and detector UAV $n_d$ |
| $d_n(t)$ | Flight distance of UAV $n$ at time $t$ |
| $D_m$ | Size of task $W_m$ |
| $f_m$ | Computational capacity of user $m$ |
| $f_{n_s}$ | Computational capacity of serving UAV $n_s$ |
| $g_{m,n_d}$ | Channel power gain between user $m$ and detector UAV $n_s$ |
| $h_{m,n_d}(t)$ | Channel gain between user $m$ and detector UAV $n_s$ after path loss |
| $H(t)$ | Cumulative signal strength on detector UAV $n_s$ at time $t$ |
| $M$ | Number of users |
| $N_d$ | Number of detector UAVs |
| $N_s$ | Number of serving UAVs |
| $r_{n_d}$ | Horizontal radius of a detector UAV |
| $r_{n_s}$ | Horizontal radius of a serving UAV |
| $t_{w_m}$ | Total delay for task $W_m$ |
| $T_m^{max}$ | Maximum tolerable delay of task $W_m$ |
| $T_m^{loc}$ | Computation delay of user $m$ |
| $T_{n_s}^{UAV}$ | Computation delay of serving UAV $n_s$ |
| $T_t^{UAV}$ | Total delay including queueing and computation at serving UAV $n_s$ |
| $T_q$ | Queueing delay at serving UAV $n_s$ |
| $T_{m,n_s}$ | Transmission delay between user $m$ and serving UAV $n_s$ |
| $v_{m,n_s}$ | Data rate (bit/sec) between user $m$ and serving UAV $n_s$ |
| $W_m$ | Task of user $m$ |
| | |
| $\alpha_{w_m}$ | Task completion variable for user $m$ |
| $\beta m, n_s$ | Offloading decision variable between user $m$ and serving UAV $n_s$ |

| | |
|---|---|
| $\lambda_m$ | Arrival rate of task $W_m$ |
| $\vartheta_n(t)$ | Flight angle of UAV $n$ at time $t$ |

# LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| ARAN | Aerial Radio Access Network |
| AR | Augmented Reality |
| CAPEX | Capital Expenditures |
| CF | Community Flying |
| DNN | Deep Neural Network |
| DRL | Deep Reinforcement Learning |
| DQN | Deep Q-Network |
| FACOM | Future Aerial Communications |
| FL | Federated Learning |
| GEO | Geosynchronous Equatorial Orbit |
| HAP | High Altitude Platform |
| IoT | Internet of Things |
| LAN | Local Area Network |
| LAP | Low Altitude Platform |
| LEO | Low Earth Orbit |
| MAN | Metropolitan Area Network |
| MAR | Mobile Augmented Reality |
| MDP | Markov Decision Process |
| MEC | Multi-Access Edge Computing |
| ML | Machine Learning |
| NFV | Network Function Virtualization |
| OPEX | Operating Expenses |
| QoS | Quality of Service |
| QoE | Quality of Experience |
| QoL | Quality of Life |
| SAGIN | Space-Air-Ground Integrated Network |
| SDN | Software-Defined Networks |

| | |
|---|---|
| SLA | Service Level Agreement |
| UAV | Unmanned Aerial Vehicle |
| UE | User Equipment |
| WAN | Wide Area Network |

# 1. INTRODUCTION

In order to meet the stringent demands of fully connected, intelligent, and computation intensive applications with low latency support, vertical networking solutions provide many opportunities [1]. Especially, considering the number of Internet of Things (IoT) connections which are estimated as 30.9 billion in 2025 [2], vertical networking would be crucial for the seamless coverage and dense connection capabilities. Moreover, since the connection of devices that have separate requirements must be processed heterogeneously to ensure Quality of Experience (QoE), reliable computation of different task types is critical in the next-generation networking systems [3].

The diverse requirements of next-generation applications are hard to satisfy with well-known practices [4]. Therefore, the deployment of Unmanned Aerial Vehicles (UAVs) as Low Altitude Platforms (LAP), airplanes as High Altitude Platforms (HAP), and Low Earth Orbit (LEO) satellites are legitimate candidates for future networks. Hence, they can provide low latency, high computation capability, reliability, and availability in order to satisfy the requirements of different applications. These properties are specifically important for the processing of the corresponding tasks of those applications in terms of content caching, resource allocation, task offloading, and extreme mobility.

Although edge computing provides promising results in urban settings in the short-run, reaching genuine ubiquitous execution of real-time, computationally intensive novel applications will require further approaches. Air components with computational processing units in this respect harmonize traditional terrestrial edge computing with a wide range of air technologies to obtain a robust, high-capacity computational infrastructure that embraces urban, suburban, and rural scenarios. Since air layers and the corresponding air components would be an essential part of the next-generation networking systems, we call this next-generation computation paradigm as air computing.

The main advantage of air computing as a new computational paradigm is that it can meet the requirements of dynamically changing QoS needs. Since infrastructure-based fixed capacity may not satisfy application requirements during an event, different air platforms can be directed to the corresponding location through air computing. Thus, the QoE of users would not be affected even though the capacity of the infrastructure can be exceeded thanks to the coordination between different components in air computing. This coordination can be through computational offloading, content caching, coverage, and mobility. Moreover, it can be performed in urban, suburban, and rural areas.

Since air computing includes both terrestrial servers and air components, edge and cloud computing related problems are also in the scope of the corresponding research issues. Accordingly, the orchestration of the task offloading, which is one of the most significant research issues that affects the QoS of the applications in the corresponding environment, is a common research problem in edge/cloud computing and therefore in air computing. In order to investigate it properly, edge computing specific solutions can be investigated first since it covers a smaller area than an air computing environment. Moreover, as related literature and simulators are mature in edge computing, developing a valid orchestrator can also provide an essential insight to apply specific methods in dynamic environments. Afterwards, the corresponding findings can be enhanced into more generic scenarios in air computing.

Since air computing is an emerging paradigm, a corresponding simulation environment in which movement of UAVs and mobile users, deployment of edge/cloud servers, and the related dynamic issues could not be developed properly. Therefore, a simulation environment that can meet the required expectations should be used in order to conduct valid air computing experiments. Hence, we can clearly observe the advantages of air computing in terms of the task success rate, mobility of UAVs, and dynamic capacity enhancement.

Considering the vast environment that air computing is able to operate, and

the dynamic capacity enhacement capabilities of the air vehicles such as UAVs, the provision of the connectivity of users is a significant challange. Therefore, ensuring a required QoS of user applications while user locations would not be known by the network can be applied on many use cases such as a disaster site, wilderness, or a rural area. Thus, a successfully deployed approach can enhance user satisfaction through air computing.

### 1.1. Thesis Contributions

The improvements in the edge computing technology pave the road for diversified applications that demand real-time interaction. However, due to the mobility of the end-users and the dynamic edge environment, it becomes challenging to handle the task offloading with high performance. Moreover, since each application in mobile devices has different characteristics, a task orchestrator must be adaptive and have the ability to learn the dynamics of the environment. For this purpose, we developed a deep reinforcement learning based task orchestrator, DeepEdge, which learns to meet different task requirements without needing human interaction even under the heavily-loaded stochastic network conditions in terms of mobile users and applications. Given the dynamic offloading requests and time-varying communication conditions, we successfully model the problem as a Markov process and then apply the Double Deep Q-Network (DDQN) algorithm [5] to implement DeepEdge.

Even though many studies and open research issues exist for air computing, there are limited test environments that cannot satisfy the performance evaluation requirements of the dynamic environment. Therefore, in this thesis, we introduce our discrete event simulator, AirCompSim, which fulfills an air computing environment considering dynamically changing requirements, loads, and capacities through its aerial mobile servers, and modular structure. To show its capabilities, a dynamic capacity enhancement scenario is used to investigate the effect of the number of users, UAVs, and requirements of different application types on the average task success rate, service time, and server utilization. The results demonstrate that AirCompSim can be used

for experiments in air computing.

One of the existing dynamic problems in air computing is the unknown user locations in an infrastructure-less environment in which users cannot connect to any communication device or computation-providing server, which is essential for task offloading in order to achieve the required QoS. Therefore, in this thesis, we investigate this problem thoroughly and propose a novel deep reinforcement learning (DRL) based scheme, DeepAir. DeepAir has four main phases including sensing, localization, resource allocation, and multi-access edge computing (MEC) to provide the corresponding QoS requirements for the offloaded tasks without violating the maximum tolerable delay. To this end, we use two types of UAVs including detector UAVs, and serving UAVs. We utilize detector UAVs as DRL agents which ensure the sensing, localization, and resource allocation phases. On the other hand, we utilize serving UAVs to provide MEC features. Our experiments show that DeepAir provides higher task success rates by deploying fewer detector UAVs in different scenarios with different numbers of users and user attraction points compared to benchmark methods.

The contributions of this thesis are listed as follows.

- We introduce air computing which is the next-generation computation paradigm. We define its components including terrestrial, LAP, HAP, and LEO layers and investigate them thoroughly.
- We analyze recent studies that focus on edge computing, UAVs, and other air components in the literature and compare them with air computing in order to show its concrete advantages and possible solutions that cannot be offered by traditional networking paradigms.
- We detail the scenarios for air computing that improve the QoS and QoE for end-users. Especially, we focus on several use cases such as natural disasters, real-time video, and outdoor activities. These use cases require seamless connection, intelligent routing, and dynamic capacity enhancement which may not be met by traditional computing schemes.

- DeepEdge can handle the diverse needs of different applications by offloading their tasks into the appropriate edge or cloud server.
- Our DeepEdge orchestrator has learning capability so that it adapts to heavily-loaded environments in terms of mobile users without human interaction. Since studies that apply DRL in the literature do not investigate the effect of the number of mobile users on a decision-making system appropriately, we explore the limits of our DRL agent by deploying varying amount of mobile devices in our simulations.
- Even though the stochastic nature of the edge computing environment due to different application types and user mobilities cause the delayed action effect, we can model the problem as a Markov process rather than applying policy gradient or semi-Markov process that are used by studies to provide a policy for task offloading. Thereby we can implement the DDQN algorithm for orchestration.
- We present AirCompSim, which provides air computing simulations considering different user/application profiles, edge servers, and air vehicles, especially UAVs. Therefore, to the best of our knowledge, AirCompSim has unique features regarding the existing simulators.
- AirCompSim supports dynamic scenarios through its scenario module in which deployed entities would be failed, or existing policies can be updated. Moreover, it provides a DRL-based event mechanism so that a researcher can implement a novel flying or offloading policy in this manner. Furthermore, based on the event mechanism, new modules based on different methods can easily be incorporated in AirCompSim.
- In DRL-based UAV studies, a subset of four categories, including (1) sensing, (2) localization, (3) resource allocation, and (4) UAV-assisted MEC, are mainly considered separately in most of the cases [6]. In this thesis, we take all of these categories into account and integrate as phases of our novel approach.
- In order to provide such a system, we developed a DRL-based multi-agent scheme, DeepAir, which is a novel approach that iteratively detects user locations in the environment using unconnected users' additive RSSI as the reward. DeepAir performs this operation iteratively, sending a single agent (detector UAV) to

the environment for each iteration, since the number of user concentrated areas cannot be known apriori in most of the cases.

- We examined the relationship between the detector and serving UAVs by conducting various experiments with different scenarios. As a result, the performance of coordinated utilization of sensing, localization, resource allocation, and UAV-assisted MEC phases on the overall task success rate is investigated.

## 1.2. Thesis Outline

The thesis is organized as follows. Chapter 2 provides the background information and presents the overview of related studies in the literature regarding edge computing, coordination and deployment of air units in different settings, and deep reinforcement learning. We detail every aspect of air computing in Chapter 3 considering its concrete advantages and possible solutions that cannot be offered by traditional networking paradigms. In Chapter 4, we present DeepEdge, our DRL-based task orchestrator in an highly dynamic edge computing environment. We explain the details of the development and the corresponding performance evaluation of our simulator, AirCompSim, in Chapter 5. We discuss the unknown user location problem and represent DeepAir, a novel multi-agent DRL-based scheme in Chapter 6. Finally, in Chapter 7, we conclude this thesis.

# 2. BACKGROUND AND RELATED WORKS

In this chapter, we provide a brief overview of edge computing, utilization of air units, and fundamentals of DRL. Moreover, we discuss the related works regarding the challanges that researchers encounter. Hence, we aim to highlight the original contributions of this thesis.

## 2.1. Edge Computing

Since several application types such as image rendering, video editing, and simulation require intensive computation, cloud computing is proposed as the possible solution regarding CPU and battery constraints of end-devices [7]. However, with the proliferation of versatile devices and corresponding applications known as the IoT, cloud computing could not meet the low latency requirements [8]. As a result, several computation architectures including Cloudlet [9], MEC [10], and Fog Computing [11] have emerged. In the literature, these architectures are named under the umbrella term edge computing [12] which is also used in this thesis. The general architecture of edge computing is shown in Figure 2.1.

The main idea behind the edge computing is that processing the computation-intensive tasks, which cannot be processed in the end-device due to CPU and battery limitation, in the suitable server in the Local Area Network (LAN). Moreover, the concept can be enhanced to Metropolitan Area Network (MAN) due to scarce capacity [8]. In that case, the most suitable server in the MAN is selected for the corresponding application. Furthermore, cloud servers can also be used with edge computing for latency insensitive applications in order to serve more users. In general, edge computing is used for many application domains such as agriculture, healthcare, smart home, robotics, data processing, video analytics, and virtual reality [13, 14].

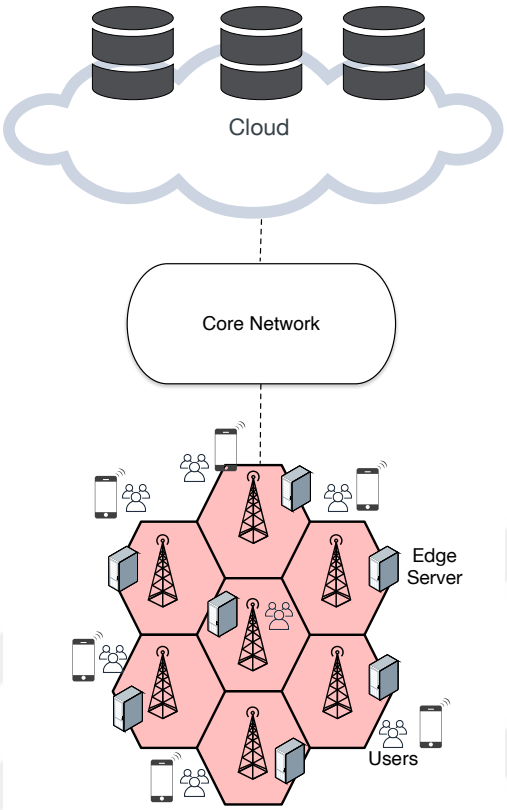



Figure 2.1. Edge computing.

### 2.1.1. Recent Edge Computing Studies

Even though vertical networking adds another dimension to the current network infrastructure in air computing, the essential element in an urban area would be edge computing as it is deployed widely. As a result, determining a profitable edge strategy is still important [15]. In this section, we investigate the most recent edge computing studies considering resource allocation, task offloading, edge caching, and energy efficiency issues. Moreover, we give an evaluation of selected edge studies in Table 2.1 and analyze them considering the benefits of the air computing.

2.1.1.1. Resource Allocation. Along with the computation/task offloading, resource allocation is one of the primary research issues in edge computing because of the new generation application requirements including transmission bandwidth, latency, energy consumption, and reliability [15,24]. In [16], authors perform intelligent task execution

Table 2.1. Evaluation of selected edge studies.

| Study | Category | Goal | Solution | Open Issue | Benefits of Air Computing |
|---|---|---|---|---|---|
| [16] | Allocation | Providing both computation efficiency and cost effectiveness to accelerate DNN-based task acceleration in the MEC | A joint method by a self-adaptive DNN partition with cost-effective resource allocation | Only a single edge server is used | With multiple components, it can increase the capacity |
| [17] | Allocation | Maximizing the sum offloading rate, quantifying MEC throughput, and minimizing the migration cost | They relax the corresponding binary variables in the original problem to overcome non-convex issue | Energy efficiency regarding edge servers is not considered | 3D network structure would alleviate the handover problem |
| [18] | Caching | A cache-assisted multi-user MEC mechanism | They formulate a non-linear programming problem which involves a joint optimization | There is no mobility | UAVs can be helpful for caching important contents |
| [19] | Caching | They investigate the cooperation problem of edge nodes in MEC | They use Lagrangian multipliers and then a distributed optimization algorithm | There is no mobility and handover consideration | Vertical networking would increase capacity for caching based on air components. |
| [20] | Offloading | A priority-differentiated offloading strategy that considers the stringent QoS requirements of mission-critical services | They use Lyapunov optimization for priority-differentiation | Only a single edge server is used | Mission-critical applications can be handled very well regarding multiple components of air computing. |
| [21] | Offloading | Task offloading for multi-user and multi-vehicle in vehicular MEC | They propose a dynamic incentive mechanism | Mobility model is not clear | Seamless connectivity can alleviate the problems in vehicular MEC systems |
| [22] | Energy | Minimizing both the energy consumption and task processing delay of the mobile devices | They propose an evolutionary algorithm that can efficiently find a representative sample of the best trade-offs | There is no mobility and cloud consideration | Multiple air components can alleviate the trade-offs between offloading and energy consumption |
| [23] | Energy | Energy-efficient offloading for DNN based smart IoT systems | They propose a swarm optimization algorithm for energy-efficient offloading strategy | There is no mobility | With the offloading to different nodes in the air, it may reduce energy consumption |

using Deep Neural Network (DNN) partitioning regarding heterogeneous edge server capacities. They propose a joint method considering cost-effective resource allocation and self-adaptive DNN partition in order to provide collaborative computation between IoT devices and edge servers. Lieang et al. focus on the challenge of handover between base stations considering the management of computation and radio resources [17]. Therefore, the goals in this study consist of maximizing the throughput and minimizing the handover cost. Chen et al. propose a cache-assisted multi-user MEC mechanism considering to cache executive codes of tasks proactively [18]. Their goal was to reduce the task execution delay and energy consumption of users. Xia et al. consider the problems including resource allocation on demand for limited edge servers,

and developing heterogeneous task offloading strategies [25]. To this end, they implement an online distributed optimization algorithm based on game theory to perform optimal offloading and energy harvesting decisions. Bahreini et al. develop an auction-based mechanism by addressing the resource allocation and monetization challenges in MEC [26]. They focus on the dynamic provisioning of computing resources since the tasks of users are heterogeneous. On the other hand, Roostaei et al. investigate Stackelberg game based distributed algorithm [27] in order to dynamically allocate and price edge resources [28]. Zhao et al. develop a hybrid system considering beamforming and resource allocation [29]. They benefit from the advantages of mmWave communications indicated in [30] in order to optimize beamforming vectors at users and base stations to minimize the maximum delay. In [31], Wang et al. focus on the challenges of limited capacity of edge servers in a heterogeneous multi-IoT environment. To address this issue, they propose a weighted cost model which is solved by a Deep Reinforcement Learning (DRL)-based algorithm considering dynamic and stochastic edge computing environments.

2.1.1.2. Task Offloading. Task offloading is the most crucial issue in edge computing regarding QoS of IoT devices and their corresponding applications [8]. The performance of task offloading is generally evaluated with other metrics such as energy efficiency and edge caching hit [32]. Peng et al. used three constrained multiobjective algorithms considering time and energy consumption in order to solve the computation offloading problem in an edge environment [33]. Feng et al. on the other hand focus on different requirements of mission-critical applications regarding their priorities [20]. To this end, they benefit Lyapunov optimization considering the energy consumption of resources [34]. However, they use only a single edge server in their experiments. Chen et al. on the other hand, develop a system that jointly optimizes task assignment and offloading scheduling in order to minimize maximum completion delay [35]. For this system, they also consider different communication and computation capabilities. Xue et al. develop a dynamic incentive mechanism to investigate the problem of the task offloading and resource allocation [21]. They consider a multi-user and multi-vehicle system. They use the Stackelberg game for the interaction between MEC service provider and user

equipments (UEs). In [36], authors focus on data caching and computing offloading in a two-tier MEC environment. They consider the constraints of tasks in terms of the delay and the minimization of the network cost at user. However, they do not include mobility for the users and cloud option for offloading. Xu et al. investigate the performance of task offloading in High-Speed Railways (HSRs) considering proper data routing paths for each offloaded task [37]. Since handovers are frequent in HSRs, they focus on how frequent handovers in uplinks and downlinks affect offloading. In [38], Zhang et al. focus on autonomous manufacturing by considering their delay sensitivity. To this end, they propose a risk-aware cloud-edge computing framework by developing a branch-and-check approach for solving the nonlinear programming problem. Yang et al. propose a machine learning (ML) solution to solve the offloading problem in MEC [39]. They train and jointly optimize the offloading decisions and resource allocation. Li et al. focus on caching techniques to optimize QoS in MEC [40]. They propose three algorithms to forecast the next executing task. Moreover, they jointly consider cache hit rate and load balance of edge servers.

2.1.1.3. Energy Efficiency. Since the battery capacity of IoT devices is limited, and service providers would like to lower their expenses regarding power consumption of edge servers, energy efficiency is an important research topic in edge computing. In [22] authors focus on minimizing the energy consumption and task processing delay in this study. For that purpose, they develop an evolutionary algorithm that finds the best trade-offs between energy consumption and processing delay. Song et al. consider minimizing the energy consumption of mobile devices when executing corresponding tasks at satellites [41]. Moreover, their model provides MEC services using LEOs for mobile devices in disaster areas. Chen et al. investigate energy-efficient offloading considering QoS requirements of DNN-based smart IoT systems [23]. To this end, they design a self-adaptive particle swarm optimization algorithm for the corresponding energy-efficient offloading strategy. In [42], authors develop a multi-armed bandit algorithm to provide a solution for the server selection problem in edge computing. They define corresponding reward and cost terms considering the energy and required time in offloading rounds. Zhou et al. focus on energy-efficient service migration in considering

MEC-enabled dense cellular networks [43]. They formulate the service migration process as a Mixed-Integer Nonlinear Programming problem. They also use the Lyapunov optimization technique to decouple the migration process.

## 2.2. Utilization of Air Units

Air units consist of three main air platforms including LAP, HAP, and LEO. Therefore we investigated the related works considering each platform and their corresponding issues.

### 2.2.1. LAP

Since providing Line of Sight (LoS) links has many advantages in terms of connectivity, service provision, and latency, UAVs have been deployed in many areas especially remote locations. However, this deployment also brings its own issues including mobility management, UAV networking management, and flight formation [44, 45]. To this end, we categorize and evaluate these issues under trajectory planning, task offloading, placement of UAVs, and energy consumption.

2.2.1.1. Trajectory Planning. In order to provide efficient on-demand services, trajectory planning is crucial for UAVs [46, 47]. Moreover, optimization of the pre-defined paths based on the dynamic events is also critical for the performance of UAVs in terms of QoS [48]. Zhao et al. investigate a proactive mobility management solution for users' trajectories in order to deploy UAVs dynamically in the network [49]. To this end, they propose a distributed learning framework in which edge servers are considered as local data owners that collect connection data. Wang et al. aim at minimizing the energy consumption of users in the network by considering the resource allocation and trajectory of UAVs [50]. They propose two solutions: (1) convex optimization based trajectory control algorithm to minimize energy consumption, and (2) DRL-based trajectory control algorithm for real-time decisions. Similarly, authors in [51] propose that the trajectory of UAVs can be approximated using traditional convex

optimization approaches and discrete variables. Liu et al. optimize UAV trajectories considering energy consumption [52]. They formulated the problem as Markov Decision Process (MDP) and proposed DRL with a double q-network. Wang et al. proposed a multi-UAV communication system for 6G in which they consider UAV trajectories and radio resource scheduling [53].

2.2.1.2. Task Offloading. One of the most important motivations for the deployment of UAVs is the computation rate maximization of applications by using task offloading [54, 55]. Through task offloading via Line of Sight, the burden on the edge servers would be alleviated and required QoS can be provided. Seid et al. study on minimization of the computation costs in terms of energy consumption and computation delay [56]. They propose a Multi-Agent Deep Reinforcement Learning (MADRL)-based approach in a multi-UAV enabled IoT edge network using a single centralized SDN controller. In [57], the authors propose an offloading system in which users can perform partial offloading regarding UAVs and MEC servers. They formulate the problem as a maximization problem and use the principles of Prospect Theory [58]. Haber et al. on the other hand focus on mission-critical applications that require ultra-reliable low-latency computation offloading [59]. They use UAVs considering the maximization of served request rate and the optimization of UAVs' positions with the offloading decision. Zhan et al. propose a framework for a multi-UAV enabled MEC system in order to maximize the number of served IoT devices regarding computation offloading and resource allocation [60]. Zhao et al. proposed a collaborative task offloading approach in a multi-UAV multi-MEC system considering energy consumption, and UAV trajectory [61]. For this purpose, they use a cooperative MADRL method in which the policy gradient algorithm is utilized. Diao et al. investigate the usage of UAVs as relay nodes considering emergency conditions [62]. Moreover, they consider energy consumption minimization by optimizing offloading and scheduling. Zeng et al. focus on multi-UAV assisted MEC environment in order to maximize revenue of ground users considering the offloading of multi-user scenarios [63]. To this end, they take different time sensitivity of each user task into account and construct a two-hierarchy Stackelberg game framework model. In this model, UAVs are considered as leaders

performing location deployments and users are the followers that perform offloading selections. In [64], Shi et al. propose a model-free DRL offloading scheme based on considering the dynamic channel state, renewable energy utilization, UAVs trajectory, and tasks offloading ratio. To perform this, they use Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm.

2.2.1.3. Energy Consumption. Even though energy consumption is considered as a performance metric that is evaluated in studies along with the other issues such as task offloading, and trajectory planning, some studies take energy efficiency into account as the main problem. Ji et al. consider nonorthogonal and orthogonal multiple access modes for a UAV-assisted MEC system and focus on weighted-sum energy consumption [65]. To this end, they propose alternating iterative algorithms in order to optimize UAV trajectory and resource allocation. The goal of Li et al. is to create a model for UAV-assisted MEC by considering energy-efficient UAV trajectory design and optimized computation offloading [66]. They also consider partial offloading in this study. Chen et al. focus on ultra-dense networks considering the resource allocation problem by maximizing energy efficiency [67]. They used UAVs as flying base stations (BS) and utilized DQN as the solution technique. Liu et al. aim to minimize the energy consumption of users by optimizing relay and computing features of UAVs [68]. They use an iterative algorithm to solve the non-convex problem. Li et al. investigate the optimization of energy efficiency in an UAV-assisted network in which UAVs are used for an energy station and MEC server [69]. Their goal is to maximize average energy efficiency in the network by considering user transmit power, user computing frequency, UAV transmit power, bandwidth allocation, and UAV trajectory planning. They use a proximal policy optimization algorithm as a DRL agent.

2.2.1.4. Placement. The placement of UAVs is substantial as coverage provides connectivity that enhances the network capacity in terms of the data transmission and computation [70]. Moreover, their utilization in poorly covered terrestrial regions would increase end-users QoE [71]. Therefore, placement optimization would be crucial for

the performance of the UAV-based network along with the trajectory planning [72,73].

In [74], Lui et al. use actor-critic methods in DRL in order to provide connectivity between UAVs so that they can cover required areas to improve QoS. Wang et al. optimize the placement of the UAVs considering the offloading decision and resource allocation in a multi-UAV-enabled MEC environment [75]. They propose a two-layer optimization method to solve the problem. On the other hand, Yuan et al. focus on the dynamic placement of UAVs in a vehicular network [76]. They utilize the actor-critic DRL approach to carry out real-time UAV placement. They also consider UAVs' flying range, communicating range, and energy resources. Abdelhakam et al. focus on strong co-channel interference that can be caused by line-of-sight channels between UAVs and the ground terminals [77]. To solve this problem, they propose a Coordinated Multi-Point (CoMP) technique considering the deployment of UAVs in a multi-UAV-assisted IoT network.

### 2.2.2. HAP Components and LEO Satellites

Considering the limited battery energy and cell-based coverage of UAVs, HAP and LEO layers provide important advantages regarding long-distance communication, energy consumption, and management opportunities. Moreover, their performance would not be affected by the weather conditions due to their high altitudes as 10 - 30km for HAP components, and 160 - 200km for LEO satellites.

The main goal of the deployment of the HAP components is to provide connectivity such as Internet access, and to perform as a controller node for the UAVs [78,79]. However, in certain circumstances, such as a congestion in the lower layers, they can be used as an edge computing server or a relay node. Since their coverage is on a regional scale, UAVs that can be considered as dynamic cells can reach the corresponding resources in a particular area via HAP components. Even though this is one of the reasons that the deployment of HAP components is generally in urban and suburban areas, maintenance issues based on energy consumption also cause an important

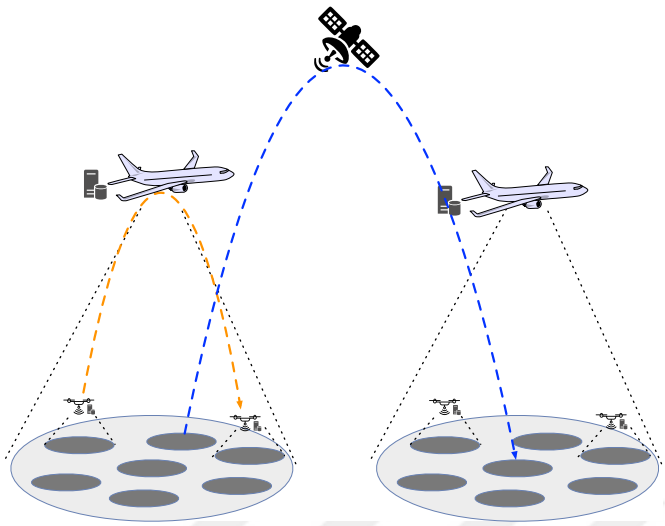


Figure 2.2. HAPs and LEO satellites provide regional and inter-regional access for LAP components.

restriction for their deployment in rural areas.

On the other hand, the essential use case of the LEO satellites is to meet the coverage problems of rural areas where infrastructure for the communication is insufficient. They can be deployed for months thanks to their efficient energy consumption, however they are not recoverable after their deployment. Moreover, they cannot be used for low latency applications due to their high propagation delay. Therefore, they can be used as a complementary resource for urban and suburban areas in order to get access to cloud computing solutions in Wide Area Network (WAN). Besides, exploiting services in continental or beyond regional distances would be more efficient using LEO satellites since terrestrial nodes may cause high latency [80]. Figure 2.2 depicts reaching regional and inter-regional resources using HAP and LEO layers.

It is important to note that when the LEO layer is exploited, SAGIN is the term that is generally used by studies to indicate the corresponding communication system [81]. In [82], Tang et al. developed an efficient offloading mechanism for SAGIN. They benefited from the communication between the LEO satellite network, LAP vehicles, and terrestrial resources. To minimize the total delay and to handle the high mobility of nodes, they proposed a deep reinforcement learning traffic offloading ap-

proach. Chen et al. [83] benefit from satellite constellations to apply Federated Learning considering communication overhead and privacy issues. They use four different modes including remote cloud learning, onboard satellite learning, federated learning with data sharing, and federated learning with no data sharing to evaluate the performance of their system. On the other hand, Guo et al. focus on service coordination between different air layers in SAGIN [84]. Therefore they separate the requirements of the environment using three service coordination scenarios: (1) fine-grained, (2) medium-grained, and (3) coarse-grained. In the fine-grained scenario, the network in the air is used as complementary regarding the need for the terrestrial network and ubiquitous coverage. Considering the delay-sensitive applications, coordination of the data processing and data communication services is provided by using medium-grained coordination. Finally, mobility and the corresponding service migration are ensured using coarse-grained service coordination.

In [85], Zhou et al. investigate dynamic scheduling problem in task offloading considering SAGIN environment. They deploy UAVs as flying gateways in order to perform the offloading decision. Considering the dynamic environment, they formulate the corresponding problem as an MDP and then apply linear programming. Similarly, Cheng et al. focus on computational offloading in SAGIN considering energy and computation constraints [86]. In their design, UAVs provide edge computing while satellites ensure access to cloud computing. To learn the optimal policy in a dynamic environment regarding large action space and mobility, they use an actor-critic DRL algorithm. In [87], Zhang et al. analyze the architecture and corresponding application scenarios of satellite MEC. They propose network function virtualization and cooperative task offloading methods in order to integrate computing resources and improve the efficiency regarding delay and energy consumption.

Resource allocation and controller placement problems are also essential for LEO studies. In [88], Chen et al. examine the dynamic assignment and placement of controllers in SDN-based LEO satellite networks. They consider two challenges including highly dynamic topology, and randomly fluctuating load. To this end, they take

propagation and queueing delays into account and then formulate the adaptive controller placement and assignment problem considering management costs. On the other hand, Zhang et al. investigate resource allocation in Non-Orthogonal Multiple Access (NOMA) terrestrial-satellite networks in which terrestrial and satellite components use the same spectrum for the communication [89]. Since the original optimization problem is non-convex, they divided the original problem into three subproblems including user association, bandwidth assignment, and power allocation. In [90], Dahrouj et al. focus on the user scheduling in integrated satellite-HAPs-ground networks considering user-connectivity, backhaul, and power constraints. To this end, they propose a deep neural network driven optimization for the user scheduling policies. Their online approach outperforms traditional model-based optimization methods which fail to meet the QoS requirements.

# 3. AIR COMPUTING

A wise deployment of terrestrial servers and air components for the computational needs will change the traditional methods such as edge and cloud computing. Therefore, in this chapter, we investigate the computing opportunities that would be the result of the intelligent communication between terrestrial servers and air vehicles which we call air computing. These opportunities manifest themselves as an enhancement of QoS and QoE considering both communication requirements and end-user needs [91, 92]. Moreover, as shown in Figure 3.1, we believe that the coordination between devices and different communication mediums through the air would open new research challenges that will shape the future of the Internet.

Even though there are many use cases for air computing as we investigate throughout this survey, we believe that air computing would be especially useful for dynamic capacity enhancement scenarios considering the battery and energy requirements of air vehicles. Therefore, air computing is beneficial in cases where the service load exceeds the capacity of the fixed infrastructure as we indicated in Section 1.1. This load can be triggered by dynamic events such as sport performances and concerts. Also in the case of a disaster existing infrastructure may be completely destroyed and air vehicles become the only remedy.

We foresee that air computing will be the next-generation computation paradigm as a result of the evolution of MEC [93]. Since the computation paradigm in MEC is typically restricted with the 2D terrestrial networks in which the resources allocated for the application tasks in either an edge server or cloud server, system bottlenecks can limit meeting the diverse requirements of heterogeneous applications. Since air computing is comprised of different air communication technologies, resource allocation alternatives are considerably increased when compared with the 2D settings. Vertical networking structure shown in Figure 3.1 depicts how different applications would use

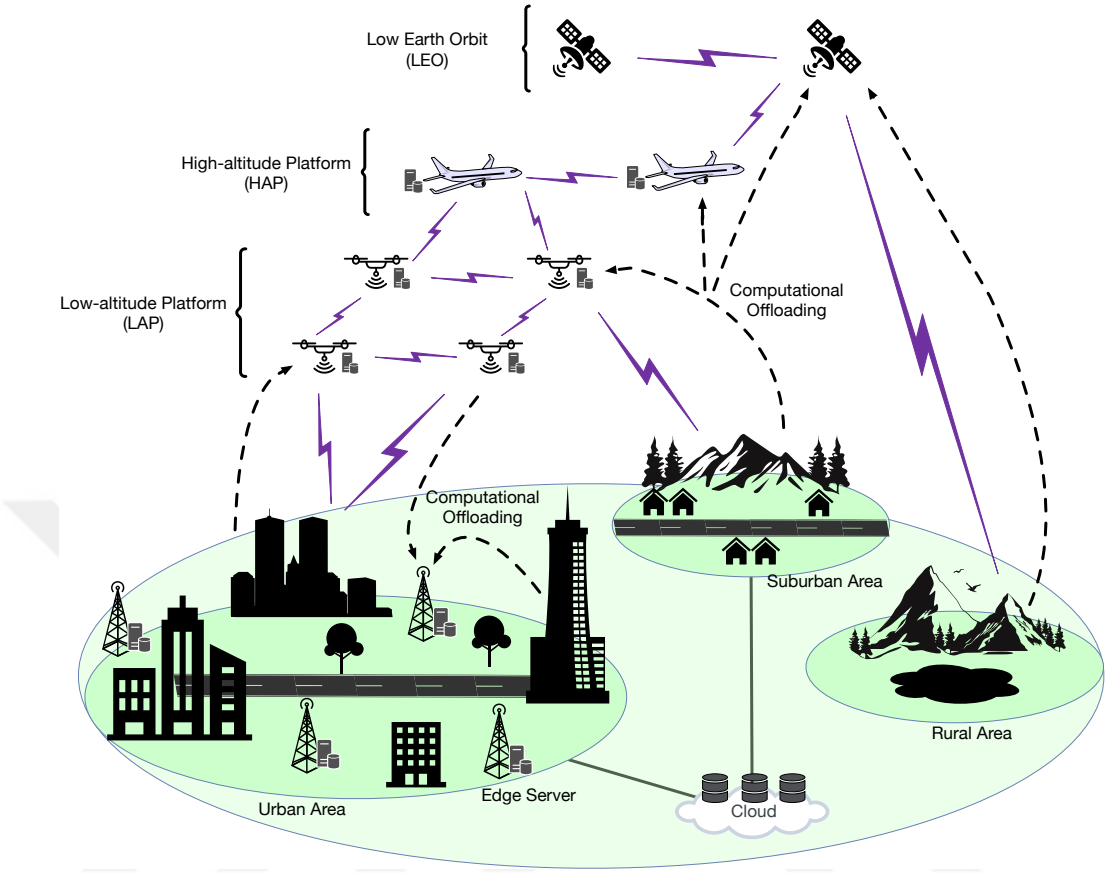



Figure 3.1. Air computing architecture.

different resources to ensure their QoS requirements.

### 3.1. Air Computing as a New Computational Paradigm

Air computing is a next-generation computational paradigm in which ubiquitous applications with radical networking and computational requirements are satisfied with the help of a family of novel communication opportunities. It provides a highly dynamic, scalable and responsive computational infrastructure in which terrestrial servers are harmonized with various air layers including LAP, HAP, and LEO as shown in Figure 3.1. Moreover, air computing augments traditional 2D edge computing with a wide spectrum of different computational servers in the air considering a highly dynamic context.

Currently in the urban area, users can enjoy the underlying terrestrial resources

to experience seamless connection based on the available infrastructure. With the help of edge computing, mobile devices can reach one of the nearest servers for the execution of their delegated tasks via offloading mechanisms. Despite these advanced approaches, ever-growing application requirements and increasing user mobility patterns push the limits of the fixed infrastructure which led to UAVs being deployed for dynamic capacity enhancement [78, 94]. Adding a vertical dimension to the network greatly enhances the possibilities in terms of the interaction of the users with computational resources. Accordingly, one of the main features of next-generation systems is expected to be their dynamically provisioned 3-Dimensional (3D) structure which leads to many opportunities in terms of QoS and user experience [95]. In a typical edge computing scenario, computational offloading would end either in an edge server or in a cloud server in the 2D terrestrial networks. Cloud servers, although providing seemingly infinite computational capacity, due to latency may be prohibitive or cause low QoE. In that respect, by adding a new dimension using UAVs and other HAP entities, the overall capacity is considerably enhanced and access becomes agile which leads to the server selection process to be more versatile. Since different application types would have instant access to the dynamically arranged computational array of resources in the air, as well as terrestrial ones, this architecture will provide a dramatically increased QoE offerings. Moreover, air computing will also address users or autonomous entities in the air. A user in an air vehicle can perform computational offloading to other air units and/or terrestrial servers, which manifests itself as a capacity enhancement. As shown in Figure 3.1, the offloading can be directly from the air vehicle or indirectly via routing over a LAP or a HAP before it is sent to the corresponding server.

A suburban area consists of residential homes and has less population density than the urban area. Hence, the communication infrastructure is not as pervasive as in the urban area which results in fewer resources for the computational needs of the applications. Even though the cloud servers can be reachable via existing infrastructure, the latency would be high for many applications which require low latency for the QoS. Therefore, using LAP and HAP layers as the vertical networking for the suburban areas will increase QoE. The first effect would be on the capacity of the area as new

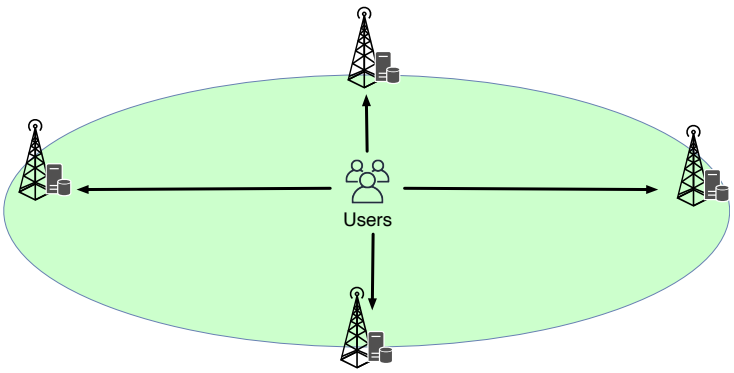
resources can be available for the applications of the users. The second influence would be on the latency since time-critical applications can use the computational sources of LAP vehicles for their corresponding tasks without using the cloud resources that cause high delay. Third, the coverage in the area can be enhanced by placing the UAVs into the corresponding places and the connection would not be interrupted [59, 78].

In the rural area, we assume that there is an environment in which people perform different activities such as sailing, kayaking, climbing, trekking, and camping. The essential fact is that the communication infrastructure may not exist or exist with extremely limited capacity. Moreover, accessing the cloud server is not always possible. In these circumstances, air computing can be used to meet essential QoS requirements since users would utilize fundamental resources for their applications. Even though LAP and HAP platforms are the backbone of the air computing, LEOs are generally used in rural areas since the replacement and energy refilling could be important issues for air vehicles such as UAVs. By deploying LEOs as the computational server or as the relay node that offloads the corresponding tasks to the suitable server, the end-user can enjoy the benefits of the applications also in rural areas.

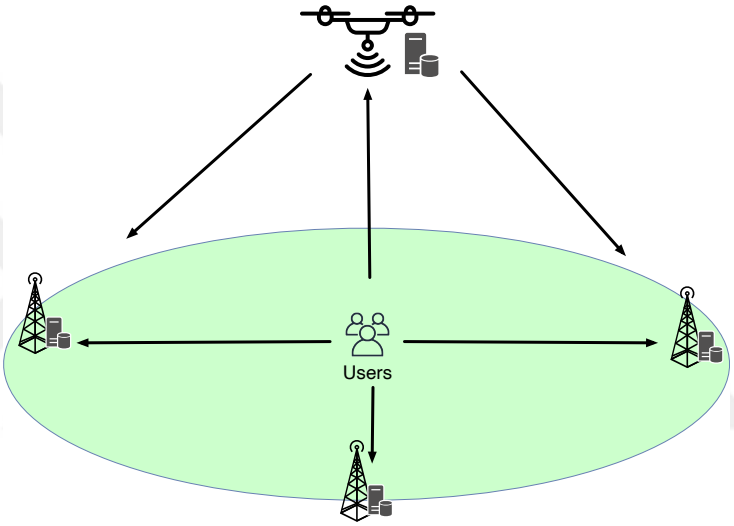
## 3.2. Differences Between Air and Edge Computing

The capacity of the networks considering 2D terrestrial resources is limited to serve very dense mobile and IoT devices. To solve these issues, air computing provides a third layer which is the air including LAP, HAP, and LEO layer to enhance the 2D computational paradigm into 3D. The 3D structure is also considered as vertical networking and it provides important solutions that cannot be given by traditional edge computing.

As shown in Figure 3.2, one of the most important differences between edge and air computing is the direction of the computational task offloading. In edge computing, the direction is horizontal as computational resources are inside the terrestrial 2D area. Moreover, these directions are always one way that is from the user device to




(a) Horizontal and one-way direction in edge computing.




(b) Horizontal, vertical and two-way direction in air computing.

Figure 3.2. The effect of the task interarrival time on the image-rendering application.

the corresponding server. An offloading of a task or a request comes from the user application to the server and then the server process the task in order to give the suitable service. On the other hand, in air computing, the direction of the computational offloading can be both horizontal and vertical. An application would benefit from terrestrial resources and air platforms at the same time. Furthermore, the vertical case of the computational offloading may be in both directions. A typical user in an urban area can use the resources in the air and, conversely, an application in an air vehicle can also benefit from the terrestrial servers. The important differences between edge and air computing are summarized in Table 3.1.

Table 3.1. Important differences between edge and air computing.

| **Feature** | **Edge Computing** | **Air Computing** |
|---|---|---|
| Network Architecture | 2D terrestrial | 3D vertical |
| Typical Latency | $< 10$ ms | $< 5$ ms |
| Mobility | $< 500$ km/hr | $< 1000$ km/hr |
| Coverage | Static cells | Dynamic cells |
| Offloading Direction | Horizontal one-way | Vertical two-way |
| Bandwidth | Static | Dynamic |

## 3.3. Advantages of Air Computing

Air computing has many advantages that can be categorized as *offloading, content caching, latency, computational capability, coverage,* and *mobility.* In this subsection, we explain those advantages in detail.

### 3.3.1. Offloading

The main advantage of air computing regarding offloading is the vertical network opportunities. Unlike traditional edge computing which is based on terrestrial resources, air computing employs a wide variety of computational technologies in air layers each with a different degree of geographical and mobility capabilities. Moreover, air components not only add a new physical dimension to the overall infrastructure but also their ability to be dynamically arranged creates a vision where the system can swiftly adapt itself to the ever-changing conditions of the users, applications, and the network itself. This allows air computing to adequately respond to the full spectrum of application profiles including ones with stringent latency and computational requirements.

We can detail the advantages of air computing for offloading in three different scenarios. In the first scenario, we can assume that a user device in a terrestrial place decides to offload an atomic task of an application which cannot be processed in the device itself. Hence, the task can be offloaded to either terrestrial resources or

air components. If terrestrial servers are selected for the offloading, the corresponding procedure would be similar to the edge computing process in which the task is processed on the server and then the results are transmitted to the user. However, in contrast to edge computing, if edge servers in the terrestrial area cannot serve due to their limited capacity, the tasks can be offloaded to air components rather than the cloud servers. This is a crucial advantage of air computing since blockage-free air routes and dynamically provisioned servers would provide lower latency.

In the second scenario, we can assume that tasks are non-atomic and different parts of the main task can be processed in different resources in an air computing environment as shown in Figure 3.3. Since coverage is not a problem in vertical networking as in the case of traditional 2D networking, the corresponding partial tasks may be sent to the terrestrial resources in other regional domains if capacity problems occur in the air. As the communication would be in very high bit rates and the air resources would be in the vicinity, the latency may not be an important issue in this situation. Thus, partial offloading can be carried out more efficiently in air computing. On the other hand, if only the terrestrial resources are used for this purpose, this scenario can cause capacity problems regarding the processing if the number of users and their corresponding tasks are high with respect to the resources. Moreover, sending partial tasks to different terrestrial resources may cause congestion in particular parts of the network regarding the link capacity and user density.

The third scenario for task offloading in air computing is that the tasks can be offloaded from the air to the ground. This is crucial since without air computing, users in the air must use the device capabilities, resources of the air vehicle, or relay capabilities of Geosynchronous Equatorial Orbit (GEO) satellites which results in low QoE. Now, through air computing, the tasks can be offloaded to terrestrial sources with low latency and applications can enjoy the benefits of the corresponding resources. Moreover, since air components can cover a large area, different tasks may be offloaded to different terrestrial areas.

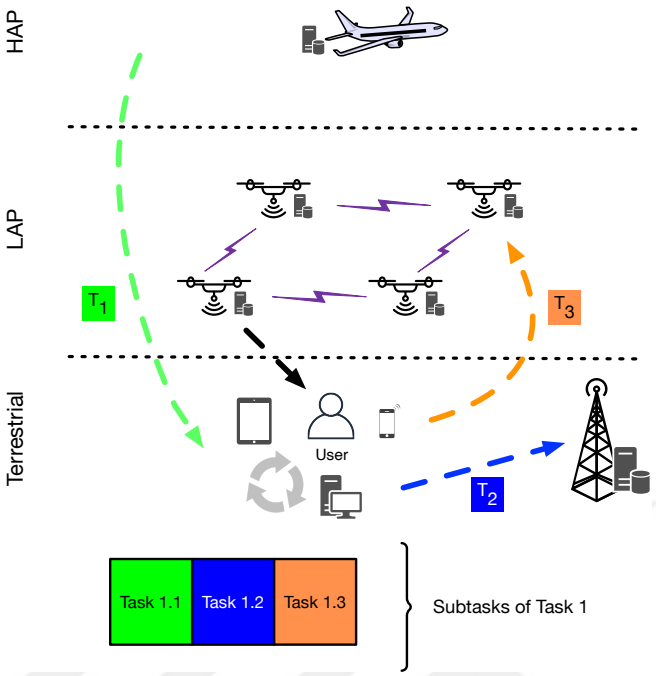



Figure 3.3. Subtasks of a single task can be processed by different components of an air computing environment.

### 3.3.2. Content Caching

Content caching is one of the important practices in order to access the requested pages, tools, and applications with low latency. Therefore, content caching optimization contains three objectives including QoS guarantee, content popularity, and utility maximization [96, 97]. To this end, hit ratio is used as the primary metric to indicate the quality of the content caching optimization. Especially, when the storage capacity of the corresponding servers is insufficient, the quality of the optimization would be more important.

By using vertical networking through air computing, the capacity is enhanced regarding two possible methods. First, storage of the air vehicles can be used for this purpose. As a result, the capacity can be improved and more content can be reachable by users. In the second method, air components would be used as a relay for the request and the corresponding content can be reached through nearby terrestrial servers. This method can provide lower latency than using WAN.

Even though air computing has important features for content caching in the urban area, its main advantage manifests itself in suburban and rural areas as the communication infrastructure is less developed. Considering the fact that even cloud resources cannot be reachable in rural areas, the importance of air computing would be better recognized. However, since using UAVs would be less efficient as they need corresponding battery charging stations which cannot be found in the rural area, HAPs and LEOs are more suitable to use. As both air platforms can provide content caching, the QoE of users would be enhanced.

### 3.3.3. Latency

Regarding the QoS, one of the most important metrics for a computation paradigm is latency. Since users would like to obtain the corresponding content or result as quickly as possible, providing low latency is critical. Air computing makes use of vertical networking opportunities to provide low latency for specific scenarios. This allows air computing to support diverse application profiles such as remote health, mobile augmented reality, and natural disaster emergency intervention. In traditional terrestrial networking paradigms such as edge computing, the range of the servers is crucial for the latency even though edge servers are located at the LAN or MAN. On the other hand, as air computing allows for the dynamic placement and provisioning of resources in places needed, typically latency would be independent of the geographical location of the users as shown in Figure 3.4. This advantage also provides important stability in terms of QoE.

### 3.3.4. Computational Capability

Even though the computational power of the air vehicles is not as powerful as the terrestrial servers, the tasks of applications can be offloaded to multiple sources to enhance the throughput. Moreover, by using both terrestrial servers and air components, a single task may be partitioned to be processed. Since the data rate would be higher and latency would be lower in air computing regarding edge computing, using

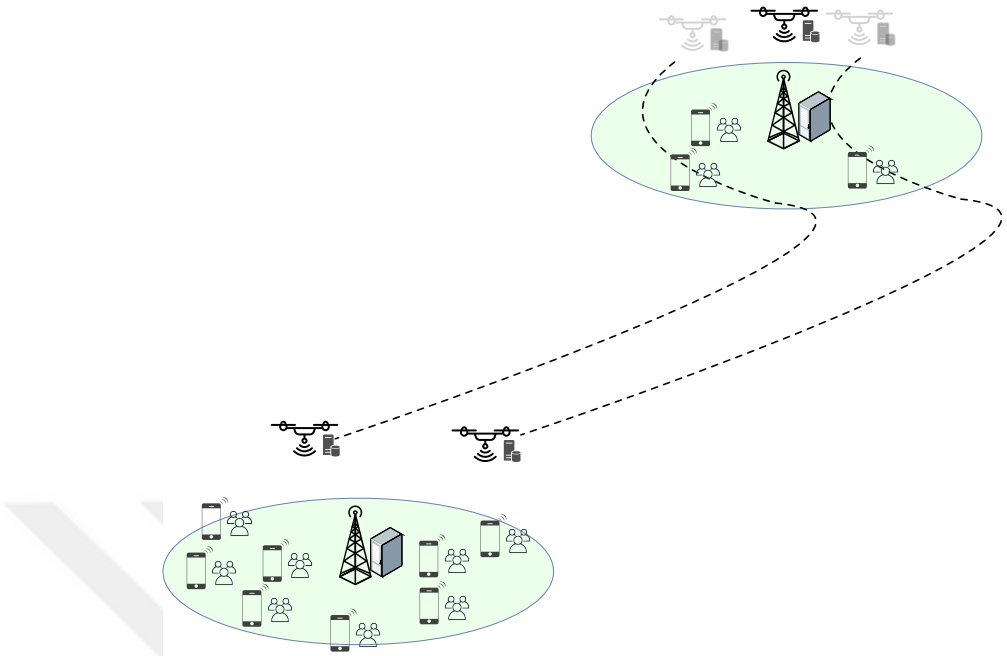


Figure 3.4. Air components can be replaced in the environment dynamically based on the changing user demands in particular locations.

several sources in different layers for a single task would increase QoE.

### 3.3.5. Coverage

Since air vehicles and their corresponding components are used in air computing, the end devices will not depend on the static cell infrastructure in which there is a limited capacity regarding the number of users. As shown in Figure 3.5, this dynamic cell structure will provide pervasive connectivity which is crucial for the heterogeneity of future applications.

### 3.3.6. Mobility

As a result of seamless coverage and pervasive connectivity, air computing provides high mobility which is over 1000 km/hr. Moreover, since air components communicate with each other, handover for the processed tasks of users that are in the air or terrestrial vehicle is carried out more smoothly. Furthermore, mobility in air

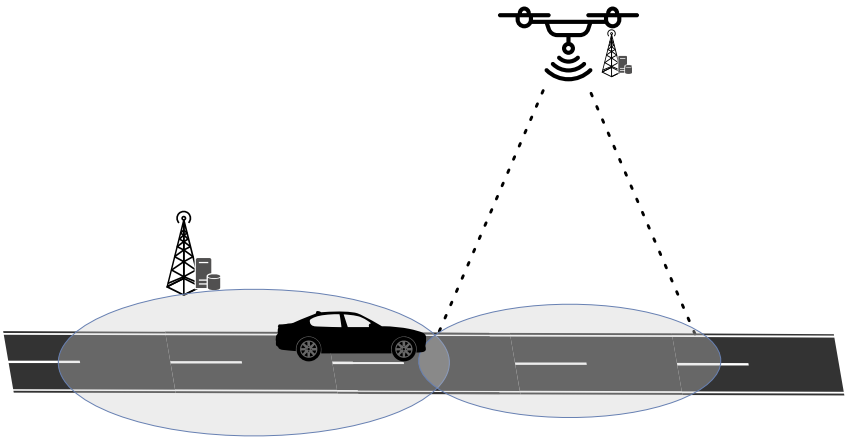


Figure 3.5. Dynamic cell structure through air components would provide seamless connection for end-users.

computing can be considered for the users in the air and ground. Accordingly, the processed tasks can also be sent to the terrestrial servers from the air or vice versa.

## 3.4. Air Computing Use Cases

The actions that can be taken in air computing are similar to edge computing as they include computation/task offloading, resource allocation, and resource provisioning. However, since air components provide important flexibility regarding vertical networking, the use cases in air computing are more versatile than edge computing. To this end, we elaborate on the potential use cases of air computing in this section.

### 3.4.1. Natural Disasters

Natural disasters such as earthquakes, hurricanes, tsunami, and floods wreak havoc on settlements and residential areas. They cause the loss of human lives and the destruction of important resources that people need. Considering communication perspective, there would be two important consequences. First, the communication facilities would be destructed by the natural disaster and as a result people can be deprived of important resources that cause isolation in the disaster site. This is crucial for those people affected by the disaster because they cannot obtain the required aid

which must be given to the heavy injury cases and also to humans that expect to be rescued. Without communication resources, the outcome of the disaster would be much worse. Secondly, even in the cases where communication facilities are not seriously damaged, bursty traffic caused by people that want to make an emergency call and to reach their friends, and relatives brings about congestion in the network. Note that this congestion can be also caused by the people outside the disaster site since they also would like to reach the people that are affected by the disaster. Similar to the first case, as a result, people can be isolated in the disaster site so that the outcome of the disaster would be heavier.

Considering both cases, the essential requirement for providing the communication and computation in a disaster site is to enhance the capacity dynamically. This can be carried out by using air components of air computing including UAVs, HAP vehicles, and LEOs [98, 99]. Note that the utilization of those components depends on the disaster type. For example, if the disaster is a hurricane, UAVs and HAP vehicles cannot be used during the disaster. Similarly, if the disaster is an earthquake or a flood, using UAVs would be more effective considering the latency which is a crucial metric for these scenarios.

### 3.4.2. Well-being Monitoring

In the event of a medical crisis elderly people who suffer from chronic diseases may require real-time action where high latency would be fatal, especially in rural areas. Therefore, well-being monitoring must be provided such that the latency should not be destructive for the patients. As traditional methods may not ensure low latency, air computing can be used for this purpose [100, 101]. For the urban areas, air computing may be utilized as the complementary resource regarding capacity since the corresponding wireless networks can handle most of the requests. On the other hand, for suburban and rural areas, air computing would be the primary resource for well-being monitoring. By using the coverage, latency, and data rate advantages of air computing, the Quality of Life (QoL) of those patients can be enhanced.

### 3.4.3. Remote Health

The issues in remote health are similar to those in well-being monitoring, however they are more critical as medical operations are carried out in this case. The scope of remote health includes the actions of doctors and monitoring the results of those actions on the patients in operations. Moreover, critical life-related metrics such as heart beating, and adrenalin level must be constantly monitored in the operation. To this end, air computing provides important opportunities in this area through its paradigm and corresponding components. For example, if the patient or doctor cannot move from one place to another due to several reasons, the operation can be carried out from a remote area where the doctor leads it.

### 3.4.4. Real-time Video

Since real-time video requires a constant bit rate through the lifetime of the video, ensuring the desired QoE is more difficult than the video on demand systems in which the delay can be compensated using dynamically changing buffers. This issue is experienced differently by two main use-cases: (1) real-time conferences/calls such as Zoom and Skype, and (2) watching sports activities.

The fundamental problem in the real-time video for sports events through the Internet is that viewers obtain the content with higher delay regarding terrestrial broadcast. As a result, QoE reduces significantly as viewers may hear the sound of terrestrial broadcast viewers when an important incident has occurred in the event including a football or basketball competition. On the other hand, in conferences and calls, the video can stall or the voice cannot be synchronized with the video due to jitter or congestion in the network.

Air computing can provide a possible solution which proposes a novel architecture for video streaming using air components as shown in Figure 3.6. In this solution, the video segments are routed via different components including terrestrial servers, UAVs,

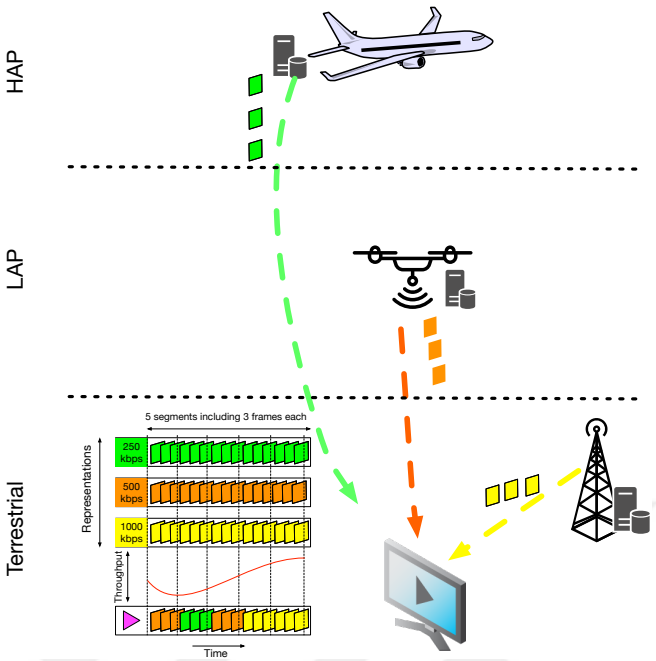



Figure 3.6. Various representations of the video segments as conveyed by different components of air computing.

HAP vehicles, and LEOs based on the current requirements of the network and video. Note that this approach may bring about its own challenges considering scheduling and management of segments and user profiles. However, by using the underlying technology and novel utilization of the air components, the problems above can be solved.

### 3.4.5. Mobile Augmented Reality

Augmented Reality (AR) in which the real and virtual objects are combined has been used in many application areas including entertainment, healthcare, and education [102]. Moreover, since they perform in real-time, augmented reality applications are delay intolerant in order to provide satisfactory QoE for end-users [103]. Especially, after the widespread usage of smartphones with their expanded capabilities, augmented reality applications are widely deployed. To this end, AR is recently named as Mobile Augmented Reality (MAR).

The most significant difference between AR and MAR is their processing methods. While traditional AR applications perform in-device processing, MAR uses offloading to carry out corresponding computations [104]. The main reason for offloading in MAR is the limited capacity of mobile devices regarding battery and CPU. Therefore, edge and cloud computing have been broadly utilized by MAR applications in recent years. However, since the application requirements of MAR has changed over the years, traditional edge solutions may not be sufficient to provide required latency and capacity.

Air computing can solve the issues related to MAR including scalability, latency, and expected data capacity indicated in [104]. The scalability problem can be handled by multiple components of air computing that can be reachable via seamless connection. Even though users are outside of the urban areas, air components can handle MAR tasks considering latency. Moreover, a huge amount of data can be transferred using the advantages of 3D network structures.

### 3.4.6. Sport and Concert Activities

The communication infrastructure in terrestrial areas is built considering fixed resources in which once the facility is deployed, it can be changed with difficulty and significant additional cost. Considering Capital Expenditures (CAPEX) and Operating Expenses (OPEX) of companies, investing on fixed resources was plausible over years. For example, if there are limited resources such as base stations for users that increase in years, improving the capacity which means adding new base stations could be the solution for this problem. However, especially after the proliferation of the versatile application types in smartphones, the fixed infrastructure may not be sufficient for the user needs that change dynamically.

One of the most important examples of this situation is the sport and concert activities where thousands of people rally and use their applications. Statically built infrastructures undergo a significant amount of traffic and computational offloading

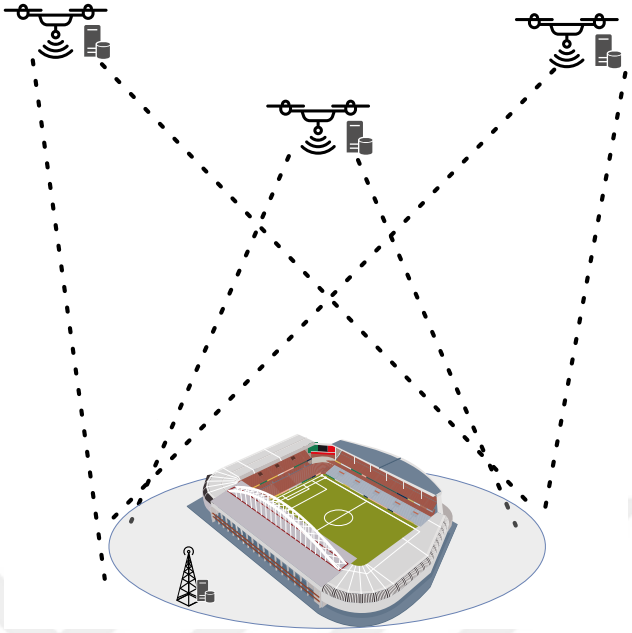


Figure 3.7. Dynamic network capacity enhancement by air components for organization with intense participation.

which may be an order of magnitude higher than the expected requests. Even though network slicing, Network Function Virtualization (NFV), Software-Defined Networks (SDN), and edge computing are used to provide a solution, the issue is still open. On the other hand, by using air components and directing them to the corresponding places where activities occur based on the current network requirements, we can increase the capacity of the network dynamically [105] as shown in Figure 3.7. Hence, even though the network resources cannot meet the number of application requests, new resources and routes can be created through air computing. As a result, QoS and QoE would be enhanced.

### 3.4.7. Outdoor Activities

Since people that perform outdoor activities including sailing, kayaking, climbing, and trekking may not access corresponding resources due to lack of communication infrastructure, they can face isolation in the natural environments. This situation would be crucial considering several issues. In the case of an injury or a health problem such as a heart attack, emergency services must be accessed immediately in order


Figure 3.8. Leveraging communication through air computing for outdoor activities.

to obtain the aid. Regarding the current networking conditions, the communication and computation from secluded areas including sea, mountain, and forest cannot be provided properly. Thus, even though there would not be such a problem, people carrying out the activities do not feel safe about those possible issues. Apart from the health, daily routines such as social media, communication via chat, and telephone call also cannot be carried out in those conditions. Hence, even though those activities provide important relief for people, their life quality may have deteriorated.

Through air computing, the issues for outdoor activities can be solved as shown in Figure 3.8. By deploying UAVs, HAP vehicles, and LEOs in suitable places, people in secluded areas can reach important contents and communication infrastructure [106]. We assume that HAP and LEOs would be more useful for such activities as UAVs may

face battery reload problems which is similar to the situation in rural areas.

## 3.5. Challenges and Future Research Directions

Even though air computing can solve many issues related to the limitation of current networking paradigms, it would face many challenges such as network architecture, regulation of air vehicles, battery issues, coverage, and a communication protocol. Considering the fact that UAV communication between different devices causes many challenges [107], applying different vehicles in the air and providing communication between them is not trivial.

Therefore, all of these issues must be investigated thoroughly in order to apply the air computing paradigm correctly. On the other hand, these challenges open new research areas regarding the provision of QoS, energy efficiency, determining air vehicle placement, and the deployment of AI. In this section, we elaborate on those challenges and corresponding research opportunities.

### 3.5.1. Challenges

We evaluate challenges considering the architecture design of air computing, corresponding protocol, and flying vehicle regulations.

3.5.1.1. Air Computing Architecture Design. Since air computing has a 3D structure with four major layers including terrestrial, LAP, HAP, and LEO, the networking architecture in terms of offloading mechanism and routing is crucial [108]. One of the main concerns in networking is how the requests would be handled considering the mesh connected different nodes in the 3D structure. To investigate this, we propose three candidate design approaches including hierarchical, free, and hybrid designs. Moreover, we also make a comparison between a distributed approach and orchestration in air computing regarding task offloading.

*Hierarchical Design* - As the name suggests, there is a systematic order between air computing layers in the hierarchical design as shown in Figure 3.9. If a task is offloaded from a user in a terrestrial network, the selection of the corresponding server for the computation is handled by another entity in the air computing environment, not by the user. For example, if the service for the corresponding application task can be met in one of the HAP vehicles due to network conditions, the task is routed regarding a predefined path rather than directly transmitted to the corresponding HAP component by the user. Therefore, the task is first sent to the nearest edge server in LAN. If the edge server cannot process the task because of its high load or its service incapability, the task is relayed to one of the UAVs in LAP. Note that we assume that the corresponding edge server is in the vicinity of the UAV. Next, if the UAV similarly cannot execute the task, it relays the task to one of the UAVs in its vicinity, one of the available edge servers on the ground, or one of the airplanes in HAP. Since the corresponding service has been given in HAP in this case, the task is pushed to the HAP.

Even though the hierarchical design provides important manageability throughout the air computing environment, it may face high delays due to its layered structure. Based on the use cases and user profiles, it can be applied in the network for specific areas.

*Direct Access Design* - In contrast to the hierarchical design, in direct access design, IoT devices in an air computing environment can select the corresponding server to meet the requirements of their tasks as shown in Figure 3.10. As a result, if the corresponding sender knows which server provides the required service, it offloads the task directly to the appropriate layer through the seamless connectivity feature of air computing.

However, this free access for each device in the network would cause congestion and underutilization of the resources. In terms of congestion, if a service is given by a particular server in the network, all devices which require the service may offload their

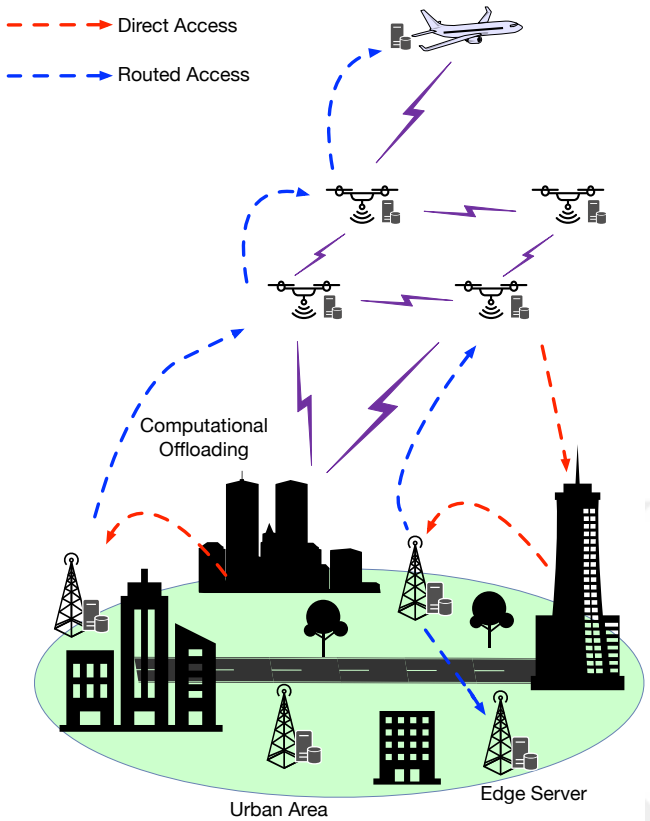



Figure 3.9. Architecture of the hierarchical design.

task to that server. As a result, communication links and server capacity would be heavily affected by this situation. On the other hand, if devices use particular servers in the network due to low delay, high processing, and energy efficiency, some resources would be underutilized.

*Hybrid Design* - Considering the advantages of hierarchical design and direct access design, a hybrid design would be applied in an air computing environment. For the urban areas, in which the network is extremely dense, the application of hierarchical design would be more suitable in order to prevent underutilization and congestion cases. Moreover, the delay issue in hierarchical design can be covered by diverse resources regarding the provision of different services, and seamless connectivity.

On the other hand, considering the suburban and rural areas, the utilization of direct access design would be more convenient since the infrastructure in those areas

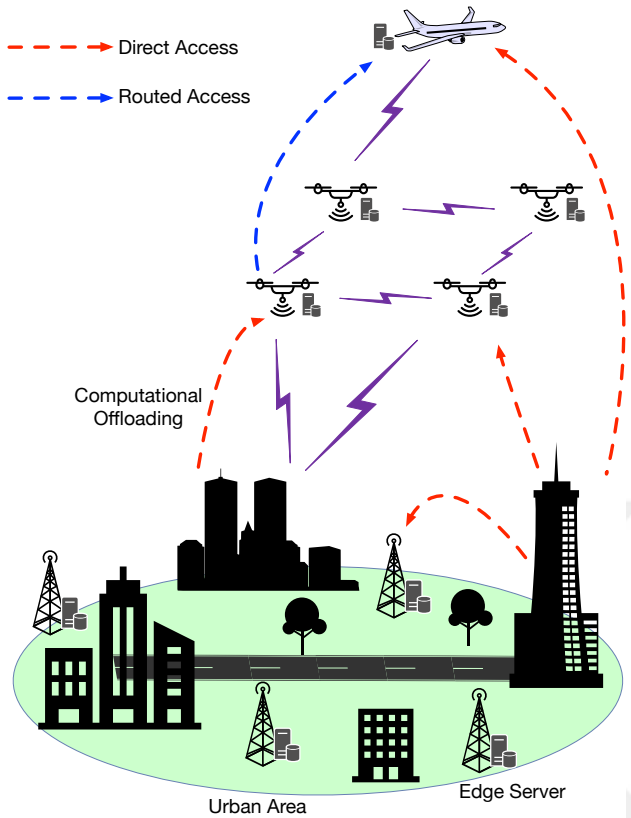



Figure 3.10. Architecture of the direct access design.

is limited. Therefore, direct access to the corresponding servers can be of benefit in terms of delay.

3.5.1.2. Air Computing Protocol. As there are many different entities in the air computing environment, the communication between them should be carried out based on predefined rules, which are defined by a protocol. An air computing protocol should be reliable, secure, and fast so that the entities carry out communication easily [109]. Moreover, it should facilitate the management of the network as it must provide data integrity.

3.5.1.3. Flying Vehicle Regulations. Since each country has different regulation policies considering flying air vehicles, the entities in an air computing environment should comply with those corresponding rules. Moreover, the air computing protocol should also be in compliance with regulations as reliable and fast communication is vital for

flying entities in the air. On the other hand, considering the existing cellular network infrastructure, the standardization between existing resources and newly deployed air computing devices must be investigated for 6G and beyond [110, 111].

## 3.6. Future Research Directions

We examine future research direction considering request management, deploying artificial intelligence, energy issue of air vehicles, and movement/coverage of air vehicles. We believe that these directions provide important research opportunities for researchers.

### 3.6.1. Request Management

In an air computing environment, handling user and IoT requests are crucial to meet the required QoS. Moreover, since there are different layers in the air, the request management would be more complex than other networking paradigms such as edge computing. Even though architecture design is essential for request management, deciding where to offload and when to offload is crucial for the performance. To this end, the benefits of a distributed system and an orchestrator-based system must be investigated thoroughly.

A distributed system in air computing can be described such that the device which offloads the task takes the decision of where to offload. For example, users can select one of the edge servers for offloading, or an edge server relays the offloaded task to one of the UAVs without consulting any intermediate device. One of the most important advantages of a distributed system is its scalability and its low latency. Since there is no intermediate element considering the offloading, the transmission of the request would be faster.

On the other hand, it has a significant drawback considering the current state of the network. An offloading device in the environment cannot be aware of the current

condition of the corresponding servers and network elements in terms of the load of the servers, and the number of requests that may affect the communication links. Therefore, if there is no additional communication regarding this information, the decision taken by the offloading device would cause a failed task offloading. Moreover, note that if an additional communication mechanism is deployed in the environment, it must be well-optimized so that the communication links cannot be affected by the transmission regarding congestion.

In contrast to distributed systems, an orchestrator to which the tasks are first sent by the offloading devices can be used in different parts of an air computing environment. In such a system, a user can directly offload tasks to the orchestrator. Here the decision of where to offload is taken by the orchestrator which has full access to the current system state regarding the communication links and corresponding servers.

Even though it has advantages, several points must be investigated and optimized in an orchestrator-based system. First, the deployment of the orchestrator is crucial for the performance of the system since many entities in the network may send their corresponding requests. Therefore, it should be reachable in a suitable time so that it can relay those requests without violating the maximum delay requirement based on application type. Second, the centralized deployment of an orchestrator results in a single point of failure such that the corresponding portion of the network would be heavily affected as users and IoT devices cannot offload their tasks. To this end, a fault management system should be considered with the deployment of an orchestrator. Finally, third, the cost of an orchestrator in terms of new communication links, delay, and congestion must be minimized for the system performance.

### 3.6.2. Deploying Artificial Intelligence

Considering the diverse requirements of various applications and IoT devices in an air computing environment, traditional optimization-based solutions and heuristics would provide limited solutions. Therefore, the application of AI-based solutions will

be inevitable. Especially, regarding edge and UAV-based systems, there are already many studies which benefit from AI-based solutions including ML, Deep Learning (DL), and DRL [112–114]. As a result, along with the requirements of the air computing paradigm, novel AI solutions should be applied to meet new challenges.

Since there are many resources that produce an enormous amount of data in an air computing environment, processing them and then learning meaningful patterns considering the performance of the system can be feasible using ML techniques. However, in recent years, DL solutions have been preferred rather than traditional ML algorithms since DL is more successful in terms of training, non-linear transformation, efficiency, and required computing power [115].

Even though DL solutions are preferred in recent studies, the data collection phase in an air computing environment would cause serious degradation of system performance. First, the huge amount of data can bring about congestion problems on communication links. Second, for such a heterogeneous environment, providing the privacy of the data would be difficult. Therefore, applying Federated Learning (FL) solutions would be more suitable with air computing [116, 117].

On the other hand, since many decisions must be taken based on dynamic events in the air computing environment, and they may not be labeled due to the nature of the problem, supervised ML and DL based solutions would be inadequate. To this end, recent studies benefit from DRL in which the agent can learn directly from the environment without needing human interaction [118]. The agent in an air computing environment can be the orchestrator, UAV, or edge server since they take actions based on the current state of the system. Thus, we believe that future studies can consider the deployment of DRL solutions along with FL.

### 3.6.3. Energy Issue of Air Vehicles

The vehicles in air layers use either batteries or fuel. Therefore, their deployment, trajectories, and computing capacities should be well-optimized to ensure energy efficiency. Energy-related issues are currently evaluated regarding edge and UAV studies. However, considering all layers of air computing, a collaboration between different air vehicles can reduce energy consumption further.

### 3.6.4. Movement and Coverage of Air Vehicles

Even though deployment is an important research issue for 2D terrestrial networks, this problem is more difficult to manage as air vehicles move. Moreover, since their coverage, transmission quality, and power consumption are heavily affected by their vertical movement, optimization of their altitude and trajectory is crucial [119, 120]. Therefore, request management can be handled along with this optimization in order to provide efficient performance.

# 4. DEEPEDGE: A DEEP REINFORCEMENT LEARNING BASED TASK ORCHESTRATOR FOR EDGE COMPUTING

The number of end-users increases dramatically that causes tremendous amount of data in networks by using applications which have diverse requirements. Even though the cloud computing solutions have been used to cope with those application requirements, it is insufficient considering the high data rates, real-time response requirements, vast amounts of data, and user mobility [14]. Thus, edge computing is recently applied in order to manage the diverse demands of networks [121].

Edge computing is an umbrella term [12] that encompasses several similar technologies including Cloudlet [9], MEC [10], and Fog Computing [11]. The idea behind the edge computing is to provide services to applications in a LAN or MAN without routing the offloaded tasks to the WAN which causes high service delays that may affect QoS. The expectation from an edge network is that the user can offload its tasks, which cannot be performed on the device, to the corresponding edge server. As a result, end-users can rapidly access and exploit computational resources, and the core network is relieved.

An edge computing environment is one of the most dynamic and heterogeneous environments since it may consist of multiple edge servers, different services, tasks, technologies, and user profiles. Therefore, deciding where and how the offloaded tasks must be processed in the edge network is crucial. The methodology of this decision is called task orchestration. The primary job of the task orchestrator is to determine whether the offloaded tasks of the users are processed in the cloud via WAN or in the edge via LAN/MAN. Afterwards, if the requested service is available in the edge, the orchestrator should choose the corresponding edge server based on several metrics including delay, length of the task, data rate, and utilization of edge servers. A typical architecture of an edge computing environment that consists of the IoT, edge, and

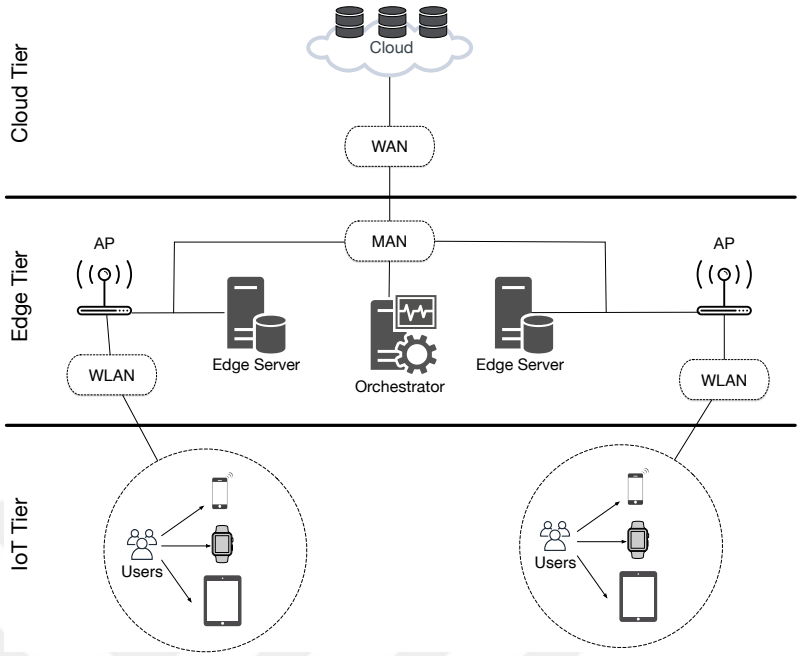



Figure 4.1. A three-tier edge computing network.

cloud tiers is shown in Figure 4.1.

A task orchestrator should be scalable and adaptive since the requirements of the applications and mobile networks may change over time due to their dynamic and heterogeneous nature. Since many parameters must be taken into account, such as delay requirements, length of the task, data rate, and utilization of edge servers, traditional rule-based orchestrators would be insufficient to make an accurate decision considering the constraints of the problem. Therefore, a policy must be created in an intelligent way and applied such that the unique demands of the applications are met.

Considering the dynamics of the edge computing networks, the policy for task orchestration should be learned from the environment in order to make accurate decisions. DRL is widely used to implement decision-making systems modeled by a MDP [122]. The most important advantage of the DRL is that the agent learns the environment dynamics on its own using the reward mechanism of the system without any human interaction. Therefore, ranging from the communication needs of unmanned aerial vehicles [123] to spectrum allocation studies [124, 125], DRL solutions are applied re-

cently in computer networks. Task offloading in edge computing is one of the areas in communication that DRL is also performed [126, 127]. We give their details and also explain what features make our study unique in Section II.

In this chapter, we introduce our DRL-based task orchestrator, DeepEdge, that is able to meet the dynamic needs of different application types including image rendering, infotainment, pervasive health, and augmented reality under various loads. We designed detailed simulation experiments to evaluate the performance of our proposal in a realistic setting where computational and network level factors are varied. We compared the results with several decision-making systems including the recently developed Fuzzy Logic approach [128].

## 4.1. DeepEdge

Mobile networks are one of the most dynamic and heterogeneous environments since they consist of multiple edge servers, different services, tasks, technologies, and user profiles. Therefore, deciding where user tasks are offloaded to is a critical and NP-hard problem [10]. In our design, we assume that a mobile device has already taken the offloading decision. The main goal of our design is to minimize the failed task rate in the network by using the autonomous policy of DRL, even in the heavily-loaded scenarios.

### 4.1.1. Challenges of Applying DRL on Edge

Even though task orchestration requires a corresponding policy suitable for the concept of DRL, ensuring essential points in a real-time and dynamic environment is a complicated operation. There are four challenges to apply DRL in the edge computing environment:

- Since tasks are generated by multiple users from multiple applications, guaranteeing the Markov property is a crucial challenge. The transition from one state

to its next state should display the change of the environment for the given action that is fundamental for MDP. Therefore, the state representation must be carefully defined in order to apply DRL on edge accurately.

- The delayed action effect is another imperative challenge due to the fact that the performed action may not immediately affect the network attributes which describe a state for DRL. As a result, there is an important risk that a state and its next state would be the same. An action in MDP must cause a change for its next state like in Atari games [129] in order to carry out DRL.
- The third important challenge is the stochastic environment because of different application tasks. Since task orchestration is modeled considering that each incoming task indicates a state in the system, applying an action would not assure that the next state is the expected state for the edge environment.
- The last challenge for this study is that we would like to implement the DRL agent which can learn online rather than using historical data. Related with the delayed action effect, online learning also needs a suitable architecture to implement.

### 4.1.2. Providing MDP on Edge

The DRL is based on the MDP which ensures the mathematical framework to model decision making systems. MDP is formally defined as a 4-tuple $< S, A, P, R >$ where

- $S$ is the set of all valid states where $s \in S$
- $A$ is the set of all valid actions where $a \in A$
- $P : SxA \rightarrow P(S)$ is the state transition probability function that provides the probabilty of $P(s_t, s_{t+1}) = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ in which taking the action $a$ as the decision at time $t$ change the current state $s_t$ to $s_{t+1}$.
- $R : SxAxS \rightarrow R$ is the reward function that defines the reward $r$ for the taken action $a$ at time $t$ considering the state transition such that $r_t = R(s_t, a_t, s_{t+1})$.

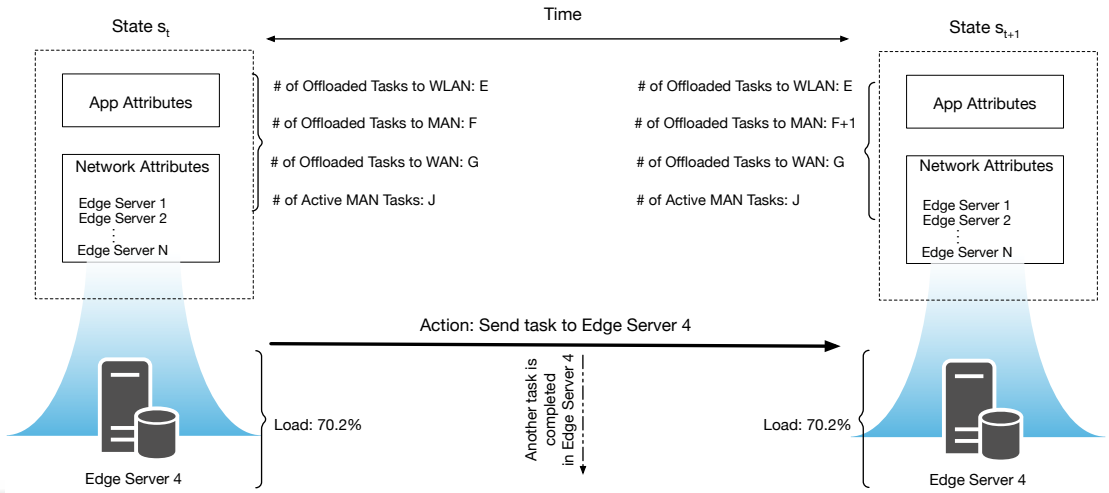



Figure 4.2. Because of the characteristics of the edge computing environment, two consecutive states would be the same regarding the application and network attributes that define the states. This is impractical for DRL since the Markov property may not be ensured. Therefore, we use four supplemental attributes for each state to identify them uniquely.

The agent takes an action based on the observation it gets from the environment so that it can apply a deterministic policy $\pi$ for a given state which is defined as $\pi(a \mid s) = P(a_t = a \mid s_t = s)$. For the popular implementations of DRL such as Q-Learning, DQN, and DDQN, MDP must be provided for the environment so that the agent can develop a successful policy for its decisions.

MDP is essentially based on the Markov property in which the next state depends only on the current state. In other words, $s_{t+1}$ depends on $s_t$; it is not determined by $s_{t-1}$, $s_{t-2}$, ... , $s_1$, $s_0$. Markov property is formulated as in Equation 4.1. If $\{s_0, s_1, s_2, ...\}$ is a sequence of discrete random variables, then the sequence is a Markov chain if it satisfies the Markov property as

$$P(s_{t+1} = s \mid s_t, s_{t-1}, s_{t-2}, ..., s_1, s_0) = P(s_{t+1} = s \mid s_t) \ . \tag{4.1}$$

Since the edge computing environment is real-time and extremely dynamic, en-

suring Markov property is not a trivial task. As shown in Figure 4.2, the orchestrator may send the offloaded task to the Edge Server 4 regarding its load which is 70.2% at $s_t$. However, if there is another similar task for which the edge server has allocated its resources and it is completed before the arrival of the offloaded task, the load of the edge server would not change at $s_{t+1}$. Moreover, if the $s_{t+1}$ consists of the same application attributes and network attributes, the Markov property cannot be satisfied. Note that this scenario for the load of an edge server is also valid for the load of MAN. This is impractical for DRL that works with the unique states which provide the Markov chain.

In order to solve this problem, we add four supplemental attributes, which are described in Section 4.3 along with other attributes, for each state:

- Number of Offloaded Tasks to WLAN
- Number of Offloaded Tasks to MAN
- Number of Offloaded Tasks to WAN
- Number of Active MAN Tasks

By adding them, we can identify each state uniquely throughout the edge computing environment so that we can satisfy the Markov property for the implementation of DRL.

### 4.1.3. State Representation

In DRL, a state is defined when the agent takes an action. Considering our system architecture, the agent is the orchestrator that takes action when a task comes to the orchestrator in order to be offloaded to the corresponding edge or the cloud server. Note that since each incoming task is produced by a mobile device by following a Poisson process, the time between consecutive states is exponentially distributed. Thus, we define a state $s$ at which the orchestrator makes a decision for the task.

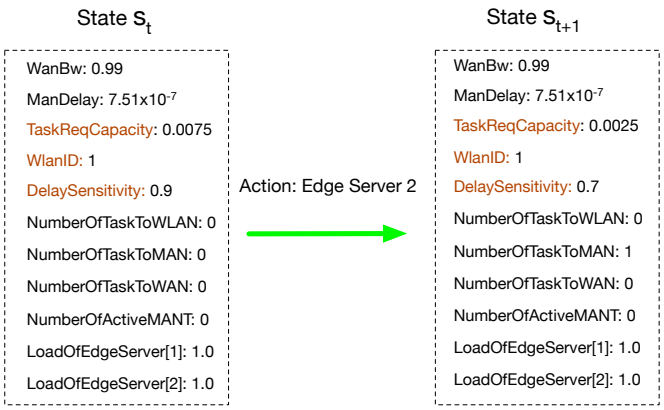



Figure 4.3. The state transition in an edge computing environment consisting of two edge servers.

There are two fundamental feature sets for the state representation in our architecture due to different application requirements: app attributes and network attributes. App attributes indicate the individual requirements of the applications in order to ensure their QoS and to identify the corresponding state. On the other hand, network attributes describe the current condition of the network. Thus, a transition between two consecutive states in a stochastic environment would be as in Figure 4.3 and the transition probability $Pss'$ is defined as $Pss' = P\left(s_{t+1} = s' \mid s_t = s, a_t = a\right)$. However, even though the taken action changes the environment through four supplemental attributes we add, the action does not fully determine the next state due to the Poisson fashion of the different applications run by multiple users.

In this study, we use ten features in which three of them indicate app attributes, while seven of them are related to the network condition. On the other hand, since the number of edge servers varies based on the network configuration, the attribute that shows the current load of an edge server depends on the environment. The attributes of a state are given in Table 4.1.

For the definition of a state we use *WanBw, ManDelay, NumberOfTaskToWLAN, NumberOfTaskToMAN, NumberOfTaskToWAN, NumberOfActiveMANT*, and *LoadOfEdgeServer[N]* as network attributes. On the other hand, *TaskReqCapacity, DelaySensitiv-*

Table 4.1. Attributes of the state.

| **Attribute** | **Description** |
|---|---|
| *WanBw* | Remaining WAN data rate of the network. |
| *ManDelay* | MAN delay of the network. |
| *TaskReqCapacity* | Required capacity of a VM to process the given task. |
| *WlanID* | Wlan ID of mobile device generating the given task. |
| *DelaySensitivity* | Indicating whether the task is delay intolerant or not. Its value is between 0 and 1. |
| *NumberOfTaskToWLAN* | The number of tasks offloaded to WLAN. |
| *NumberOfTaskToMAN* | The number of tasks offloaded to MAN. |
| *NumberOfTaskToWAN* | The number of tasks offloaded to WAN. |
| *NumberOfActiveMANT* | The number of tasks offloaded to MAN but have not arrived their destinations yet. |
| *LoadOfEdgeServer[N]* | The active load of an edge server. There would be N number of edge server in the network. |

*ity*, and *WlanID* attributes are used for tasks to indicate their QoS requirements and location of offloading. *WanBw, ManDelay*, and *LoadOfEdgeServer* refers to the recent conditions of the WAN data rate, MAN delay, and load of the edge servers, respectively. Note that, *LoadOfEdgeServer[N]* includes the information of all of the edge servers in the network so that the $N$ is the variable that represents the number of edge servers. As a result, for example, if there are N edge servers in the network, the corresponding state includes $6 + N$ network related attributes. Hence, along with the application related attributes, the state consists of $9 + N$ attributes in total. We demonstrate an example state transition in Figure 4.3 considering the scenario in which there are two edge servers in the environment.

The other network attributes including *NumberOfTaskToWLAN, NumberOfTaskToMAN, NumberOfTaskToWAN, NumberOfActiveMANT* are self-explanatory. We increase corresponding values if the tasks are offloaded through WLAN, MAN, and WAN, respectively. Considering application attributes on the other hand, *TaskReqCapacity* indicates the required capacity for the offloaded task regarding edge servers. *DelaySensitivity* refers to the level of application tolerance to delay so that corresponding action should select WLAN edge server, MAN edge server, or the cloud server wisely. Finally,

*WlanID* specifies from where the task is offloaded so that the agent can identify which edge server is local for the user.

### 4.1.4. Action Space

The actions of an orchestrator are limited considering the number of edge servers in the network and the cloud server. Therefore, if there are $N$ edge servers, the orchestrator selects one of the $N + 1$ decisions as an action $a_t \in \{a_1, a_2, ..., a_{N+1}\}$.

The actions are taken by the neural network part of the orchestrator regarding DRL. When an offloaded task comes to the orchestrator, the current state of the environment is given to the neural network as an input for action. The corresponding generic neural network model is shown in Figure 4.4. Considering the $N$ edge servers in the network, the input features are defined regarding the state of the environment, which is $9+N$. Based on the network condition and the expected number of users, there would be several hidden layers that consist of multiple neurons. Since the orchestrator determines the corresponding offloading decision among $N + 1$ possible choices, as $a_t \in \{a_1, a_2, ..., a_{N+1}\}$, the size of the output layer is equal to the number of edge servers and the cloud server.

Even though the action space is limited so that the increasing number of tasks would not affect the decision performance of the orchestrator based on the neural network, there is a limited area including several edge servers for which an orchestrator is responsible. We assume that the orchestrator resembles the controller in a Software-Defined Network. Thus, for larger areas, several similar orchestrators will be needed.

### 4.1.5. Reward Mechanism

As a DRL agent, the long-term goal of the orchestrator is to minimize the failed task rate by making successful offloading decisions. In other words, the orchestrator finds a policy $\pi(s_t)$ that maximizes the expected sum of future rewards at $s_t$ by taking

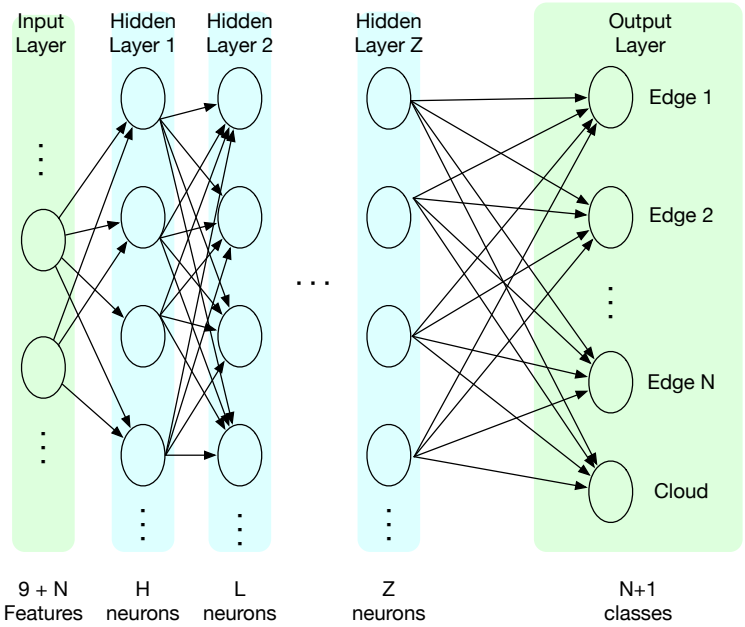



Figure 4.4. The generic deep neural network model for DeepEdge.

the corresponding action $a_t$ to the $s_{t+1}$, such that

$$R_t = \sum_{i=t}^{\infty} \gamma * R(s_i, s_{i+1}) \tag{4.2}$$

where $\gamma \in [0, 1]$ is the discount factor to indicate the importance of the immediate or long-term rewards, and $R_t$ is the long-term reward. We define the reward for each state transition so that if the offloaded task is successfully completed, we assign $R(s_t, s_{t+1}) = 1$, otherwise $R(s_t, s_{t+1}) = -1$.

Since our objective is to maximize the number of successfully offloaded tasks, we formulate the reward function regarding the individual success so that the cumulative reward would reflect it. It is important to note that if our objective consisted of the minimization of the service time or processing time, the reward mechanism would be different for our system. Moreover, since the orchestrator treats each task equally, it takes the same reward for each offloaded task depending on its decision. We do not apply any prioritization for different applications. However, if needed this could easily be employed.

### 4.1.6. Applying DDQN on Edge

In order to achieve its objective function of maximizing the expected sum of future rewards, a DRL agent should choose actions under a deterministic policy $\pi(s_t)$ that also maximizes the action-value function, $Q_\pi(s_t, a_t)$, which is also named as the Q-function. Therefore, under a policy $\pi$, the value of an action $a_t$ in a state $s_t$ is

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[R_t \mid s_i = s, a_i = a\right] \tag{4.3}$$

where $R_t$ is defined in Equation 6.20. As a result, an optimal policy is to select the highest valued action in each state such that

$$\pi(s_t) = arg\max_{a'}(Q(s_t, a')) \tag{4.4}$$

where $a'$ indicates the set of all possible actions.

In the enhanced DQN algorithm [129], there is a target network along with the experience replay in addition to the online network to carry out a smoother learning process. The target network has the same configuration as the online network in terms of the network size and number of the hidden layers. The only difference between them is the network parameters. The online network parameters, $\theta_t$, are copied to the target network having parameters $\theta_t^-$ for each $\tau$ step, and both of them are fixed in other steps. Thus, the target in DQN is determined as

$$z_t = R_{a_t}(s_t, s_{t+1}) + \gamma * Q(s_{t+1}, arg\max_{a'} Q(s_{t+1}, a'; \theta_t^-); \theta_t^-) \ . \tag{4.5}$$

However, as investigated in [5], the DQN algorithm causes overestimation of the target values since the target network is used for both selecting and evaluating the action. Thus, in this study, we use the DDQN algorithm [5] in which the online network is used for the selection of the action, and the target network is exploited for evaluating

the action as follows

$$z_t = R_{a_t}(s_t, s_{t+1}) + \gamma * Q(s_{t+1}, arg\max_{a'} Q(s_{t+1}, a'; \theta_t); \theta_t^-) \ . \tag{4.6}$$

Note that the learning objective of the orchestrator is to minimize the temporal difference error, $\delta_t$, of $Q_\pi(s_t, a_t)$ regarding the target value $z_t$:

$$\delta_t = \mid Q(s_t, a_t) - z_t \mid \ . \tag{4.7}$$

### 4.1.7. The Delayed Action Effect

In DRL, the agent gets the next state information from the environment immediately and updates its parameters based on the result of the value function, which is the reward. Since the next state information is a direct result of its action, the agent updates its parameters accurately. However, in our problem, because of the service time, MAN delay, and WAN delay, the agent cannot immediately get the next state information which is the outcome of its action. This is an important issue since the agent cannot learn the dynamics of the environment as it cannot observe the effect of its actions instantaneously. To solve this delayed action effect, we adapt DDQN into our environment as shown in Figure 4.5.

In our solution, we use memory items each of which holds 5-tuple information including *state*, *next state*, *action*, *value*, and *isDone* elements. These elements are essential for the DDQN algorithm in order to feed the agent's memory considering the experience replay and the neural network. After an action is taken, traditionally, they are given to the agent immediately. Since we cannot provide the effect of the action immediately due to our environment, we first insert the *state* and *action* elements of the current *task* into a memory item. Note that since each state is also the next state of its previous state, except for the very first state of the system, we also insert the *state* into the memory item of the previous task. By using the same fashion, the *next*

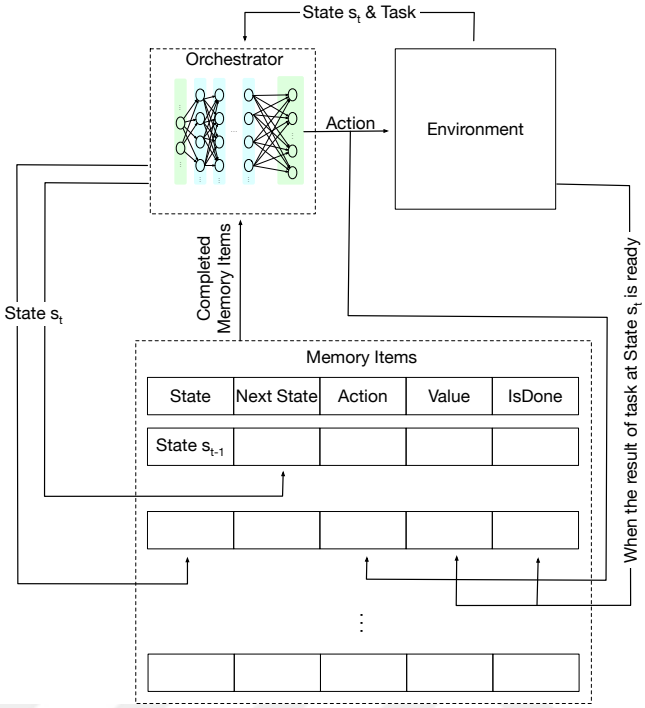



Figure 4.5. 5-tuple memory item structure to feed the DDQN algorithm.

*state* is filled for our current *task*. Afterwards, when the result of the current *task* is available in the environment, we place the value, regarding the reward mechanism, into the memory item. Moreover, if the task is the last task in the system, we set the *isDone* element as true. Note that indicating the last element is also imperative in the DDQN algorithm in order to ensure MDP. Finally, if all elements of a memory item are set, it is sent to the memory of the agent considering the experience replay which is used to feed the online network for training. Thus, we can perform online training by overcoming the delayed action effect. We also give the details of our scheme in the corresponding algorithms shown in Figure 4.6 and Figure 4.7, respectively. Since DDQN is a well-known algorithm, we only give the steps of our approach until DDQN is fed by a memory item in the algorithm shown in Figure 4.7.

In the algorithm given in Figure 4.6, our input is the task and our goal, the output, is the creation of the memory item considering the delayed action effect. To this end, we first create an empty memory item in the first line and get the *currentState*. When the agent takes the action, we store this information in the *action* variable and increase

**Input:** $task$

**Output:** action $a_t$ and memory item of $task$

1: $mItem \leftarrow CreateMemoryItem()$
2: $currentState \leftarrow GetState()$
3: $action \leftarrow agent.doAction(currentState)$ // using online network
4: **if** $action = EdgeServerOfTheTask$ **then**
5: $\quad NumberOfTaskToWLAN{+}{+}$
6: **else if** $action = OtherEdgeServer$ **then**
7: $\quad NumberOfTaskToMAN{+}{+}$
8: **else**
9: $\quad NumberOfTaskToWAN{+}{+}$
10: **end if**
11: $mItem.currentState \leftarrow currentState$
12: $mItem.nextState \leftarrow null$
13: $mItem.value \leftarrow null$
14: $mItem.action \leftarrow action$
15: $mItem.isDone \leftarrow false$
16: $stateToMemItem(currentState.id, mItem)$
17: **if** $stateToMemItem(currentState.id - 1) \neq null$ **then**
18: $\quad prevItem \leftarrow stateToMemItem(currentState.id - 1)$
19: $\quad prevItem.setNextState(currentState)$
20: **end if**
21: $taskToStatePair(task, currentState)$
22: $OffloadTaskToDestination(action, task)$

Figure 4.6. The algorithm of handling the incoming tasks considering the delayed action effect.

the corresponding values from line 4 to 10 considering to where the task is offloaded. Those values are used as features for the next state. Afterwards, from line 11 to 16, we

**Input:** A completed *task*

**Output:** training of neural networks

1: **if** $task = isFailed$ **then**
2: $reward \leftarrow -1$
3: **else**
4: $reward \leftarrow 1$
5: **end if**
6: $state \leftarrow taskToStatePair(task)$
7: **if** $stateToMemItem(state.id) \neq null$ **then**
8: $memItem \leftarrow stateToMemItem(state.id)$
9: $memItem.reward \leftarrow reward$
10: **if** $task.id = 75000$ **then**
11: $memItem.isDone \leftarrow true$
12: **end if**
13: **if** $memItem.nextState \neq null$ **then**
14: $agent.DDQN(memItem)$ //which saves the memory item into the agent's memory for the experience replay
15: **end if**
16: **end if**

Figure 4.7. The algorithm of handling completed tasks and memory items.

fill the memory item with the corresponding values. Since we do not know the value and the next state information, *null* is assigned to those elements initially. Next, we store the $(currentState, memoryItem)$ tuple into the $stateToMemItem$ dictionary in order to change the memory item when we obtain the result of the task. If the current state is not the first state in the environment, we set the next state element of the previous memory item as the $currentState$. Finally, we store the $(task, currentState)$ pair in the $taskToStatePair$ dictionary and then offload the task to the destination based on the selected action. As a result, the time complexity of the algorithm is based on the

feedforwarding operation of the online network considering the worst-case. Therefore, the multiplication of the online network parameters and the corresponding state values in the *doAction* method in line three define the time complexity. On the other hand, the space complexity is based on the size of the created memory item.

In the algorithm given in Figure 4.7, our input is the completed task and the output is the completed memory item to feed the agent's memory. Hence, first, the reward is set based on the fail or success condition of the offloaded task. Afterwards, the corresponding state is obtained from the *taskToStatePair* dictionary in line 6 in order to acquire the memory item related to the state. Next, the *value*, and *isDone* elements of the memory item is set from line 7 to 12. Finally, if the next state information is set, then the memory item is given to the memory of the agent for the training. Here, considering the *DDQN* method which initiates the training, the time complexity of the algorithm is based on the multiplication of the online network parameters, corresponding state values, and batch size. Similarly, the space complexity is based on the multiplication of the batch size and memory item size.

## 4.2. Performance Evaluation of DeepEdge

We carried out our experiments in the EdgeCloudSim [130] simulator for the performance evaluation. Our scenario consists of 14 locations, each of which operates an edge server with equal capacity. We used different numbers of mobile devices in the edge computing environment to evaluate the performance of the algorithms under variable load. Mobile devices run different applications including augmented reality, pervasive health, image rendering, and infotainment. Each application generates tasks that have different characteristics based on their requirements given in Table 4.2. Task interarrival time indicates the average task generation rate. Delay sensitivity refers to an application's tolerance to latency. This value is high for delay-intolerant applications such as video conferencing. VM utilization species the percentage of processing resources required to execute the related task.

Table 4.2. Application properties in the simulator.

| **Property** | **Augment. Reality** | **Perv. Health** | **Image Rend.** | **Infot. App** |
|---|---|---|---|---|
| Task Interarrival | 2 sec | 3 sec | 20 sec | 7 sec |
| Delay Sensitivity | 0.9 | 0.7 | 0.1 | 0.3 |
| Data Upload/Download (KB) | 1500/25 | 20/1250 | 2500/200 | 25/1000 |
| VM Utilization on Edge (%) | 6 | 2 | 30 | 10 |
| VM Utilization on Cloud (%) | 0.6 | 0.2 | 3 | 1 |
| Usage Percentage (%) | 30 | 20 | 20 | 30 |

### 4.2.1. Configuration

4.2.1.1. Simulator Parameters. We deployed 14 edge servers in the network each of which has 8 VMs, and one cloud server that has 4 VMs. While the computational resource capacity of each VM in an edge server is 10 billion instructions per second (GIPS), a VM in the cloud server has 100 GIPS capacity. For each experiment, the number of mobile devices in the network ranged from 200 to 2400. The average number of offloaded tasks is based on both the number of mobile devices and the average task generation rates given in Table 4.2. Thus, the expected number of generated tasks would be 6500 and 80000 for 200 and 2400 mobile users, respectively.

We repeated our experiments 40 times with different random seeds. The duration of each experiment in simulator time was five minutes. Since we want to conduct a valid comparison with the competitors, we used the empirical results for the WAN and WLAN delays explained in [128]. These values can also be observed in EdgeCloudSim. For MAN delay, we used the M/M/1 queueing model implementation of EdgeCloudSim with an additional propagation delay of 5 ms.

4.2.1.2. Training the Agent. Since we used 14 edge servers in the network, the output layer of the neural networks regarding online and target networks consisted of 15 nodes with respect to our generic model shown in Figure 4.4. Our final model includes two hidden layers each of which consists of 128 neurons. We applied the ReLU activation

Table 4.3. Hyperparameters and their values.

| **Hyperparameter** | **Value** |
|---|---|
| Learning Rate | 0.0001 |
| Discount Factor | 0.8 |
| Initial Exploration | 1 |
| Exploration Factor | 0.99 |
| Final Exploration | 0.1 |
| Replay Memory Size | 1000000 |
| Minibatch Size | 4 |
| Target Network Update Frequency | 10 |

function for each neuron in the input and hidden layers. On the other hand, the linear activation function was used for the output layer. We used Stochastic Gradient Descent (SGD) for the optimization of the model. Other important hyperparameters for DDQN are given in Table 6.2. In order to define hyperparameters for the *discount factor*, *minibatch size*, and *target network update frequency*, we used random search regarding our experience in the domain. On the other hand, for the *initial exploration*, *exploration factor*, *final exploration*, and *replay memory size*, we used well-known values from the literature.

Since we use the same agent for the simulations of the different numbers of users, training should provide a robust online neural network considering the different dynamics. Therefore, there are two options: (1) we can train the agent for each network condition regarding 200 to 2400 users, or (2) we can train the agent only in the scenario of 2400 users since it will reduce the training time, and this scenario represents the most challenging load for the system. Due to prolonged execution times, we opted for the second option. To give a numeric comparison, completion of a single episode took over three hours for the first option whereas it took around 0.3 hours for the second option. It took 101 episodes (over 30 hours) to train the agent and converge to the range of 10% - 15% failed task rate.

Figure 4.8 shows the rewards and percentage of failed tasks for four different learning rates during the training of the agent. Since the learning rate is a crucial

hyperparameter to train a neural network, we selected rates based on a range from 0.00001 to 0.001. Small learning rates cause a slow learning process, which is reflected in the case of 0.00001. Therefore, as shown in Figure 4.8, the reward converges to its maximum value after episode 40 for this learning rate while other learning rates result in the same reward range after episode 20. On the other hand, increasing the learning rate does not affect the speed of the learning process in the case of 0.0001, 0.0005, and 0.001. The learning rate of 0.001 gives poor results after episode 60 because of the exploding gradient problem which is a result of both the linear activation function in the output layer and the neural network architecture including the number of layers and nodes. The exploding gradient problem causes extremely high weights in the neural network and therefore the model cannot learn anymore. It is also an indicator that the learning rate must be lowered. Hence, in our training phase, we focused on the other three learning rates considering 0.001 as the upper bound. Finally, we selected the final model which is trained by the learning rate of 0.0001 since it is the most successful regarding the total reward. The learning rates 0.0005 and 0.00001 are not as successful since they cannot exceed local optima due to their smaller values.

On the other hand, considering the rewards in the training phase, the upper bound is 75000 on average due to the total number of tasks for the scenario of 2400 mobile users. Similarly, the lower bound for rewards is -75000. Since we used the DDQN algorithm for the learning, the oscillation between episodes is smoother so that it is limited between rewards of 55000 and 45000. This is important since the time to converge would be higher in the traditional DQN algorithm as shown in [5]. As a result, since the corresponding rewards do not significantly change, we choose our final model after the 100th episode.

4.2.1.3. Competitors. We evaluated the performance of DeepEdge considering our primary goal, which is minimizing the failed tasks. Hence, we applied four different approaches for the orchestration that are used in [128] in order to examine the performance of the DeepEdge orchestrator appropriately. In these approaches, the *network based* method first investigates the WAN data rate, which is initially 20 Mbps for each

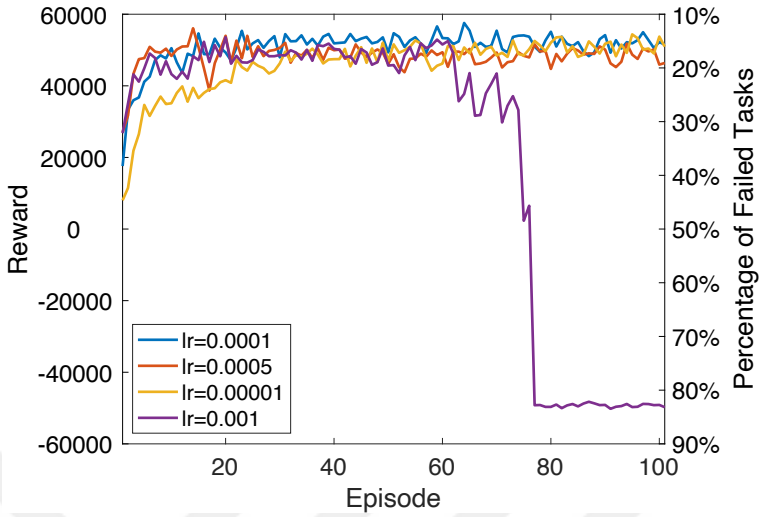



Figure 4.8. Change of the failed task rate over episodes for the scenario of 2400 mobile users.

edge server. Afterwards, if the data rate is above the predefined threshold which is 6 Mbps, it forwards the task to the cloud. Otherwise, the task is offloaded to one of the edge servers that have the capacity to complete the task. On the other hand, the *utilization based* approach checks the utilization of edge servers. If the total utilization of the edge servers is under 80%, the task is offloaded to one of the edge servers in the network. However, if the total utilization of edge servers is higher than 80%, the task is offloaded to the cloud. This threshold value assignment is based on a set of exploratory simulation experiments with different parameters. According to the simulation results, the best values for minimum WAN data rate and maximum CPU utilization are observed as 6 Mbps and 80%, respectively. Apart from the results, the candidate threshold values were 12, 8, 6, 3 Mbps and 90, 80, 70, 60 percent for the minimum WAN data rate and the maximum CPU utilization, respectively. The *hybrid* approach is a joint method of the network and utilization based techniques. Lastly, we used the *fuzzy based* approach that is implemented in [128] as the main competitor. We used the same input parameters in this study for the fuzzy logic process in order to perform a valid comparison.

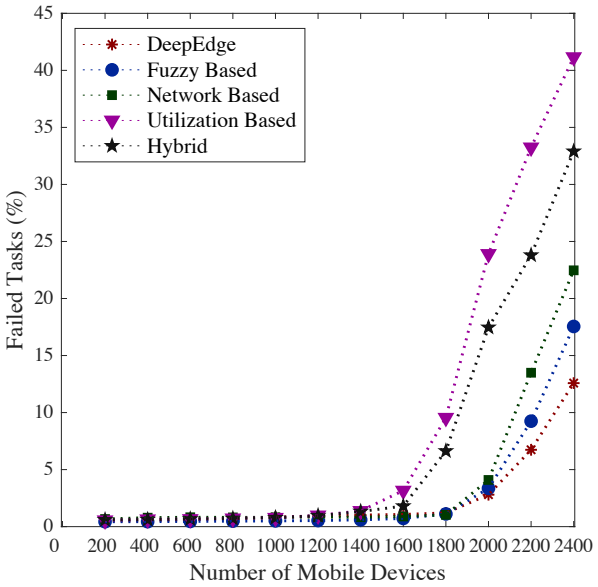



Figure 4.9. The percentage of failed tasks.

### 4.2.2. Experiments

4.2.2.1. Overall Results. We first examined the overall performance of DeepEdge in terms of the percentage of failed tasks. As shown in Figure 4.9, the performance of all approaches is similar until 1600 mobile devices in the network. However, as the load builds up from 1600 to 2000 mobile devices, the accuracies of *hybrid* and *utilization based* approaches decrease. Especially when the number of mobile devices is higher than 1800, DeepEdge outperforms all other approaches. Considering the fact that the competitors are based on heuristic methods, and *fuzzy logic* has 81 hand-crafted rules, it is a notable achievement that the agent explored the environment on its own and created the corresponding rules.

We also analyzed how DeepEdge efficiently uses network and edge resources under high load. Figure 4.10 shows that DeepEdge utilizes the resources in the network more efficiently than the other approaches, especially after 2000 users. This result is the effect of its accurate decisions for the offloaded tasks in the network considering the conditions of the edge servers and network data rate in long term. Since the decision

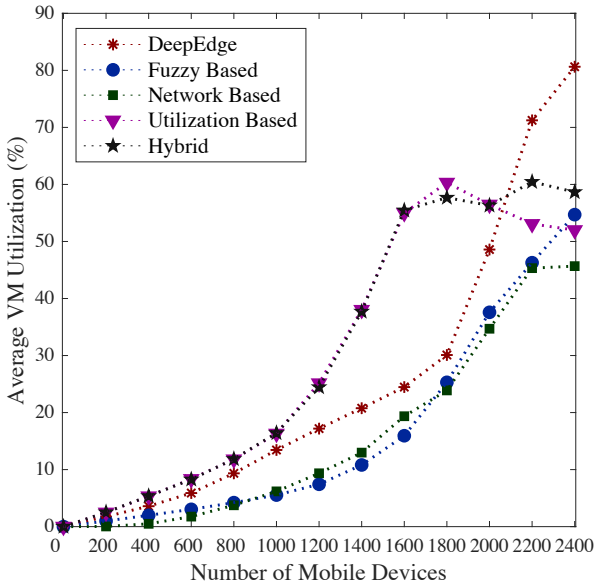



Figure 4.10. Average VM utilization of edge and cloud servers based on the number of mobile devices.

of other approaches may cause congestion on the WAN or MAN, they cannot utilize computational resources well for the heavily-loaded cases. Thus, there is a ceiling effect in heavily-loaded cases since the overload on the network due to their inappropriate policies causes underutilized VMs on servers.

Next, we examined the overall service time for our proposed approach considering the network load in terms of tasks. Figure 4.11 exposes that the service time of DeepEdge is higher than the other approaches, particularly for the high number of tasks. Actually, this is not surprising since our objective is that maximizing the success rate of the offloaded tasks, not minimizing the service time. Due to we design the reward mechanism based on our objective, DeepEdge sacrifices the service time to maximize the cumulative reward. On the other hand, the cause of this effect can also be seen in Figure 4.12 more clearly in which we investigate the processing time on the cloud. It can be observed that DeepEdge offloads the tasks to the edge tier more when the number of tasks increase. This choice increases the overall service time in the end since some of the CPU-intensive apps' tasks would be sent to the edge rather than the

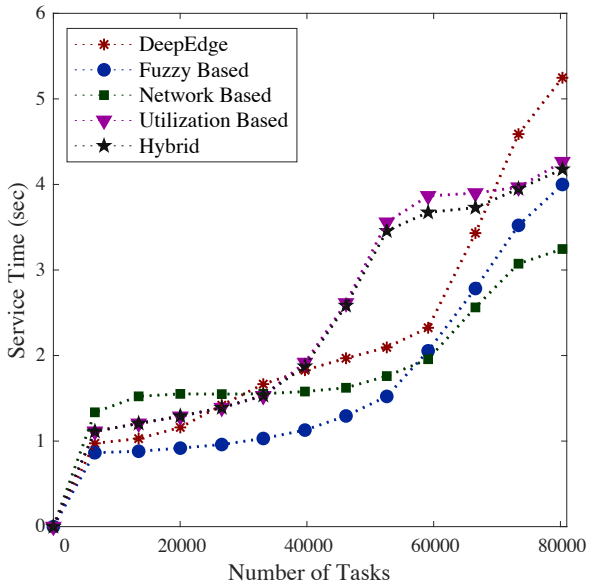



Figure 4.11. The overall service time.

cloud. However, it also provides the minimization of the failure rate, which is the goal of the DRL agent.

4.2.2.2. Application-based Results. Based on the overall results, we then investigated the performance of DeepEdge for each application as shown in Figure 4.13. Considering the augmented reality, pervasive health, and infotainment applications, the robustness of DeepEdge manifests itself for the heavily-loaded environments. For the augmented reality application, our orchestrator offloads the generated tasks generally to the edge considering its delay sensitivity, which is relatively higher. The same requirement is also valid for the pervasive health application. However, since both applications cause different loads on a VM, our orchestrator also takes these properties into account regarding the *TaskReqCapacity* feature. On the other hand, the delay sensitivity of the infotainment application is low. However, since its impact on VM utilization in terms of the load is high, the orchestrator must choose edge or cloud for the offloading wisely. The results show that DeepEdge learned those requirements well. Thus, its performance is better than its competitors.

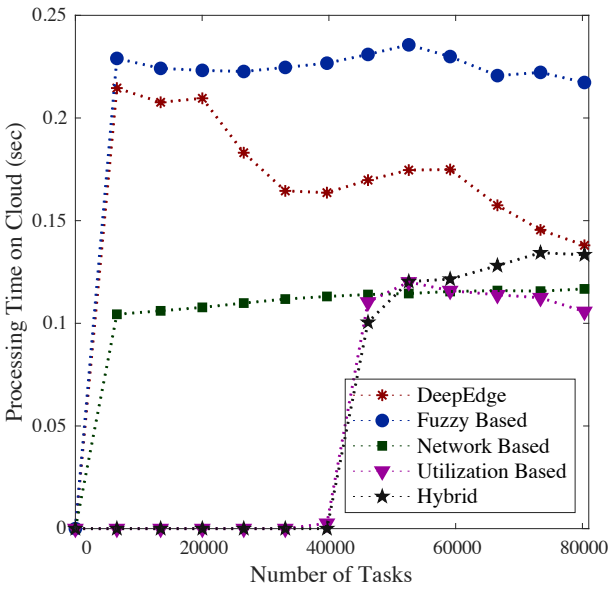
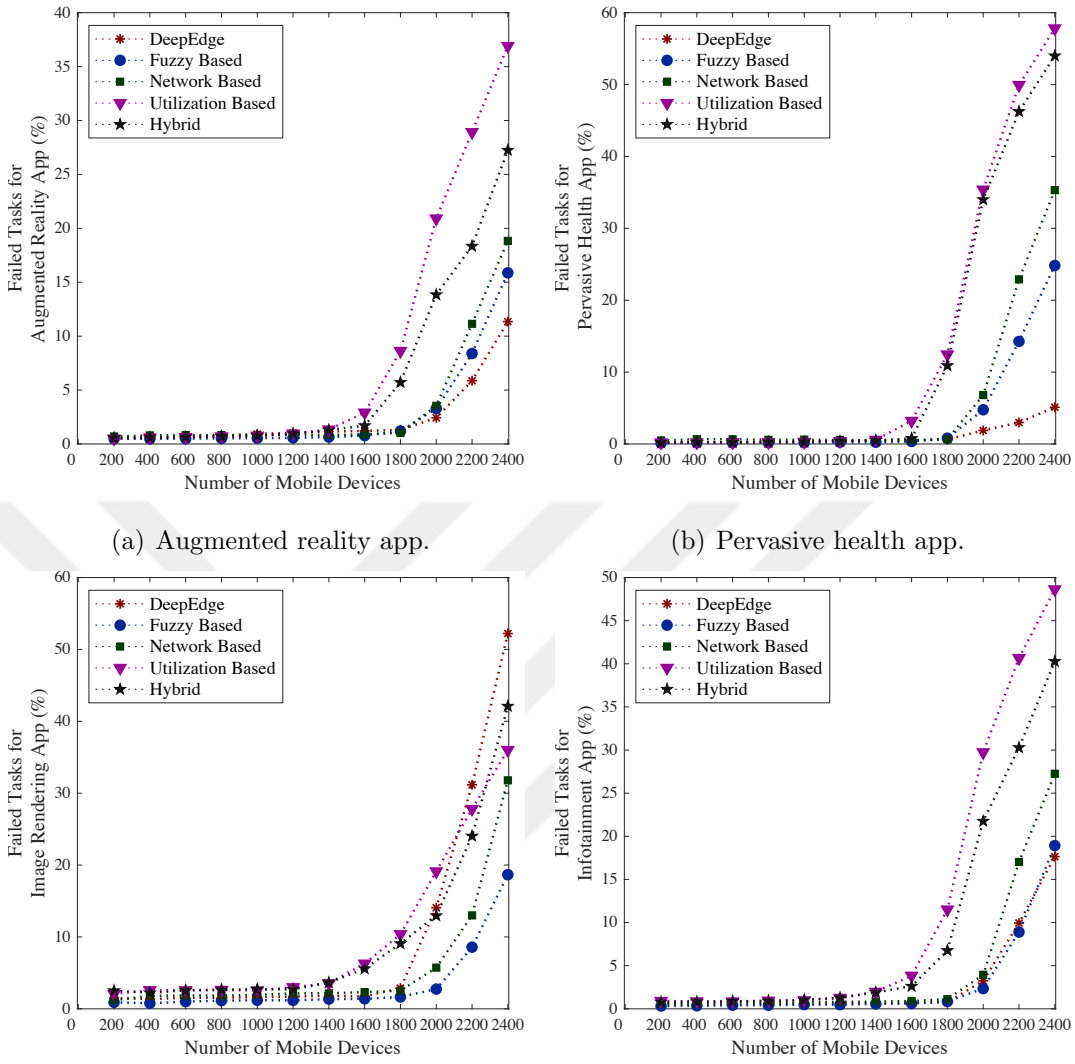



Figure 4.12. The processing time on cloud.

On the other hand, even though DeepEdge outperforms other approaches considering the different application types which have diverse task requirements, it cannot provide good results for the image rendering app. Therefore, in order to investigate this result, we performed several experiments using only the image rendering application. Hence, we trained three DDQN agents as the orchestrator by using the same hyperparameters given in Table 6.2. For these three agents, we only changed task interarrival times of the image rendering application for training. To this end, we applied 20 sec, as the original value, 10 sec, and 5 sec interarrival times. Finally, we compared the performance of DeepEdge with the Fuzzy Based approach as it gives the best performance for the image rendering application shown in Figure 4.13(c).

The results given in Figure 4.14 show that DeepEdge outperforms the Fuzzy Based approach when the task interarrival time is 5 sec. On the other hand, there is no significant difference between the performance of 10 sec and 20 sec interarrival times. Thus, apart from the results in Figure 4.14, the reason that DeepEdge is not as successful for the image rendering app after 1800 users can be concluded by examining Table 4.2. Since the average number of arriving image rendering tasks is 10, 6.66,




(a) Augmented reality app.

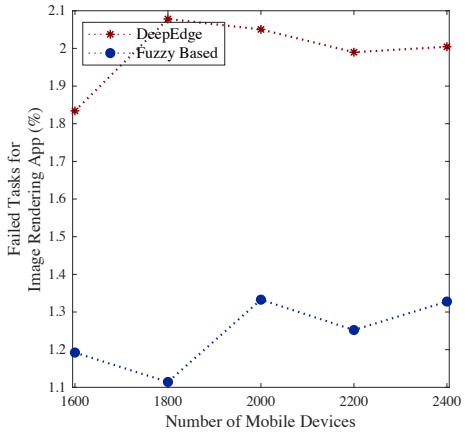
(b) Pervasive health app.

(c) Image rendering app.

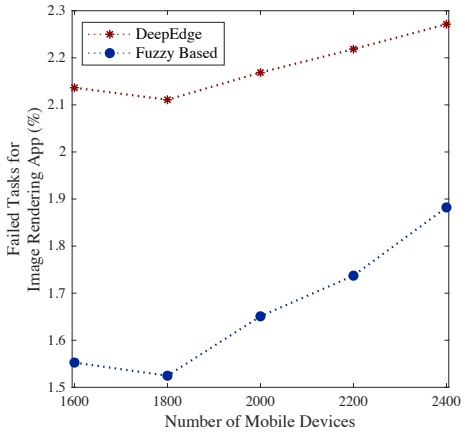
(d) Infotainment app.

Figure 4.13. The change of failed tasks based on different application types. If the number of mobile devices increase in the network, DeepEdge outperforms other approaches in terms of successfully completed tasks.

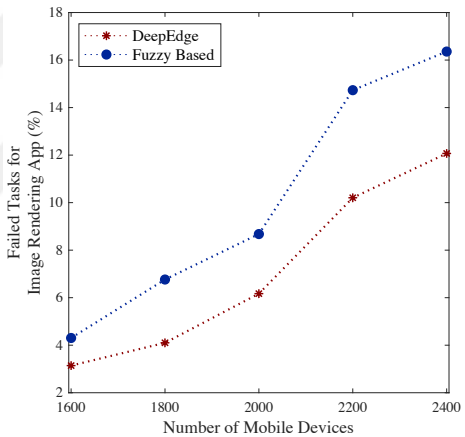
and 2.85 times lower than the other applications respectively, states related to the decision of the image rendering tasks are relatively less for the training. Therefore, the agent could not learn the specific details of the requirements of the image rendering app. Furthermore, note that, since the image rendering tasks constitute only 3% of the overall task offloading requests, the overall performance of DeepEdge is still significantly better for the heavily-loaded cases as shown in Figure 4.9.




(a) 20 sec task interarrival.




(b) 10 sec task interarrival.




(c) 5 sec task interarrival.

Figure 4.14. The effect of the task interarrival time on the image-rendering application.

## 4.3. Discussion

We implemented a DRL based task orchestrator, namely DeepEdge, for edge computing. Throughout this chapter, we assumed that the offloading decision had been taken by mobile devices, and the task must be offloaded to one of the edge servers in the network or to the cloud. Due to the different characteristics of each application, generated tasks by mobile users must be handled by the orchestrator considering the application requirements and the current network conditions. For this reason, we de-

veloped DeepEdge by applying the DDQN algorithm. Even though providing MDP and coping with the delayed action effect are challenging due to the real-time, and stochastic nature of the edge computing environment, we successfully implemented a DRL agent based on DDQN for the orchestration. We conducted experiments in the EdgeCloudSim simulator by comparing our model with other approaches in the literature. Our results showed that DeepEdge has the capability to perform orchestration for different task types and outperforms its competitors. On the one hand, DeepEdge improved the average number of successfully completed tasks considering the different application requirements. On the other hand, DeepEdge used VM resources more efficiently since it had prevented the network bottlenecks so that offloaded tasks could arrive at their destinations. To the best of our knowledge, this is the first study in the domain of edge computing that evaluates the performance of a DRL model for different application types under various loads in terms of the number of mobile users.

# 5. AIRCOMPSIM: A DISCRETE EVENT SIMULATOR FOR AIR COMPUTING

The extensive utilization of smart devices by end-users results in various application types that have different goals. Therefore, providing a necessary QoS based on the corresponding SLA requirements is currently challenging for the existing network paradigms. Even though multi-access edge computing (MEC) [93] has been widely used for this purpose, its static deployment in which the capacity of the edge server(s) cannot be changed based on the dynamic needs would cause an issue to meet the requests of the user tasks. Thus, UAVs and similar aerial units have recently been used by many studies in order to enhance the performance of the network environment in terms of QoS [131–133].

Considering varied deployment cases of UAVs and related air vehicles to alleviate the restricted 2D networking environment, we recently proposed air computing which is a next-generation computational paradigm as a result of the evolution of MEC [134]. In this regard, air computing represents a dynamic and responsive computation environment for all spectrum of applications through different air platforms. Hence, in addition to the ground layer, there are three air platforms in air computing. Each of these air platforms can provide different opportunities in terms of latency, data rate, computational capability, coverage, and mobility to the corresponding applications using the benefits of 3D networking. The joint operation of those air platforms in a dynamic environment to meet different requirements are still investigated.

Even though air platforms, especially UAVs, have recently been used by many studies for varied goals such as task offloading, content caching, and trajectory planning, there are few simulators that consider these scenarios in the literature. Moreover, they do not allow mobile users, applicable UAV policies, and the definition of different application types, which may have different SLA requirements, such as maximum tolerable delay. In most of the cases, existing simulators focus on the flight controls,

visualization and physics [135]. In [136], Shah et al. proposed AirSim which is built on Unreal Engine and focused on realistic flying simulation of aerial vehicles. Their simulator includes a physics engine for real-time hardware-in-the-loop simulations. D'Urso et al. represented an integrated simulator for UAVs for the realistic simulations in [137]. Therefore, they developed a software middleware that coordinates the tools including Gazebo [138], ArduCopter [139], and ns-3 [140] to work together. Similarly, Baidya et al. proposed a middleware for the realistic UAV simulation in [141]. They focused on the network dynamics and modeling UAV operations so that they interfaced ArduPilot [139] and ns-3 tools through their simulator. To the best of our knowledge, none of the existing simulators consider application performance in terms of task completion, user mobility, and UAV flying policy based on the system load and computational resources.

In this chapter, we introduce a new simulator, namely AirCompSim, which ensures a discrete event simulation environment to conduct complex air computing experiments based on varied scenarios [142]. AirCompSim achieves that by providing a modular structure. Therefore, it is straightforward to alter the corresponding parameters, add new scenarios including dynamic events such as server failure, and create new flying policies for UAVs. Moreover, it also provides an event mechanism for deep reinforcement learning (DRL) so that smart and sophisticated UAV policies such as where to fly and when to fly can be investigated. Furthermore, since the essential results are logged and stored, it provides an easy mechanism to plot the related results after the experiments. To demonstrate the capabilities of AirCompSim, we present a dynamic capacity enhancement scenario and evaluate the performance of different settings in an air computing environment.

## 5.1. AirCompSim Architecture

AirCompSim provides a modular architecture in which each module can interact with each other through the corresponding instances and interfaces. Moreover, the modular architecture based on the discrete event system ensures that a new module

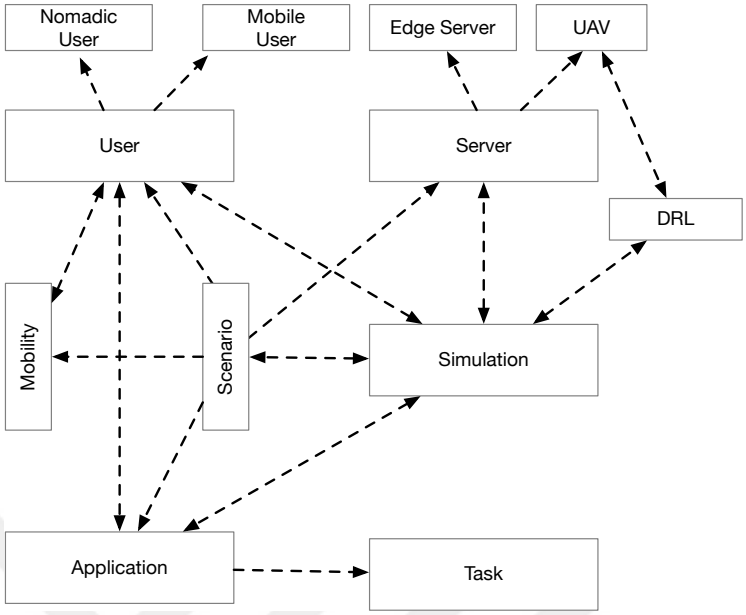



Figure 5.1. Relationship of the AirCompSim modules.

can be easily incorporated into AirCompSim. To this end, AirCompSim has five core modules namely Simulator, Server, User, Application, and Task. The relationship of these five modules is shown in Figure 5.1.

### 5.1.1. Simulation Module

The simulation module is the main module of AirCompSim. It is responsible for handling the events in a simulation via the event queue and creating the corresponding user types, user mobility, and UAV policies. Moreover it provides a detailed logging mechanism including the events, task offloading, UAV flying, and task processing so that it allows a detailed debugging option for the researchers. Furthermore, it saves the results of the repeated simulations as a *csv* file therefore after a detailed experiment it is straightforward to process the results using various libraries such as *pandas* [143].

### 5.1.2. Server Module

The server module defines a generic class for the entities having server features. To this end, this module is used as a parent class for edge servers, UAVs, and the

cloud server. Note that in the current version of AirCompSim [142], UAVs are the only flying vehicles in the simulator. However, it is easy to add additional flying servers with different altitudes based on the server module. This module also provides the utilization, processing delay for the given tasks, and earliest idle time based on the queueing. A server module-based entity has a computational capacity to serve the offloaded tasks based on the $M/M/1$ queueing model.

5.1.2.1. Edge Server. In our simulator, edge servers are located in Local Area Networks (LAN) on the ground. They have a coverage radius in which users can connect to them to offload their tasks. Their capacity and location are fixed throughout an experiment. Even though the corresponding capacity can be set by researchers in the simulation module, in the default settings of AirCompSim, the capacity of an edge server is less than that of a cloud server and higher than that of a UAV.

5.1.2.2. UAV. We consider UAVs as flying edge servers with less capacity in our simulator. Therefore, various flying policies can be applied in AirCompSim in order to observe the effects of different proposals on task success rate, utilization, and service delay. In the default settings of AirCompSim, UAVs fly to the areas that have already been covered by an edge server in order to enhance the corresponding capacity.

5.1.2.3. Cloud Server. A cloud server is the most powerful entity in terms of computational capacity in an air computing environment. However, since it is accessed through the Wide Area Network (WAN), it is prone to higher network delay than LAN and Metropolitan Area Network (MAN). Hence, it is critical to select a cloud server for task offloading when the corresponding maximum tolerable delay is high. In the default settings of AirCompSim, a cloud server is selected by a user if there is neither an edge server nor a UAV to connect.

### 5.1.3. User Module

The user module supports different user types, including mobile users, nomadic users, and users in the air. In the current implementation of AirCompSim there are mobile users and nomadic users. The main difference between them is their mobility pattern. A user can connect to multiple servers simultaneously if it is in the coverage of them. In the default mode of AirCompSim, if a user is connected to multiple servers, such as a UAV and an edge server, the task is offloaded to one of them, which has a lower queueing delay. Here, we assume that users are informed by their connected servers in a separate channel. Note that this default behavior can be changed by a researcher to seek efficient offloading decisions.

### 5.1.4. Application Module

Each user in the simulator runs at least one application, which randomly produces a computation-intensive task based on the application type. Therefore, each task based on an application type consists of a size, required computational units for processing, arrival rate, and maximum tolerable delay tuple. Each task is successfully completed if the total delay, which includes network, processing, and queueing delays, is lower than or equal to the maximum tolerable delay. The performance of the applied policies is evaluated based on the total task success rate, which also indicates the success rate of the corresponding application. For this reason, the location of the users affects the performance since some areas in the environment may not have an infrastructure, such as edge servers, and other places may face congestion.

### 5.1.5. Mobility Module

The mobility module provides user mobility and also heuristic UAV flying policies. Based on the research goals, a user mobility module or a flying policy for UAVs can be added to the simulator. By default, a random waypoint model runs for the mobility of each user in the environment. On the other hand, UAVs fly between areas where

edge servers are deployed in the center based on capacity calculations. A capacity calculation for areas is carried out by including the number of user and their application profiles, and also the total capacity of edge servers. Afterwards, the required number of additional capacity and therefore the number of UAVs are computed. Finally, UAVs are deployed in the corresponding areas. Note that if an area has a higher need for UAVs, that area has a priority for UAV deployment considering a constraint in the number of UAVs.

### 5.1.6. Scenario Module

The scenario module includes the experimental setup which is essential for the performance evaluation. To this end, the number of users, edge servers, UAVs, the location of all kinds of servers, the user mobility model, and also the corresponding capacities of servers are first defined in this module. Moreover, if the experiment includes dynamic updates such as a server failure or sudden increase in capacity demand, those can also be defined in this module.

### 5.1.7. DRL Module

The DRL module consists of the implementation of value-based and policy-based DRL algorithms in addition to the event mechanism which provides the corresponding state information along with the rewards. The implementation of DRL algorithms is supported by PyTorch-based neural networks. Note that since DRL-based studies require more sophisticated settings, AirCompSim only provides the skeleton that can be adapted to the related scenarios by researchers.

## 5.2. Use-Case: Dynamic Capacity Enhancement

In order to demonstrate the capabilities of AirCompSim, we developed a dynamic capacity enhancement scenario in which we investigated the effect of the number of UAVs and users on the task success rate, edge utilization, UAV utilization, service

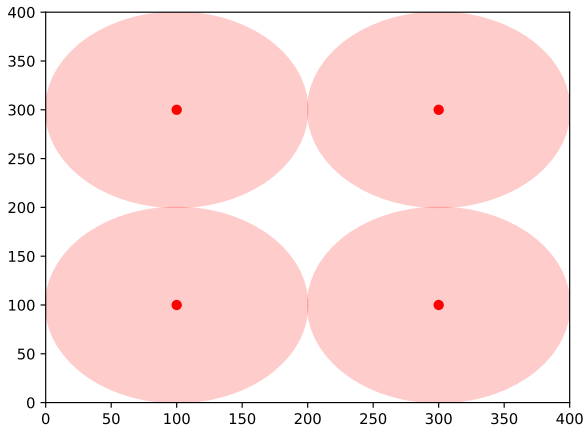



Figure 5.2. Distribution and coverage of edge servers in the example scenario.

time, and the ratio of the offloaded tasks between edge, cloud, and UAV. Moreover, we also evaluated their effect on different application types in terms of task success rate, which is, to the best of our knowledge, provided by none of the existing simulators.

### 5.2.1. Scenario

In our scenario, we have a $400 \times 400 m^2$ environment. We divided the environment into four equal areas and deployed four edge servers in the center of them, as shown in Figure 5.2. On the other hand, each user in the environment is mobile and runs four different application types. Since we run the default mode of AirCompSim, deployed UAVs can only move between four edge server areas. Therefore, as depicted in Figure 5.2, there are areas in the environment that cannot be covered by any server. Users in those areas offload their tasks to the cloud server. To control the variance of results, we repeated our experiments 50 times. Throughout the experiments, we used Python 3.10. The corresponding simulation and application parameters are given in Tables 6.1 and 5.2, respectively.

### 5.2.2. Performance Evaluation

We first evaluated the effect of the number of users and UAVs on the average task success rate in the environment. As shown in Figure 5.3, the task success rate decreases

Table 5.1. Simulation parameters.

| **Parameter** | **Value** |
|---|---|
| Size of a task | 500 Kb |
| Capacity of a serving UAV | 1000 cmp. units/sec |
| Edge Server Radius | 100 m |
| UAV Radius | 100 m |
| Data Rate | 100 Mbps |
| $X_{max}$ | 400 m |
| $Y_{max}$ | 400 m |
| Altitude of UAVs | 200 m |
| Simulation Time | 1000 sec |
| User Mobility Model | Random Waypoint |

Table 5.2. Application parameters.

| **Application** | **Arrival Rate** | **Comp. Load** | **Max. Tolerable Delay** |
|---|---|---|---|
| Entertainment | 10 sec/task | 100 units | 0.3 sec |
| Multimedia | 10 sec/task | 100 units | 3 sec |
| Rendering | 20 sec/task | 200 units | 1 sec |
| Image Classification | 20 sec/task | 600 units | 1 sec |

when the number of users increases since the load in the system affects the utilization of deployed servers. However, this effect would be mitigated by deploying UAVs since they can fly dynamically between edge server areas to enhance the existing capacity. As a result of this, the task success rate is improved when more UAVs are deployed in the system. On the other hand, regarding the simulation and application parameters, the task success rate would be at most 80% in this setting. The first reason for this result is the infrastructure-less areas as shown in Figure 5.2. Since UAVs move to the edge server areas in the default policy of the simulator, users in infrastructure-less areas can offload their tasks only to the cloud which causes high WAN delay. Therefore, application tasks, except the multimedia whose delay tolerance is 3 seconds, would not be completed successfully when they are offloaded to the cloud. The second reason for the task success rate results is related to queueing delay at servers based on their capacity and arrival rate of the offloaded tasks. Hence, the average service time in the system increases exponentially as shown in Figure 5.4. In these results, when no UAVs

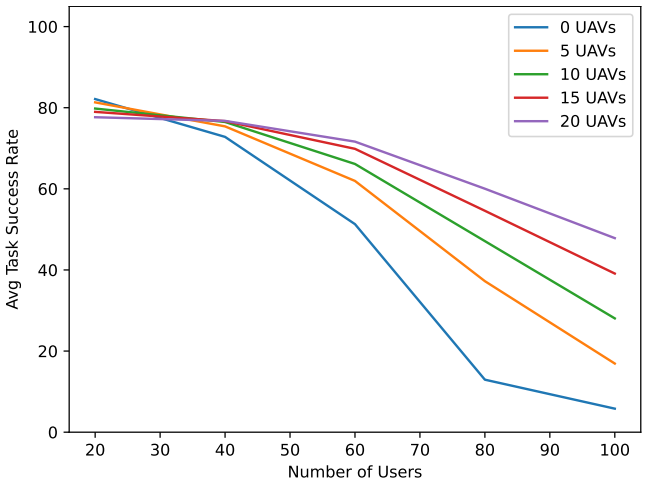



Figure 5.3. Average task success rate.

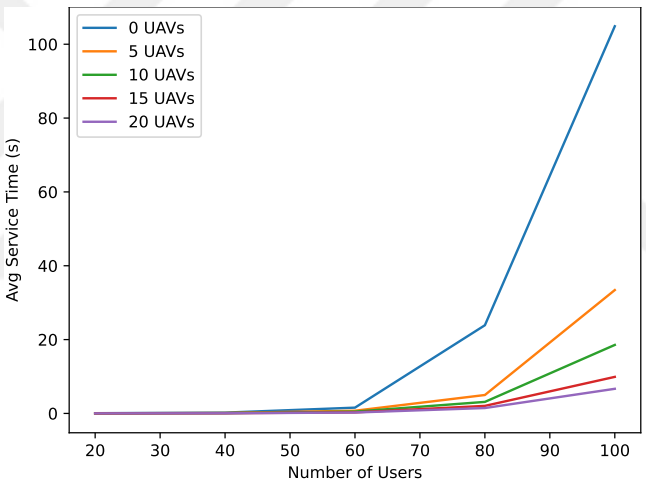
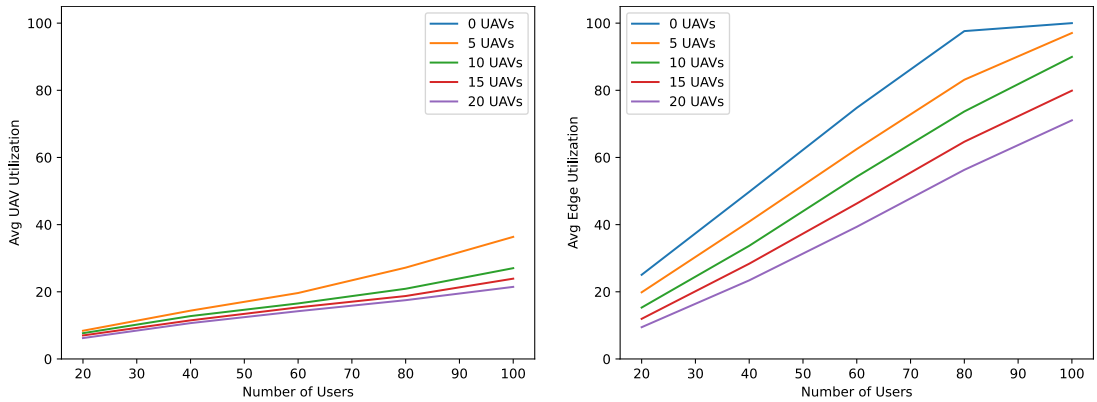



Figure 5.4. Average service time.

are deployed, offloaded tasks undergo longer service time which affects the task success rate heavily. However, when we deploy UAVs, the service time decreases since users can offload the corresponding tasks to UAVs based on the existing queueing delays of all types of servers.

Observing the utilization of the servers in the environment is also essential to analyze system resources. To this end, UAV and edge utilizations based on the number of users and UAVs are shown in Figure 5.5. When there is no UAV in the environment, edge servers are fully utilized while there are 80 and 100 users. On the other hand, for more UAV deployments, the edge utilization decreases since users offload their tasks to



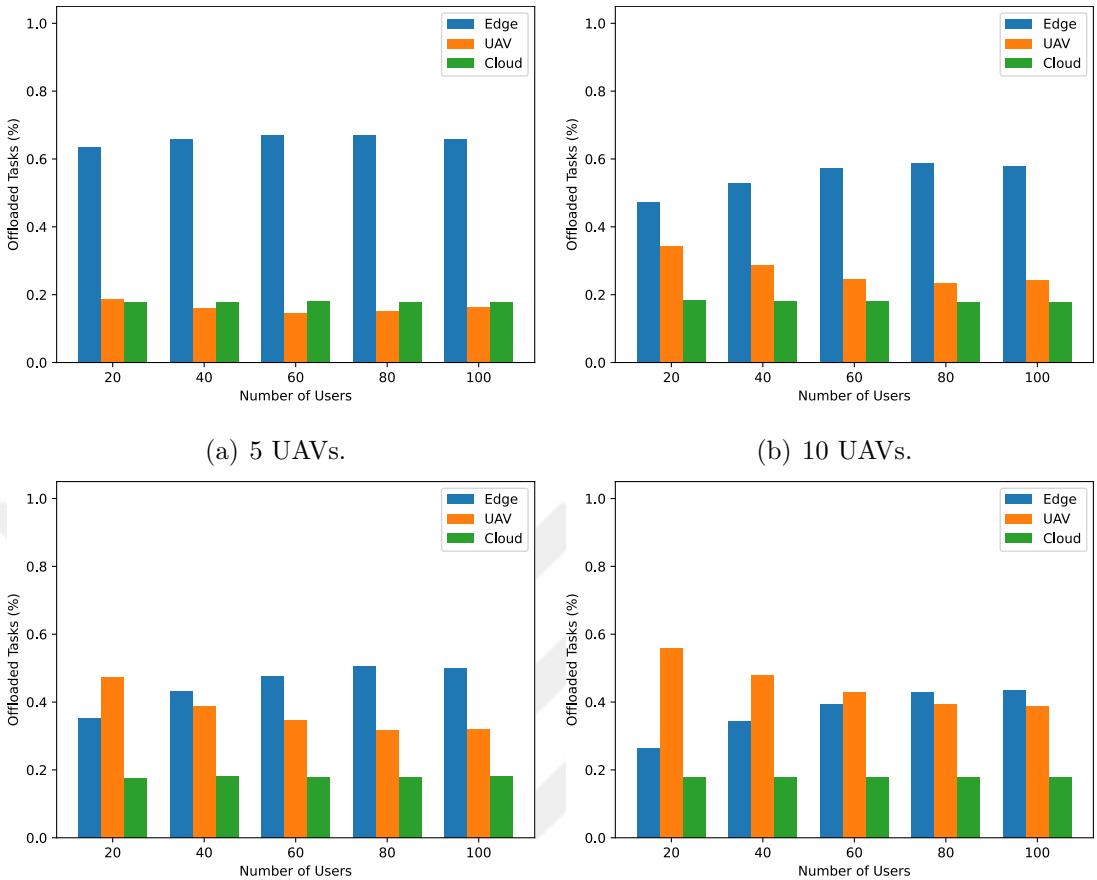

(a) UAV utilization. (b) Edge utilization.

Figure 5.5. Average UAV and edge utilization.

the available UAVs. Considering the UAV utilization as shown in Figure 5.5(a), when there are fewer UAVs in the environment, their average utilization is higher as in this case the total capacity is more limited.

Another effect of the deployment of UAVs is shown in Figure 5.6 which manifests the distribution of offloaded tasks between edge, cloud, and UAVs. Initially, the largest portion of tasks are offloaded to edge servers since most of the environment is covered by them and there are a small number of UAVs. However, for each increment of the number of UAVs, the corresponding offloading distribution shifts to UAVs. The main reason for this case is that users select the less congested server for their tasks which is the default task offloading policy in AirCompSim. On the other hand, the offloading percentage for the cloud server would not change since the infrastructure-less area is fixed in the environment.

Finally, we investigated the average task success rate for each application, which is shown in Figure 5.7. Considering the application parameters given in Table 5.2, these are the expected results. The Entertainment application requires 100 computational units, which should be completed in 0.3 seconds, while the multimedia application requires the same computational units with 3 seconds of time constraint. Note that the arrival rate for both application types is also the same. As a result, tasks of multimedia are completed more successfully than entertainment tasks. Similarly, rendering and




(a) 5 UAVs.

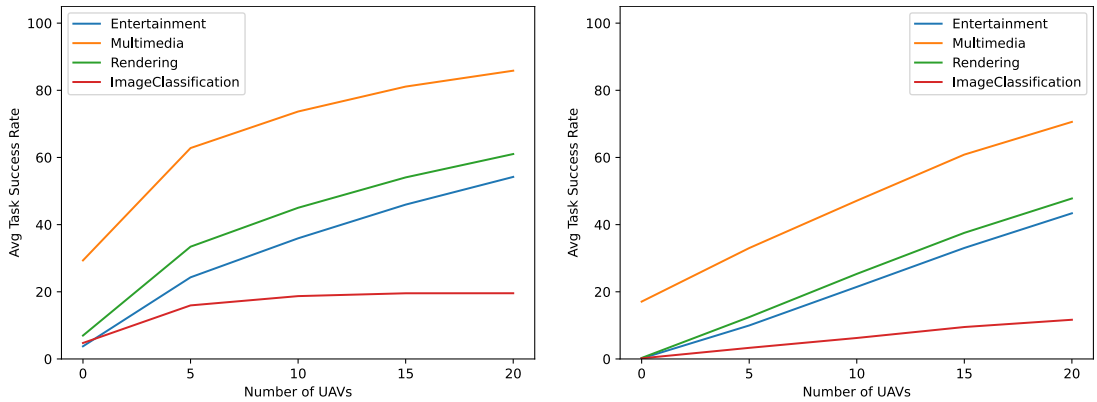
(b) 10 UAVs.

(c) 15 UAVs.

(d) 20 UAVs.

Figure 5.6. Effect of deployed UAVs on offloading.

image classification applications have the same time constraint and arrival rate of 1 and 20 sec/task, respectively. However, image classification requires three times computational unit than that of rendering. Therefore, rendering tasks are processed better than image classification. Moreover, even though rendering requires two times computational unit than that of the entertainment, their tasks are processed more successfully as the arrival rate is two times bigger than the entertainment. Furthermore, the maximum tolerable delay of the rendering application is three times bigger than the entertainment, which affects task completion. On the other hand, considering image classification, the number of UAVs in the environment would not affect the task success rate. The fundamental reason for this result is that image classification requires 600 computational units with 1-second delay tolerance while the capacity of a UAV is 500 computational units/sec. Therefore, none of the deployed UAVs can process image classification tasks successfully in this setting. However, since UAV




(a) 80 users. (b) 100 users.

Figure 5.7. Task success rate of each application type.

deployment relieves edge server resources, it can only slightly affect the task success rate of image classification, as shown in Figure 5.7(b). Note that these are the results of the demonstrative scenario. AirCompSim is flexible enough to evaluate different scenarios.

## 5.3. Discussion

In this chapter, we proposed a new simulator, AirCompSim, in order to provide a development and research environment in which researchers can conduct experiments for the optimization of task offloading, mobility, and flying policies in an air computing environment. Considering the dynamic requirements of air computing, we developed a modular structure for the implementation so that users can extend the existing policies and event mechanism based on their needs. Moreover, we designed a scenario to represent the capabilities of our simulator. To this end, we evaluated task success rate, service time of different server types, utilization of the resources, and application-based performance based on the varying number of users and UAVs. Our results showed that AirCompSim can operate well and therefore would be used as a testbed for air computing research.

# 6. DEEPAIR: A MULTI-AGENT DEEP REINFORCEMENT LEARNING BASED SCHEME FOR AN UNKNOWN USER LOCATION PROBLEM

Among different air vehicles, UAVs are the most studied units since their deployment is easier considering their energy consumption, flying altitude, and configuration [95]. To this end, they are used for dynamic capacity enhancement in environments in which the fixed capacity would not be sufficient to meet the application requirements of an increasing number of users. Therefore, this feature solves variety of problems such as communication in a disaster site, and enhancing services in infrastructure-less environments [81, 84]. Moreover, their deployment provides significant vertical networking opportunities such as high mobility support, coverage, latency, and two-way task offloading [94]. As a result, the requirements of users living in urban, suburban, and rural areas can be met efficiently through these vertical networking opportunities.

There are many studies in which UAVs are used as flying computational units to assist either deployed edge servers or are solely deployed to enhance network capacity for task offloading [81, 144]. Since the battery and CPU capacity of the end users would not be sufficient to process the corresponding tasks, UAVs therefore can be used as a flying edge server. However, since there are many different scenarios, various methods and algorithms are developed to meet the service requirements. DRL is one of those methods that is applied in the literature since the traditional heuristic methods and convex optimization cannot solve the corresponding dynamic problems [145]. To this end, DRL solutions would be used for trajectory optimization, energy-efficient offloading, UAV placement, and generic task offloading.

User connectivity is a primary issue in accessing the related resources for task offloading and service differentiation. However, in order to provide a required service, the corresponding technology such as edge or UAV should first detect the user, and then the connection should be established. However, in an environment which is

infrastructure-less and user locations are unknown, providing those services is a crucial technical challange.

Even though UAVs are used in many different cases, their utilization in an infrastructure-less environment in which users cannot connect to any cellular operator, edge/cloud server, or satellite has not been investigated properly. That environment can be a disaster site, wilderness, or a natural area that is open to visitors. In such an environment, the detection of user locations, localization, and then the measurement of required capacity for user tasks are major issues to ensure the necessary service. Therefore, in this study, we focus on an environment in which there are users at unknown locations where we locate them through a novel method using DRL. Afterwards, as the second step, we estimate the corresponding requirements of the connected users at detected locations and decide how many UAVs are needed in those corresponding areas.

On the other hand, as detailed by Bai et al. [6], DRL-based UAV studies have four main categories including (1) sensing, (2) localization, (3) resource allocation, and (4) UAV-assisted MEC. However, most of the studies in the literature focus on a subset of those categories. Therefore, in addition to the challenging environment, providing a solution for each of those categories is our overall goal.

In this chapter, we develop a DRL-based scheme, DeepAir, which takes unconnected users' emitted RSSI signals into account as a reward and then finds the corresponding user attraction points in the environment. Since the number of user attraction points could not be known, DeepAir iteratively detects those points. Afterwards, based on the quality of those detected locations, users that are in the coverage can connect to agents, which we also call detector UAVs. After these phases, the corresponding user task profiles are extracted through the requests of connected users. Next, the required capacity based on those task profiles is computed, and then the necessary number of serving UAVs, which have MEC features, is measured for the detected user-concentrated areas. Thus, we increase the task success rate based on their Service

Level Agreement (SLA) requirements.

## 6.1. System Model and Problem Formulation

We consider an environment as a set of users denoted as $\mathcal{M} = \{1, 2, ..., M\}$, a set of serving UAVs represented as $\mathcal{N}_s = \{1, 2, ..., N_s\}$, and a set of detector UAVs specified as $\mathcal{N}_d = \{1, 2, ..., N_d\}$. Detector UAVs are used for sensing and localization of users in the environment whose locations are unknown. Serving UAVs, on the other hand, are used as a computational resource for offloaded tasks of users. In other words, serving UAVs are flying edge servers in the environment. Each UAV type has a horizontal radius, $r$, for communicational or computational coverage. Each user $m \in \mathcal{M}$ randomly produces a computation-intensive task $W_m = (D_m, C_m, \lambda_m, T_m^{max})$, where $D_m$ is the size of the task as bits, $C_m$ is the required CPU cycle for processing as cycle/bit, $\lambda_m$ is the arrival rate as task/sec, and $T_m^{max}$ is the maximum tolerable latency as seconds.

In the environment, the location of a UAV $n \in \mathcal{N}_s$ or $n \in \mathcal{N}_d$ can be denoted as $u_n(t) = [x_n(t), y_n(t), z_n(t)]$, where $x_n(t), y_n(t)$, and $z_n(t)$ are the $X, Y, Z$ coordinates at time time $t$. Therefore, the position of UAV $n$ at the next time step for a horizontal flight can be expressed as

$$x_n(t+1) = x_n(t) + d_n(t) \times cos(\vartheta_n(t)) \tag{6.1}$$

$$y_n(t+1) = y_n(t) + d_n(t) \times sin(\vartheta_n(t)) \tag{6.2}$$

where $d_n(t)$ is the flight distance, and $\vartheta_n(t) \in [0, 2\pi]$ is the flight angle. Moreover, the following movement constraints should be satisfied during the horizontal flight considering the area of the environment as

$$0 \leq x_n(t) \leq X_{max} \tag{6.3}$$

$$0 \leq y_n(t) \leq Y_{max} \tag{6.4}$$

where $X_{max}$ and $Y_{max}$ are the maximum lengths of the environment. Similarly, to avoid collision between any two UAVs, including serving and detector UAVs, minimum distance should be satisfied as follows

$$||u_i(t) - u_j(t)|| \geq d_{min} \quad \forall i, j, i \neq j \tag{6.5}$$

where $d_{min}$ denotes the minimum distance. On the other hand, location of a user $m \in \mathcal{M}$ at time $t$ is represented as $L_m(t) = [x_m(t), y_m(t), 0]$ where $x_n(t)$, and $y_n(t)$ are the $X$, and $Y$ coordinates, respectively. Since users are in the ground, the Z coordinates are zero.

If a user $m \in \mathcal{M}$ processes its task locally, the corresponding local computation delay is measured as follows

$$T_m^{loc} = \frac{D_m \times C_m}{f_m} \tag{6.6}$$

where $f_m$ is the computational capacity of user $m$ as CPU cycle per second. On the other hand, if the task $W_m$ is offloaded to a serving UAV $n_s \in \mathcal{N}_s$, the computation delay at $n_s$ is measured as

$$T_{n_s}^{UAV} = \frac{D_m \times C_m}{f_{n_s}} \tag{6.7}$$

where $f_{n_s}$ is the computational capacity of UAV $n_s$ as CPU cycles per second. Since multiple users can be connected and therefore offload their tasks to a serving UAV, $M/M/1$ queueing model is used for the overall delay measurement at the corresponding serving UAV as

$$T_t = \frac{\sum_m^{\mathcal{M}_{sub}} D_m \times C_m}{f_{n_s} - \sum_m^{\mathcal{M}_{sub}} (\lambda_m \times D_m \times C_m)} \tag{6.8}$$

where $\mathcal{M}_{sub}$ denotes a subset of users concentrated in the corresponding area. Therefore, queueing delay at the serving UAV is measured as

$$T_q = T_t - T_{n_s}^{UAV} \ . \tag{6.9}$$

Considering the task offloading case for task $W_m$, transmission delay between a user $m \in \mathcal{M}$ and a serving UAV $n_s \in \mathcal{N}_s$ is measured as

$$T_{m,n_s} = \frac{D_m}{v_{m,n_s}} \tag{6.10}$$

where $v_{m,n_s}$ is the data rate between $m$ and $n_s$ as bit/sec. It is important to note that users connect and communicate multiple serving UAVs via orthogonal frequency-division multiple access (OFDMA). Therefore, the transmission interference between different users can be ignored.

Channel gain, which indicates to the measurement of the strength of the signal between the transmitter and receiver in wireless communication, is computed between a user $m$ and a detector UAV $n_d \in \mathcal{N}_d$ using free-space path loss model as

$$h_{m,n_d}(t) = \frac{g_{m,n_d}}{|d_{m,n_d}(t)|^2} \tag{6.11}$$

where $g_{m,n_d}$ denotes the channel power gain between the user $m$ and detector UAV $n_d$, and $d_{m,n_d}(t)$ is the distance at time $t$. Considering the cumulative signal strength of users at a detector UAV $n_d$, it is measured as

$$H(t) = \sum_{m}^{\mathcal{M}} h_{m,n_d}(t) \ . \tag{6.12}$$

Since serving UAVs are sent to the locations that detector UAVs has already provided, we assume that the channel quality between users and serving UAVs is already above an acceptable threshold and therefore is not included in the formulation.

### 6.1.1. Problem Formulation

In the environment, the total delay for a task of user $m$ is computed as

$$t_{w_m} = t_{network} + t_{service} \tag{6.13}$$

where $t_{network}$ is the network delay, and $t_{service}$ is the service delay. The network delay is computed as

$$t_{network} = \begin{cases} T_{m,n_s}, & \text{if} \quad \beta_{m,n_s} = 1 \\ 0, & \text{if} \quad \beta_{m,n_s} = 0 \end{cases} \tag{6.14}$$

where $\beta_{m,n_s}$ is a binary variable that indicates whether the task is offloaded to a serving UAV or not. While calculating the network delay, the propagation delay is ignored. The service delay on the other hand is computed as

$$t_{service} = \begin{cases} T_{n_s}^{UAV} + T_q, & \text{if} \quad \beta_{m,n_s} = 1 \\ T_{n_s}^{loc}, & \text{if} \quad \beta_{m,n_s} = 0 \ . \end{cases} \tag{6.15}$$

In our environment, a task of user $m$, $W_m$, is successfully completed if the total delay is lower than or equal to the maximum tolerable delay of the task. Hence, we define $\alpha_{w_m}$ as a success variable for task completion as

$$\alpha_{w_m} = \begin{cases} 1, & \text{if} \quad t_{w_m} \leq T_m^{max} \\ 0, & otherwise \ . \end{cases} \tag{6.16}$$

Thus, in this study, our main goal is to maximize the overall task success in the

environment. To this end, our objective function is defined as

$$\max \sum_{m}^{\mathcal{M}} \sum_{w_m}^{W} \alpha_{w_m} \tag{6.17}$$

which subject to

$$\sum_{n_s}^{\mathcal{N}_s} \beta_{m,n_s} \leq 1 \quad \forall m \in \mathcal{M} \tag{Constraint 1}$$

$$d_{m,n_s} \leq r_{n_s} \quad \forall m \in \mathcal{M}, \quad \forall n_s \in \mathcal{N}_s \tag{Constraint 2}$$

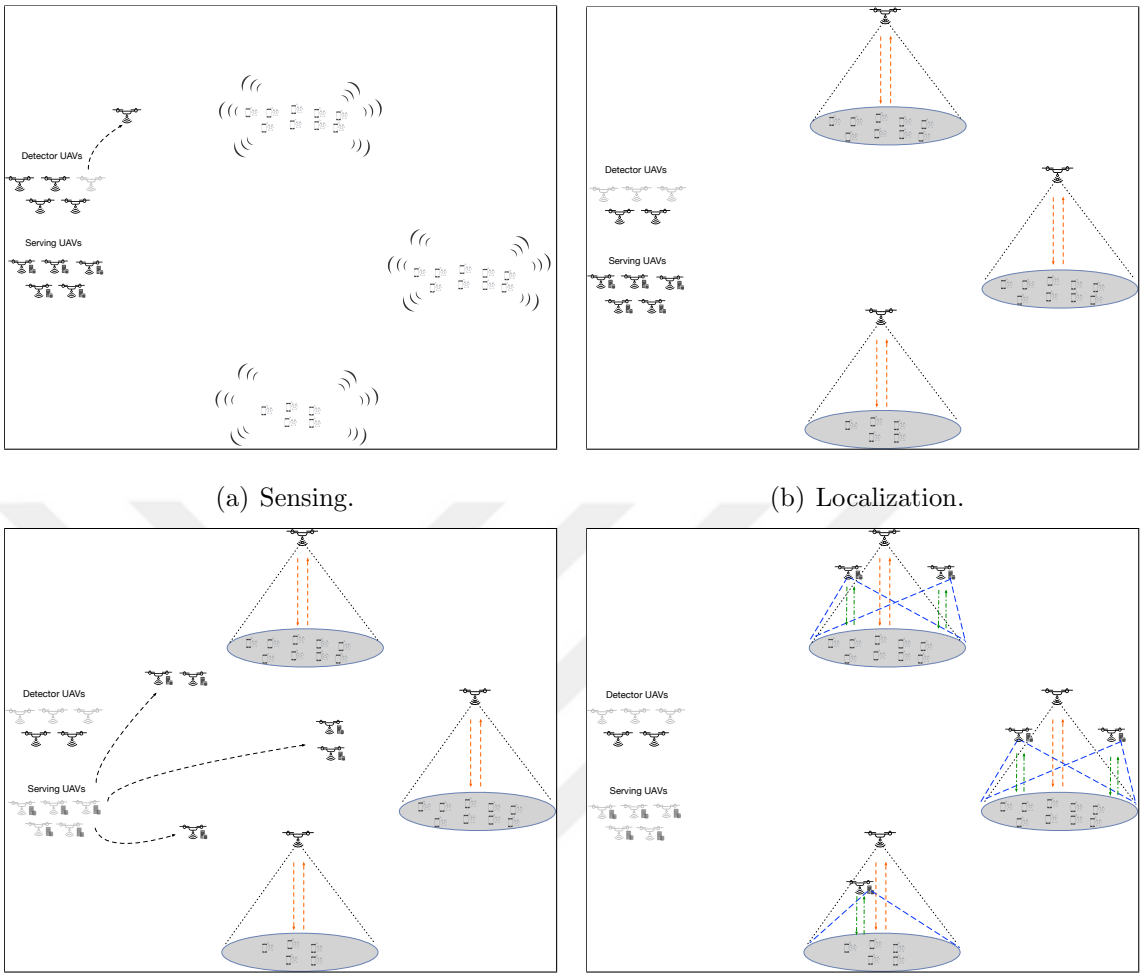
$$f_{n_s} - (\lambda_m \times D_m \times C_m) > 0 \quad \forall m, n_s \tag{Constraint 3}$$

$$\text{Equations } (6.3) - (6.5) \tag{Constraint 4}$$

where $d_{m,n_s}$ is the distance between user $m$ and serving UAV $n_s$. Constraint 1 represents that a task can be offloaded to only a single serving UAV even though the user can connect to multiple serving UAVs. Constraint 2 denotes that a user should be in the coverage of a serving UAV in order to connect it and then offload a task to it. Constraint 3 ensures that offloaded tasks cannot exceed the capacity of a serving UAV. Finally, Constraint 4 describes the movement constraints.

## 6.2. DeepAir

Our DeepAir operation has four main phases that should be considered to provide the required QoS for user tasks. As formulated in Section 6.1, a user task is successfully completed if the total delay in the system is smaller than or equal to its maximum tolerable delay. Therefore, each phase, including sensing, localization, resource alloca-




(a) Sensing.

(b) Localization.

(c) Resource allocation.

(d) MEC.

Figure 6.1. Phases of DeepAir.

tion, and MEC, is significant for the efficient operation in the environment. To this end, we use detector UAVs for the sensing, localization, and resource allocation phases. Note that a detector UAV is also used as a DRL agent in the environment to find the user-concentrated areas. On the other hand, based on the reporting of detector UAVs, we deploy serving UAVs for the offloading and processing of tasks as part of the MEC resources. The whole operation including four phases is depicted in Figure 6.1.

Each type of UAV in the environment can communicate with each other via a separate channel. Thus, they know their existing location. Moreover, they can also communicate with the base, which is at $(0, 0)$ coordinates, so that they can notify the existing situation in the corresponding areas in their horizontal coverage. As a result,

the system can react to the events in those corresponding areas in real-time.

### 6.2.1. Sensing

Due to the nature of the infrastructure-less deployment, initially, users in the environment are not connected to any system component, emitting only signals for a possible connection. Therefore, as shown in Figure 6.3(a), a detector UAV can sense signal strength at some point in the environment at time $t$. Based on the location of users and the detector UAV, that signal strength can change in different areas of the environment considering the cumulative signal strength measurements in Equations 6.11 and 6.12.

In this study, we assume that there are different user attraction points in the environment whose locations are also unknown. Based on those user attraction points as shown in Figure 6.3(a), users are gathered in certain areas. Therefore, the corresponding additive signal strength would be higher when the detector UAV is close to that area. However, considering the fact that there would be many user attraction points whose locations could be in different parts of the environment, and some of those parts may have similar user densities, sensing levels can turn out to have identical or similar values for different points in the environment. Thus, this fact should be taken into account in the localization phase in which we use DRL.

### 6.2.2. Localization

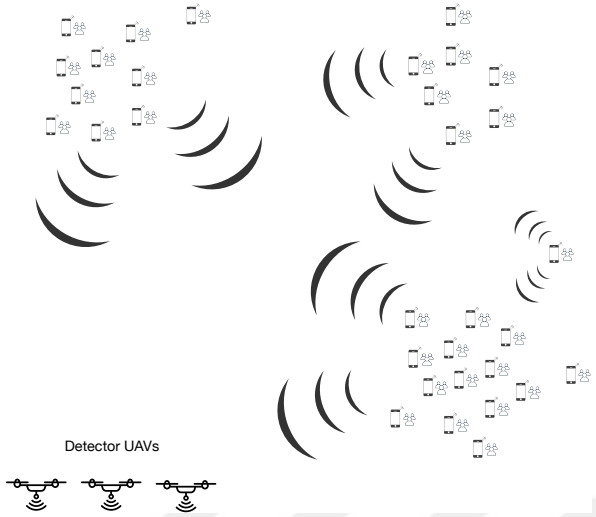
In the localization phase, information gathered in sensing is initially used for the movement of detector UAVs (agents) and then to locate the corresponding user attraction points. One of the crucial factors here is that the number of user attraction points in the environment is also unknown along with the number of users, and user locations. Therefore, we cannot apply multiple agents simultaneously in the environment since if the number of deployed agents is less than the number of user attraction points then users at some of the attraction points cannot be detected and served. Thus, we apply

```
1: Input: Threshold and the environment
2: Output: Found Locations
3: isThresholdMet = True
4: locations = [ ]
5: while isThresholdMet do
6:    singleLocation, connectionCount = DQN()
7:    if connectionCount is smaller than Threshold  then
8:       isThresholdMet = False
9:    else
10:       Add singleLocation to locations
11:    end if
12: end while
13: return  locations
```

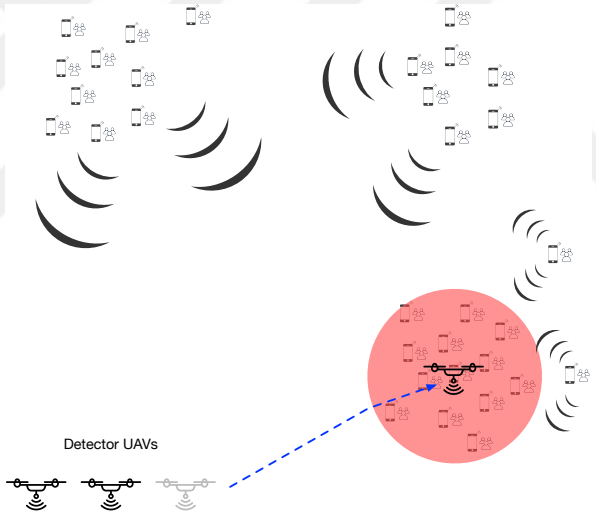
Figure 6.2. The algorithm of finding locations.

an iterative approach using DRL in the algorithm given in Figure 6.2.

In the algorithm given in Figure 6.2, we find the user attraction points in the environment. To perform this, we initially set a threshold for new connected users in an iteration. For each iteration we send a DRL agent to the environment flying from the base at $(0, 0)$ coordinates, and after the convergence it returns the corresponding location information along with how many new users are connected to it. If the number of connections is smaller than the threshold, the execution of the algorithm is terminated. Otherwise, we continue to send an agent to the environment. Note that when a user device connects to a detector UAV, it stops emitting the connection signal. As a result, the additive signal power would be less in a particular place in the next iteration for the agent. This situation is depicted in Figure 6.3. As a result, the time complexity of the algorithm is based on the multiplication of the Q-network parameters, state values, number of episodes, and number of users, considering the




(a) Environment consisting of users that emits connection signals.




(b) Emitted signals of connected users stop after the connection.

Figure 6.3. Depiction of the state of environment regarding the emitted signals.

worst-case. On the other hand, the space complexity is based on the number of user attraction points which, in the worst case, depends on the number of users.

6.2.2.1. MDP Definition. The DRL is based on MDP which is formally defined as a 4-tuple $< S, A, P, R >$ where

- $S$ is the set of states where $s \in S$

- $A$ is the set of actions where $a \in A$
- $P : S \times A \rightarrow P(S)$ is the state transition probability function that provides the probability of $P(s_t, s_{t+1}) = P(s_{t+1} = s' \mid s_t = s, a_t = a)$. This function denotes that the current state $s_t$ at time $t$ changes to the state $s_{t+1}$ by taking the action $a$.
- $R : S \times A \times S \rightarrow R$ is the reward function. It defines the corresponding reward $\mathcal{R}$ at time $t$ by taking an action $a$ in a state $s_t$. Therefore, it can be also defined as $\mathcal{R}_t = R(s_t, a_t, s_{t+1})$.

Our environment provides the corresponding MDP definition through the state representation, action space, reward mechanism, and state transition. Therefore, an applied DRL algorithm in the environment using a detector UAV can create a policy $\pi$ based on a given state as $\pi(a \mid s) = P(a_t = a \mid s_t = s)$. Through this policy, the agent can learn the dynamics in the environment for a given state and therefore takes the most effective action.

6.2.2.2. State Space $S$. Based on the algorithm given in Figure 6.2, there is only a single agent in the environment for each iteration. Therefore, the state at time $t$ consists of the current location of the agent as

$$s(t) = u_n(t) = [x_n(t), y_n(t), z_n(t)] \ . \tag{6.18}$$

6.2.2.3. Action Space $A$. In our environment, the action space $A$ consists of five discrete actions considering the horizontal movements. Therefore, it is defined as

$$a(t) = [Left, Right, Up, Down, NoMove] \ . \tag{6.19}$$

Based on this definition, the horizontal speed of each agent is fixed during their flight. Considering the selected action at time $t$, they can stay fixed at their horizontal

coordinates by $NoMove$ action. Otherwise, they can move into four different areas of the environment.

6.2.2.4. Reward Function $R$. Based on the policy $\pi$, the agent takes the corresponding actions in the environment to maximize its cumulative reward. To this end, for a given state $s_t$, the agent maximizes the expected sum of future reward by applying policy $\pi(s_t)$ as follows

$$R_t = \sum_{i=t}^{\infty} \gamma * R(s_i, s_{i+1}) \tag{6.20}$$

where $\gamma \in [0, 1]$ is the discount factor that denotes the importance of the long-term rewards if its value is close to one. Otherwise, its value would be close to zero. Thus, the reward function is defined as follows based on the consideration above

$$R(t) = \begin{cases} H(t), & \text{if} \quad \text{satisfying constraints} \\ -1, & \text{if} \quad otherwise\ . \end{cases} \tag{6.21}$$

6.2.2.5. Application of DRL. Since our action space is discrete, applying a value-based DRL algorithm is more convenient in our environment. Therefore, for each DRL agent, we implement Deep Q-Learning (DQN) algorithm [146], which manifests a high success in many different environments with different state spaces.

In value-based DRL algorithms such as DQN, the agent should select the best state-action pair among different options through its policy by a given state $s_t$ by maximizing Q-function, $Q_\pi(s_t, a_t)$. Therefore, under the policy $\pi$, the Q-function is defined as

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[R_t \mid s_i = s, a_i = a\right] \tag{6.22}$$

which denotes the value of an action $a_t$ in a state $s_t$. Thus, an optimal policy is defined

as selecting the highest valued action in each state,

$$\pi(s_t) = arg\max_{a'}(Q(s_t, a')) \tag{6.23}$$

where $a'$ indicates the set of all possible actions. As a result, the value of a state-action pair is computed as

$$z_t = R_{a_t}(s_t, s_{t+1}) + \gamma * Q(s_{t+1}, arg\max_{a'} Q(s_{t+1}, a'; \theta_t); \theta_t) \tag{6.24}$$

where $\theta_t$ represents the Q-network parameters. On the other hand, considering the convergence through the Q-network, the agent minimize the temporal difference error, $\delta_t$, of $Q_\pi(s_t, a_t)$ regarding $z_t$:

$$\delta_t = \mid Q(s_t, a_t) - z_t \mid \ . \tag{6.25}$$

### 6.2.3. Resource Allocation

After the completion of sensing and localization phases through detector UAVs, the next step is the resource allocation for serving UAVs which provide computational serving capabilities. Since a serving UAV has a limited capacity, measuring how many of them should be deployed is the main problem in this phase.

As users have already connected to the corresponding detector UAVs, and have started to send their task offloading requests, the system can create a task profile in those user-connected areas by conducting several capacity calculations. To this end, we perform a capacity calculation that includes the task profile of each user as we defined $W_m = (D_m, C_m, \lambda_m, T_m^{max})$. The measurements are essentially based on Equation 6.8 as the delay at serving UAVs is based on $M/M/1$ queueing model.

After the measurement of the required number of serving UAVs for each detected

area, the other important issue is the deployment of available serving UAVs into those areas, each of which may have different task profiles. Note that the available serving UAVs may not meet the total capacity requirements in the environment. In this case, available serving UAVs are first deployed to the areas which have a higher need for serving UAVs regarding task profiles. On the other hand, if the required numbers of serving UAVs are equal for the corresponding areas, then a round-robin approach is applied for the deployment. Note that our primary goal for this computation and then deployment is to maximize the overall task success in the environment as defined in our objective function in Equation 6.17.

### 6.2.4. MEC

After the resource allocation, the next and final phase is providing the MEC services to users. To this end, users offload their tasks to serving UAVs and expect a service without violating the task's maximum tolerable delay, $T_m^{max}$.

A user in a corresponding area can connect to multiple serving UAVs simultaneously in the environment. Therefore, a user should select one of its connected serving UAVs at a particular time to offload the corresponding task. It performs this selection based on the existing queueing condition for each serving UAV. Note that since multiple users can connect to a single serving UAV, the $M/M/1$ queueing model is used for the total delay measurement at the serving UAV as explained in Section 6.1. Here, we assume that a user can receive the recent queueing delay information from each serving UAV to which it is connected via a separate channel. Thus, it selects a serving UAV for task offloading considering the minimum queueing delay. As a result, our objective function defined in Equation 6.17 can be provided by minimizing the total delay regarding task profiles in the long term.

Table 6.1. Simulation parameters.

| **Parameter** | **Value** |
|---|---|
| Number of User Attraction Points | [3, 5] |
| Size of a task ($D_m$) | 500 Kb |
| Required CPU cycles of a task ($C_m$) | 90 cycles/bit |
| Arrival rate of a task ($\lambda_m$) | 0.30 task/sec |
| Maximum tolerable delay of a task ($T_m^{max}$) | 1 sec |
| Capacity of a serving UAV ($f_{n_s}$) | 300 cycles/sec |
| UAV Radius ($r_n$) | 75 m |
| Data Rate ($v_{m,n_s}$) | 100 Mbps |
| Channel Power Gain ($g_{m,n_d}$) | $1.42 \times 10^{-4}$ |
| New Connected User Threshold | 1 |
| $X_{max}$ | 500 m |
| $Y_{max}$ | 500 m |
| Altitude of UAVs ($z_n(t)$) | 200 m |

## 6.3. Performance Evaluation

We conducted experiments using a discrete event simulation for the performance evaluation. In these experiments, we have an environment whose size is $500 \times 500$ $m^2$. In this environment, there are various number of user attraction points around which the user densities are higher. Note that the location of those users and attraction points are initially not known by the system. The corresponding simulation parameters are given in Table 6.1. Throughout the experiments, we used Python 3.10. Moreover, we used PyTorch version 2.2.0 for the training of DRL agents.

In our experiments, we assumed that each user in the environment produces a task with the parameters $D_m$, $C_m$, $\lambda_m$, and $T_m^{max}$. Moreover, each produced task should be offloaded to one of the serving UAVs since we assumed that the computational capabilities of user devices are not sufficient to meet the task requirements. Similarly, each detector and serving UAV is identical in terms of radius, and altitude. Considering the offloading, we ignored the propagation delay for simplicity. Note that our essential goal was to maximize the overall task success as defined in Equation 6.17. We repeated our experiments 50 times with different seeds. The duration of each experiment in

Table 6.2. Hyperparameters of DRL algorithm.

| **Parameter** | **Value** |
|---|---|
| Learning Rate | 0.005 |
| Discount Factor | 0.99 |
| Initial Exploration | 1 |
| Exploration Factor | 0.99 |
| Final Exploration | 0.01 |
| Replay Memory Size | 1000000 |
| Batch Size | 64 |

simulator time was 1000 seconds.

### 6.3.1. Training Step

Regarding algorithm given in Figure 6.2, we train a DRL agent through detector UAVs for each iteration as long as it provides new connections. To this end, we tested several hyperparameters throughout the training step in order to achieve convergence for different user distributions in the environment considering the final model. Generally, we used random search based on our experience in the domain [147].

The neural network in our final model consists of three layers each of which includes 128 neurons. Rectified Linear Unit (ReLU) is applied for each neuron as the activation function. We used mean squared error (MSE) for the loss, and stochastic gradient descent (SGD) as the optimizer. On the other hand, for the initial exploration, exploration factor, final exploration, and replay memory size, we used well-known values from the literature as given in Table 6.2.

Since the learning rate is the most crucial hyperparameter for the performance of the model, we exclusively tested different settings. Thus, we determined 0.005 as the final value for the neural network of the agents. Figure 6.4 shows the effect of two different learning rates over episodes. As shown in the figure, a lower value of learning rate, such as 0.0005, causes a late convergence in the environment. On the other hand, if we use 0.005 as the learning rate, the model converges earlier which provides time

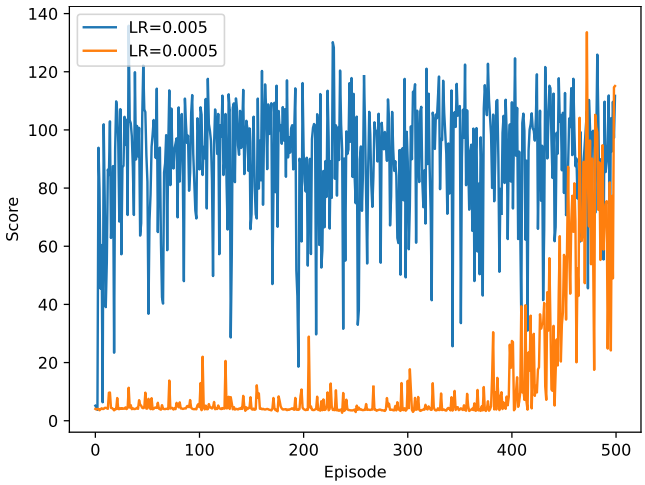



Figure 6.4. Effect of learning rate on scores for each episode using 100 users.

efficiency.

### 6.3.2. Competitors

We used two competitors as benchmark methods namely the Community Flying (CF) and Random methods similar to [148]. In CF, we divided the environment into equal communities, and the center of each community was evaluated as a possible user attraction point. Accordingly, detector UAVs are sent to those centers for possible connections and corresponding QoS measurements. Afterwards, the required number of serving UAVs is deployed based on the needs of those areas. On the other hand, in the random method, the possible attraction centers are selected randomly in the environment based on the available number of detector UAVs. We named the random methods based on the available number of detector UAVs as in the case of CF. To this end, for example, if we use 8 detector UAVs, then the method is named as Random-8.

### 6.3.3. Performance Experiments

We first evaluate the performance of DeepAir considering the effect of varying numbers of users, and different serving UAVs. As shown in Figure 6.5, the success rate of DeepAir is quite high based on the successfully placed detector UAVs through DRL.

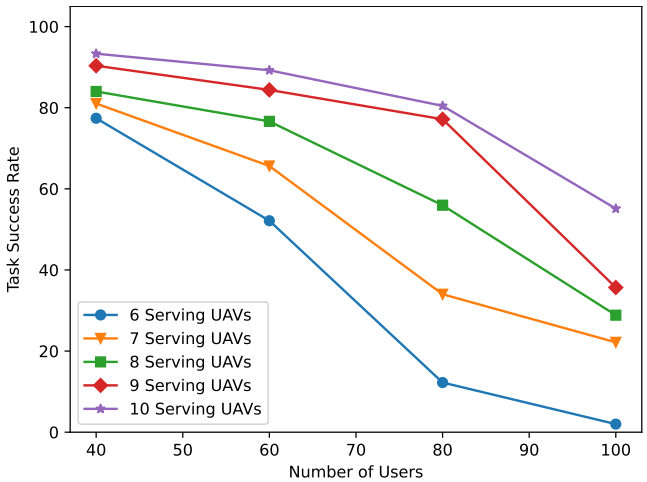



Figure 6.5. Effect of the number of users and serving UAVs on DeepAir.

Note that based on the configuration in the experiments, at most six detector UAVs are used when we apply DeepAir. On the other hand, when the number of users increases based on the same number of attraction points, the task success rate for each serving UAV case decreases. This is an expected result since the computational capacity of those serving UAVs would not be sufficient to meet the higher number of tasks produced by each user in the environment. Similarly, a higher number of serving UAVs results in a higher task success rate since they provide more computational capacity.

Prior to the evaluation of the performance of benchmark methods, we first conducted experiments to observe their success rate using different numbers of detector UAVs. To this end, as shown in Figure 6.6, we compared 4, 6, 8, and 16 detector UAV cases using 80 users. As expected, using an increased number of detector UAVs provided a higher task success rate since the probability of covered users in the environment is higher in that case. Therefore, we used CF-16 and Random-16 as the benchmark methods in the experiments.

The performance of Random-16 and CF-16 methods based on different numbers of users and serving UAVs are shown in Figure 6.7. The results manifest that CF-16 provides a better task success rate since it is a more structured approach. However, even though random decisions are taken in Random-16, when we use 9-10 serving

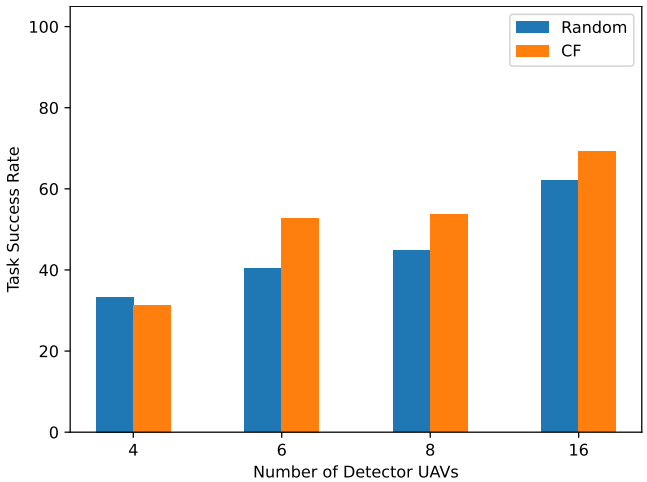
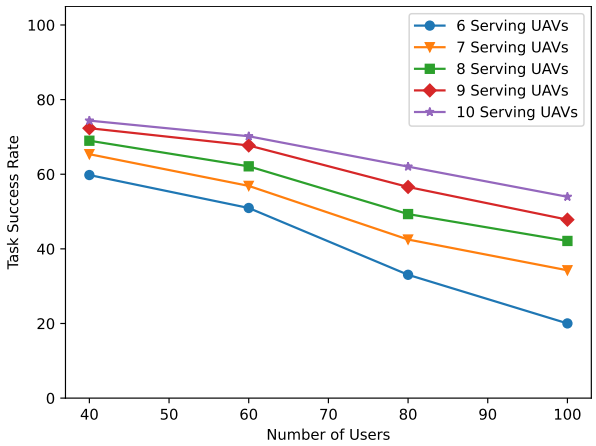


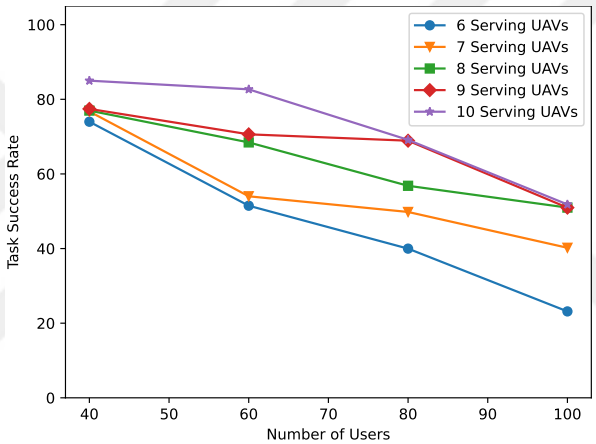

Figure 6.6. Effect of the number of detector UAVs on benchmark methods using 80 users.

UAVs, the task success rate is above 70% which is quite good. On the other hand, when the number of users increases, the task success rate decreases in both methods for the same reasons that we elaborated on regarding the results of DeepAir.

After the evaluation of benchmark methods and DeepAir, next we compared their most successful deployments in which we used ten serving UAVs. Figure 6.8 shows the results of this comparison. Based on the results, DeepAir outperforms both benchmark methods for different user counts. Note that since the arrival rate for each task, $\lambda_m$, is 0.30 task/sec, each user produces 300 tasks on average for each experiment. Moreover, both benchmark methods deployed 16 detector UAVs in this comparison, while DeepAir utilized at most six detector UAVs. On the other hand, the task success rate is close for each method when there are 100 users in the environment. This is the result of the insufficient computational capacity of serving UAVs deployed in the environment. Therefore, even if the best locations are selected for detector UAVs, the task success rate cannot exceed a certain value under those circumstances. Moreover, since the accuracy of location selection in Random-16 and CF-16 methods is less than that of DeepAir, the less number of connected users are served better using ten serving UAVs. This situation results in close values in terms of the task success rate when there are 100 users in the environment. In summary, although the user locations are




(a) Performance of Random-16.




(b) Performance of CF-16.

Figure 6.7. Results of benchmark methods.

successfully detected, if the number of serving UAVs is not sufficient for a certain load, the task success rate improvement requires more UAVs.

Since the relationship between the number of detector UAVs and serving UAVs is also significant, and Figure 6.6 depicts only the performance of benchmark methods, we conducted additional tests. To this end, we configured an experiment based on the same parameters as given in Table 6.1. However, in this case, to evaluate and show the corresponding relationship more clearly, we fixed the number of user attraction points and users as five, and 50, respectively. To perform a fair evaluation, each user attraction point included 10 users along with different user distributions. We tested the effect of 2, 4, and 6 detector UAVs by deploying 8, 10, and 12 serving UAVs.

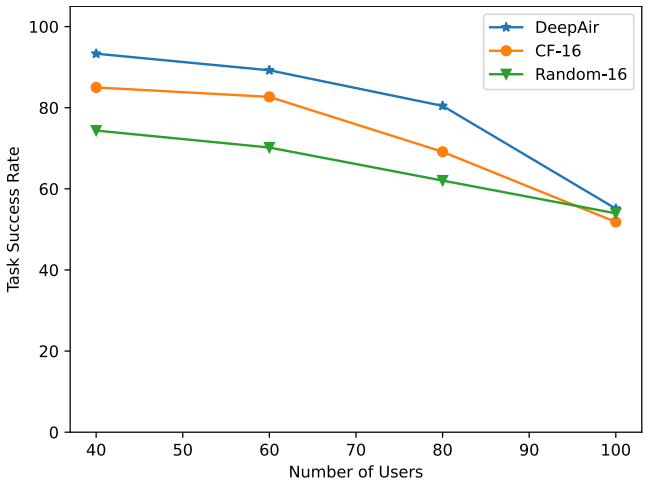



Figure 6.8. Performance comparison between DeepAir and benchmark methods using 10 serving UAVs. While CF-16 and Random-16 use 16 detector UAVs, DeepAir outperforms them using at most six detector UAVs.

We first evaluated the percentage of connected users considering the number of detector UAVs based on each method as shown in Figure 6.9. The results show that when the number of detector UAVs increases, more users can be connected to the system as expected. As DeepAir uses a smart approach through DRL, it finds user locations efficiently even using a small numbers of detector UAVs. Therefore, it outperforms benchmark methods. On the other hand, considering the performance of benchmark methods, as detector UAVs are placed more strategically in the CF method, it detects users better than the Random method.

Next, we evaluated the task success rate by deploying 8, 10, and 12 serving UAVs based on the connected users by detector UAVs. Each user produces 300 tasks on average for each simulation, hence approximately 15000 tasks were produced for each experiment in total. The results given in Figure 6.10 show the relationship between the detector and serving UAVs more clearly. When there is a higher number of serving UAVs in the environment, the task success rate is close to the percentage of connected users. Moreover, the task success rate follows the pattern of the connected users through detector UAVs regarding different numbers of serving UAVs. Therefore, increasing the number of serving UAVs until the total capacity requirements are met

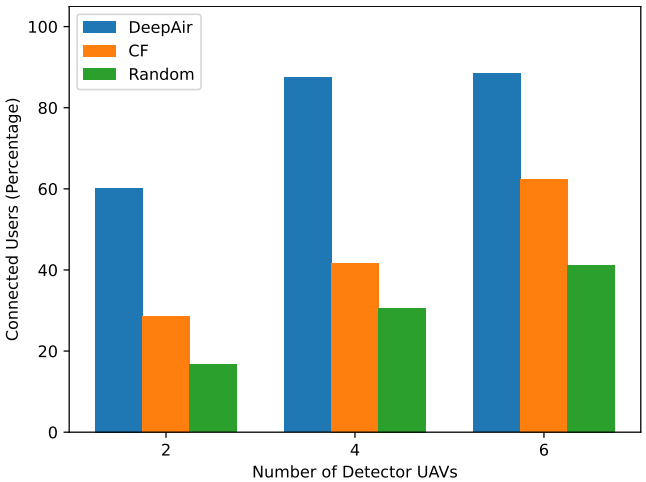



Figure 6.9. The effect of detector UAVs on the connected users.

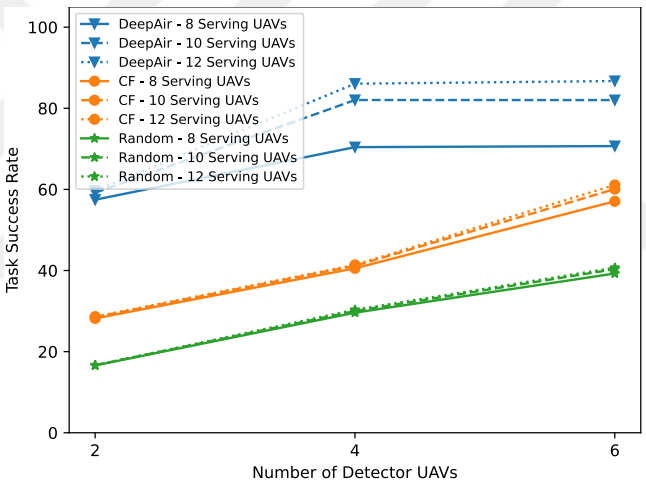



Figure 6.10. The effect of detector and serving UAVs on task success rate using 50 users.

in a detected area would increase the task success rate consistently.

## 6.4. Discussion

Considering the fact that the unknown user location problem has not been deeply investigated in the literature, we think that several points should be discussed based on our observations and experiments throughout this study. We first noted about the discrete action space for each type of UAV. Since continuous horizontal actions would complicate the already complex problem regarding DRL, we applied a discrete action

space. Moreover, and more importantly, applying a discrete action space alleviates the problem since it turns that into a maze problem in which there are several different prizes (RSSI power) in different sections of the environment. The agent learns to follow those small prizes to reach a bigger prize through episodes. Therefore, the convergence of the agents would be more quickly compared to the case of a continuous action space. As a result we could apply the DQN algorithm, which is a less complex regarding other DRL algorithms such as PPO, and DDPG.

Throughout our experiments, we also observed that higher RSSI due to a bigger number of users provides more accurate location prediction in DRL. As a result of this, more users can connect to the corresponding detector UAVs. Therefore, if the capacity of serving UAVs is so high, it is not so affected by the number of users, then the task success rate would be larger even though the load is increased. This is an important observation since, otherwise, the corresponding results would be evaluated incorrectly. For this reason, the selection of the capacity of serving UAVs and the required CPU cycles for user tasks are significant for manifesting the accuracy of the experiments.

# 7. CONCLUSION

In this study, we defined a novel, next-generation computational paradigm called air computing. In the face of ever-growing application resource demands, air computing strives to solve bottlenecks and inefficiencies of the computational infrastructure by intelligently harmonizing 2D legacy terrestrial resources with novel 3D vertical networking technologies.

Air computing is indeed based on a family of technologies such as UAV, LAP, HAP, LEO, and edge computing. In order to give a complete overview, we first investigated air computing as a whole regarding its main features and how it contrasts with former systems such as edge and cloud computing. We then described the individual technological components and how they fit in the overall architecture.

Moreover, we examined the advantages that would be put forward by a possible air computing implementation regarding the QoS requirements of the next-generation applications and QoE expectations of end-users. Then, we elaborated on the potential use cases where the current paradigms experience difficulty in meeting the dynamic user demands. Afterwards, we analyzed the opportunities and the corresponding challenges in an overall context from the perspectives of both the end users and service providers.

Next, we implemented a DRL based task orchestrator, namely DeepEdge, for edge computing. Throughout the study, we assumed that the offloading decision had been taken by mobile devices, and the task must be offloaded to one of the edge servers in the network or to the cloud. Due to the different characteristics of each application, generated tasks by mobile users must be handled by the orchestrator considering the application requirements and the current network conditions. For this reason, we developed DeepEdge by applying the DDQN algorithm. Even though providing MDP and coping with the delayed action effect are challenging due to the real-time, and stochastic nature of the edge computing environment, we successfully implemented a

DRL agent based on DDQN for the orchestration. We conducted experiments in the EdgeCloudSim simulator by comparing our model with other approaches in the literature. Our results showed that DeepEdge has the capability to perform orchestration for different task types and outperforms its competitors.

In order to provide a development and research environment in which researchers can conduct experiments for the optimization of task offloading, mobility, and flying policies in an air computing environment, we designed and implemented a new simulator, AirCompSim. Considering the dynamic requirements of air computing, we developed a modular structure for the implementation so that users can extend the existing policies and event mechanisms based on their needs. Moreover, we designed a scenario to represent the capabilities of our simulator. To this end, we evaluated task success rate, service time of different server types, utilization of the resources, and application-based performance based on the varying number of users and UAVs. Our results showed that AirCompSim can operate well and therefore would be used as a testbed for air computing research.

Utilizing the capabilities of AirCompSim, we investigated the unknown user location problem in a UAV-assisted environment. The corresponding environment can be a disaster site, wilderness, or a rural area in which user devices cannot connect to any communication device and edge servers because of the lack of infrastructure. Moreover, each user device produces tasks that should be completed regarding their maximum tolerable delay which is not met by the computational capabilities of user devices. Therefore, those tasks should be offloaded to the related computational units. In order to achieve this in such an environment, sensing, localization, resource allocation, and MEC capabilities should be provided together, sequentially. Therefore, we proposed DeepAir, a novel approach which uses DRL iteratively via detector UAVs that are responsible for sensing, localization, and resource allocation. Afterwards, MEC features are provided to those connected users by serving UAVs. Conducted experiments show that DeepAir provides a high task success rate by using a small number of detector UAVs in the environment compared to the benchmark methods.

Regarding the research and corresponding studies we have performed through this thesis, one of the most important features of air computing is that it can provide dynamic capacity enhancement for any type of environment. Therefore, any system that may face a bottleneck due to insufficient capacity of network infrastructure, computing/caching server, or high mobility can use air computing and its related components in order to increase QoS and QoE. Thus, we believe that air computing can provide holistic solutions for the next-generation computer networks.

## 7.1. Future Work

Air computing can affect and therefore solve many different issues through its 3D networking structure. To this end, in the future, we first plan to take energy consumption into account since energy efficiency is crucial for the movements of the UAVs which would affect the performance. Therefore, we plan to efficiently optimize the trade-off between energy consumption and task success rate.

Even though the existing capabilities of AirCompSim provide many benefits, we plan to add new modules including energy calculations and also dynamically changing UAV radius based on the altitude. Moreover, we extend its structure considering a multi-threaded structure. Thus, it can be utilized more swiftly for different scenarios.

Finally, we plan to meet air computing with large language models (LLM). The ubiquitous execution of LLM-powered applications in a wide variety of environments through air computing can allow local devices to run LLM and Generative AI (GenAI) applications easily.

# REFERENCES


1. Mao, B., F. Tang, Y. Kawamoto and N. Kato, "Optimizing Computation Offloading in Satellite-UAV-Served 6G IoT: A Deep Learning Approach", *Network*, Vol. 35, No. 4, pp. 102–108, 2021.

2. Statista, 2022, "Internet of Things (IoT) and Non-IoT Active Device Connections Worldwide From 2010 to 2025 (in Billions)", `https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/`, accessed on March 07, 2025.

3. Khan, W. Z., M. Y. Aalsalem, M. K. Khan, M. S. Hossain and M. Atiquzzaman, "A Reliable Internet of Things Based Architecture for Oil and Gas Industry", *19th International Conference on Advanced Communication Technology (ICACT)*, pp. 705–710, Pyeongchang, South Korea, 2017.

4. Li, T., Y. Fan, Y. Li, S. Tarkoma and P. Hui, "Understanding the Long-Term Evolution of Mobile App Usage", *Transactions on Mobile Computing*, Vol. 22, No. 2, pp. 1213–1230, 2021.

5. Van Hasselt, H., A. Guez and D. Silver, "Deep Reinforcement Learning With Double Q-Learning", *Proceedings of the Conference on Artificial Intelligence*, Vol. 30, Phoenix, Arizona, USA, 2016.

6. Bai, Y., H. Zhao, X. Zhang, Z. Chang, R. Jäntti and K. Yang, "Towards Autonomous Multi-UAV Wireless Network: A Survey of Reinforcement Learning-Based Approaches", *Communications Surveys & Tutorials*, Vol. 25, No. 4, pp. 3038–3067, 2023.

7. Senyo, P. K., E. Addae and R. Boateng, "Cloud Computing Research: A Review of Research Themes, Frameworks, Methods and Future Research Directions",

*International Journal of Information Management*, Vol. 38, No. 1, pp. 128–139, 2018.

8. Laroui, M., B. Nour, H. Moungla, M. A. Cherif, H. Afifi and M. Guizani, "Edge and Fog Computing for IoT: A Survey on Current Research Activities & Future Directions", *Computer Communications*, Vol. 180, pp. 210–231, 2021.

9. Satyanarayanan, M., V. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *Pervasive Computing*, Vol. 8, No. 4, pp. 14–23, 2009.

10. Chen, X., L. Jiao, W. Li and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing", *Transactions on Networking*, Vol. 24, No. 5, pp. 2795–2808, 2015.

11. Cisco, "Fog Computing and The Internet of Things: Extend The Cloud to Where The Things Are", *Cisco White Paper*, 2017.

12. Baktir, A. C., A. Ozgovde and C. Ersoy, "How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions", *Communications Surveys & Tutorials*, Vol. 19, No. 4, pp. 2359–2391, 2017.

13. Mach, P. and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", *Communications Surveys & Tutorials*, Vol. 19, No. 3, pp. 1628–1656, 2017.

14. Mao, Y., C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective", *Communications Surveys & Tutorials*, Vol. 19, No. 4, pp. 2322–2358, 2017.

15. Horner, L. J., "Edge Strategies in Industry: Overview and Challenges", *Transactions on Network and Service Management*, Vol. 18, No. 3, pp. 2825–2831, 2021.

16. Dong, C., S. Hu, X. Chen and W. Wen, "Joint Optimization With DNN Partitioning and Resource Allocation in Mobile Edge Computing", *Transactions on Network and Service Management*, Vol. 18, No. 4, pp. 3973–3986, 2021.

17. Liang, Z., Y. Liu, T.-M. Lok and K. Huang, "Multi-Cell Mobile Edge Computing: Joint Service Migration and Resource Allocation", *Transactions on Wireless Communications*, Vol. 20, No. 9, pp. 5898–5912, 2021.

18. Chen, Z., Z. Zhou and C. Chen, "Code Caching-Assisted Computation Offloading and Resource Allocation for Multi-User Mobile Edge Computing", *Transactions on Network and Service Management*, Vol. 18, No. 4, pp. 4517–4530, 2021.

19. Yuan, P., S. Shao, L. Geng and X. Zhao, "Caching Hit Ratio Maximization in Mobile Edge Computing With Node Cooperation", *Computer Networks*, Vol. 200, p. 108507, 2021.

20. Feng, L., Y. Zhou, T. Liu, X. Que, P. Yu, T. Hong and X. Qiu, "Energy-Efficient Offloading for Mission-Critical IoT Services Using EVT-Embedded Intelligent Learning", *Transactions on Green Communications and Networking*, Vol. 5, No. 3, pp. 1179–1190, 2021.

21. Xue, J., Q. Hu, Y. An and L. Wang, "Joint Task Offloading and Resource Allocation in Vehicle-Assisted Multi-Access Edge Computing", *Computer Communications*, Vol. 177, pp. 77–85, 2021.

22. Bozorgchenani, A., F. Mashhadi, D. Tarchi and S. A. S. Monroy, "Multi-Objective Computation Sharing in Energy and Delay Constrained Mobile Edge Computing Environments", *Transactions on Mobile Computing*, Vol. 20, No. 10, pp. 2992–3005, 2020.

23. Chen, X., J. Zhang, B. Lin, Z. Chen, K. Wolter and G. Min, "Energy-Efficient Offloading for DNN-based Smart IoT Systems in Cloud-Edge Environments",

*Transactions on Parallel and Distributed Systems*, Vol. 33, No. 3, pp. 683–697, 2021.

24. Luo, Q., S. Hu, C. Li, G. Li and W. Shi, "Resource Scheduling in Edge Computing: A Survey", *Communications Surveys & Tutorials*, Vol. 23, No. 4, pp. 2131–2165, 2021.

25. Xia, S., Z. Yao, Y. Li and S. Mao, "Online Distributed Offloading and Computing Resource Management With Energy Harvesting for Heterogeneous MEC-Enabled IoT", *Transactions on Wireless Communications*, Vol. 20, No. 10, pp. 6743–6757, 2021.

26. Bahreini, T., H. Badri and D. Grosu, "Mechanisms for Resource Allocation and Pricing in Mobile Edge Computing Systems", *Transactions on Parallel and Distributed Systems*, Vol. 33, No. 3, pp. 667–682, 2021.

27. Zhang, J. and Q. Zhang, "Stackelberg Game For Utility-Based Cooperative Cognitiveradio Networks", *Proceedings of the Tenth International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 23–32, New Orleans, USA, 2009.

28. Roostaei, R., Z. Dabiri and Z. Movahedi, "A Game-Theoretic Joint Optimal Pricing and Resource Allocation for Mobile Edge Computing in NOMA-Based 5G Networks and Beyond", *Computer Networks*, Vol. 198, p. 108352, 2021.

29. Zhao, C., Y. Cai, A. Liu, M. Zhao and L. Hanzo, "Mobile Edge Computing Meets mmWave Communications: Joint Beamforming and Resource Allocation for System Delay Minimization", *Transactions on Wireless Communications*, Vol. 19, No. 4, pp. 2382–2396, 2020.

30. Wang, C.-X., F. Haider, X. Gao, X.-H. You, Y. Yang, D. Yuan, H. M. Aggoune, H. Haas, S. Fletcher and E. Hepsaydir, "Cellular Architecture and Key Technologies for 5G Wireless Communication Networks", *Communications Magazine*,

Vol. 52, No. 2, pp. 122–130, 2014.

31. Wang, Z., M. Goudarzi, M. Gong and R. Buyya, "Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments", *Future Generation Computer Systems*, Vol. 152, pp. 55–69, 2023.

32. Dziyauddin, R. A., D. Niyato, N. C. Luong, A. A. A. M. Atan, M. A. M. Izhar, M. H. Azmi and S. M. Daud, "Computation Offloading and Content Caching and Delivery in Vehicular Edge Network: A Survey", *Computer Networks*, Vol. 197, p. 108228, 2021.

33. Peng, G., H. Wu, H. Wu and K. Wolter, "Constrained Multi-Objective Optimization for IoT-enabled Computation Offloading in Collaborative Edge and Cloud Computing", *Internet of Things Journal*, Vol. 8, No. 17, pp. 13723–13736, 2021.

34. Neely, M. J., "Stochastic Network Optimization With Application to Communication and Queueing Systems", *Synthesis Lectures on Communication Networks*, Vol. 3, No. 1, pp. 1–211, 2010.

35. Chen, Y., X. Zhou, W. Wang, H. Wang, Z. Zhang and Z. Zhang, "Delay-Optimal Closed-Form Scheduling for Multi-Destination Computation Offloading", *Wireless Communications Letters*, Vol. 10, No. 9, pp. 1904–1908, 2021.

36. Feng, H., S. Guo, L. Yang and Y. Yang, "Collaborative Data Caching and Computation Offloading for Multi-Service Mobile Edge Computing", *Transactions on Vehicular Technology*, Vol. 70, No. 9, pp. 9408–9422, 2021.

37. Xu, J., Z. Wei, Z. Lyu, L. Shi and J. Han, "Throughput Maximization of Offloading Tasks in Multi-Access Edge Computing Networks for High-Speed Railways", *Transactions on Vehicular Technology*, Vol. 70, No. 9, pp. 9525–9539, 2021.

38. Zhang, Y. and H.-Y. Wei, "Risk-Aware Cloud-Edge Computing Framework for

Delay-Sensitive Industrial IoTs", *Transactions on Network and Service Management*, Vol. 18, No. 3, pp. 2659–2671, 2021.

39. Yang, B., X. Cao, J. Bassey, X. Li and L. Qian, "Computation Offloading in Multi-Access Edge Computing: A Multi-Task Learning Approach", *Transactions on Mobile Computing*, Vol. 20, No. 9, pp. 2745–2762, 2020.

40. Li, C., J. Liu, Q. Zhang and Y. Luo, "Efficient Cooperative Cache Management for Latency-Aware Data Intelligent Processing in Edge Environment", *Future Generation Computer Systems*, Vol. 123, pp. 48–67, 2021.

41. Song, Z., Y. Hao, Y. Liu and X. Sun, "Energy-Efficient Multiaccess Edge Computing for Terrestrial-Satellite Internet of Things", *Internet of Things Journal*, Vol. 8, No. 18, pp. 14202–14218, 2021.

42. Ghoorchian, S. and S. Maghsudi, "Multi-Armed Bandit for Energy-Efficient and Delay-Sensitive Edge Computing in Dynamic Networks With Uncertainty", *Transactions on Cognitive Communications and Networking*, Vol. 7, No. 1, pp. 279–293, 2020.

43. Zhou, X., S. Ge, T. Qiu, K. Li and M. Atiquzzaman, "Energy-Efficient Service Migration for Multi-User Heterogeneous Dense Cellular Networks", *Transactions on Mobile Computing*, Vol. 22, No. 2, pp. 890–905, 2021.

44. Lyu, J., Y. Zeng and R. Zhang, "UAV-aided Offloading for Cellular Hotspot", *Transactions on Wireless Communications*, Vol. 17, No. 6, pp. 3988–4001, 2018.

45. Gupta, L., R. Jain and G. Vaszkun, "Survey of Important Issues in UAV Communication Networks", *Communications Surveys & Tutorials*, Vol. 18, No. 2, pp. 1123–1152, 2015.

46. Shi, W., J. Li, N. Cheng, F. Lyu, S. Zhang, H. Zhou and X. Shen, "Multi-Drone 3-D Trajectory Planning and Scheduling in Drone-Assisted Radio Access

Networks", *Transactions on Vehicular Technology*, Vol. 68, No. 8, pp. 8145–8158, 2019.

47. Zhou, Y., N. Cheng, N. Lu and X. S. Shen, "Multi-UAV-Aided Networks: Aerial-Ground Cooperative Vehicular Networking Architecture", *Vehicular Technology Magazine*, Vol. 10, No. 4, pp. 36–44, 2015.

48. Jeong, S., O. Simeone and J. Kang, "Mobile Edge Computing via A UAV-Mounted Cloudlet: Optimization of Bit Allocation and Path Planning", *Transactions on Vehicular Technology*, Vol. 67, No. 3, pp. 2049–2063, 2017.

49. Zhao, Z., L. Pacheco, H. Santos, M. Liu, A. Di Maio, D. Rosário, E. Cerqueira, T. Braun and X. Cao, "Predictive UAV Base Station Deployment and Service Offloading With Distributed Edge Learning", *Transactions on Network and Service Management*, Vol. 18, No. 4, pp. 3955–3972, 2021.

50. Wang, L., K. Wang, C. Pan, W. Xu, N. Aslam and A. Nallanathan, "Deep Reinforcement Learning Based Dynamic Trajectory Control for UAV-Assisted Mobile Edge Computing", *Transactions on Mobile Computing*, Vol. 21, No. 10, pp. 3536–3550, 2021.

51. Li, Z., M. Chen, C. Pan, N. Huang, Z. Yang and A. Nallanathan, "Joint Trajectory and Communication Design for Secure UAV Networks", *Communications Letters*, Vol. 23, No. 4, pp. 636–639, 2019.

52. Liu, Q., L. Shi, L. Sun, J. Li, M. Ding and F. Shu, "Path Planning for UAV-Mounted Mobile Edge Computing With Deep Reinforcement Learning", *Transactions on Vehicular Technology*, Vol. 69, No. 5, pp. 5723–5728, 2020.

53. Wang, J., Z. Na and X. Liu, "Collaborative Design of Multi-UAV Trajectory and Resource Scheduling for 6G-Enabled Internet of Things", *Internet of Things Journal*, Vol. 8, No. 20, pp. 15096–15106, 2020.

54. Zhou, F., Y. Wu, R. Q. Hu and Y. Qian, "Computation Rate Maximization in UAV-Enabled Wireless-Powered Mobile-Edge Computing Systems", *Journal on Selected Areas in Communications*, Vol. 36, No. 9, pp. 1927–1941, 2018.

55. Motlagh, N. H., T. Taleb and O. Arouk, "Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives", *Internet of Things Journal*, Vol. 3, No. 6, pp. 899–922, 2016.

56. Seid, A. M., G. O. Boateng, B. Mareri, G. Sun and W. Jiang, "Multi-Agent DRL for Task Offloading and Resource Allocation in Multi-UAV Enabled IoT Edge Network", *Transactions on Network and Service Management*, Vol. 18, No. 4, pp. 4531–4547, 2021.

57. Apostolopoulos, P. A., G. Fragkos, E. E. Tsiropoulou and S. Papavassiliou, "Data Offloading in UAV-Assisted Multi-Access Edge Computing Systems Under Resource Uncertainty", *Transactions on Mobile Computing*, Vol. 22, No. 1, pp. 175–190, 2021.

58. Kahneman, D. and A. Tversky, "Prospect Theory: An Analysis of Decision Under Risk", *Handbook of the Fundamentals of Financial Decision Making: Part I*, chap. Chapter 6, pp. 99–127, World Scientific, 2013.

59. El Haber, E., H. A. Alameddine, C. Assi and S. Sharafeddine, "UAV-Aided Ultra-Reliable Low-Latency Computation Offloading in Future IoT Networks", *Transactions on Communications*, Vol. 69, No. 10, pp. 6838–6851, 2021.

60. Zhan, C., H. Hu, Z. Liu, Z. Wang and S. Mao, "Multi-UAV-Enabled Mobile-Edge Computing for Time-Constrained IoT Applications", *Internet of Things Journal*, Vol. 8, No. 20, pp. 15553–15567, 2021.

61. Zhao, N., Z. Ye, Y. Pei, Y.-C. Liang and D. Niyato, "Multi-Agent Deep Reinforcement Learning for Task Offloading in UAV-assisted Mobile Edge Comput-

ing", *Transactions on Wireless Communications*, Vol. 21, No. 9, pp. 6949–6960, 2022.

62. Diao, X., W. Yang, L. Yang and Y. Cai, "UAV-Relaying-Assisted Multi-Access Edge Computing With Multi-Antenna Base Station: Offloading and Scheduling Optimization", *Transactions on Vehicular Technology*, Vol. 70, No. 9, pp. 9495–9509, 2021.

63. Zeng, Y., D. Lu and J. Du, "Joint Optimized Multi-User Access and UAV Deployments Based on Heterogeneous Revenue in IoT Network", *Computer Networks*, Vol. 234, p. 109919, 2023.

64. Shi, J., C. Li, Y. Guan, P. Cong and J. Li, "Multi-UAV-Assisted Computation Offloading in DT-Based Networks: A Distributed Deep Reinforcement Learning Approach", *Computer Communications*, Vol. 210, pp. 217–228, 2023.

65. Ji, J., K. Zhu, C. Yi and D. Niyato, "Energy Consumption Minimization in UAV-Assisted Mobile-Edge Computing Systems: Joint Resource Allocation and Trajectory Design", *Internet of Things Journal*, Vol. 8, No. 10, pp. 8570–8584, 2020.

66. Li, M., N. Cheng, J. Gao, Y. Wang, L. Zhao and X. Shen, "Energy-Efficient UAV-Assisted Mobile Edge Computing: Resource Allocation and Trajectory Optimization", *Transactions on Vehicular Technology*, Vol. 69, No. 3, pp. 3424–3438, 2020.

67. Chen, X., X. Liu, Y. Chen, L. Jiao and G. Min, "Deep Q-Network Based Resource Allocation for UAV-Assisted Ultra-Dense Networks", *Computer Networks*, Vol. 196, p. 108249, 2021.

68. Liu, Z., X. Tan, M. Wen, S. Wang and C. Liang, "An Energy-Efficient Selection Mechanism of Relay and Edge Computing in UAV-Assisted Cellular Networks",

*Transactions on Green Communications and Networking*, Vol. 5, No. 3, pp. 1306–1318, 2021.

69. Li, B., W. Liu, W. Xie and X. Li, "Energy-efficient Task Offloading and Trajectory Planning in UAV-Enabled Mobile Edge Computing Networks", *Computer Networks*, Vol. 234, p. 109940, 2023.

70. Borralho, R., A. Mohamed, A. U. Quddus, P. Vieira and R. Tafazolli, "A Survey on Coverage Enhancement in Cellular Networks: Challenges and Solutions for Future Deployments", *Communications Surveys & Tutorials*, Vol. 23, No. 2, pp. 1302–1341, 2021.

71. Mozaffari, M., W. Saad, M. Bennis and M. Debbah, "Unmanned Aerial Vehicle With Underlaid Device-to-Device Communications: Performance and Tradeoffs", *Transactions on Wireless Communications*, Vol. 15, No. 6, pp. 3949–3963, 2016.

72. Lyu, J., Y. Zeng, R. Zhang and T. J. Lim, "Placement Optimization of UAV-Mounted Mobile Base Stations", *Communications Letters*, Vol. 21, No. 3, pp. 604–607, 2016.

73. Alzenad, M., A. El-Keyi, F. Lagum and H. Yanikomeroglu, "3D Placement of An Unmanned Aerial Vehicle Base Station (UAV-BS) for Energy-Efficient Maximal Coverage", *Wireless Communications Letters*, Vol. 6, No. 4, pp. 434–437, 2017.

74. Liu, C. H., Z. Chen, J. Tang, J. Xu and C. Piao, "Energy-Efficient UAV Control for Effective and Fair Communication Coverage: A Deep Reinforcement Learning Approach", *Journal on Selected Areas in Communications*, Vol. 36, No. 9, pp. 2059–2070, 2018.

75. Wang, Y., Z.-Y. Ru, K. Wang and P.-Q. Huang, "Joint Deployment and Task Scheduling Optimization for Large-Scale Mobile Users in Multi-UAV-Enabled Mobile Edge Computing", *Transactions on Cybernetics*, Vol. 50, No. 9, pp. 3984–

3997, 2019.

76. Yuan, T., C. E. Rothenberg, K. Obraczka, C. Barakat and T. Turletti, "Harnessing UAVs for Fair 5G Bandwidth Allocation in Vehicular Communication via Deep Reinforcement Learning", *Transactions on Network and Service Management*, Vol. 18, No. 4, pp. 4063–4074, 2021.

77. Abdelhakam, M. M., M. M. Elmesalawy, I. I. Ibrahim and S. G. Sayed, "Collaborative CoMP and Trajectory Optimization for Energy Minimization in Multi-UAV-Assisted IoT Networks With QoS Guarantee", *Computer Networks*, Vol. 237, p. 110074, 2023.

78. Kurt, G. K., M. G. Khoshkholgh, S. Alfattani, A. Ibrahim, T. S. Darwish, M. S. Alam, H. Yanikomeroglu and A. Yongacoglu, "A Vision and Framework for The High Altitude Platform Station (HAPS) Networks of the Future", *Communications Surveys & Tutorials*, Vol. 23, No. 2, pp. 729–779, 2021.

79. Qiu, J., D. Grace, G. Ding, M. D. Zakaria and Q. Wu, "Air-Ground Heterogeneous Networks for 5G and Beyond via Integrating High and Low Altitude Platforms", *Wireless Communications*, Vol. 26, No. 6, pp. 140–148, 2019.

80. Kodheli, O., E. Lagunas, N. Maturo, S. K. Sharma, B. Shankar, J. F. M. Montoya, J. C. M. Duncan, D. Spano, S. Chatzinotas and S. Kisseleff, "Satellite Communications in the New Space Era: A Survey and Future Challenges", *Communications Surveys & Tutorials*, Vol. 23, No. 1, pp. 70–109, 2020.

81. Liu, J., Y. Shi, Z. M. Fadlullah and N. Kato, "Space-Air-Ground Integrated Network: A Survey", *Communications Surveys & Tutorials*, Vol. 20, No. 4, pp. 2714–2741, 2018.

82. Tang, F., H. Hofner, N. Kato, K. Kaneko, Y. Yamashita and M. Hangai, "A Deep Reinforcement Learning-Based Dynamic Traffic Offloading in Space-Air-Ground

Integrated Networks (SAGIN)", *Journal on Selected Areas in Communications*, Vol. 40, No. 1, pp. 276–289, 2021.

83. Chen, H., M. Xiao and Z. Pang, "Satellite-Based Computing Networks With Federated Learning", *Wireless Communications*, Vol. 29, No. 1, pp. 78–84, 2022.

84. Guo, Y., Q. Li, Y. Li, N. Zhang and S. Wang, "Service Coordination in the Space-Air-Ground Integrated Network", *Network*, Vol. 35, No. 5, pp. 168–173, 2021.

85. Zhou, C., W. Wu, H. He, P. Yang, F. Lyu, N. Cheng and X. Shen, "Delay-Aware IoT Task Scheduling in Space-Air-Ground Integrated Network", *Global Communications Conference (GLOBECOM)*, pp. 1–6, Waikoloa, HI, USA, 2019.

86. Cheng, N., F. Lyu, W. Quan, C. Zhou, H. He, W. Shi and X. Shen, "Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach", *Journal on Selected Areas in Communications*, Vol. 37, No. 5, pp. 1117–1129, 2019.

87. Zhang, Z., W. Zhang and F.-H. Tseng, "Satellite Mobile Edge Computing: Improving QoS of High-Speed Satellite-Terrestrial Networks Using Edge Computing Techniques", *Network*, Vol. 33, No. 1, pp. 70–76, 2019.

88. Chen, L., F. Tang and X. Li, "Mobility and Load-Adaptive Controller Placement and Assignment in LEO Satellite Networks", *International Conference on Computer Communications (INFOCOM)*, pp. 1–10, Virtual Conference, 2021.

89. Zhang, Y., H. Zhang, H. Zhou, K. Long and G. K. Karagiannidis, "Resource Allocation in Terrestrial-Satellite-Based Next Generation Multiple Access Networks With Interference Cooperation", *Journal on Selected Areas in Communications*, Vol. 40, No. 4, pp. 1210–1221, 2022.

90. Dahrouj, H., S. Liu and M.-S. Alouini, "Machine Learning-Based User Scheduling

in Integrated Satellite-HAPS-Ground Networks", *Network*, Vol. 37, No. 2, pp. 102–109, 2023.

91. Sisinni, E., A. Saifullah, S. Han, U. Jennehag and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions", *Transactions on Industrial Informatics*, Vol. 14, No. 11, pp. 4724–4734, 2018.

92. Siddiqi, M. A., H. Yu and J. Joung, "5G Ultra-Reliable Low-Latency Communication Implementation Challenges and Operational Issues With IoT Devices", *Electronics*, Vol. 8, No. 9, p. 981, 2019.

93. Wang, H., G. Ding, F. Gao, J. Chen, J. Wang and L. Wang, "Power Control in UAV-Supported Ultra Dense Networks: Communications, Caching, and Energy Transfer", *Communications Magazine*, Vol. 56, No. 6, pp. 28–34, 2018.

94. Cheng, N., W. Xu, W. Shi, Y. Zhou, N. Lu, H. Zhou and X. Shen, "Air-Ground Integrated Mobile Edge Networks: Architecture, Challenges, and Opportunities", *Communications Magazine*, Vol. 56, No. 8, pp. 26–32, 2018.

95. Li, B., Z. Fei and Y. Zhang, "UAV Communications for 5G and Beyond: Recent Advances and Future Trends", *Internet of Things Journal*, Vol. 6, No. 2, pp. 2241–2263, 2018.

96. Ouyang, T., Z. Zhou and X. Chen, "Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing", *Journal on Selected Areas in Communications*, Vol. 36, No. 10, pp. 2333–2345, 2018.

97. Zhang, X. and Q. Zhu, "Hierarchical Caching for Statistical QoS Guaranteed Multimedia Transmissions Over 5G Edge Computing Mobile Wireless Networks", *Wireless Communications*, Vol. 25, No. 3, pp. 12–20, 2018.

98. Zhao, N., W. Lu, M. Sheng, Y. Chen, J. Tang, F. R. Yu and K.-K. Wong, "UAV-Assisted Emergency Networks in Disasters", *Wireless Communications*, Vol. 26,

No. 1, pp. 45–51, 2019.

99. Erdelj, M., E. Natalizio, K. R. Chowdhury and I. F. Akyildiz, "Help From The Sky: Leveraging UAVs for Disaster Management", *Pervasive Computing*, Vol. 16, No. 1, pp. 24–32, 2017.

100. Dong, P., X. Wang, S. Wang, Y. Wang, Z. Ning and M. S. Obaidat, "Internet of UAVs Based Remote Health Monitoring: An Online E-health System", *Wireless Communications*, Vol. 28, No. 3, pp. 15–21, 2021.

101. Ullah, S., K.-I. Kim, K. H. Kim, M. Imran, P. Khan, E. Tovar and F. Ali, "UAV-Enabled Healthcare Architecture: Issues and Challenges", *Future Generation Computer Systems*, Vol. 97, pp. 425–432, 2019.

102. Manuri, F. and A. Sanna, "A Survey on Applications of Augmented Reality", *Advances in Computer Science: An International Journal (ACSIJ)*, Vol. 5, No. 1, pp. 18–27, 2016.

103. Chatzopoulos, D., C. Bermejo, Z. Huang and P. Hui, "Mobile Augmented Reality Survey: From Where We Are to Where We Go", *Access*, Vol. 5, pp. 6917–6950, 2017.

104. Siriwardhana, Y., P. Porambage, M. Liyanage and M. Ylianttila, "A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects", *Communications Surveys & Tutorials*, Vol. 23, No. 2, pp. 1160–1192, 2021.

105. Montero, E., C. Rocha, H. Oliveira, E. Cerqueira, P. Mendes, A. Santos and D. Rosário, "Proactive Radio and QoS-Aware UAV as BS Deployment to Improve Cellular Operations", *Computer Networks*, Vol. 200, p. 108486, 2021.

106. Zhao, Z., P. Cumino, C. Esposito, M. Xiao, D. Rosário, T. Braun, E. Cerqueira and S. Sargento, "Smart Unmanned Aerial Vehicles as Base Stations Placement to

Improve the Mobile Network Operations", *Computer communications*, Vol. 181, pp. 45–57, 2022.

107. Fotouhi, A., H. Qiang, M. Ding, M. Hassan, L. G. Giordano, A. Garcia-Rodriguez and J. Yuan, "Survey on UAV Cellular Communications: Practical Aspects, Standardization Advancements, Regulation, and Security Challenges", *Communications Surveys & Tutorials*, Vol. 21, No. 4, pp. 3417–3442, 2019.

108. Huang, X., J. A. Zhang, R. P. Liu, Y. J. Guo and L. Hanzo, "Airplane-Aided Integrated Networking for 6G Wireless: Will It Work?", *Vehicular Technology Magazine*, Vol. 14, No. 3, pp. 84–91, 2019.

109. Hassija, V., V. Chamola, A. Agrawal, A. Goyal, N. C. Luong, D. Niyato, F. R. Yu and M. Guizani, "Fast, Reliable, and Secure Drone Communication: A Comprehensive Survey", *Communications Surveys & Tutorials*, Vol. 23, No. 4, pp. 2802–2832, 2021.

110. Abdalla, A. S., A. Yingst, K. Powell, A. Gelonch-Bosch and V. Marojevic, "Open Source Software Radio Platform for Research on Cellular Networked UAVs: It Works!", *Communications Magazine*, Vol. 60, No. 2, pp. 60–66, 2022.

111. Bor-Yaliniz, I., M. Salem, G. Senerath and H. Yanikomeroglu, "Is 5G Ready for Drones: A Look into Contemporary and Prospective Wireless Networks From A Standardization Perspective", *Wireless Communications*, Vol. 26, No. 1, pp. 18–27, 2019.

112. Wang, X., Y. Han, V. C. Leung, D. Niyato, X. Yan and X. Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey", *Communications Surveys & Tutorials*, Vol. 22, No. 2, pp. 869–904, 2020.

113. Chen, X., S. Leng, J. He and L. Zhou, "Deep-Learning-Based Intelligent Intervehicle Distance Control for 6G-Enabled Cooperative Autonomous Driving", *Internet*

*of Things Journal*, Vol. 8, No. 20, pp. 15180–15190, 2020.

114. Sun, C., X. Wu, X. Li, Q. Fan, J. Wen and V. C. Leung, "Cooperative Computation Offloading for Multi-Access Edge Computing in 6G Mobile Networks via Soft Actor-Critic", *Transactions on Network Science and Engineering*, Vol. 11, No. 6, pp. 5601–5614, 2021.

115. LeCun, Y., Y. Bengio and G. Hinton, "Deep Learning", *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

116. Nguyen, D. C., M. Ding, P. N. Pathirana, A. Seneviratne, J. Li and H. V. Poor, "Federated Learning for Internet of Things: A Comprehensive Survey", *Communications Surveys & Tutorials*, Vol. 23, No. 3, pp. 1622–1658, 2021.

117. Samarakoon, S., M. Bennis, W. Saad and M. Debbah, "Federated Learning for Ultra-Reliable Low-Latency V2V Communications", *Global Communications Conference (GLOBECOM)*, pp. 1–7, Abu Dhabi, UAE, 2018.

118. Chen, W., X. Qiu, T. Cai, H.-N. Dai, Z. Zheng and Y. Zhang, "Deep Reinforcement Learning for Internet of Things: A Comprehensive Survey", *Communications Surveys & Tutorials*, Vol. 23, No. 3, pp. 1659–1692, 2021.

119. He, H., S. Zhang, Y. Zeng and R. Zhang, "Joint Altitude and Beamwidth Optimization for UAV-Enabled Multiuser Communications", *Communications Letters*, Vol. 22, No. 2, pp. 344–347, 2017.

120. Huang, W., Z. Yang, C. Pan, L. Pei, M. Chen, M. Shikh-Bahaei, M. Elkashlan and A. Nallanathan, "Joint Power, Altitude, Location and Bandwidth Optimization for UAV With Underlaid D2D Communications", *Wireless Communications Letters*, Vol. 8, No. 2, pp. 524–527, 2018.

121. Wu, C., Z. Liu, D. Zhang, T. Yoshinaga and Y. Ji, "Spatial Intelligence Toward Trustworthy Vehicular IoT", *Communications Magazine*, Vol. 56, No. 10, pp.

22–27, 2018.

122. Luong, N. C., D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey", *Communications Surveys & Tutorials*, Vol. 21, No. 4, pp. 3133–3174, 2019.

123. Lin, Y., M. Wang, X. Zhou, G. Ding and S. Mao, "Dynamic Spectrum Interaction of UAV Flight Formation Communication with Priority: A Deep Reinforcement Learning Approach", *Transactions on Cognitive Communications and Networking*, Vol. 6, No. 3, pp. 892–903, 2020.

124. Lei, W., Y. Ye and M. Xiao, "Deep Reinforcement Learning Based Spectrum Allocation in Integrated Access and Backhaul Networks", *Transactions on Cognitive Communications and Networking*, Vol. 6, No. 3, pp. 970–979, 2020.

125. Li, Y., W. Zhang, C.-X. Wang, J. Sun and Y. Liu, "Deep Reinforcement Learning for Dynamic Spectrum Sensing and Aggregation in Multi-Channel Wireless Networks", *Transactions on Cognitive Communications and Networking*, Vol. 6, No. 2, pp. 464–475, 2020.

126. Gong, S., Y. Xie, J. Xu, D. Niyato and Y.-C. Liang, "Deep Reinforcement Learning for Backscatter-Aided Data Offloading in Mobile Edge Computing", *Network*, Vol. 34, No. 5, pp. 106–113, 2020.

127. Ning, Z., P. Dong, X. Wang, J. J. Rodrigues and F. Xia, "Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System", *Transactions on Intelligent Systems and Technology (TIST)*, Vol. 10, No. 6, pp. 1–24, 2019.

128. Sonmez, C., A. Ozgovde and C. Ersoy, "Fuzzy Workload Orchestration for Edge Computing", *Transactions on Network and Service Management*, Vol. 16, No. 2,

pp. 769–782, 2019.

129. Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, "Human-level Control Through Deep Reinforcement Learning", *Nature*, Vol. 518, No. 7540, p. 529, 2015.

130. Sonmez, C., A. Ozgovde and C. Ersoy, "EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems", *Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 39–44, Valencia, Spain, 2017.

131. Zhang, B., X. Lin, Y. Zhu, J. Tian and Z. Zhu, "Enhancing Multi-UAV Reconnaissance and Search Through Double Critic DDPG With Belief Probability Maps", *Transactions on Intelligent Vehicles*, Vol. 9, No. 2, pp. 3827–3842, 2024.

132. Hao, H., C. Xu, W. Zhang, S. Yang and G.-M. Muntean, "Joint Task Offloading, Resource Allocation, and Trajectory Design for Multi-UAV Cooperative Edge Computing with Task Priority", *Transactions on Mobile Computing*, Vol. 23, No. 9, pp. 8649–8663, 2024.

133. Shi, H., Y. Tian, H. Li, J. Huang, L. Shi and Y. Zhou, "Task Offloading and Trajectory Scheduling for UAV-Enabled MEC Networks: An MADRL Algorithm With Prioritized Experience Replay", *Ad Hoc Networks*, Vol. 154, p. 103371, 2024.

134. Yamansavascilar, B., A. Ozgovde and C. Ersoy, "Air Computing: A Survey on A New Generation Computation Paradigm", *Computer Networks*, Vol. 251, p. 110653, 2024.

135. Ebeid, E., M. Skriver, K. H. Terkildsen, K. Jensen and U. P. Schultz, "A Survey of Open-Source UAV Flight Controllers and Flight Simulators", *Microprocessors*

*and Microsystems*, Vol. 61, pp. 11–20, 2018.

136. Shah, S., D. Dey, C. Lovett and A. Kapoor, "Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles", *Field and Service Robotics: Results of the 11th International Conference*, pp. 621–635, Zurich, Switzerland, 2018.

137. DUrso, F., C. Santoro and F. F. Santoro, "An Integrated Framework for the Realistic Simulation of Multi-UAV Applications", *Computers & Electrical Engineering*, Vol. 74, pp. 196–209, 2019.

138. Koenig, N. and A. Howard, "Design and Use Paradigms for Gazebo, an Open-Source Multi-robot Simulator", *International Conference on Intelligent Robots and Systems (IROS)*, Vol. 3, pp. 2149–2154, Sendai, Japan, 2004.

139. ArduPilot, 2009, "ArduPilot Software", `https://ardupilot.org`, accessed on February 23, 2025.

140. Riley, G. F. and T. R. Henderson, "The NS-3 Network Simulator", *Modeling and Tools for Network Simulation*, pp. 15–34, Springer, 2010.

141. Baidya, S., Z. Shaikh and M. Levorato, "FlyNetSim: An Open Source Synchronized UAV Network Simulator Based on NS-3 and Ardupilot", *Proceedings of the 21st International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 37–45, Montreal, Canada, 2018.

142. Yamansavascilar, B., 2024, "AirCompSim: A Discrete Event Simulator for Air Computing", `https://github.com/Anamort/AirCompSim`, accessed on February 23, 2025.

143. McKinney, W., "Pandas: A Foundational Python Library for Data Analysis and Statistics", *Python for High Performance and Scientific Computing*, Vol. 14, No. 9, pp. 1–9, 2011.

144. Baltaci, A., E. Dinc, M. Ozger, A. Alabbasi, C. Cavdar and D. Schupke, "A Survey of Wireless Networks for Future Aerial Communications (FACOM)", *Communications Surveys & Tutorials*, Vol. 23, No. 4, pp. 2833–2884, 2021.

145. Xiao, Y., J. Liu, J. Wu and N. Ansari, "Leveraging Deep Reinforcement Learning for Traffic Engineering: A Survey", *Communications Surveys & Tutorials*, Vol. 23, No. 4, pp. 2064–2097, 2021.

146. Arulkumaran, K., M. P. Deisenroth, M. Brundage and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey", *Signal Processing Magazine*, Vol. 34, No. 6, pp. 26–38, 2017.

147. Bergstra, J. and Y. Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, Vol. 13, No. 1, pp. 281–305, 2012.

148. Ning, Z., Y. Yang, X. Wang, Q. Song, L. Guo and A. Jamalipour, "Multi-Agent Deep Reinforcement Learning Based UAV Trajectory Optimization for Differentiated Services", *Transactions on Mobile Computing*, Vol. 23, No. 5, pp. 5818–5834, 2024.